Query Result Size Estimation Techniques in Database Systems

by

Banchong Harangsri

A dissertation submitted to the

The University of New South Wales

School of Computer Science and Engineering

Sydney, NSW 2052, Australia

in fulfillment of the requirements

for the degree of

Doctor of Philosophy

April 1998

Abstract

Query optimisers are critical to the efficiency of modern relational database systems. If a query optimiser chooses a poor query execution plan, the performance of the database system in answering the query can be very poor. In fact, the differences in cost between the least and most expensive query execution plans can be several orders of magnitude. On the other hand, it can be prohibitively expensive for the query optimiser to search exhaustively for the least-cost (strictly optimal) query execution plan. Most query optimisers, therefore, compromise by using a reasonably cheap search to obtain a reasonably cheap query execution plan.

Accurate, but inexpensive, query size estimation is fundamental to the success of real query optimisers. A number of studies [Christodoulakis 1984; Ioannidis and Christodoulakis 1991, 1993] have demonstrated that optimisers can select very expensive query execution plans if they are forced to rely on poor or inaccurate query size estimates. This thesis will address the problem of how to obtain reliable and accurate query size estimation for the cost calculation of query execution plans.

The thesis focuses on improving query size estimation for the two main relational database operations: selection and join. We propose, analyse and compare a number of novel query size estimation techniques in order to come up with practical solutions which can be implemented with low overhead. The novel techniques we introduce range from machine learning, neural network, local regression and sampling-based techniques. The thesis also involves extensive analytical and experimental investigation of the effectiveness of the novel and traditional estimation techniques.

Our conclusions are that no one approach is "best" over the entire spectrum of database configurations (distributed vs. centralised, normal vs. skewed data distributions, etc). However, we have identified the strengths and weaknesses of all major techniques and analysed the time and space complexities required by those techniques, all of which can be used as a guide to the implementation of query optimisers in a variety of practical situations.

Our specific conclusions include:

- Sampling-based techniques typically give more accurate query size estimation than non-sampling-based techniques but require more run-time overhead. This run-time overhead may be unacceptable in a number of situations (e.g. in distributed databases where the cost of sampling over the network is prohibitive).

- The most accurate of the non-sampling-based techniques is local regression. Moreover, we show that it is a superset of the earlier well-known techniques such as histogram, parametric (namely, uniform-distribution-based) or curve-fitting, and is more flexible than those earlier techniques in the control of the degree of the polynomial and in the window type used.
- We propose a bootstrap method to improve the quality of join selectivities.
- We show that histograms built by a new sampling-based technique is more accurate in query size estimation than the histograms built by two traditional sampling-based techniques, i.e., simple random sampling with and without replacement. These two traditional techniques are ones typically used by most current database systems to build histograms.

Declaration

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any degree or diploma of a university or other institute of higher learning, except where due knowledge is made in the text of the thesis.

Banchong Harangsri

April, 1998

Acknowledgements

The four years of my PhD research at UNSW have had both good times and bad times. In spite of the bad times, I'm very, very pleased to have had great supervisors, a fabulous place to work (CSE@UNSW) and to have been able to live in Australia. The place is great, and I really don't want to leave!

I wish to express my heartfelt thanks to Dr. John Shepherd, my thesis supervisor and Dr. Anne Ngu, my co-supervisor.

To John, "carrot and stick" is the thing that you always use with me. Thanks for your patience; I hope you can avoid such recalcitrants in the future. You are one of the "coolest dudes" I've ever met. Despite the hard times and disappointments, I always seemed to learn something useful to improve myself from you.

Anne, you helped me become a knowledgeble researcher. You know that I started from nearly zero background, zero knowledge about database systems but you got me through my Masters degree, and I doubt that I would have been able to start my PhD without you? Your comments on my work really did help everything to work out nicely in the end.

I am indebted to the Thai government, more correctly to Thai people, for their taxes they have been paying for me to do my Masters and PhD over the last seven years. I'll try my best to give you back every cent's worth of it by using my knowledge to help improve Thailand in any way I can.

To dad, I'm really sorry that I kept you waiting so long (seven years) that you couldn't make it to the end. But dad, I've now managed to reach the day that you wanted to see and I wish that your soul can see it. Dad, if you hear me, this thesis is dedicated for you.

A big "thank you" goes to Bing Hiong for many good points about the Christian philosophy that she told me over the last five years. I'm sorry that I'm not yet converted ... but you nevertheless taught me many important things about life via quotes from the Bible!

Sunee, this small paragraph will convey nothing to fit your goodness here but I will attempt to. I really appreciated your time and effort in helping me proof-read my thesis and picking up many little mistakes. Thanks for your putting up with me in all those chats, both via `talk` and via the telephone. I'm sorry that many times I'd hit `^c` too soon to quit talking with you; I hope you understand that it was due to my thesis commitment.

I would also like to thank to Long, his wife Villa, his mother, his brother and everyone else in his family. They constantly cheered me up whenever I've felt depressed or have undergone any hardships in life or my study. You and Villa treated me like one in your family, and your generosity in feeding me almost every night was simply *too much*. My memories of shared meals with you will be one of my most treasured memories from Australia.

Chien-Chung, how can I forget to mention you here? I remember every single moment we spent together in the evening over the four years of your thesis. You are a kind of fun loving person who can make me smile and burst out laughing all the time. You would almost never get into anything serious. Thanks for all the talks at any time, night or day – no problem for you – and simply for being there.

To Shinichi, the best time of the day for us was when we went for a dinner together to a restaurant: Indian, Tum, Indonesian, Chinese, Korean, etc. All I had to do was name one ... anywhere ... and you'd drive us there. You've been great company and a great friend. And thanks for coming along to all those Chinese restaurants, even though you don't like Chinese food!

To Ruth, thanks for your constant encouragement whenever I had any hard times. You are always sincere and willing to help. I will never forget your offer to help financially if I ran out of time and scholarship. Thanks for all the good times at the movies with Shinichi. Of course, you know I'm an action fan, right :-)

Raymond, thanks for your moral support. Whenever I listened to your advice, I would strengthen myself to fight back against whatever difficulties I was confronting. Forgive me for looking arrogant and being ignorant :-)

Martin, thanks for being simply a nice person. I like the way you think and speak your mind. That made me understand myself better and sometimes I just have to accept the bitter facts, don't I?

Swee Yew, thanks for a number of recommendations you'd given me whenever I went to see you at your desk. You are absolutely a very approachable person whenever I want to ask something. I'll remember and value your friendship.

Thanks to Liping and her family for keeping asking (pushing) me about submitting my thesis. Thanks for cutting me off whenever I tried to talk for more than five minutes, and also for reminding that every single minute of my time from now until I hand in my thesis was precious (expensive, you call it the Chinese way) – "time is gold". Nice words.

Thanks to Larry Wall for Perl, Guido van Rossum for Python, B. Kernighan and D. Ritchie for C, Bjarne Stroustrup for C++, University of California at Berkeley for its Berkeley Database library, Donald E. Knuth for TeX, Leslie Lamport for LaTeX, B. Smith and S. Sutanthavibul for the XFig drawing package, Thomas Williams and Colin Kelley for their `gnuplot` graph plotting package, Richard Stallman for his excellent Emacs editor and C. R. Birchenhall for the MatClass library. Without all your good work, my thesis would not have seen the light of the day for another seven years. All of the tables and graphs for the copious experiments in this thesis were automatically generated by Perl.

Thanks guys for all the great help.

Tong

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Highlight

Despite the use of the exhaustive search algorithm, the "optimal" plan selected by an optimiser can be not truly optimal if the size estimation method used by the optimiser inaccurately estimates sizes of temporary intermediate relations. This is no matter what sizes (small or large) of search spaces are considered. This also in turn indicates whatever search algorithms used by an optimiser will be in vain if the size estimation method used by the optimiser inaccurately approximates sizes of intermediate relations.

1.1 Overview

Database management systems are large and complicated software systems, typically constructed from components such as: file management, transaction management, recovery and concurrency control, query processing, security and protection. Query processing is one of the critical components in any database system. An obvious reason is that queries provide one of the most common forms of interaction between humans and a database system. (We define queries as requests by users in some form of languages to a database system for some information stored in the database.) If the speed of answering queries is unacceptably slow, one of the primary goals of database management systems (timely provision of useful information) will *not* have been met.

With traditional navigational database systems, e.g. network and hierarchical, databases are linked-based, containing physical links to connect records together. To retrieve information from a database of this kind, a program to implement a user request for some information in the database must be written by a skilled programmer in some low-level *procedural* data manipulation language (DML) such as NDML (Network DML) and HDML (Hierarchical DML) in order to retrieve the information required by the user. The programmer is required to have knowledge of storage structures and access paths in order for efficient information access to be achieved. The programmer is thus given the opportunity to construct the "optimal" query execution plan (if at all possible) to retrieve data from storage. (A query execution plan is a plan which basically has a sequence to access data stored in database files.) Hence for these database systems, there is no need for extensive query optimisation to be done.

On the other hand, with the advent of relational systems accompanied by higher-level *declarative* languages such as SQL, QUEL and etc., users are able to specify what they want in a high-level query language – namely, what the desired result would be like, rather than focusing on the implementation details of how the result can be obtained. In a high-level declarative query, there is no clear exact order stated in the query of how the data should be accessed. On one hand, this facilitates the use of the language for users; they require no knowledge of storage structures and access paths in order to request a database system for some information. On the other hand, this also requires the relational database system to perform a non-trivial optimisation task on the user's behalf.

The example of university INGRES [INGRES 1988], a popular public domain relational database system, shows that query optimisation is a critical and complex part of the implementation of relational database systems. University INGRES consists of thirteen modules and two of the modules are dedicated for the query optimisation task. The entire system contains 59,967 lines of C source code with the two query optimisation modules having 10,782 lines of C code. In other words, about 18% of the entire database system is dedicated to the query optimisation task.

A query processing architecture [Elmasri and Navathe 1991] fundamentally comprises 4 main components as shown in Figure 1.1: Scanner and Parser, Query Rewriter, Query Optimiser and Query Code Generator. Here are the tasks of each component:

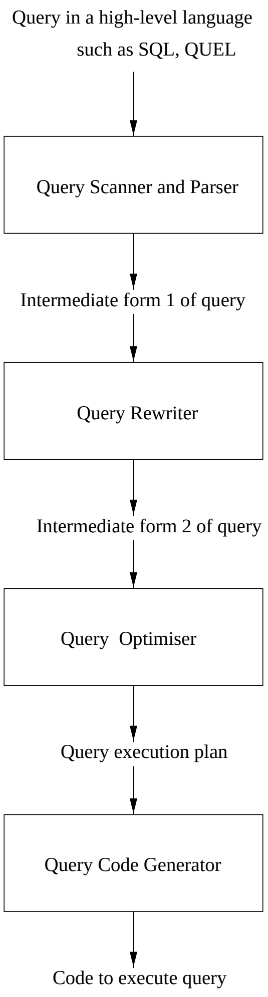


Figure 1.1: Query processing architecture

Scanner and Parser A query is considered by the scanner and parser as a stream of words called *tokens*. The scanner feeds tokens received from a query word-by-word to the parser which would then check the syntax of the query. The task of scanning and parsing queries is fundamentally similar to the parsing translation done by compilers [Aho et al. 1985]. The output of this component is an internal representation (intermediate form 1) of the query normally denoted by a graph or tree data structure [Elmasri and Navathe 1991].

Query Rewriter is involved with simplification of expressions in a query, transforming expressions via equivalence-preserving re-write rules, identifying and removing common sub-expressions, as well as performing semantic optimisation [Lanzelotte 1990; Straube 1990]. These processes are referred to as *logical query optimisation*; their aim is to produce a simpler (more efficient) query expression. The output of this component is a modified query graph (intermediate form 2 as shown in the figure).

Query Optimiser For any given query, there are generally several equivalent ways called *query execution plans* to evaluate it. Each of the execution plans produces the same final result but they all have different execution costs.

Definition 1.1 ***A query execution plan*** *is a plan which has an order for accessing data stored in database files and reading/writing with temporary files.*

The aim of a query optimiser is to find the plan which incurs the lowest cost to execute the query. In practice, we are more often satisfied with quickly finding a cheap plan, rather than attempting to find the optimal (absolute cheapest) plan.

Query Code Generator simply generates code (as a sequence of low-level database operations) using a given query execution plan. This sequence of operations is then passed to the query evaluation engine to produce the answer to the query.

The decision of a query optimiser to select a plan as the final execution plan for a query is typically based on the *estimated* costs of plans and all these estimated costs are rudimentarily approximated by a *query size estimation method* or *size estimation method* for short. Thus, it is crucial that a size estimation method used by an optimiser provide estimated plan costs reasonably close to the corresponding actual ones so that the decision by the query optimiser in most of the times provides a low cost plan, but not necessarily cheapest.

The reason for not being the cheapest is that a query optimiser relies on size estimates, not the *actual* sizes, and so there can be errors in the plan cost approximation – each error comes from the difference between an actual size and its estimated size used – such that the selected plan is not the cheapest. However, all these errors can be reduced if a query optimiser employs an accurate size estimation method.

To raise the importance of a size estimation method used, let us give a simple example. Suppose there are two plans X and Y which cost 600 and 1000 cost units respectively, to be selected by a query optimiser. If the query optimiser employs an accurate size estimation method, it can turn out that the estimated costs for the two plans are 400 and 900, respectively. Thus the optimiser will select plan X as the final execution plan which is the optimal plan.

On the other hand, if the query optimiser employs a poor size estimation method, it can turn out that the estimated costs for the two plans are 800 and 600, respectively. Thus the optimiser will select plan Y as the execution plan. This clearly points that the optimiser now selects the suboptimal plan.

Over recent years, research in query optimisation has been getting more and more advanced and complicated due to problem issues such as the following four. We notice, however that one common thing which all the problem issues below call for is an optimiser which employs an accurate size estimation method. In this thesis we will investigate a series of size estimation methods.

- Query optimisation in object-oriented database systems

First there are many new applications such as computer-aided design and office integration [Delobel et al. 1995] which call for data models with more expressive power, e.g., object-oriented data models than the *atomic* flat data types such as integers, reals and character strings used by the relational data model. This is where the research in relational query optimisation has been shifted into object-oriented query optimisation. See the research theses for examples, for the architectures of extensible object-oriented query optimisers in [Straube 1990; Lanzelotte 1990; Mitchell 1993; Munoz 1994].

Despite the shift of the query optimisation research as mentioned above, the query size estimation methods, i.e., a sampling-based and a non-sampling based method, developed in this thesis can still be used with the object-oriented query optimisation. As part of the cost model [Gardalin et al. 1995; Gardarin et al. 1996] for the Flora object-oriented query optimiser [Gruser et al. 1996], the authors imply to a choice in adopting a method for estimating sizes of selection predicates in object-oriented queries. Such a method of choice in fact could be any of a sampling-based or non-sampling-based method.

Furthermore, in the MOOD project (METU Object-Oriented DBMS) [Dogac et al. 1996], the cost of path expressions [Ozkan et al. 1996] in object-oriented queries is involved with selectivity estimation (or size estimation) and hence, our work in this thesis fits very well to back up the previous work.

• Knowledge-based and deductive systems with "large" joins

Second, there are applications for knowledge-based systems and deductive database systems which require processing of queries with a large number of joins [Krisnamurthy et al. 1986]. This implies that the search space as a result of "large" joins has been substantially increased, which then makes it more difficult to search for the optimal plan in the very large search space.

The problem of query optimisation, in the strict sense of finding the minimal cost query execution plan, is known to be NP-complete [Ullman 1988*a*; Swami and Gupta 1988]. For a query with m relations, an *exhaustive search* algorithm, the very first search algorithm proposed by Selinger et al. [1979*a*], has a worst case time complexity of $O(m!)$ [Krisnamurthy et al. 1986]. This time complexity grows substantially faster than an exponential time complexity $O(2^m)$. (They both are formidable but the former is considerably worse!.) As the size of the optimisation problem increases (e.g. m becomes larger than 7), then exhaustive search over the entire space to find the optimal plan becomes infeasible. This is where more advanced search algorithms, such as Simulated Annealing [Kirkpatrick et al. 1983], Genetic Search [Holland 1975] and A* [Winston 1984; Pearl 1984], have been introduced and investigated to replace the exhaustive search for applications such as of knowledge-based systems.

• Multiple query optimisation

Third, there is a problem called *multiple query optimisation* [Jarke 1984; Sellis 1988; Park and Segev 1988; Yoo 1990; Chakravarthy 1991; Cosar et al. 1993; Choenni et al. 1996]. In many current multi-user database systems, there can be a number of user queries coming to a database system simultaneously. Those "concurrent" queries which exist in a database system at the same time may share some common expressions. By sharing the results for common expressions (one result for one common expression) over a number of queries, overall query processing cost can be significantly reduced. There are two main issues to be dealt with for this problem: (1) identifying common expressions from the concurrent queries and (2) constructing a global execution plan by a global query optimiser to execute the queries which exploits the identified common expressions.

- Web-based query processing and optimisation

Fourth, there is an open new problem with *web-based query processing and optimisation* which perhaps deals with a vast amount of *unstructured* data over the internet (as compared with *structured* data stored in relational or object-oriented databases for examples). This is probably an ongoing research project under investigation in many research institutions.

1.2 Query optimisation problem: a search problem and its significance

To define the problem of query optimisation, let us give an example query Q in Figure 1.2 together with its query graph in Figure 1.3. A query graph is a graph which represents how relations are joined by join predicates and restricted by selection predicates in the query. For example, in Figure 1.3 relation A joins with B on a join predicate $A.a_1 = B.b_1$ and B joins with C on a join predicate $B.b_2 = C.c_1$. In addition, A must be restricted by a selection predicate $A.a_3 =$ "xyz" and C must be restricted by a selection predicate $C.c_3 > 999$.

From the query graph, one possible plan to execute Q in order is shown in Figure 1.4. That is, select tuples from A which satisfy predicate $A.a_3 =$ "xyz" and write them into A', a temporary intermediate relation. A' then joins with the original relation B which produces a temporary intermediate relation $A'B$. Next C is restricted by the condition $C.c_3 > 999$ which results in a temporary relation C'. Last, the two temporary relations $A'B$ and C' join each other to produce the final temporary relation $A'BC'$ which is what the user requires.

Query Q:

select $A.a_2, C.c_2$
from A, B, C
where $A.a_1 = B.b_1$ **and**
$B.b_2 = C.c_1$ **and**
$A.a_3 =$ "xyz" **and**
$C.c_3 > 999$;

Figure 1.2: A join query

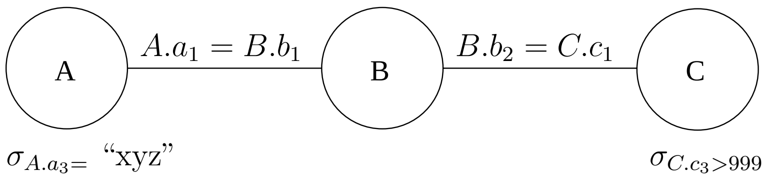


Figure 1.3: The join graph of Q

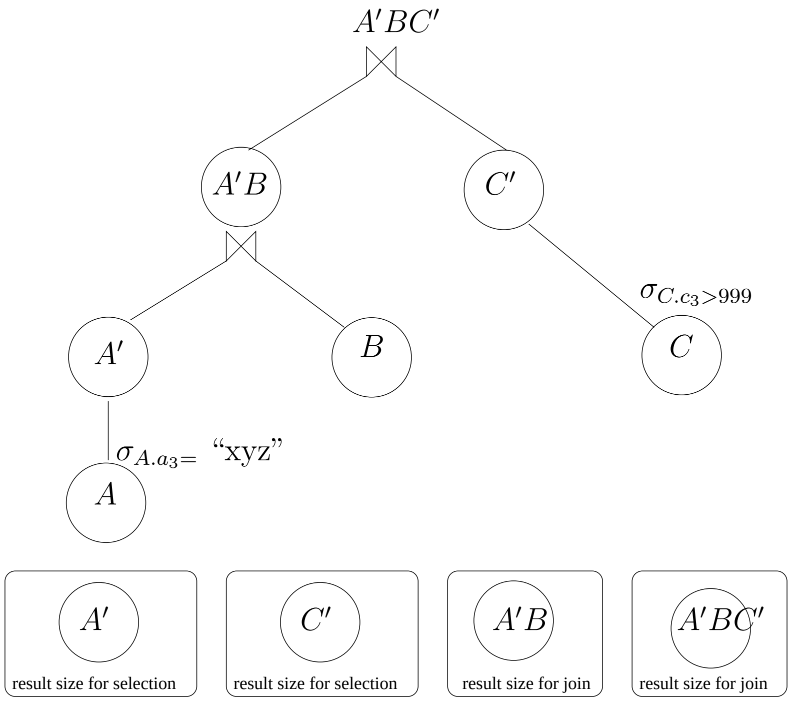


Figure 1.4: An example of a query execution plan for Q

To execute a query with m relations involved, the total number of possible unique plans in the search space is calculated from $m!$. In the given query Q, $m = 3$; Table 1.1 shows all unique 3! plans (join orders) to execute Q.

1	$C' \bowtie B \bowtie A'$
2	$B \bowtie C' \bowtie A'$
3	$C' \bowtie A' \bowtie B$
4	$A' \bowtie C' \bowtie B$
5	$B \bowtie A' \bowtie C'$
6	$A' \bowtie B \bowtie C'$
	total $3! = 6$

Table 1.1: Unique plans

Each of the unique plans can produce the same final output relation, i.e., $A'BC'$ but with different execution costs. Different execution costs can result basically due to different amounts of reading and writing to/from storage.

The optimisation problem is to find which one out of the $m!$ plans results in the minimum and thus optimal cost to execute the query. Given a query Q, an optimisation algorithm to find the optimal plan with the minimum cost is shown in Figure 1.5.

```
STEP 1. let a unique plan, say $optimal\_plan$, be the so-far optimal plan to execute $Q$
        in the entire search space of $Q$. Let $optimal\_plan = \{\}$ and its estimated cost
        $estcost(optimal\_plan) = \infty$.
STEP 2. generate a unique plan $plan$ to execute query $Q$ from the search space.
STEP 3. estimate the cost of the plan $estcost(plan)$.
STEP 4. if $estcost(plan) < estcost(optimal\_plan)$, then:
            •$optimal\_plan = plan$
            •$estcost(optimal\_plan) = estcost(plan)$.
STEP 5. repeat steps 2–4 until all unique plans in the search space are exhausted. If so, re-
        turn the optimal plan $optimal\_plan$ and its estimated cost $estcost(optimal\_plan)$.
```

Figure 1.5: Exhaustive search query optimisation algorithm, given a join query Q

In Figure 1.5, to select the optimal plan *optimal_plan* with the minimum cost, one would need to evaluate the cost of each unique plan *plan* in the search space. Typically a query optimiser would rely on the *estimated* cost of each unique plan to decide upon which plan is optimal.

Queries considered in this thesis are of the form like the one in Figure 1.2 which comprise two important operations: *selection* and *join*. We call these queries *join queries*. Given a query of the form, it is reasonable to consider that the estimated cost *estcost*(*plan*) for a unique plan *plan* consists of:

$$estcost(plan) = \texttt{estcost(selection)} + \texttt{estcost(exeplan)} \tag{1.1}$$

where `estcost(selection)` is the total estimated cost for all selections in the query and estcost(exeplan) is the estimated execution plan cost, which is the estimated cost for a join order, such as any of the 6 join orders as shown in Table 1.1. In query Q in Figure 1.2, there are two selections: $A.a_3 =$ "xyz" and $C.c_3 > 999$. Equation (1.1) is the optimisation objective which we want to minimise. Whichever plan in the search space has the minimum estimated cost would be selected as optimal.

Query optimisation researchers view and tackle the query optimisation problem as a twofold one: (1) Query size estimation methods and (2) Search algorithms. The details are described below.

1.2.1 Query size estimation methods

The optimisation objective in (1.1), in essence, entails the estimation of sizes of temporary intermediate relations. Figure 1.4 shows the underlying need for the estimation of the sizes of temporary relations: $A', C', A'B$ and $A'BC'$.

Definition 1.2 *The term* ***"size"*** *used throughout the thesis is equivalent to (1) a number of tuples or (2) a cardinality. These 3 terms can be used interchangeably.*

Therefore, researchers in this area attempt to find out a good method to accurately approximate sizes of temporary intermediate relations. This would then assist in obtaining the reliable estimated cost *estcost*(*plan*) for any unique plan *plan* in the search space. That is, the estimated cost *estcost*(*plan*) for any plan *plan* should accurately reflect the actual cost for the plan *plan* and hence if a plan is selected optimal, its minimum estimated cost will likely reflect the minimum actual cost for the plan in the search space.

Christodoulakis [1984]; Ioannidis and Christodoulakis [1991, 1993] imply that poor size estimation methods can lead to the selection of more expensive query execution plans. The theoretical studies mainly show the severe problem particularly when the search space considered is large, e.g., the number of relations involved in queries is more than 10. We have also found a common evidence to the earlier work and moreover, the search space we have considered is rather small, i.e., 4–5 relations involved in queries.

For a large search space and with a poor size estimation method used, given an “optimal” plan, an initial error may be negligible for the first subplan (such as the first join), but subsequent errors (errors in the next subplans such as next joins) can grow very rapidly, namely, exponentially [Ioannidis and Christodoulakis 1991]. Thus the plan which is selected as “optimal” by an optimiser may indeed no longer be optimal as a consequence of the “explosive” inaccuracy of the size estimation method used.

In summary, the evidences from the earlier work and our work indicate that a size estimation method used by an optimiser definitely plays a critical role to the selection of low-cost query execution plans no matter what sizes of search spaces are considered.

By the queries we consider in this thesis, the intrinsic problem of query size estimation is to approximate the *selectivity* for the join or selection operation. A selectivity is a ratio, namely, a numerical value between 0 and 1 and is defined as:

Definition 1.3 ***Selectivity*** *is the ratio of the size of the output relation due to a join or selection operation over the cartesian product of sizes of all the relations which participate in the operation.*

For a selection: $$selectivity = \frac{|out|}{|in|}$$

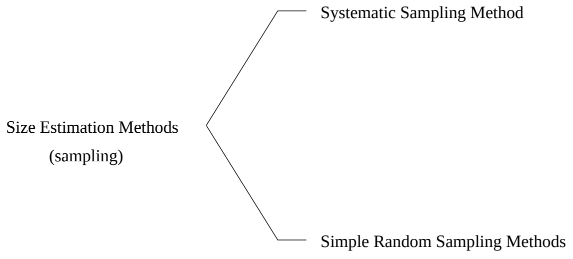
For a join: $$selectivity = \frac{|out|}{|in_1| * |in_2| * \cdots * |in_m|}$$

where $|out|$ is the size of the output relation, and $|in|$ or $|in_i|, i = 1, 2, \ldots, m$ is the size of the input relation. In the case of a join operation, there are at least two participating relations (although this may be the same relation being used twice). In the case of a selection operation, there is only one participating relation.

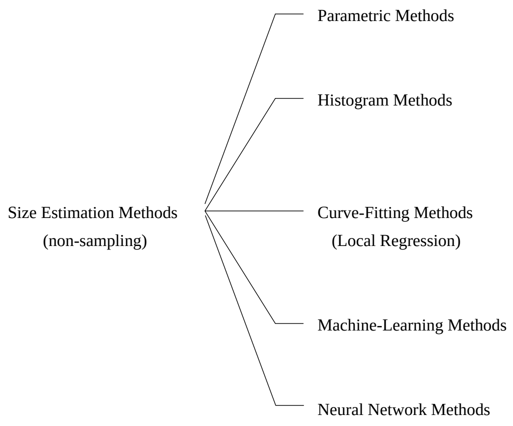
The selectivity obtained then can be used to estimate the size of a temporary intermediate relation, e.g., the sizes of A', C' and $A'B$ as shown in Figure 1.4. As an example, the estimated size of A' is calculated by:

$$\text{(estimated selectivity of selection } A.a_3 = \text{“}xyz\text{”)} \; * \; \text{(the size of } A\text{)}$$

Similarly, the estimated size of $A'B$ is calculated by:



(a) Sampling-based methods



(b) Non-sampling-based methods

Figure 1.6: Size estimation methods

(estimated selectivity of join $A.a_1 = B.b_1$) $*$ (the size of cartesian product of A' and B)

where the estimated size of A' is as calculated above.

Because of the close relationship between the notions of size and selectivity (the size of a temporary relation is determined using the selectivity), we use the terms ***size estimation*** and ***selectivity estimation*** interchangably throughout this thesis.

Mannino et al. [1988] have made an extensive survey of size estimation methods. There are two important lines of research for size estimation methods. One deals with *query size estimation for selections*, the other with *query size estimation for joins*. Figure 1.6 shows the majority of known size estimation methods. Any of the methods in the figure can rudimentarily be used for both selection and join selectivity estimation. (Some of the earlier work proposed a method for join selectivity estimation only, some proposed a method for selection selectivity estimation only and others proposed a method for both.)

Definition 1.4 ***Sampling-based methods*** *are ones which call for sampling so as to estimate sizes of temporary intermediate relations.*

That is, samples are drawn from the original relations and an estimated size of a temporary relation (due to a selection or a join) is calculated using these samples. In Figure 1.6(a), simple random sampling has many variants [Hou et al. 1988, 1989; Lipton et al. 1990; Haas and Swami 1992, 1995]. In this thesis, we propose a novel sampling-based method called *systematic sampling* (SYSSMP) which we have found superior to existing sampling-based methods.

Definition 1.5 ***Non-sampling-based methods*** *conventionally make use of statistical parameters about the data distributions for selectivity estimation. These statistical parameters are stored in the database profile catalogue.*

In this thesis, we propose several new approaches to non-sampling-based query size estimation, e.g., curve-fitting (or local regression), machine-learning and neural network from Figure 1.6(b). The other methods in the figure are ones that were previously proposed in the literature. Although curve-fitting methods have already been studied by Sun et al. [1993] and Chen and Roussopoulos [1994], they all belong to a class of what we call *local regression* [Cleveland 1979] methods. Furthermore, the existing curve-fitting methods are less effective and efficient than the local regression method proposed in this thesis.

Table 1.2 shows the list of acronyms for all sampling-based and non-sampling-based methods we will consider in this thesis. More details about the acronyms in the table are given below. Throughout the thesis, we will refer to these methods by their acronyms.

acronym	description
SRSWOR	Simple Random Sampling WithOut Replacement
SRSWR	Simple Random Sampling With Replacement
SS	Sequential Sampling, a variant of SRSWR
SYSSMP	SYStematic SaMPling
HYBRID	SS + SYSSMP

(a) sampling-based methods

acronym	description
UNF	uniform-distribution-based parametric method
HIST	equi-height histogram method
IASE	curve-fitting method (unweighted regression)
ASE	curve-fitting method (unweighted regression)
M5	machine-learning method
NN	neural network method

(b) non-sampling-based methods

Table 1.2: Sampling-based and non-sampling-based methods

1.2.1.1 Representative methods as targets for comparison

It is generally known that sampling-based methods produce more accurate selectivities and thus more accurate sizes of temporary intermediate relations than non-sampling-based methods (see also the results in [Sun et al. 1993; Harangsri et al. 1996*c*]).

Second, there are two types of database systems with which the sampling-based methods are not appropriate. One is distributed database systems where sampling must be done across a network just to obtain selectivities. The other is systems which prefer an instant and quick way for query size estimation. (By the nature of sampling-based methods, they are not quick methods for size estimation.) For these two types of systems, non-sampling-based methods will suit them rather than sampling-based methods.

Therefore we scope down all experiments in this thesis into:

- Comparing among sampling-based methods only.
- Comparing among non-sampling-based methods only.

In spite of the scoping down, there are still a large number of size estimation methods both sampling-based and non-sampling-based in the literature available as targets for comparison with the methods proposed in this thesis. We selected the following methods as representatives to compare with the SYSSMP and local regression methods proposed here and compared them, using a wide range of various *data distributions*, various *numbers of distinct values*, various *join predicates* and various *selection predicates* in all experiments conducted in the thesis.

Sampling-based methods

Ling and Sun [1995] extensively compared three most representative simple random sampling algorithms with replacement. Their analytical and experimental work apparently demonstrates that the most effective algorithm is SS (Sequential Sampling). Hence, SS is the target to be compared with SYSSMP (SYStematic SaMPling).

Non-sampling-based methods

All the following representative methods will be compared with the local regression method proposed in this thesis.

UNF (UNiForm) This is a parametric method which is basically selected as a ground for comparing with other more sophisticated methods below. Every other method should pass the performance test by comparing with this method.

HIST (HISTogram) This is the equi-height histogram. The main reason we selected this "vanilla" histogram method as a target for comparison with our method here is that equi-height histograms are state-of-the-art ones currently in use in most of commercial database systems. Page 29 of Poosala's thesis [Poosala 1997] contains a table of histograms currently in use in 7 commercial databases, including Oracle, Sybase and Informix. 6 out of 7 use the equi-height histogram. Hence we see that there is enough value to compare with this type of histograms.

IASE (Instant and Accurate Size Estimation) and ASE (Adaptive Size Estimation) These are only two curve-fitting methods that we have found in the literature.

M5 (Learning Machine #5) As one of machine-learning methods, M5 demonstrated its superiority over many other machine-learning methods compared in [Quinlan 1993*b*] by using many real-world databases from the University of California at Irvine [Merz and Murphy 1996].

NN (Neural Network) Known as a function approximator, neural networks are applied successfully to many estimation problems. Standard neural networks with the most popular training algorithm *backpropagation* are a target to be compared with our method.

1.2.2 Search algorithms

Researchers in this area attempt to find out a good search algorithm by which a low-cost, not necessarily optimal, query execution plan can be quickly obtained from a search space. Search algorithm researchers typically assume the existence of good size estimation methods.

Figure 1.7 shows the search algorithms which have been proposed in the literature as search engines for query optimisers. Except for the exhaustive search algorithm,

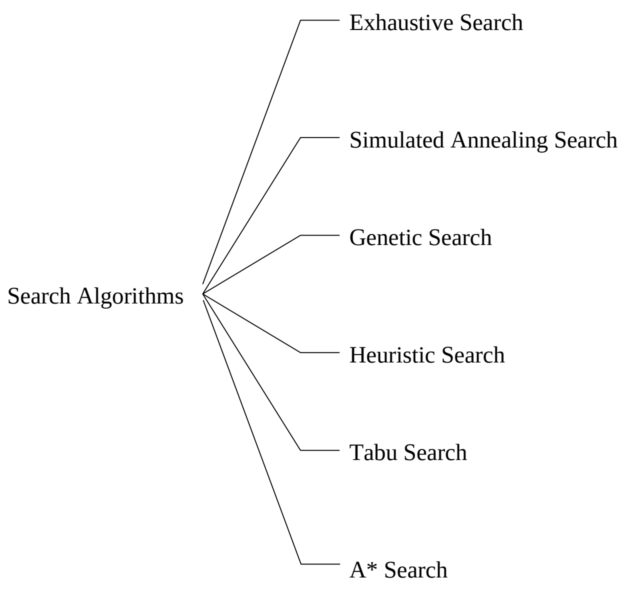


Figure 1.7: Search algorithms as a search engine for query optimisers

all search algorithms in the figure are *limited search algorithms* which perform only a *partial search* over the entire space of query execution plans.

1.2.2.1 Exhaustive search algorithm

The exhaustive search algorithm was the first search algorithm proposed in the query optimisation literature for IBM's System-R query optimiser.

Definition 1.6 ***The exhaustive search algorithm*** *enumerates all $m!$ plans in a search space where m is the number of relations involved in a query. By using a heuristic which avoids performing any cartesian product in any plan, this can prune down many likely expensive plans from the search space. The algorithm then considers a smaller but still exponential number of plans, i.e.,* 2^m *[Selinger et al. 1979a] in the search space.*

1.2.2.2 Limited search algorithms

Definition 1.7 *A* ***limited search algorithm*** *searches (considerably) less than the $m!$ plans in the search space. Only a portion of the entire space will be searched. The query optimiser chooses the minimal cost plan from the portion searched; this may not be the (globally) optimal execution plan.*

Simulated Annealing has been successfully applied to many hard combinatorial problems (see [Laarhoven and Aarts 1988] for examples), i.e., which have very large search spaces. The first application of Simulated Annealing to the query optimisation problem was done by Ioannidis and Wong [1987]. Subsequently, there have been a series of variants of Simulated Annealing suggested to improve the original proposal [Swami and Gupta 1988; Swami 1989*b*,*a*; Ioannidis and Kang 1990; Kang 1991; Pongpinigpinyo 1996].

A*, one of very well-known search algorithms used in AI, was proposed by Yoo [1990]. A Genetic search algorithm was proposed by Bennett et al. [1991]; Stillger and Spiliopoulou [1996] and the Tabu search algorithm was proposed by Morzy et al. [1994]. There have also been a number of heuristic algorithms proposed in the literature, e.g. ones proposed by Wong and Youssefi [1976]; Krisnamurthy et al. [1986].

1.2.2.3 Limited search algorithms with large search spaces

When the number of relations involved in a query is small (less than about 8 relations), then the entire search space $m!$ will be small and thus can be thoroughly searched for the optimal plan by the exhaustive search algorithm.

However, future database applications will require processing of queries with a large number of joins [Krisnamurthy et al. 1986; Swami 1989*b*] perhaps involving more than 7 relations. This would expand the search space to a size that made exhaustive search infeasible and so the development of limited search algorithms are essential.

Regardless of the search algorithm used, a critical factor is the accuracy of size estimation for intermediate results. In this thesis, we thus focus on methods for accurate query size estimation, assuming that they will be applicable to any existing or future query optimisation search algorithm.

1.3 Definitions and notations

The following are definitions and notations that we will use throughout the thesis.

Simple predicate A simple predicate on a relation is a condition specified on a single attribute (any) of the relation to restrict only some tuples of the relation which satisfy the condition.

Let R be a relation of interest with a cardinality (a total number of tuples)

N. The schema of R consists of u attributes, namely, $b_1, b_2, \ldots, b_u$. A simple predicate is of the form ($b_\#$ *relopt* x), where $b_\#$ is an attribute of R, $\# = 1, 2, \ldots, u$, *relopt* could be any of the relational operators $<, >, \neq, =, \geq, \leq$ and x could be any value in the domain of attribute $b_\#$. A simple predicate query is a selection query with a simple predicate on R.

Complex predicate A complex predicate on a relation is formed by any combination of AND-ed (conjunctive) or OR-ed (disjunctive) simple predicates on the relation, e.g., ($b_1 > 10$ and $b_2 \neq 120$), ($b_1 > 10$ or $b_2 = 120$), ($b_1 \leq 89$ or $b_2 = 12$ and $b_4 \geq 123$), where each attribute $b_\#$, $\# = 1, 2, 4$ is an attribute of R. A complex predicate query is a selection query with a complex predicate on R.

Frequency means the number of times an attribute value of $R.b_\#$ occurs in relation R. Let $f(x)$ be either a function of a *frequency distribution* of x — how many times each x value appears in R under $b_\#$ — or a function of a *cumulative frequency distribution* of x — how many times any value less than or equal to x appears in R under $b_\#$. We use the term *data distribution* as a synonym for the term *frequency distribution*.

On-line sampling In on-line sampling, the query optimiser obtains selectivity estimates by taking a (hopefully) representative sample of an original relation and then determining the likely result size based on the result size for the query on the sample.

The sample is taken immediately on-line from its original relation stored in the secondary storage for the selectivity estimation. This is the main reason why any of sampling-based methods never suffers against effects of low/high updates to a database, in contrast with virtually all non-sampling-based methods that do suffer. As a non-sampling-based method, M5 proposed by Harangsri et al. [1997] does not suffer against effects of database updates.

Note that the description above is strictly applicable for selection selectivity estimation. For join selectivity estimation, the approach would also be similar, i.e., by taking samples of each original relation in the join and then joining them together for an estimated join selectivity.

The on-line sampling would clearly incur some delay in processing queries since apart from reading and writing to/from storage for temporary intermediate relations created on the fly, the database system has to do some extra reading and writing to obtain samples from which selectivities can be estimated. In this thesis, whenever we refer to the term "sampling", we mean on-line sampling.

Off-line sampling Unlike on-line sampling, in off-line sampling, the database is not sampled directly for the selectivity estimation purpose but is sampled typically to collect necessary statistical parameters which are then used by those non-sampling-based methods for selection and join selectivity estimation. Normally off-line sampling a database is performed when the system has a light load, for example, at night.

In this thesis, we will normally use the whole phrase "off-line sampling" to refer to this kind of sampling and use simply "sampling" to mean "on-line sampling".

1.4 Thesis scope and contributions

This thesis places the main emphasis on size estimation methods, not search algorithms. We will propose a series of size estimation methods which have been found to improve the size estimation and thus cost evaluation of query optimisers.

There are two major contributions in the thesis. One is a novel sampling-based size estimation method for centralised database systems. We describe its scope in Section 1.4.1. The other is a novel non-sampling-based size estimation method for (1) distributed database systems or (2) database systems which call for an instant and quick way of query size estimation. We describe its scope in Section 1.4.2.

Although sampling-based methods are generally known to produce more accurate selectivities and thus more accurate sizes of intermediate relations than the non-sampling-based methods, they also have some disadvantages in practice.

Firstly, in distributed database systems, relations may be allocated to different nodes of the network. In this case, on-line sampling may become infeasible because we introduce the significant overhead of network latency into some sampling operations.

Secondly, some database systems may prefer an instant and quick way of size estimation. Then any of non-sampling-based methods should be selected as the choice. Compared with non-sampling-based methods, sampling-based methods are not an instant and quick size estimator but can generally yield more accurate selectivities. Hence this is a tradeoff in using sampling-based and non-sampling-based methods.

1.4.1 Centralised database systems

For centralised database systems, we propose that:

- For joins and selections, a novel sampling-based method SYSSMP be used for estimating sizes of temporary intermediate relations.
- For selections, if:
    - histograms [Piatetsky-Shapiro and Connell 1984], a current state-of-the-art method, are still the favored choice as used by many commercial database systems such as INGRES, Sybase, DB2, Informix, MS-sqlserver, Oracle and Teradata for query size estimation,
    - and because current databases are getting larger and larger and perhaps change frequently,

  then building histograms from entire relations would be very expensive and time-consuming. Hence building histograms from a sample of an original relation would play a more important role to the very large and rapidly changed databases. We propose that SYSSMP be used to build histograms, in place of SRSWR or SRSWOR. The results can be found in [Harangsri et al. 1998].
- The bootstrap method [Efron 1979] can be used to improve the quality of join selectivities.

In this thesis, all experiments for joins are conducted through *star joins*. A star join is a join in which any join attribute of the two or more participating relations can join one another on a common join domain. The reason star joins are used is that (1) they are amenable to simulation and analysis, (2) Haas and Swami [1995] also used star joins in their experiments and (3) they also play a significant role in decision-support system applications [Haas and Swami 1995].

1.4.2 Distributed database systems

For distributed database systems where non-sampling-based methods are more appropriate, we propose that:

- For selections, local regression [Cleveland 1979] be used for estimating sizes of temporary intermediate relations. Compared with other non-sampling-based methods:

  - novel machine-learning and neural network methods proposed in this thesis,
  - the parametric uniform distribution based method (UNF), the equi-height histogram (HIST) and two curve-fitting methods ASE and IASE, proposed earlier in the literature,

  local regression achieves the best results in query size estimation for selections.

  If histograms, which is also a special form of local regression, are the preferred method for query size estimation and very large and dynamically-changing databases are considered, then we propose like in the case of centralised database systems that SYSSMP be used to build histograms, in place of SRSWR or SRSWOR.

- For joins, since the experimental work is not yet done, we give very sound justifications in Section 5.7 of Chapter 5 of why local regression would perform for joins equally as well as when it performs for selections. The following is a brief justification.

  In essence, the selectivity estimation for any join can use the frequency distribution approximating functions for the attributes which are in the join. And each of these approximating functions is fundamentally built for both selection and join selectivity estimation. Many of previous non-sampling-based methods, e.g., UNF [Selinger et al. 1979*a*] and histogram [Piatetsky-Shapiro and Connell 1984; Poosala 1997] share the approximating functions for both join and selection selectivity estimation.

  As the join selectivity estimation shares the same underlying approximating functions used for selection selectivity estimation, there should be no doubt for the efficiency of join selectivity estimation as long as the efficiency of selection selectivity estimation is justified.

This thesis also contributes a comprehensive experimental evaluation of all of the size estimation methods, both our new methods and the existing ones. To our knowledge, a unified experimental analysis such as this has never been undertaken previously, and provides a valuable insight into the effectiveness of all of these methods.

For the feasibility issue of using on-line sampling for selectivity estimation, some query optimisation researchers, e.g., [Ioannidis and Poosala 1995; Ioannidis 1993; Chen and Roussopoulos 1994] question about how feasible it is to use sampling particularly for the estimation of join selectivities (because a join is a most time-consuming operation). We will show by a simplified analysis in Chapter 7 that a sampling fraction can be selected such that the total cost for all necessary sampling would be far less than the total query execution cost. This further suggests that the total sampling time will only slightly interfere in the total query execution time. To our best knowledge to date we have seen no such analysis in the literature of query optimisation and selectivity estimation.

1.5 Cost-based vs. rule-based approaches to query optimisation

The following is a justification that all the size estimation methods developed in this thesis can be used by cost-based query optimisers.

Definition 1.8 ***Cost-based query optimisers*** *are ones that generate a number of equivalent query plans, use query size estimates to approximate query plan costs and select one which is optimal or near-optimal as the final execution plan for the query.*

Although there are other approaches to query optimisation, e.g., rule-based optimisation [Freytag 1987; Haas et al. 1989], parametric optimisation [Ioannidis et al. 1992], most of current query optimisers in use nowadays are cost-based, e.g., Oracle, INGRES, DB2, MS-sqlserver, Sybase, Informix, InterBase [InterBase 1998] and SOLID [SOLID 1998].

Furthermore, although modern query optimisers like Oracle, offer two optimisation approaches [Oracle 1996*a*]: rule-based and cost-based, a note [Software AG Americas 1998] against the performance of the rule-based optimisation in practice evidences that the Oracle cost-based optimiser significantly improves the speed of query processing over the Oracle rule-based optimiser. In addition, according to [Oracle 1996*b*], the following are the reasons why one should use the cost-based approach:

- The main problem against the rule-based approach is that users must have the knowledge of or experience in how to compose SQL queries in order for the most efficient performance to be attained. That is, requesting the same information from a database by writing different SQL statements can achieve different performances.

- The cost-based approach generally chooses an execution plan that is as good as or better than the plan chosen by the rule-based approach, especially for queries with multiple joins and indices.

Due to the performance problem existing in practice, Oracle will no longer support the rule-based approach in future versions of its query optimiser [Oracle 1996*b*].

As a result, we hope that all the size estimation methods developed in this thesis can be plugged into those cost-based query optimisers.

1.6 Thesis organisation

We start in Chapter 2 by giving fundamentals and detailed survey of query size estimation methods that have been proposed in the literature.

For centralised database systems, we propose in Chapter 3 that systematic sampling be used for query size estimation for both joins and selections.

To improve the quality of join selectivities used by query optimisers of centralised database systems, the bootstrap method is proposed in Chapter 4.

For distributed database systems or systems which call for an instant and quick way of size estimation, we propose in Chapter 5 that local regression be used for query size estimation for selections.

The evaluation of the sampling-based method proposed in this thesis in an experimental cost-based query optimiser is made in Chapter 6.

We summarise our achievements made by this thesis and discuss about some future work in Chapter 7.

CHAPTER 2

Fundamentals and survey on query size estimation methods

Summary

In the previous chapter, we showed that query size estimation is a critical component for the effective functioning of query optimisers for relational database systems. In this chapter we describe the major approaches to the problem of query size estimation for both centralised and distributed database systems. We characterise these approaches in terms of their underlying assumptions and their strengths and shortcomings. The conclusion is that existing approaches have complementary advantages and disadvantages, so that no one method is outstanding for all purposes.

2.1 Introduction

There has been a considerable amount of work on the issue of selectivity estimation over one and a half decades. During this period, five major approaches to solving the problem have been identified: *parametric* [Selinger et al. 1979*a*; Makinouchi et al. 1981; Christodoulakis 1983*b*], *histogram* [Ioannidis 1993; Ioannidis and Poosala 1995; Poosala 1997], *curve-fitting* [Sun et al. 1993; Chen and Roussopoulos 1994], *machine learning* [Harangsri et al. 1997] and *sampling* [Hou et al. 1988, 1989; Lipton et al.

1990; Hou et al. 1991*b*; Haas and Swami 1992, 1995]. We will describe each of the categories in detail in Sections 2.4, 2.5, 2.6, 2.7, 2.8, respectively.

The structure of each section (with the methods in the same category) would be the same. That is:

1. review a body of work in the literature that belongs to the category,
2. give a detailed description of one (or two) representative methods in the category for selection and join query result size estimation, The reasons we chose each representative method were described earlier in Section 1.2.1.1 of Chapter 1.
3. and analyse the algorithm and storage complexity for the representative method (s).

The analysis of the algorithm complexity to estimate selectivities for complex predicate queries is done based on the worst case that all attributes of a relation, say R, are specified in those complex predicate queries on R.

The storage complexity for a natural join is analysed per relation which participates in the join. It is analysed based on the assumption that there can be only one join attribute on each joining relation but the extension of the analysis to cover more join attributes on a joining relation should be obvious to be done.

For each respective category, we describe UNF (parametric), equi-width and equi-height histograms (histogram), IASE and ASE (curve-fitting), M5 (machine learning) and SS (sampling) as for the representative methods. We also make a summary of the algorithm and storage complexities for these representative methods in Section 2.9.

The first four categories: parametric, histogram, curve-fitting and machine learning are all non-sampling-based techniques and the last is sampling-based techniques. Through the first four, typically query size estimation will rely on *attribute independence* assumption — there is no correlation among attributes of a relation — and the reason is described below. Through the last, query size estimation can naturally handle *attribute dependence* — there are some correlations among attributes of the relation.

The description till the end of this section is applicable for any of non-sampling-based techniques in the first four categories.

To estimate sizes of complex predicate queries (multiple-attribute queries), multi-dimensional query size estimation techniques are the most appropriate. However, the following two important reasons are the ones that prevent the use of multi-dimensional techniques and thus multi-dimensional models yielded by the techniques.

- To create multi-dimensional models, one needs to know all dependencies among attributes in a relation – there can be more than one attribute dependency in a relation. Finding all attribute dependencies in a relation is a separate and hard problem in itself which requires another approach to solving the problem. Ziarko [1991] proposed an approach to the problem and the paper also contains a number of references to other approaches.

- Assuming that in an attribute dependency, K attributes of the relation depend on one another, the storage requirement for a multi-dimensional model would be $(p+1)^K$ for curve-fitting methods or $bucket^K$ for histogram methods (see the derivation for the storage requirement in Chapter 5). p is the coefficient of the polynomial used by the curve-fitting method and *bucket* is the number of buckets used by a dimension of a multi-dimensional histogram.

  In other words, curve-fitting and histogram methods call for a nontrivial exponential amount of storage to the number of attributes involved in the dependency, to maintain statistical parameters in database systems.

Consequently many query optimisation researchers simply resort to the *attribute independence* assumption so as to make the selectivity estimation problem *tractable.* As a consequence of using the assumption, multi-dimensional techniques reduce themselves to a single-dimensional one which yields single-dimensional models as the outcome. Moreover, most of the estimation techniques used in commercial systems even use single-dimensional histograms (models).

However, the attribute independence assumption is not the only way to hinge on for the size estimation of complex predicate queries. A technique like SVD proposed by Poosala and Ioannidis [1997]; Poosala [1997] can also be used to create single-dimensional models (single-dimensional histograms, for example). The resulting histograms can then be combined to yield an approximation of the actual joint frequency distribution of two attributes that depend on each other. But the severe disadvantage against this technique is that it cannot be extended to deal with the attribute dependence with more than two attributes involved.

Hence, in Section 2.2 we describe how to use the attribute independence assumption to estimate the size of a complex predicate query. Using the assumption, it is sufficient to know how to calculate the selectivities for two types of simple predicates, namely, $b_{\#} = x$ and $b_{\#} < x$, where $b_{\#}$ is an attribute of R. These two types are sufficient for the calculation of selectivities for any complex predicate query. As a result, for all the representative methods in the first four categories (the last category does not depend on the attribute independence assumption), we only consider how to estimate the selectivities for the two simple predicates.

Likewise in Section 2.3 we describe how to estimate the size of a natural join. The selectivity of a natural join calculated by each representative method will be described in the corresponding sections 2.4, 2.5, 2.6, 2.7 and 2.8.

In the last section 2.10, we review previous studies on the cost model derivation for multidatabase systems which fundamentally relies on curve-fitting techniques. The main reason of the review is to raise the significance of query size estimation methods to the cost model derivation problem.

2.2 Query size estimation for selections using attribute independence assumption

Below we consider two important types of complex predicate queries: *conjunctive* and *disjunctive* queries.

Let us first begin with a conjunctive query Q with a conjunctive predicate *cpred*:

$$cpred \equiv (pred_1 \text{ and } pred_2 \text{ and } \cdots \text{ and } pred_{P_1}) \tag{2.1}$$

where each of the $pred_i$'s is a simple predicate of the form $b_{\#}$ *relopt* x and P_1 is the number of all simple predicates. Using the attribute independence assumption, the estimated result size for the conjunctive query is calculated by $sel\_ind * N$ where sel_ind is the "combined" selectivity $sel_{\text{and}}(cpred)$ for the entire conjunction:

$$sel\_ind = sel_{\text{and}}(cpred) = \prod_{i=1}^{P_1} sel(pred_i) \tag{2.2}$$

where $sel(pred_i)$ is the selectivity of a simple predicate. Thus, for example,

$$\begin{aligned} & sel_{\text{and}}(b_1 > 10 \text{ and } b_2 \neq 120 \text{ and } b_4 < 230) \\ = \;& sel(b_1 > 10) \; * \; sel(b_2 \neq 120) \; * \; sel(b_4 < 230) \end{aligned}$$

Now consider a disjunctive query Q of the form:

$$dpred \equiv (cpred_1 \text{ or } cpred_2 \text{ or } \cdots \text{ or } cpred_{P_2})$$

where each of the $cpred_i$'s is a conjunctive predicate as in (2.1) and P_2 is the number of all conjunctive predicates. The estimated result result size for the disjunctive query Q is calculated by $sel\_ind * N$, where sel_ind is the "combined" selectivity $sel_{\text{or}}(dpred)$ for the entire disjunction:

$$sel\_ind = sel_{\text{or}}(dpred) = \sum_{i=1}^{P_2} sel_{\text{and}}(cpred_i) \tag{2.3}$$

where $sel_{\text{and}}(cpred_i)$ is calculated by (2.2). Thus, for example,

$$\begin{aligned} & sel_{\text{or}}(b_1 \leq 89 \text{ or } b_2 = 12 \text{ and } b_4 \geq 123) \\ = \;& sel(b_1 \leq 89) \; + \; sel_{\text{and}}(b_2 = 12 \text{ and } b_4 \geq 123) \end{aligned}$$

To calculate the selectivity of a simple predicate, it is sufficient to consider two types of simple predicates, namely of the form $b_\# = x$ and $b_\# < x$. For other types of simple predicates, i.e., $>, \geq, \leq$ and $\neq$, Figure 2.1 is true.

2.3 Query size estimation for natural joins

Let us consider a natural join $(R'_1.b_x \bowtie R'_2.b_y)$, where R'_1 and R'_2 can:

$$\begin{aligned}
&sel(b_{\#} > x) = 1 - (sel(b_{\#} < x) + sel(b_{\#} = x)) \\
&sel(b_{\#} \geq x) = 1 - sel(b_{\#} < x) \\
&sel(b_{\#} \leq x) = sel(b_{\#} < x) + sel(b_{\#} = x) \\
&sel(b_{\#} \neq x) = 1 - sel(b_{\#} = x)
\end{aligned}$$

Figure 2.1: Selectivity calculation for $>, \geq, \leq$ and $\neq$ relational operators

1. both be an original relation,
2. be one temporary intermediate relation due to a selection, projection, or join operation and the other original relation,
3. or both be a temporary intermediate relation due to a selection, projection, or join operation.

The size of the natural join $(R'_1.b_x \bowtie R'_2.b_y)$ is calculated by: $sel(R_1.b_x \bowtie R_2.b_y) * (N_{R'_1} * N_{R'_2})$, where $sel(R_1.b_x \bowtie R_2.b_y)$ is the selectivity of the natural join and $N_{R'_1}$ and $N_{R'_2}$ are the cardinalities of relations R'_1 and R'_2, respectively.

2.4 Parametric methods

2.4.1 Review on parametric methods

Selinger et al. [1979*a*]; Makinouchi et al. [1981]; Christodoulakis [1983*b*]

The parametric methods are ones which depend upon some underlying assumptions of a data distribution such as Uniform, Normal, Poisson, Zipf distributions and so on. The parametric methods proposed in [Selinger et al. 1979*a*; Makinouchi et al. 1981] rely on the uniform distribution in approximating selectivities while the method in [Christodoulakis 1983*b*] relies on the Normal and Pearson Type 2 and 7 distributions. The methods will approximate selectivities effectively if the actual data distribution follows the *a priori* assumption. However, the data distribution in a real database may not match the expected one; for example, the initial assumption may be wrong, or the data may change over time so that the distribution changes. In such cases, the resulting selectivities could be unreliable.

Epstein and Stonebraker [1980]

Epstein and Stonebraker [1980] proposed a parametric method for the join query size estimation. Three values: 1, $\frac{1}{2}$ and $\frac{1}{10}$ were proposed as for join selectivities between any two relations. The three values were then compared by using two search algorithms – each algorithm acts as a search engine for the query optimiser used for comparison. One is an exhaustive search algorithm [Selinger et al. 1979*b*] which searches everywhere in a search space and the other is a limited search [Wong and Youssefi 1976] which is a kind of heuristic algorithms and searches only a portion of the entire search space.

The conclusion based on experiments with two databases is that the exhaustive search performs dramatically better than the limited search in choosing low-cost query execution plans. For the exhaustive search, the selectivity $\frac{1}{10}$ seems to be the best choice for any join between two relations. But for the limited search the results obtained were inconclusive – there is no clear trend about which selectivity is the best choice.

Our question here is that even though the exhaustive search algorithm can be used as a search engine for a query optimiser, how can the proposal for the join selectivity value $\frac{1}{10}$ be validated for any join between two relations ?

Christodoulakis [1983*a*]

(In fact, this work is not under any of the five categories of methods. The reason we classified it into here is because it is one of very primitive methods like any of parametric methods and is related to some of the parametric work described below.)

Christodoulakis [1983*a*] proposed a method for the estimation of join selectivities (see the formula for the estimation below which is taken from formula (18) in the reference). The method can call for a very large amount of storage because the method would keep "comprehensive" statistical information in the database profile catalog. For some cases where an attribute does not have many distinct values appearing in the relation, the method works fine. However, for other cases where an attribute does have a large number of distinct values, the method will not be practical.

The reason is that the method requires storing in the database profile catalog, all pairs of [a distinct value, the number of its occurrences], i.e., the complete frequency distribution for all distinct values of an attribute. Hence, a large number of distinct values of an attribute will incur a large amount of storage to maintain all the pairs in the profile catalog.

Conventionally, most query size estimation work attempts to reduce such a huge storage requirement by storing "brief or summary" statistical information of the frequency distribution of an attribute in the profile catalog, instead of the "comprehensive" information. For instance, histograms [Piatetsky-Shapiro and Connell 1984] require storage for summary information by the number of buckets used to create a histogram, normally 10-20 buckets. This means that 10-20 parameters per attribute would be required to be maintained in the catalog. A curve-fitting method [Chen and Roussopoulos 1994] requires storage approximately by the degree of the polynomial used, which is 6 and hence, 6 parameters per attribute would be required in the catalog.

Given a natural join $R_1.b_x \bowtie R_2.b_y$, the formula for the join selectivity proposed by Christodoulakis [1983*a*] is given by:

$$\frac{\sum_{i=1}^{d} f_x(x_i) * f_y(y_i)}{(N_{R_1} * N_{R_2})}$$

where d is the number of distinct values in the common domain of the join attributes $R_1.b_x$ and $R_2.b_y$. x_i and y_i, $i = 1, 2, \ldots, d$ are distinct values in the common join domain. $f_x(x_i)$ and $f_y(y_i)$ are the frequency distribution of attribute b_x in relation R_1 and that of attribute b_y in relation R_2, respectively. N_{R_i}, $i = 1, 2$ is the cardinality of R_i. Table 2.1 shows a calculation for a join selectivity using the formula. The cardinalities of R_1 and R_2 are 9 and 8 tuples, respectively.

Although the join selectivity estimation is exact as illustrated by the example in Table 2.1(b), due to the comprehensive information maintained, the huge storage requirement makes the proposed method impractical to be implemented in a general database environment.

Chao and Egyhazy [1986]

Chao and Egyhazy [1986] proposed a parametric method for the query size estimation of selections and joins. There are two severe shortcomings of the work:

x's	y's
1	1
1	3
2	3
2	3
2	4
3	4
3	4
4	4
4	

(a) $R_1.b_1$ and $R_2.b_2$

domain value	$f_x(x_i)$	$f_y(y_i)$	$f_x(x_i) * f_y(y_i)$
1	2	1	$= 2$
2	3	0	$= 0$
3	2	3	$= 6$
4	2	4	$= 8$
$d = 4$			total=$\frac{16}{(9*8)}$

(b) Frequency distributions

Table 2.1: Calculation for a join selectivity

• **Large storage requirement** Like the work [Christodoulakis 1983*a*], the storage requirement can be very large because the authors also took the same assumption of comprehensive information as the earlier work. As for selections, although query size estimation can be exact for simple predicate queries and very accurate for complex predicate queries, the huge storage requirement makes the proposed method impractical for a general database environment.

• **Unjustified join selectivities** Let us consider a natural join $R'_1.b_x \bowtie R'_2.b_y$, where R'_1 and R'_2 can both be original relations or one temporary intermediate relation due to a selection or projection operation and the other original relation. The join selectivity for this case is calculated by $\frac{coeff_1 * coeff_2}{2}$ where $coeff_1$ is $N_{R'_1}/$(the number of distinct values of b_x) and $coeff_2$ is $N_{R'_2}/$(the number of distinct values of b_y). $N_{R'_i}$, $i = 1, 2$ is the cardinality of relation R'_i.

First we argue that since $coeff_i$, $i = 1, 2$ in many cases (see also its formula above) will be more than or equal to 1, the formula $\frac{coeff_1 * coeff_2}{2}$ will also in many cases produce a value greater than or equal to 1. Since the join selectivity in the worst case (i.e. the cartesian product of two relations) is always equal to one, the above result is rather surprising.

Second even assuming that the formula is acceptable, there is no justification of why such a formula would be accurate for any join between $R'_1.b_x$ and $R'_2.b_y$.

Consider another join $R'_1.b_x \bowtie R'_2.b_y$, where both R'_1 and R'_2 are a temporary intermediate relation but at least one of the two must be as a result of a join operation. The join selectivity for this case is simply equal to $\frac{1}{2}$, which is also one of the join selectivity values earlier proposed by Epstein and Stonebraker [1980] (see above). Why does the earlier work seems to favor the join selectivity $\frac{1}{10}$, not the $\frac{1}{2}$? We also do not see any experiments done in support of the value $\frac{1}{2}$ for this subsequent work. Hence we consider that $\frac{1}{2}$ is another ad-hoc join selectivity unjustified.

Gardy and Puech [1989]

[Gardy and Puech 1989] proposed a parametric method for the query size estimation of joins, semijoins and outerjoins. The main shortcoming is that most of the work relies on the assumption that the underlying distribution of attribute values is uniform. This assumption is also taken by earlier work such as [Selinger et al. 1979*a*; Makinouchi et al. 1981]. Based on this assumption, many formulas (which are called *generating functions*) for query size estimation for joins, semijoins and outerjoins were derived. As all the formulas rely on an *a priori* uniform distribution assumption, which may or may not be true for relations in a real database, the approximated join (semijoin or outerjoin) result sizes can be unreliable.

Belussi and Faloutsos [1995]

Belussi and Faloutsos [1995] claims that *Fractals* [Mandelbrot 1983] can be used to describe the behavior of real world data, more particularly data distributions and proceeded to use the theory of Fractals for the size estimation of spatial queries. Two kinds of spatial queries were considered: range queries and spatial join queries.

From a generalised fractal dimension, the authors picked one of the generalised dimension D_2 which is called *correlation fractal dimension* and used it in formulas for selectivity estimation. The following two formulas (quoted from equations (15) and (16) in [Belussi and Faloutsos 1995]) were given to estimate the selectivities of a range query and a spatial join query, respectively:

$$Sel_{range}(\epsilon, \text{“shape”}) = \overline{nb}(\epsilon, \text{“shape”}) + 1$$
$$Sel_{join}(\epsilon, \text{“shape”}) = \frac{\overline{nb}(\epsilon, \text{“shape”})}{(N-1)}$$

where N is the number of all points in a point set and $\overline{nb}(\epsilon, \text{“shape”})$ is defined by:

$$\overline{nb}(\epsilon, \text{“shape”}) = K_{\text{“shape”}} * \epsilon^{D_2} \tag{2.4}$$

where $\overline{nb}(\epsilon, \text{“shape”})$ is the average number of neighbors in the radius ϵ for a specified query shape “shape” which could be square, circle, diamond, etc. and $K_{\text{“shape”}}$ is the *proportionality* constant which depends upon the specific query shape.

The following are two important issues which remain unclear in the work and hence, make the value of the method proposed dubious.

- The derivation of (2.4) is based on the relationship that the average number of neighbors follows the power law:

  $$\overline{nb}(\epsilon, \text{“shape”}) \propto \epsilon^{D_2}$$

  It is not quite clear whether such a relationship is really true or not in practice and whether it can be described by the correlation fractal dimension D_2.

- The proportionality constant $K_{\text{“shape”}}$ in (2.4) is based on the *unproved* assumption which the author claimed “sounds right”. Whether the assumption is really true or not, this remains as a question.

2.4.2 Query size estimation for selections using UNF

The uniform distribution assumption assumes that the number of occurrences of any value in a domain of an attribute is the same. Using the assumption, the following are two selectivity formulas for simple predicates $b_{\#} < x$ and $b_{\#} = x$ proposed by Selinger et al. [1979*a*].

$$sel(b_{\#} < x) = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{2.5}$$

$$sel(b_{\#} = x) = \frac{1}{d} \tag{2.6}$$

where x_{min} and x_{max} are the minimum and maximum values, respectively of attribute $b_{\#}$ and d is the number of distinct values of $b_{\#}$ in the relation.

The first formula $sel(b_{\#} < x)$ uses the rationale of a linear scale; every x value in the predicate will be linerly scaled by the formula.

The second formula $sel(b_{\#} = x)$ uses the rationale that every distinct value of $b_{\#}$ has roughly the same number of tuples $\frac{N}{d}$ to appear in R. Hence the selectivity of any distinct value x would be $\frac{(\frac{N}{d})}{N}$ which is then equal to $\frac{1}{d}$.

2.4.2.1 Algorithm and storage complexity for selections

Algorithm

The calculation for $sel(b_\# < x)$ or $sel(b_\# = x)$ in (2.5) and (2.6) can be done in a fixed number of steps. Hence the selectivity estimation for a simple predicate can be done in $O(1)$.

Thus, to calculate the query result size for a complex predicate query on R with a certain number of attributes ($\leq u$) involved in the query, the worst case time complexity would be $O(u)$, where u is the number of attributes of R.

Storage

Since there are only three fixed values x_{min}, x_{max} and d to be maintained in the profile catalog for an attribute of R, the storage requirement for all u attributes of R is $(3 * u)$ and by ignoring the constant factor 3, the storage complexity is $O(u)$.

2.4.3 Query size estimation for joins using UNF

Let us consider a natural join between relations R_1 and R_2, namely $R_1.b_x \bowtie R_2.b_y$. Based on the uniform distribution assumption, Figure 2.2 shows the calculation towards the estimated selectivity $sel(R_1.b_x \bowtie R_2.b_y)$ of the join. $N_{R_i},\ i = 1, 2$ is the cardinality of relation R_i. Suppose that the number of distinct values of b_x in relation R_1 is d_x and thus the number of tuples per distinct value would be $\frac{N_{R_1}}{d_x}$. Similarly for $R_2.b_y$, the number of distinct values of b_y in relation R_2 is d_y and thus the number of tuples per distinct value would be $\frac{N_{R_2}}{d_y}$. Also let $d = \min(d_x, d_y)$, which is the smaller value between the two.

dist.	freq.(R_1)	freq.(R_2)	tuples per dist.
1	$\frac{N_{R_1}}{d_x}$	$\frac{N_{R_2}}{d_y}$	$\frac{N_{R_1}}{d_x} * \frac{N_{R_2}}{d_y}$
2	$\frac{N_{R_1}}{d_x}$	$\frac{N_{R_2}}{d_y}$	$\frac{N_{R_1}}{d_x} * \frac{N_{R_2}}{d_y}$
.	$\cdots$	$\cdots$	$\cdots$
.	$\cdots$	$\cdots$	$\cdots$
d	$\frac{N_{R_1}}{d_x}$	$\frac{N_{R_2}}{d_y}$	$\frac{N_{R_1}}{d_x} * \frac{N_{R_2}}{d_y}$
	total tuples		$d * \frac{N_{R_1}}{d_x} * \frac{N_{R_2}}{d_y}$

Figure 2.2: The estimated total number of tuples based on UNF

Using the estimated total number of tuples in Figure 2.2, the selectivity of the natural join between $R_1.b_x \bowtie R_2.b_y$, $sel(R_1.b_x \bowtie R_2.b_y)$, is thus defined by:

$$\begin{aligned} sel(R_1.b_x \bowtie R_2.b_y) &= \frac{d * \frac{N_{R_1}}{d_x} * \frac{N_{R_2}}{d_y}}{N_{R_1} * N_{R_2}} \\ &= \frac{1}{\max(d_x, d_y)} \end{aligned} \tag{2.7}$$

where $\max(d_x, d_y)$ is the higher value between the two.

2.4.3.1 Algorithm and storage complexity for joins

Algorithm

The retrieval of d_x and d_y can be done in a fixed number of steps. Hence the selectivity estimation in (2.7) can be done in $O(1)$.

Storage

As per relation in the join $(R_1.b_x \bowtie R_2.b_y)$, since the join selectivity estimation uses the number of distinct values d_x (and d_y) for the calculation in (2.7), the storage complexity per relation in the join is $O(1)$, if only one attribute in R_1 (and R_2) can be a join attribute. Even if all of the attributes are potential join attributes, the storage overhead is not significant.

In fact, this storage complexity is shared between both selections and the join – both are responsible for this cost – because d_x (or d_y) is also used by selections.

2.5 Histogram methods

2.5.1 Review on histogram methods

The basic idea behind the histogram method is to build (and store) a model of the data distribution of attributes and then use this model as a basis for determining selectivities and thus query result sizes. Typically, the models are implemented via one frequency histogram for each attribute where the histogram contains the occurrence frequency of ranges of values in the domain of the attribute. These histograms are typically stored as part of the meta-data in many database management systems.

Traditional histograms

Traditional histograms in the literature of query size estimation are *equi-width* and *equi-height* ones. A traditional histogram is built by first splitting a domain of an attribute into a certain number of buckets (intervals) – resulting in ranges of attribute values in the buckets – and then counting the number of tuples that fall into the ranges of the buckets. We call a total number of tuples falling in a bucket *total frequency.*

The difference between the two types of histogram is in which aspect is kept constant. For equi-width histograms, the interval length of each bucket is the same but the total frequency of each bucket could be different. In contrast, for equi-height histograms, the total frequency of each bucket is the same, while the interval length may vary. Equi-width and equi-height histograms were proposed by Piatetsky-Shapiro and Connell [1984].

Piatetsky-Shapiro and Connell [1984] concluded that in order to control the error in query result size estimation, the height (total frequency), not the width, of histogram intervals is the critical factor. They suggested that keeping the interval height constant would lead to better size estimates than using equal-width intervals. In other words, equi-height histograms are superior to equi-width ones in estimating query result sizes.

Single-dimensional histograms consider the frequency distribution of an individual attribute of a relation while *multi-dimensional histograms* are a histogram whose construction considers the joint frequency distribution of several attributes of a relation.

Multi-dimensional histograms such as those proposed by Muralikrishna and DeWitt [1988]; Muralikrishna [1988] and Poosala [1997] are known to be superior to single-dimensional ones in estimating result sizes of complex predicate (multiple-attribute) queries. The superiority stems from the fact that multi-dimensional histograms can deal with both (1) the *attribute dependence* assumption (some correlation among attributes of a relation) if there is any and (2) the *attribute independence* assumption (no correlation among attributes of the relation) while single-dimensional histograms can deal very well only with the attribute independence assumption but can deal poorly with the attribute dependence assumption [Christodoulakis

1984; Poosala 1997]. An obvious reason of the superiority that anyone can see is that building multi-dimensional histograms takes into account the joint frequency distribution among the correlated attributes of the relation while building single-dimensional ones does not.

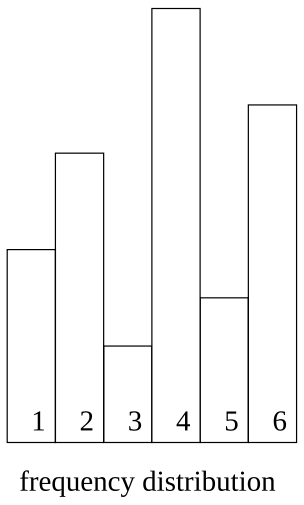
On the other hand, building multi-dimensional histograms to handle complex predicate queries basically calls for an exponential amount of storage to the number of correlated attributes in a relation. This is the major drawback preventing the use and implementation of this type of histogram in commercial database systems. Currently, single-dimensional equi-height and equi-width histograms are the most commonly used query size estimation approaches employed in commercial database systems. These systems are willing to tolerate a small storage overhead per relation (or even per attribute), but the storage demands of multi-dimensional histograms are well beyond this.

Serial histograms (version 1)

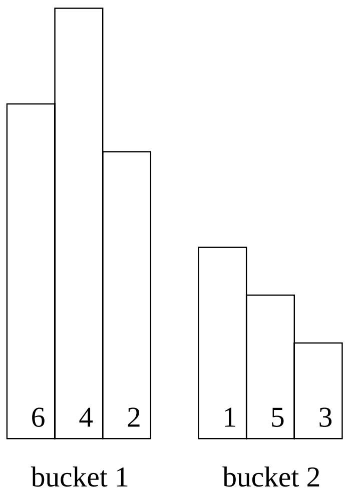
Ioannidis and Christodoulakis [1993]; Ioannidis [1993]; Ioannidis and Poosala [1995] proposed a new class of histograms called *serial histograms*. A serial histogram is built by grouping distinct values with similar frequencies, i.e., similar number of tuples, into the same bucket. The distinct values falling into the same bucket may or may not have any contiguity (proximity) in their numerical/alphabetical order while the traditional, i.e., equi-width and -height histograms [Piatetsky-Shapiro and Connell 1984; Muralikrishna and DeWitt 1988] do have. The theoretical foundation developed proves that compared with other types of histograms, serial histograms are optimal for natural join and selection queries in that they minimise the variance of frequencies grouped into the same bucket (and thus minimise the variance for all the buckets of the histogram).

Figure 2.3(a) shows the frequency distribution of 6 distinct values of an attribute: 1, 2, 3, 4, 5 and 6 labelled at the base of each frequency bar. Figure 2.3(b) shows a 2-bucket serial histogram as the outcome of grouping the distinct values with similar frequencies into the two buckets. Note also the rearranging of distinct values labelled at the base of each frequency bar in the two figures.

The criterion for the optimality of a histogram is defined by the squared error



(a) A frequency distribution before grouping



(b) After grouping, a 2-bucket serial histogram

Figure 2.3: A serial histogram with *uncontiguous* distinct values in 2 buckets

criterion:

$$\sum_{i=1}^{d}(f(x_i) - g(x_i))^2 \tag{2.8}$$

where d is the number of distinct values of the attribute, $f(x_i)$ is the frequency of the distinct value x_i and $g(x_i)$ is its estimate by a size estimation method. In the case of serial histograms (as a method), in a bucket of a serial histogram, a fixed average frequency – calculated by all frequencies of the distinct values in the bucket divided by the number of all distinct values in it – is used as $g(x_i)$ for the bucket; that is, $g(x_i)$ in the same bucket is all the same. This means, independent of whatever distinct value in the bucket is considered, its estimated frequency will always be the fixed average frequency for the bucket.

The squared error in (2.8) is a significant measure for the success of a size estimation method. It points that if a size estimation method can achieve the lowest squared error, then the approximating function $g(x)$ used by the method will best capture the frequency distribution of the attribute. Hence, query size estimation by such an approximating function will also yield the lowest error in size estimation.

Note that the squared error criterion in (2.8) is *naturally* used by any of curve-fitting methods including local regression proposed in this thesis.

In fact, if an approximating function $g(x)$ used is of the form:

$$\begin{aligned} g(x) &= \text{the fixed average frequency for 1st bucket} \\ &= \text{the fixed average frequency for 2nd bucket} \\ &= \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ &= \text{the fixed average frequency for } I\text{th bucket} \end{aligned} \tag{2.9}$$

then this function is a polynomial with the degree 0 which is used by serial histograms, where I is the number of buckets used by a serial histogram. This way of fitting in a bucket by a constant value, i.e., an average frequency is called *local constant fitting* [Cleveland and Loader 1996]. This polynomial in (2.9) is a special case of a general polynomial with the degree 0. Using other higher degrees of a polynomial, e.g., 1, 2 or 3, instead of the degree 0 is a main idea of local regression. According to [Cleveland and Loader 1996], the local constant fitting very infrequently proves to be the best choice in practice to fit data in a local area (bucket) and was only widely appreciated in the early smoothing literature.

Through serial histograms, the grouping of distinct values into buckets will assist in obtaining the lowest squared error in (2.8), compared with other ways of groupings by other types of histograms. However, this does not necessarily mean that such an error will be the absolute minimum compared with the squared error yielded by other approaches such as any of curve-fitting methods and local regression.

Thus by the lowest squared error attained by serial histograms, query size estimation through serial histograms will achieve the best result size estimates, compared with other types of histograms. Once again this does not necessarily imply that such result size estimates by serial histograms will be superior to the estimates yielded by other approaches.

There are two disadvantages in using serial histograms. First, all distinct values falling into a bucket must be recorded and stored in the database profile catalog together with the average frequency for the bucket. This is a severe drawback and makes it impractical to build and maintain such histograms in database systems because this implies that all distinct values of an attribute must be maintained (i.e., stored in the catalog). This would require a substantial amount of storage for any attribute that has a large number of distinct values, e.g., license number, social security number.

Second, the algorithm for constructing an optimal histogram requires an exhaustive enumeration (namely, the exhaustive search) over all possible serial histograms, which calls for an exponential time complexity. The authors then proposed a randomised algorithm [Swami and Gupta 1988; Ioannidis and Kang 1990] for the histogram construction. But since the randomised algorithm is not exhaustive in nature, the resulting constructed histogram may or may not be optimal.

Serial histograms (version 2)

Poosala et al. [1996]; Poosala and Ioannidis [1997] studied a new and large taxonomy of histograms. Among many types of histograms in the taxonomy, V-Optimal(V,A) and MaxDiff(V,A) histograms are the most attractive, i.e., producing least errors in query result size estimation, compared with others considered. V as the first parameter in V-Optimal(V,A) and MaxDiff(V,A) means the grouping[1] of contiguous distinct values into the same bucket and A as the second parameter means the attempt to minimise the variance of *areas* grouped into the same bucket.

Consider an attribute domain with values $x_1, x_2, \ldots, x_d$ where d is the number of distinct values of the attribute in a relation. Let $f(x_1), f(x_2), \ldots, f(x_d)$ be the frequency distribution of all the values x_i's, $i = 1, 2, \ldots, d$. An *area* of a distinct value x_i is defined by $(x_{i+1} - x_i) * f(x_i)$. Note that the serial histograms described in the previous section are characterised as V-Optimal(F,F) where the first F means the grouping of *contiguous frequencies* into the same bucket (in place of the grouping of *contiguous distinct values* by V-Optimal(V,A) and MaxDiff(V,A)) and the second F means the attempt to minimise the variance of frequencies grouped into the same bucket.

The construction of V-Optimal(V,A) histograms is similar to that of V-Optimal (F,F) histograms and hence, uses the same randomised algorithm. The construction of MaxDiff(V,A) histograms aims to insert bucket boundaries between two contiguous distinct values whose area difference produces one of the largest area differences. In other words, the grouping of distinct values into a bucket is to avoid grouping

[1]The grouping of contiguous distinct values into the same bucket by these two types of histograms is the same as that by the traditional histograms.

the distinct values with vastly different areas into the same bucket.

The extensive comparison among many histogram types, including with the equi-width, equi-height and V-Optimal(F,F) histograms, demonstrated that the most efficient histograms are V-Optimal(V,A) and MaxDiff(V,A) both in construction time and low error in query result size estimation.

However, both histogram types: V-Optimal(V,A) and MaxDiff(V,A) fall short commonly in the same problem of local constant fitting – the problem against the fitting in a bucket by a constant value, which is the average frequency for the bucket.

Figure 2.4 graphically shows the problem against the local constant fitting, compared with the local fitting by a polynomial with a higher degree. The "bell-like" graph represents the frequency distribution of an attribute. There are two dashed lines fitting the small curve (solid line) in the jth bucket. One line (horizontal) is the fitting by an average frequency for the jth bucket and the other line is the fitting by local regression.

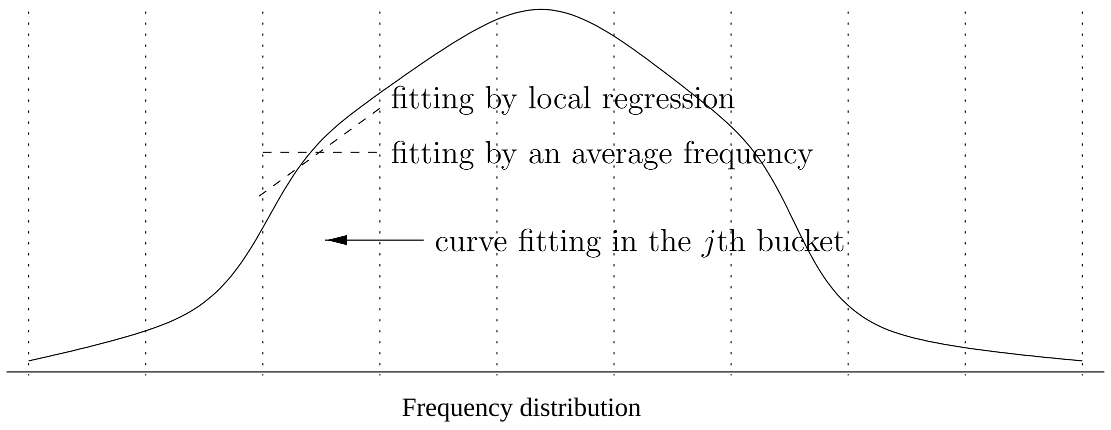

fitting by local regression
fitting by an average frequency
curve fitting in the jth bucket
Frequency distribution

Figure 2.4: Problem against local constant fitting

Regardless of any way of the groupings employed by the two histogram types, there may still exist some small curves (a small curve is a small partition of the entire graph for a frequency distribution) remaining in some buckets and fitting a small curve in a bucket, such as the small curve in the jth bucket of the figure, by the average frequency for the bucket would not basically be as efficient as fitting the small curve in the bucket by a polynomial with a slightly higher degree, such as 1, 2, or 3.

As a consequence of the problem against local constant fitting, the second and important clue that can be used to generally say that any histogram would be inferior to local regression is the squared error criterion in (2.8) which is used to define an optimal histogram. This criterion is always *naturally* used by any curve-fitting method including local regression to find the best-fit coefficients of the polynomial used which would then result in the optimal squared error between $f(x_i)$ and $g(x_i)$, where $i = 1, 2, \ldots, d$. The squared error yielded by the two histogram types will very likely be higher than the squared error by local regression. The reason is as follows.

Consider any bucket of a histogram. A fixed average frequency, e.g., the straight horizontal line in Figure 2.4 will be used to approximate all the frequencies $f(x_i)$'s in the bucket, regardless of what the actual values of $f(x_i)$'s are like. Compared with local regression in the bucket, because the principle of least squared error always naturally applies to any curve-fitting method, a better curve (most probably not the straight horizontal line) with *various* $g(x_i)$'s like the diagonal line in Figure 2.4 will be used to capture the corresponding actual frequencies $f(x_i)$'s in the bucket, instead of the fixed average frequency. Hence, the squared error by local regression will very likely be lower than the one by the two histogram types.

2.5.2 Query size estimation for selections using histograms

In this section, we show how to build single-dimensional equi-width and equi-height histograms for individual attributes of relation R and use these histograms together to estimate sizes of complex predicate queries.

This section is mainly developed from [Harangsri et al. 1998] in which we used SYSSMP to build from sample relations, more efficient histograms than the histograms built by SRSWR and SRSWOR. The histograms considered in the paper are equi-width and -height.

The description throughout this section is generalised for both equi-width and equi-height single-dimensional histograms. We start by giving examples of equi-width and -height histograms and histogram notations, followed by how to estimate result sizes of complex predicate queries.

A schema of relation R consists of u attributes, namely, $b_1, b_2, \ldots, b_u$. A histogram of attribute $b_\#$, $\# = 1, 2, \ldots, u$ has the structure as shown in Table 2.2.

The two histograms in Table 2.2(a) and 2.2(b) are respective examples of equi-width and -height histograms. Each of them shows a *total frequency* distribution of attribute $b_\#$ of a sample relation (5% sampling) from an original relation with 10k tuples. The histograms each have 10 buckets of which each *upper bound value* is shown in the middle column of the histograms. The histogram 2.2(c) shows the histogram notations which generalise the two example equi-width and -height histograms. Notation $I_{b_\#}$ is the number of buckets desired for attribute $b_\#$ (=10 in the two histogram examples). Notation $E_{b_\#,k}$ is the upper bound value for bucket k where $k = 0, 1, 2, \ldots, I_{b_\#}$. Bucket 0 contains the minimum value of attribute $b_\#$ and bucket $I_{b_\#}$ contains the maximum value of attribute $b_\#$. $F_{b_\#,k}$ denotes the total frequency of bucket k (the total number of tuples falling into this bucket).

Let Q be a complex predicate query on relation R. The cardinality of relation R is N and the size of a sample relation is n ($\leq N$). This sample relation is used to build histograms for each attribute $b_\#$ of relation R — one histogram is for one attribute of R. Using the attribute independence assumption, an estimated result size for a query Q is calculated by $sel\_ind * N$ where sel_ind is the selectivity based on the attribute independence assumption (consult sel_ind in Section 2.2).

As we noted in that section, we only need to consider how to calculate the selectivity of an individual simple predicate in Q and it is sufficient to consider only 2 types of simple predicates, namely, $b_\# < x$ and $b_\# = x$. In the following two sections 2.5.2.1 and 2.5.2.2, we describe the formulas for the selectivities of $sel(b_\# = x)$ and $sel(b_\# < x)$, respectively.

2.5.2.1 Selectivity of $sel(b_\# = x)$

Piatetsky-Shapiro and Connell [1984] proposed that the *attribute density* be used for the selectivity of $b_\# = x$ instead of $\frac{1}{d}$, where d is the number of distinct values of $b_\#$ in relation R. An attribute density of $b_\#$ is defined by $\frac{\sum_{i=1}^{d} f(x_i)^2}{N^2}$ where $f(x_i)$ is the frequency distribution of attribute value x_i. The attribute density was used in Piatetsky-Shapiro and Connell's experiments and the authors justified that the attribute density takes an unequal frequency distribution of $b_\#$ into consideration (if there is any) while the formula $\frac{1}{d}$, which is based on the uniform distribution, always presumes that each distinct value of $b_\#$ has the same number of tuples to appear in relation R. In our equi-width and -height histogram implementation for the

no.	upper bound val	total freq
0	38907.0	0
1	38956.3	37
2	39005.6	54
3	39054.9	48
4	39104.2	51
5	39153.5	47
6	39202.8	47
7	39252.1	55
8	39301.4	54
9	39350.7	54
10	39400.0	53

(a) Equi-width histogram for attribute $b_{\#}$

no.	upper bound val	total freq
0	38907	0
1	38973	50
2	39016	50
3	39070	50
4	39116	50
5	39174	50
6	39218	50
7	39263	50
8	39307	50
9	39356	50
10	39400	50

(b) Equi-height histogram for attribute $b_{\#}$

no.	upper bound val	total freq
0	$E_{b_{\#},0}$ (min)	0
1	$E_{b_{\#},1}$	$F_{b_{\#},1}$
2	$E_{b_{\#},2}$	$F_{b_{\#},2}$
3	$E_{b_{\#},3}$	$F_{b_{\#},3}$
.	$\dots$	$\dots$
.	$\dots$	$\dots$
.	$\dots$	$\dots$
.	$\dots$	$\dots$
.	$\dots$	$\dots$
$I_{b_{\#}} - 1$	$E_{b_{\#},(I_{b_{\#}}-1)}$	$F_{b_{\#},(I_{b_{\#}}-1)}$
$I_{b_{\#}}$	$E_{b_{\#},I_{b_{\#}}}$ (max)	$F_{b_{\#},I_{b_{\#}}}$

(c) Histogram notations

Table 2.2: Equi-width and equi-height histograms and histogram notations

comparison in [Harangsri et al. 1998], we followed Piatetsky-Shapiro and Connell's proposal.

2.5.2.2 Selectivity of $sel(b_{\#} < x)$

Obtaining the selectivity for predicate $sel(b_{\#} < x)$ can be described by Figure 2.5. It holds that $sel(b_{\#} < x)$ is a value between $sel(b_{\#} < E_{b_{\#},3})$ in the third bucket and $sel(b_{\#} < E_{b_{\#},4})$ in the fourth bucket. Generalising this, we obtain that:

$$sel(b_{\#} < E_{b_{\#},(j-1)}) < sel(b_{\#} < x) < sel(b_{\#} < E_{b_{\#},j})$$

where x falls into the jth bucket. There are two estimation schemes to calculate an average selectivity of predicate $b_{\#} < x$. The first is called *half scheme* which is proposed by Piatetsky-Shapiro and Connell [1984] and the second is called *uniform scheme* which is proposed by Muralikrishna and DeWitt [1988]; Muralikrishna [1988]. The experimental results in the latter work (multi-dimensional histograms) indicate that the uniform scheme is superior to the half scheme, which corresponds to our experimental results in [Harangsri et al. 1998].

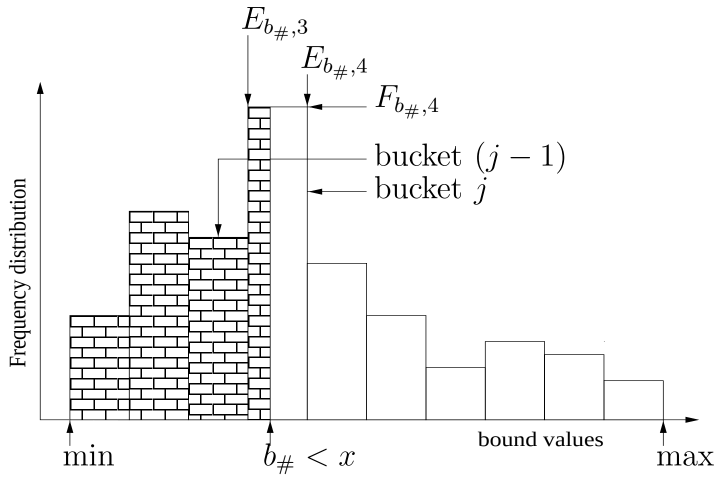


Figure 2.5: Selectivity calculation for simple predicate $b_{\#} < x$

Half scheme The half scheme calculates the average selectivity of $b_{\#} < x$ using the midvalue between buckets $j-1$ and j by:

$$\frac{sel(b_{\#} \leq E_{b_{\#},(j-1)}) + sel(b_{\#} \leq E_{b_{\#},j})}{2}$$

In the histogram example (which is equi-width) in Figure 2.5, x falls into bucket $j = 4$. $sel(b_{\#} \leq E_{b_{\#},3}) = \frac{F_{b_{\#},1}+F_{b_{\#},2}+F_{b_{\#},3}}{n}$. Likewise, $sel(b_{\#} \leq E_{b_{\#},4}) = \frac{F_{b_{\#},1}+F_{b_{\#},2}+F_{b_{\#},3}+F_{b_{\#},4}}{n}$. Generalising this, we obtain the midvalue selectivity:

$$\frac{(\sum_{l=1}^{j-1} F_{b_{\#},l})/n + (\sum_{l=1}^{j} F_{b_{\#},l})/n}{2}$$

as the average selectivity of predicate $b_{\#} < x$.

For equi-height histograms, since each bucket contains the same number of tuples, if x falls into bucket j, then the midvalue selectivity would be:

$$\frac{\frac{(j-1)}{I_{b_{\#}}} + \frac{j}{I_{b_{\#}}}}{2} = \frac{j - 0.5}{I_{b_{\#}}}$$

Uniform scheme The uniform scheme calculates the average selectivity of $b_{\#} < x$ by:

$$sel(b_{\#} \leq E_{b_{\#},(j-1)}) + \Delta$$

where Δ:

$$\Delta = \frac{F_{b_{\#},j} * \frac{x - E_{b_{\#},(j-1)}}{E_{b_{\#},j} - E_{b_{\#},(j-1)}}}{n}$$

In other words, Δ is based on the displacement of x from the lower bound $E_{b_{\#},(j-1)}$ relative to the entire displacement between the lower ($E_{b_{\#},(j-1)}$) and upper bounds ($E_{b_{\#},j}$). Described by the equi-width histogram example in Figure 2.5, Δ is a fraction of the fourth partly-shaded bucket and $sel(b_{\#} \leq E_{b_{\#},(j-1)})$ is equal to $\frac{\sum_{l=1}^{j-1} F_{b_{\#},l}}{n}$. Therefore, for equi-width histograms, we obtain the formula:

$$\frac{\sum_{l=1}^{j-1} F_{b_{\#},l}}{n} + \Delta \tag{2.10}$$

as for the selectivity of $b_{\#} < x$.

For equi-height histograms, since each bucket contains the same number of tuples, if x falls into bucket j, then the

$$sel(b_{\#} \leq E_{b_{\#},(j-1)}) = \frac{(j-1)}{I_{b_{\#}}}$$

while the Δ is equal to:

$$\Delta = \frac{F_{b_{\#},j} * \frac{x-E_{b_{\#},(j-1)}}{E_{b_{\#},j}-E_{b_{\#},(j-1)}}}{n} = \frac{\frac{n}{I_{b_{\#}}} * \frac{x-E_{b_{\#},(j-1)}}{E_{b_{\#},j}-E_{b_{\#},(j-1)}}}{n} = \frac{x-E_{b_{\#},(j-1)}}{E_{b_{\#},j}-E_{b_{\#},(j-1)}} * \frac{1}{I_{b_{\#}}}$$

where $F_{b_{\#},j} = \frac{n}{I_{b_{\#}}}$ as each bucket contains the same number of tuples, i.e., $\frac{n}{I_{b_{\#}}}$. Therefore, for equi-height histograms, we obtain the formula:

$$\frac{(j-1) + \frac{x-E_{b_{\#},(j-1)}}{E_{b_{\#},j}-E_{b_{\#},(j-1)}}}{I_{b_{\#}}} \tag{2.11}$$

as for the selectivity of $b_{\#} < x$.

2.5.2.3 Algorithm and storage complexity for selections

Algorithm

The attribute density of $b_{\#}$ can be retrieved in a fixed number of steps. Hence the calculation for $sel(b_{\#} = x)$ can be done in an $O(1)$ time.

The calculation for $sel(b_{\#} < x)$ is a search in primary memory to find which bucket among $I_{b_{\#}}$ buckets x in the predicate falls into. Normally the number of $I_{b_{\#}}$ buckets used by an attribute $b_{\#}$ is small, e.g., 10-15 buckets and thus the linear search can be used effectively for the search. The calculation for $sel(b_{\#} < x)$ can then be done in $O(I_{b_{\#}})$ [Ullman 1988*b*] by the linear search.

Consequently, to calculate the query result size for a complex predicate query on R with a certain number of attributes ($\leq u$) involved in the query, the worst case time complexity would be $O(\sum_{\#=1}^{u} I_{b_{\#}})$, where u is the number of attributes of R. In many times, the number of buckets used is the same for all the u attributes of R, say I; therefore, the complexity $O(\sum_{\#=1}^{u} I_{b_{\#}})$ can then reduce to $O(u * I)$.

Storage

Consider the example equi-width histogram in Table 2.2(a). For this histogram, the total frequencies in each bucket vary from bucket to bucket and hence are needed to be stored in the database profile catalog. Of the histogram, all the intervals of the upper bound values have a fixed interval length $h = E_{b_{\#},j} - E_{b_{\#},(j-1)}$ for any $j = 1, 2, \ldots, I_{b_{\#}}$ and hence, all the upper bound values can be calculated by

$min + j * h$, where $j = 1, 2, \ldots, I_{b_\#}$. Therefore, there is no need to store the upper bound values in the profile catalog. The storage requirement for an attribute $b_\#$ of R would be:

$$1 \text{ (for } min) + 1 \text{ (for } max) + 1 \text{ (for fixed interval length } h)$$
$$+1 \text{ (for attribute density of } b_\#) + I_{b_\#} \text{ (for total frequencies)}$$

As a result, the storage requirement for all u attributes of R would be $\sum_{\#=1}^{u}(4+I_{b_\#})$. In many times, the number of buckets used is the same for all the u attributes of R, say I; therefore, $\sum_{\#=1}^{u}(4 + I_{b_\#})$ can then reduce to $u * (4 + I)$. But since $I > 4$ in most of the times, the storage complexity is $O(u * I)$.

Consider the example equi-height histogram in Table 2.2(b). For this histogram, there is no need to store the total frequencies of each bucket in the profile catalog as they are all the same. However, each interval of the upper bound values would have a variable interval length and hence, all the upper bound values must be stored in the profile catalog. Therefore, the storage complexity for all u attributes of R would be $\sum_{\#=1}^{u}(4 + I_{b_\#})$, namely:

$$1 \text{ (for } min) + 1 \text{ (for } max) + 1 \text{ (for fixed total frequency)}$$
$$+1 \text{ (for attribute density of } b_\#) + I_{b_\#} \text{ (for upper bound values)}$$

which can then reduce to $O(u * I)$ if all the u attributes of R use the same number of buckets I. This complexity is the same as that for the equi-width histogram.

2.5.3 Query size estimation for joins using histograms

Let us consider a natural join between relations R_1 and R_2, namely $R_1.b_x \bowtie R_2.b_y$. $N_{R_i},\ i = 1, 2$ is the cardinality of relation R_i. d_x and d_y are each the number of distinct values of attributes b_x and b_y, respectively.

Using Piatetsky-Shapiro and Connell's proposal for the attribute density, this inherently implies that for $R_1.b_x$ instead of using $\frac{N_{R_1}}{d_x}$ (as employed by UNF) as the number of tuples per distinct value, they use an average number of tuples per distinct value which takes into account the unequal frequency distribution of attribute b_x. Similarly for $R_2.b_y$, the unequal frequency distribution of attribute b_y would also be taken into account.

Figure 2.6 shows the calculation towards the estimated selectivity $sel(R_1.b_x \bowtie R_2.b_y)$ of the join. Note that the calculation in this figure is similar to that in Figure 2.2. See Figure 2.2 for comparison. The main difference between the two figures is the number of tuples per distinct value which takes the *unequal frequency distribution* into account here and which takes the *equal frequency distribution* into account by UNF.

The attribute density of b_x is defined by $\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}^2}$ where $f(x_i)$ is the frequency distribution of attribute value x_i. Thus the average number of tuples per distinct value, taking the unequal frequency distribution of attribute b_x into account would be $N_{R_1} * \frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}^2}$ which is then equal to $\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}}$.

Likewise, for $R_2.b_y$, the average number of tuples per distinct value would be $N_{R_2} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}^2}$ which is then equal to $\frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$. d is $\min(d_x, d_y)$.

dist.	freq.(R_1)	freq.(R_2)	tuples per dist.
1	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}}$	$\frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$
2	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}}$	$\frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$
.	...	...	... ...
.	...	...	... ...
d	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}}$	$\frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$	$\frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$
	total tuples		$d * \frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}$

Figure 2.6: The estimated total number of tuples

Using the estimated total number of tuples in Figure 2.6, the selectivity of the natural join between $R_1.b_x \bowtie R_2.b_y$, $sel(R_1.b_x \bowtie R_2.b_y)$, is thus defined by:

$$\begin{aligned} & \frac{d * \frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}}}{N_{R_1} * N_{R_2}} \\ &= d * \frac{\sum_{i=1}^{d_x} f(x_i)^2}{N_{R_1}^2} * \frac{\sum_{i=1}^{d_y} f(y_i)^2}{N_{R_2}^2} \\ &= d * (\text{attr. density of } b_x) * (\text{attr. density of } b_y) \end{aligned} \tag{2.12}$$

2.5.3.1 Algorithm and storage complexity for joins

Algorithm

Consider (2.12). The retrieval of the attribute densities of b_x and b_y and the numbers of distinct values d_x and d_y can be done in a fixed number of steps. Hence the selectivity estimation in (2.12) can be done in $O(1)$.

Storage

The selectivity estimation for the join $(R_1.b_x \bowtie R_2.b_y)$, uses the attribute density for b_x (and b_y) and the number of distinct values d_x (and d_y) for the calculation in (2.12). Thus, these two parameters per relation need to be stored. The storage complexity per relation in the join is $O(1)$, if only one attribute in R_1 (and R_2) can be a join attribute.

Consider now the two parameters for R_1: attribute density of b_x and d_x. The storage required for d_x is introduced particularly for the computation of the join, while the density of b_x is shared between selections $(b_x = \mathit{const})$'s and the join.

However, as per relation in the join, the newly introduced storage for d_x still incurs the same $O(1)$ storage complexity if only one attribute in R_1 can be a join attribute.

2.6 Curve-fitting methods

2.6.1 Review on curve-fitting methods

The main theme of curve-fitting methods [Sun et al. 1993; Chen and Roussopoulos 1994] is to attempt to fit either a frequency distribution or a cumulative frequency distribution of an individual attribute of a relation by a polynomial with a high degree. The principle of least square error is used to find the best-fit coefficients of a polynomial used. The earlier study called IASE (Instant and Accurate Size Estimation) in [Sun et al. 1993] proposed to fit a frequency distribution via a polynomial $g(x) = \sum_{i=-p_1}^{p_2} a_i x^i$ where p which is equal to $p_1 + p_2$ is the degree of the polynomial and $p_1, p_2 \geq 0$. The subsequent study called ASE (Adaptive Size Estimation) in [Chen and Roussopoulos 1994] proposed to use a polynomial $g(x) = \sum_{i=0}^{p} a_i x^i$ where p is the degree of the polynomial used to fit a cumulative frequency distribution. In fact, by IASE if $p_1 = 0$ and p_2 is equal to p which is used by ASE, then the two polynomials are the same.

One of the main differences between these two studies is that the earlier study proposed to scan an entire relation for the frequency distributions of all attributes in the relation and then fit each frequency distribution obtained by the polynomial defined above, whereas the subsequent study proposed to use already-processed queries together with their query result sizes as the source for acquiring the cumulative frequency distributions for all the attributes in the relation. This is a clever and simple idea which no other previous studies on query size estimation had used. The method by the subsequent study would be particularly useful, especially for very large databases as scanning a large database for building up statistical parameters for all relations in the database is definitely costly and time-consuming.

Table 2.3 shows a summary of the relationship between IASE and ASE. In the table, the polynomial degree p used by IASE is 10 ($p_1 = 4$ and $p_2 = 6$) in all experiments while the degree used by ASE is 6 also in all experiments.

feature	IASE	ASE
polynomial	$g(x) = \sum_{i=-p_1}^{p_2} a_i x^i$ where $p = p_1 + p_2$	$\sum_{i=0}^{p} a_i x^i$
polynomial degree p	10	6
fitting type	freq. distribution	cumulative freq. distribution
building stat. parameters	scan relations	use query feedback

Table 2.3: Relationship between IASE and ASE

Known as the *oversmoothing* problem (details of the problem are described in Chapter 5), the main drawback of these curve-fitting methods is that many times they cannot nicely fit (cumulative) frequency distributions and therefore, the resulting polynomial fails to effectively capture many of the points in the (cumulative) frequency distribution. To overcome this problem we will propose in this thesis a new curve-fitting method called *local regression* to improve the quality of fitting. In fact, the earlier curve-fitting methods are known as a *global regression* and can be viewed as a special form of the local regression proposed in this thesis. The details of local and global regression will be described in depth in Chapter 5.

2.6.2 Query size estimation for selections using IASE

In this section, we describe the method of IASE to estimate query result sizes for selections. IASE proposed to fit frequency distributions, as opposed to ASE which proposed to fit cumulative frequency distributions. The details of fitting frequency distributions by IASE are described below.

$f(x)$ is a frequency distribution function of an attribute $b_{\#}$ of R, namely: $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_d, f(x_d))$ where d is the number of distinct values of $b_{\#}$ which appear in R. Let x_{max} be the maximum value in the domain of attribute $b_{\#}$ and x_{min} the minimum value in the domain. IASE uses a polynomial of the form:

$$g(x) = \sum_{i=-p_1}^{p_2} a_i x^i \tag{2.13}$$

to fit the frequency distribution $f(x)$, where $p = p_1 + p_2$ is the degree of the polynomial used and $p_1, p_2 \geq 0$.

The coefficients a_i's in equation (2.13) can be found by the principle of least square error. That is, choose some a_i's values to minimise:

$$\sum_{i=1}^{d} (g(x_i) - f(x_i))^2 \tag{2.14}$$

This is the *unweighted* or *ordinary least square* problem. Transform the sum of squares in (2.14) to matrix form as follows:

$$(Y - XA)^T (Y - XA) \tag{2.15}$$

Solve equation (2.15) for the coefficients in A:

$$\widehat{A} = (X^T X)^{-1} X^T Y \tag{2.16}$$

where X^T is the transpose of X. Note that $\widehat{A}$ is an approximate value of A. This is due to the fact that the resulting coefficients a_i's obtained may or may not be able to make the least squares in (2.14) completely equal to zero. The following are the definitions for all the matrices in (2.16).

X is an $d \times (p+1)$ matrix and defined by:

$$X = \begin{bmatrix} x_1^{-p_1} & x_1^{-p_1+1} & \dots & x_1^{-1} & 1 & x_1 & x_1^2 & \dots & x_1^{p_2-1} & x_1^{p_2} \\ x_2^{-p_1} & x_2^{-p_1+1} & \dots & x_2^{-1} & 1 & x_2 & x_2^2 & \dots & x_2^{p_2-1} & x_2^{p_2} \\ \dots & \dots & \dots & \dots & 1 & \dots & \dots & \dots & \dots & \dots \\ x_d^{-p_1} & x_d^{-p_1+1} & \dots & x_d^{-1} & 1 & x_d & x_d^2 & \dots & x_d^{p_2-1} & x_d^{p_2} \end{bmatrix} \tag{2.17}$$

Matrix X consists of elements with values x^i's through the polynomial (2.13). A is a $(p+1) \times 1$ matrix of the coefficients and Y is a $d \times 1$ response matrix, consisting of $f(x_1), f(x_2), \dots, f(x_d)$. The following are A and Y:

$$A = \begin{bmatrix} a_{-p_1} \\ \dots \\ a_{-1} \\ a_0 \\ a_1 \\ \dots \\ a_{p_2} \end{bmatrix}, \quad Y = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \dots \\ f(x_{d-1}) \\ f(x_d) \end{bmatrix} \tag{2.18}$$

After solving the ordinary least squares in (2.16) for the coefficients a_i's $i = -p_1, \dots, p_2$, given the solved coefficients:

- $sel(b_{\#} = x)$ is calculated by $g(x) = \sum_{i=-p_1}^{p_2} a_i x^i$, divided by N.
- $sel(b_{\#} < x)$ is calculated by:

$$\frac{\int_{x_{min}}^{x} g(x)\ dx}{\int_{x_{min}}^{x_{max}} g(x)\ dx} \tag{2.19}$$

The two integrations above do not give us a simple algorithmic solution, and so we now describe how the expression in (2.19) can be transformed into a more tractable form.

Since $g(x) = \sum_{i=-p_1}^{p_2} a_i x^i$, the indefinite integral of $g(x)$ is defined by:

$$G(x) = \int g(x)\ dx = \int \sum_{i=-p_1}^{p_2} a_i x^i\ dx = \sum_{i=0}^{p_2} \frac{a_i x^{(i+1)}}{(i+1)} + a_{-1}\ln(x) + \sum_{i=-2}^{-p_1} \frac{a_i x^{(i+1)}}{(i+1)} \tag{2.20}$$

$G(x)$ can be called the *cumulative frequency distribution* of x and $\ln(x)$ is the natural logarithm of x. The definite integral of $g(x)$ between x' and x'' is defined by:

$$\begin{aligned}\int_{x'}^{x''} g(x)\ dx &= G(x'') - G(x') \\ &= \left(\sum_{i=0}^{p_2} \frac{a_i x''^{(i+1)}}{(i+1)} + a_{-1}\ln(x)'' + \sum_{i=-2}^{-p_1} \frac{a_i x''^{(i+1)}}{(i+1)}\right) \\ &\quad - \left(\sum_{i=0}^{p_2} \frac{a_i x'^{(i+1)}}{(i+1)} + a_{-1}\ln(x)' + \sum_{i=-2}^{-p_1} \frac{a_i x'^{(i+1)}}{(i+1)}\right) \qquad (2.21)\end{aligned}$$

Using (2.21), equation (2.19) can be solved for $sel(b_{\#} < x)$.

2.6.2.1 Algorithm and storage complexity for selections

Algorithm

For $sel(b_{\#} = x)$, consider (2.13). To retrieve all the $(p+1)$ coefficients for the equation, we need $(p+1)$ steps, where $p = (p_1 + p_2)$.

For $sel(b_{\#} < x)$, consider (2.21). The equation also requires the retrieval of all the $(p+1)$ coefficients.

Thus, to calculate the query result size for a complex predicate query on R with a certain number of attributes ($\leq u$) involved in the query, the worst case time would be $u*(p+1)$, where u is the number of attributes of R. But since p in $(p+1)$ is normally more than 1, the worst case time complexity would be $O(u*p)$.

The above analysis is based on using the same polynomial degree p for all u attributes of R. Let $p_{b_1}, p_{b_2}, \ldots, p_{b_u}$ be the various different polynomial degrees used by each attribute of R. Suppose that all of the degrees are bound by p (i.e., $p_{b_i} \leq p$ for all $i = 1, 2, \ldots, u$. Then the worst case time complexity by using various different polynomial degrees for all u attributes would still be $O(u*p)$.

Storage

Both the calculation for $sel(b_{\#} = x)$ and $sel(b_{\#} < x)$ shares the same $(p+1)$ coefficients. Hence, the storage requirement for all u attributes of R is $u*(p+1)$, giving a storage complexity of $O(u*p)$.

2.6.3 Query size estimation for joins using IASE

IASE proposed an idea to fit a cartesian product between two join attributes by a two-dimensional polynomial and reduce the resulting polynomial to suit a natural join.

Let us consider a natural join between relation R_1 and R_2, namely $R_1.b_x \bowtie R_2.b_y$. A function $ff(x,y)$ is defined by a cartesian product of $f_x(x)*f_y(y)$, where $f_x(x)$ and $f_y(y)$ are a frequency distribution function of attributes $R_1.b_x$ and $R_2.b_y$, respectively. Let d_x be the number of distinct values of the join attribute $R_1.b_x$ and d_y the number of distinct values of the join attribute $R_2.b_y$.

Figure 2.7 shows the cartesian product between the two join attributes $R_1.b_x$ and $R_2.b_y$, for all combinations between attribute values which appear in the two relations under $R_1.b_x$ and $R_2.b_y$.

row #	x's val	y's val	$ff(x,y)$
1	x_1	y_1	$f_x(x_1)*f_y(y_1)$
2	x_1	y_2	$f_x(x_1)*f_y(y_2)$
...	...	...	...
d_y	x_1	y_{d_y}	$f_x(x_1)*f_y(y_{d_y})$
(d_y+1)	x_2	y_1	$f_x(x_2)*f_y(y_1)$
(d_y+2)	x_2	y_2	$f_x(x_2)*f_y(y_2)$
...	...	...	...
$2d_y$	x_2	y_{d_y}	$f_x(x_2)*f_y(y_{d_y})$
...	...	...	...
r-th	x_l	y_k	$f_x(x_l)*f_y(y_k)$
...	...	...	...
$(d_{x-1}d_y+1)$	x_{d_x}	y_1	$f_x(x_{d_x})*f_y(y_1)$
$(d_{x-1}d_y+2)$	x_{d_x}	y_2	$f_x(x_{d_x})*f_y(y_2)$
...	...	...	...
$m=(d_xd_y)$	x_{d_x}	y_{d_y}	$f_x(x_{d_x})*f_y(y_{d_y})$

Figure 2.7: Cartesian product between two join attributes

A two-dimensional polynomial is used of the form:

$$g(x,y) = \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij}x^i y^j \tag{2.22}$$

to fit the function $ff(x,y)$, where $p_{x_1}, p_{x_2} \geq 0$ and $p_{y_1}, p_{y_2} \geq 0$. $p_x = (p_{x_1}+p_{x_2})$ is the degree of the polynomial for the first dimension and $p_y = (p_{y_1}+p_{y_2})$ is the degree of the polynomial for the second dimension.

In order to find the "optimal" coefficients a_{ij}'s in (2.22), the principle of least square error can be used to minimise:

$$\sum_{l=1}^{d_x}\sum_{k=1}^{d_y}(g(x_l,y_k)-ff(x_l,y_k))^2 \tag{2.23}$$

The least square problem in (2.23) can be transformed into matrix form as follows:

$$(Y-XA)^T(Y-XA) \tag{2.24}$$

Solve equation (2.24) for the coefficients in A:

$$\widehat{A}=(X^TX)^{-1}X^TY \tag{2.25}$$

where X^T is the transpose of X. The following are the definitions for all the matrices in (2.25).

X is a $((d_x \times d_y) \times ((p_x+1)\times(p_y+1)))$ matrix and defined in Figure 2.8.

$$X=\begin{bmatrix} X_{1,-p_{x_1}} & X_{1,(-p_{x_1}+1)} & \dots & X_{1,i} & \dots & X_{1,(p_{x_2}-1)} & X_{1,p_{x_2}} \\ X_{2,-p_{x_1}} & X_{2,(-p_{x_1}+1)} & \dots & X_{2,i} & \dots & X_{2,(p_{x_2}-1)} & X_{2,p_{x_2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ X_{r,-p_{x_1}} & X_{r,(-p_{x_1}+1)} & \dots & X_{r,i} & \dots & X_{r,(p_{x_2}-1)} & X_{r,p_{x_2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ X_{m-1,-p_{x_1}} & X_{m-1,(-p_{x_1}+1)} & \dots & X_{m-1,i} & \dots & X_{m-1,(p_{x_2}-1)} & X_{m-1,p_{x_2}} \\ X_{m,-p_{x_1}} & X_{m,(-p_{x_1}+1)} & \dots & X_{m,i} & \dots & X_{m,(p_{x_2}-1)} & X_{m,p_{x_2}} \end{bmatrix} \tag{2.26}$$

Figure 2.8: A matrix X

In the figure, $m = d_xd_y$ and the submatrix $X_{r,i}$, where $r = 1, 2, \dots, m$ and $i = -p_{x_1}, \dots, p_{x_2}$, is defined by:

$$X_{r,i}=\begin{bmatrix} x_l^i y_k^{-p_{y_1}} & x_l^i y_k^{(-p_{y_1}+1)} & \dots & x_l^i & \dots & x_l^i y_k^{(p_{y_2}-1)} & x_l^i y_k^{p_{y_2}} \end{bmatrix} \tag{2.27}$$

Here is an example of a submatrix $X_{r,i}$, given the degree of the polynomial: $p_{x_2} = 2, p_{x_1} = 2, p_{y_2} = 2$ and $p_{y_1} = 2$. Thus the polynomial would be like: $g(x,y) = \sum_{i=-2}^{2}\sum_{j=-2}^{2} a_{ij}x^iy^j$. Suppose also that the r-th row is a combination of (x_l, y_k).

$$
\begin{array}{l|lllll}
X_{r,i=-2} & x_l^{-2}y_k^{-2} & x_l^{-2}y_k^{-1} & x_l^{-2} & x_l^{-2}y_k^{1} & x_l^{-2}y_k^{2} \\
X_{r,i=-1} & x_l^{-1}y_k^{-2} & x_l^{-1}y_k^{-1} & x_l^{-1} & x_l^{-1}y_k^{1} & x_l^{-1}y_k^{2} \\
X_{r,i=0} & y_k^{-2} & y_k^{-1} & 1 & y_k^{1} & y_k^{2} \\
X_{r,i=1} & x_l^{1}y_k^{-2} & x_l^{1}y_k^{-1} & x_l^{1} & x_l^{1}y_k^{1} & x_l^{1}y_k^{2} \\
X_{r,i=2} & x_l^{2}y_k^{-2} & x_l^{2}y_k^{-1} & x_l^{2} & x_l^{2}y_k^{1} & x_l^{2}y_k^{2}
\end{array}
$$

In each row of the example above, x_l^i will be fixed for the whole row while y_k^j will vary from $j = -2, -1, 0, 1, 2$.

A is a matrix of dimension $(((p_x + 1) \times (p_y + 1)) \times 1)$ and Y is a response matrix of dimension $((d_x \times d_y) \times 1)$. Matrices A and Y are shown in Figure 2.9.

$$
A = \left[\begin{array}{c}
a_{-p_{x_1},-p_{y_1}} \\
a_{-p_{x_1},(-p_{y_1}+1)} \\
\cdots \\
a_{-p_{x_1},p_{y_2}} \\ \hline
a_{(-p_{x_1}+1),-p_{y_1}} \\
a_{(-p_{x_1}+1),(-p_{y_1}+1)} \\
\cdots \\
a_{(-p_{x_1}+1),p_{y_2}} \\ \hline
\cdots \\
\cdots \\
\cdots \\ \hline
a_{p_{x_2},-p_{y_1}} \\
a_{p_{x_2},(-p_{y_1}+1)} \\
\cdots \\
a_{p_{x_2},p_{y_2}}
\end{array}\right], \quad
Y = \left[\begin{array}{c}
ff(x_1, y_1) = f_x(x_1) * f_y(y_1) \\
ff(x_1, y_2) = f_x(x_1) * f_y(y_2) \\
\cdots \\
ff(x_1, y_{d_y}) = f_x(x_1) * f_y(y_{d_y}) \\ \hline
ff(x_2, y_1) = f_x(x_2) * f_y(y_1) \\
ff(x_2, y_2) = f_x(x_2) * f_y(y_2) \\
\cdots \\
ff(x_2, y_{d_y}) = f_x(x_2) * f_y(y_{d_y}) \\ \hline
\cdots \\
ff(x_l, y_k) = f_x(x_l) * f_y(y_k) \\
\cdots \\ \hline
ff(x_{d_x}, y_1) = f_x(x_{d_x}) * f_y(y_1) \\
ff(x_{d_x}, y_2) = f_x(x_{d_x}) * f_y(y_2) \\
\cdots \\
ff(x_{d_x}, y_{d_y}) = f_x(x_{d_x}) * f_y(y_{d_y})
\end{array}\right] \tag{2.28}
$$

Figure 2.9: Matrices A and Y

After solving for the coefficients a_{ij}'s, to suit the natural join, the fitting polynomial $g(x, y)$ is reduced to $g(x, x)$:

$$
\begin{aligned}
g_{\bowtie_=}(x) = g(x, x) &= \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij} x^i x^j \\
&= \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij} x^{i+j}
\end{aligned}
$$

which can be read that x must be equal to y only in order for attribute values between $R_1.b_x$ and $R_2.b_y$ to be joinable by the natural join and produce tuples in the output relation.

If we consider that attribute values from the two join attributes are not necessarily equal to one another in order to be joinable and produce tuples in the output relation, then this is not the equi-join or natural join as considered above. This is other kinds of θ-joins where θ can be any of $\neq, <, \leq, >, \geq$ and in fact, by sharing the same polynomial in (2.22), the method for join selectivity estimation proposed by IASE can be used to approximate any θ-join selectivity between two join attributes where θ can be any of $=, \neq, <, \leq, >, \geq$. (In retrospect, this is the main rationale that IASE proposed to fit the cartesian product between two join attributes.)

Thus $sel(R_1.b_x \bowtie R_2.b_y)$, the selectivity of the natural join between $R_1.b_x \bowtie R_2.b_y$ is defined by:

$$sel(R_1.b_x \bowtie R_2.b_y) = \frac{\int_{low}^{high} g_{\bowtie_=}(x)\ dx}{\int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} g(x,y)\ dx\ dy} \tag{2.29}$$

where x_{max} and x_{min} are the maximum and minimum values of attribute $R_1.b_x$, respectively. Likewise, y_{max} and y_{min} are the respective maximum and minimum values of attribute $R_2.b_y$. low in (2.29) is the larger value between x_{min} and y_{min} and $high$ is the smaller value between x_{max} and y_{max}.

The indefinite integral of $\int g_{\bowtie_=}(x)\ dx$ is equal to:

$$\begin{aligned} \int g_{\bowtie_=}(x)\ dx &= \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} \frac{a_{ij} x^{(i+j+1)}}{(i+j+1)} \qquad \text{such that } \ i+j \neq -1 \\ &+ \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij} \ln(x) \qquad \text{such that } \ i+j = -1 \end{aligned} \tag{2.30}$$

and the indefinite integral of $\int\int g(x,y)\ dx\ dy$ is equal to:

$$\begin{aligned} \int\int g(x,y)\ dx\ dy &= \int\int \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij} x^i y^j\ dx\ dy \\ &= \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} \frac{a_{ij} x^{(i+1)} y^{(j+1)}}{(i+1)(j+1)} \qquad \text{such that } \ i \text{ and } j \neq -1 \\ &+ a_{-1,-1} \ln(x) \ln(y) \\ &+ \ \ln(x) \sum_{j=-p_{y_1}}^{p_{y_2}} \frac{a_{-1,j} y^{(j+1)}}{(j+1)} \qquad \text{such that } \ j \neq -1 \\ &+ \ \ln(y) \sum_{i=-p_{x_1}}^{p_{x_2}} \frac{a_{i,-1} x^{(i+1)}}{(i+1)} \qquad \text{such that } \ i \neq -1 \end{aligned} \tag{2.31}$$

The two definite integrals in (2.29) (the numerator and denominator) can easily be obtained by the substitutions of the derived integrals in (2.30) and (2.31) which would then give the desired join selectivity $sel(R_1.b_x \bowtie R_2.b_y)$.

2.6.3.1 Algorithm and storage complexity for joins

Algorithm

For $sel(R_1.b_x \bowtie R_2.b_y)$, consider (2.30) and (2.31). Both equations require the retrieval of all $(p_x + 1) * (p_y + 1)$ coefficients a_{ij}'s, where $p_x = p_{x_1} + p_{x_2}$ and $p_y = p_{y_1} + p_{y_2}$. Hence, the time requirement for the retrieval is $(p_x * p_y + p_x + p_y + 1)$. But since $p_x \leq p_x * p_y$ and $p_y \leq p_x * p_y$, the time complexity would be $O(p_x * p_y)$.

Storage

It one were to store all the coefficients $(p_x + 1) * (p_y + 1)$ in the database profile catalog for the calculation of $sel(R_1.b_x \bowtie R_2.b_y)$ in (2.29), this would not, in practice, be a practical solution and the storage complexity for just a single join selectivity $sel(R_1.b_x \bowtie R_2.b_y)$ would be $O(p_x * p_y)$, which is too expensive.

A better approach is to off-line precompute $sel(R_1.b_x \bowtie R_2.b_y)$ in advance and store it in the profile catalog. This would then require only $O(1)$.

As per relation in the join $(R_1.b_x \bowtie R_2.b_y)$, since $R_1.b_x$ (and $R_2.b_y$) must share the (1-parameter) storage cost for the precomputed join selectivity stored in the catalog, the storage complexity per relation in the join is $O(1)$, if only one attribute in R_1 (and R_2) can be a join attribute.

Note that the storage complexity $O(1)$ above is introduced particularly for the computation of the join. That is, this storage requirement for the join is separate from that for selections $O(u * p)$.

2.6.4 Query size estimation for selections using ASE

ASE (Adaptive Selectivity Estimation) [Chen and Roussopoulos 1994] uses already-processed queries to build a fitting polynomial to capture a cumulative frequency distribution of an attribute $b_{\#}$ of relation R.

The resulting polynomial will then be gradually adjusted when more queries are processed by the database system. Already-processed queries are of the form $l \leq R.b_{\#} \leq h$ — i.e., select tuples on R which satisfy the selection condition $l \leq b_{\#} \leq h$, where l and h are a value in the domain of attribute $b_{\#}$, l is the lower bound of the condition, h is the upper bound and $l \leq h$. Let τ be the number of such queries which have been processed by the database system. Query feedback from a query (in those τ queries) is information about the query of the form (l_i, h_i, s_i) where $i = 1, 2, ..., \tau$. l_i is the lower bound value of the ith query, h_i is the upper bound value of the query and s_i is the number of tuples in relation R which satisfy the query.

The polynomial used by ASE is of the form:

$$g(x) = \sum_{i=0}^{p} a_i x^i \tag{2.32}$$

where p is the degree of the polynomial. Alternatively, $g(x)$ is the approximating function to fit the frequency distribution of attribute $b_{\#}$. Recall that $G(x)$ is the *cumulative frequency distribution function* of $g(x)$.

$$G(x) = \int g(x)\ dx = \int \sum_{i=0}^{p} a_i x^i\ dx = \sum_{i=0}^{p} \frac{a_i x^{(i+1)}}{(i+1)} \tag{2.33}$$

Using $G(x)$ in (2.33) and given a query with (l, h), the estimated number of tuples in R which satisfy the query would be:

$$G(l, h) = \sum_{i=0}^{p} \frac{a_i h^{(i+1)}}{(i+1)} - \sum_{i=0}^{p} \frac{a_i l^{(i+1)}}{(i+1)} = \sum_{i=0}^{p} a_i [\frac{h^{(i+1)}}{(i+1)} - \frac{l^{(i+1)}}{(i+1)}] \tag{2.34}$$

and this is the function which ASE uses to fit the cumulative frequency distribution of $b_{\#}$. That is, ASE uses the function $G(l, h)$ to fit the cumulative frequency distribution of $b_{\#}$ between low and high values (namely, l_i's and h_i's, respectively) of the already-processed queries.

The coefficients a_i's in equation (2.34) can be found by the principle of least square error. That is, choose some a_i's values to minimise:

$$\sum_{i=1}^{\tau} (G(l_i, h_i) - s_i)^2 \tag{2.35}$$

Transform the sum of squares in (2.35) to matrix form as follows:

$$(Y - XA)^T (Y - XA) \tag{2.36}$$

Solve equation (2.36) for the optimal coefficients in A:

$$\widehat{A} = (X^T X)^{-1} X^T Y \tag{2.37}$$

where X^T is the transpose of X. The following are the definitions for all the matrices in (2.37).

X is a $\tau \times (p+1)$ matrix and defined by:

$$X = \begin{bmatrix} (\frac{h_1^1}{1} - \frac{l_1^1}{1}) & (\frac{h_1^2}{2} - \frac{l_1^2}{2}) & \dots & (\frac{h_1^{(p+1)}}{(p+1)} - \frac{l_1^{(p+1)}}{(p+1)}) \\ (\frac{h_2^1}{1} - \frac{l_2^1}{1}) & (\frac{h_2^2}{2} - \frac{l_2^2}{2}) & \dots & (\frac{h_2^{(p+1)}}{(p+1)} - \frac{l_2^{(p+1)}}{(p+1)}) \\ \dots & \dots & \dots & \dots \\ (\frac{h_\tau^1}{1} - \frac{l_\tau^1}{1}) & (\frac{h_\tau^2}{2} - \frac{l_\tau^2}{2}) & \dots & (\frac{h_\tau^{(p+1)}}{(p+1)} - \frac{l_\tau^{(p+1)}}{(p+1)}) \end{bmatrix} \tag{2.38}$$

Matrix X consists of elements with values defined inside the squared brackets of the cumulative distribution function in (2.34). A is a $(p+1) \times 1$ matrix of the coefficients and Y is a $\tau \times 1$ response matrix. The following are A and Y:

$$A = \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_p \end{bmatrix}, \quad Y = \begin{bmatrix} s_1 \\ s_2 \\ \dots \\ s_\tau \end{bmatrix} \tag{2.39}$$

After solving the ordinary least squares in (2.37) for the coefficients a_i's $i = 0, 1, \dots, p$, given the solved coefficients:

- $sel(b_\# = x)$ is calculated by $g(x) = \sum_{i=0}^{p} a_i x^i$, divided by N.
- Using $G(l, h)$ in (2.34), $sel(b_\# < x)$ is calculated by:

$$sel(b_\# < x) = \frac{G(x_{min}, x) - g(x)}{N} = \frac{\sum_{i=0}^{p} a_i [\frac{x^{(i+1)}}{(i+1)} - \frac{x_{min}^{(i+1)}}{(i+1)}] - g(x)}{N} \tag{2.40}$$

The reason for the subtraction of $G(x_{min}, x)$ by $g(x)$ is that the selection predicate by $G(x_{min}, x)$ is $x_{min} \leq R.b_\# \leq x$. This predicate indicates the inclusion of the number of tuples with value x; since we only want any tuples with values less than x, we have to subtract $G(x_{min}, x)$ by $g(x)$, which is the number of tuples with value

x.

2.6.4.1 Algorithm and storage complexity for selections

Algorithm

The analysis for the algorithm complexity here is the same as that for IASE in Section 2.6.2.1. See that section for more details.

Thus, to calculate the query result size for a complex predicate query on R with a certain number of attributes ($\leq u$) involved in the query, the worst case time complexity would be $O(u * p)$, where u is the number of attributes of R.

Storage

The analysis for the storage complexity here is the same as that for IASE in Section 2.6.2.1. Hence the storage complexity is $O(u * p)$.

2.6.5 Query size estimation for joins using ASE

ASE does not propose any method to deal with joins. The experimental results have been done only with simple predicate queries – no results done for complex predicate queries.

However, it is not very hard to see that the method proposed by IASE for joins in Section 2.6.3 could also be used for ASE for this problem with a very small modification. The following is our modification.

The two-dimensional polynomial used by IASE is of the form:

$$g(x, y) = \sum_{i=-p_{x_1}}^{p_{x_2}} \sum_{j=-p_{y_1}}^{p_{y_2}} a_{ij} x^i y^j$$

which is used to fit the function $ff(x, y)$, the cartesian product between two join attributes. If p_{x_2} and p_{y_2} both are equal to 0, p_{x_1} is equal to p_x and p_{y_1} is equal to p_y, then the polynomial above by IASE would reduce to the equivalent form of the polynomial used by ASE, namely $g(x) = \sum_{i=0}^{p} a_i x^i$. Thus here could be a polynomial (a more strict form of the above) that can be used by ASE:

$$g(x, y) = \sum_{i=0}^{p_x} \sum_{j=0}^{p_y} a_{ij} x^i y^j$$

to fit the cartesian product function $ff(x, y)$ in Figure 2.7.

Like IASE, the join selectivity $sel(R_1.b_x \bowtie R_2.b_y)$ by ASE is also:

$$sel(R_1.b_x \bowtie R_2.b_y) = \frac{\int_{low}^{high} g_{\bowtie_=}(x)\ dx}{\int_{x_{min}}^{x_{max}} \int_{y_{min}}^{y_{max}} g(x,y)\ dx\ dy} \tag{2.41}$$

The indefinite integral of $\int g_{\bowtie_=}(x)\ dx$ is equal to:

$$\int g_{\bowtie_=}(x)\ dx \;=\; \sum_{i=0}^{p_x}\sum_{j=0}^{p_y} \frac{a_{ij}x^{(i+j+1)}}{(i+j+1)} \tag{2.42}$$

and the indefinite integral of $\int\int g(x,y)\ dx\ dy$ is equal to:

$$\begin{aligned}\int\int g(x,y)\ dx\ dy &= \int\int \sum_{i=0}^{p_x}\sum_{j=0}^{p_y} a_{ij}x^i y^j\ dx\ dy \\ &= \sum_{i=0}^{p_x}\sum_{j=0}^{p_y} \frac{a_{ij}x^{(i+1)}y^{(j+1)}}{(i+1)(j+1)} \end{aligned}\tag{2.43}$$

The two definite integrals in (2.41) (the numerator and denominator) can easily be obtained by the substitutions of the derived integrals in (2.42) and (2.43) which would then give the desired join selectivity $sel(R_1.b_x \bowtie R_2.b_y)$.

2.6.5.1 Algorithm and storage complexity for joins

Algorithm

The analysis for the algorithm complexity here is the same as that for IASE in Section 2.6.3.1. See that section for more details. Hence, the time complexity would be $O(p_x * p_y)$.

Storage

The storage computation for ASE is the same as that for IASE in Section 2.6.3.1. Hence, as per relation in the join $(R_1.b_x \bowtie R_2.b_y)$, the storage complexity per relation in the join is $O(1)$, if only one attribute in R_1 (and R_2) can be a join attribute.

2.7 Machine learning methods

2.7.1 Review on machine learning methods

Machine learning is a sub-field of artificial intelligence that aims to develop algorithms that derive relationships based on examples.

Quinlan [1993*b*] studied an extensive number of machine learning techniques and proposed to combine two learning techniques: *model-based learning* and *instance-based learning* into a new combined technique. In the study,

- Among three model-based learning approaches considered, linear regression [Draper and Smith 1966], model trees [Quinlan 1992] and neural networks [Rumelhart et al. 1986], the best approach to the model-based learning is through *model trees*.
- The instance-based learning technique proposed in the study was adopted from Aha et al. [1991].

According to the extensive results in the study using several real-world databases from the University of California at Irvine [Merz and Murphy 1996], any of the three combined learning techniques (with instance-based learning): (instance-based + linear regression), (instance-based + neural network) or (instance-based + model tree) performs better than its individual techniques. For example, the combined technique (instance-based + linear regression) outperforms the pure instance-based learning technique and the pure linear regression.

Moreover, among the three combined learning techniques above, the combined technique (instance-based + model tree) performs the best in many of the problem domains considered. In this thesis, we call this best combined learning technique M5. (In fact M5 was named after Quinlan [1993*b*] as one of his learning machines.)

Harangsri et al. [1997] proposed M5 for the problem of query size estimation for selections. Using feedback from already-processed queries (called *instances*), a model tree is created whose leaf nodes consist of linear regression functions (one function for one leaf node). When a new query is required to estimate its size, the most similar queries to the new query are picked up from the stored already-processed queries and the result size of the new query is calculated based on (1) some linear regression functions in the model tree and (2) the actual result sizes of the most similar already-processed queries.

Harangsri et al. [1997, 1996*c*] compared the performance between M5 and a curve-fitting method ASE (Adaptive Selectivity Estimation) [Chen and Roussopoulos 1994]. It appeared that M5 significantly outperforms ASE. The first comparison in [Harangsri et al. 1997] was made using simple predicate queries and the second one in [Harangsri et al. 1996*c*] was made using complex predicate queries. (In fact, the second comparison was made among 3 sampling-based methods and 3 non-sampling-based methods: UNF (parametric), ASE (curve-fitting) and M5 (machine learning).)

A main and important strength of the combined learning technique M5 is that like its predecessor ASE, it does not require scanning databases to build statistical parameters. This strength is very important because in recent years current databases have been growing larger and larger and scanning entire databases is expensive and time-consuming.

Since the presentation below is slightly different from other sections, i.e., which comprise query size estimation for selections and joins by one or two methods in a category, here we give the structure for the presentation of M5 below. We start in Section 2.7.2 by giving notations and definitions that we will use to describe the combined machine learning technique M5. Next in Section 2.7.3 we apply the technique to the query size estimation problem for selections and with a small extension to the estimation problem for joins in Section 2.7.4.

2.7.2 Notations and definitions

Field M5 deals with data items that are composed of a number of fields. In our context, a field is simply a component of a simple predicate query. Recall that a simple predicate query on relation R is one of the form: $b_{\#}$ *relopt* x where $b_{\#}$ is an attribute of R, *relopt* is one of the relational operators in $\{<, >, \leq, \geq, =, \neq\}$ and x is a value in the domain of $b_{\#}$. Simple predicate queries can be viewed as $(attribute, operator, value)$ tuples composed of three fields. For example, $(age, \leq, 23)$ and $(department, =,$"Computer Systems"$)$ are representations of two simple predicate queries.

Numerical field A field whose values are either integer or real. For example, the third field of the query $(age, \leq, 23)$ is numerical.

Notice that under this machine learning scheme we do not consider *age* to be numerical even though its underlying domain is clearly numerical; we are concerned primarily wth the attribute's *name*, not its semantics.

Discrete field A discrete field is one whose values primarily are strings. The first and second fields of simple predicate queries are discrete as they always contain attribute names and relational operators while the third field could be either numerical or string. For example, the third field of query $(departname, =$,"Computer Systems") is discrete.

Training set of queries $\mathcal{Q}$ A set consisting of $(query, size)$ pairs, where *query* is a simple predicate query on a certain attribute name of relation R and *size* is the result size of the *query*. All queries in the set each must be specified only on that particular attribute name. For simple predicate queries on other attribute names, the treatment is the same as that for the particular attribute name shown below.

Those queries in the training set have been collected from already-processed queries along with their result sizes; namely, each query q_i in the set is of the form:

$$q_i : \quad (b_{\#}, relopt, x_{scaled}, s_{q_i})$$

where $b_{\#}$ is a certain attribute name of R and s_{q_i} is the result size of this query (the number of tuples in R which satisfy the query). x_{scaled} is a linearly scaled version of x which appears in the original query:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where $0 \leq x_{scaled} \leq 1$, x_{min} is the minimum value and x_{max} is the maximum value of attribute $b_{\#}$.

Only values of the numerical field are scaled. Thus, for any simple predicate query, only when the third field x is numerical (while the other two fields $b_{\#}$

and *relopt* are always discrete) will the values of x be scaled by the formula above.

For numerical computation by M5, all discrete (string) values of a discrete field each must be assigned a *rank*, i.e., a numerical value. By following Quinlan's proposal in [Quinlan 1993*b*], ranks of a discrete field can be given as follows. Supposing in a training set $\mathcal{Q}$, all values of the second field (which consists of relational operators) are either of $\{=, <, >, \neq\}$, then the respective ranks of the relational operators can be given by: $\{1, 2, 3, 4\}$.

Unseen query q_u A simple predicate query $q_u : \quad (b_{\#}, relopt, x_{scaled})$ whose result size we want to estimate. After this unseen query is processed by a database system, it may proceed to be added to the training set of queries $\mathcal{Q}$. If the third field of the unseen query is numerical, then the value of this field must be scaled by the formula above prior to its size estimation.

2.7.3 Query size estimation for selections using M5

To approximate the size of an unseen query (the number of tuples in R which satisfy the query), there are two main steps. First, a model tree for attribute $b_{\#}$ must be constructed through a training set of queries $\mathcal{Q}$. Second, using (1) the model tree constructed and (2) three queries selected from the training set $\mathcal{Q}$ which are of the most similarity to the unseen query, the result size of the unseen query can then be approximated. Here are brief descriptions of each step.

First step (model tree construction) The construction for a model tree is described in Section 2.7.3.1 and the resulting tree with a number of leaf nodes will contain one linear regression function for one leaf node.

The construction for a model tree usually occurs off-line. This is analogous to building a histogram for attribute $b_{\#}$ off-line.

Second step (size estimation) Given the model tree constructed in the first step and an unseen query, the unseen query will be parsed down the tree to a leaf node where the size of the query can be approximated using the linear regression function in the leaf node. The query parsing and size approximation is described in Section 2.7.3.2.

Given an unseen query, the selection of the three most similar queries to the unseen query is described in Section 2.7.3.3. We then show how to combine the three most similar queries together with their linear regression (leaf node) functions to produce the estimated result size of the unseen query in Section 2.7.3.4.

As usual, this step of query size estimation occurs at query optimisation time.

2.7.3.1 Constructing a model tree (Constructing leaf node functions)

Given in Figure 2.10 is the **Partition** algorithm to construct a model tree — the tree with linear regression functions in its leaf nodes. The algorithm in the figure was implemented to suit our own use for the query size estimation problem. Two major differences between our modified version and the original version [Quinlan 1993*b*] are that our version here has no pruning procedure and no smoothing procedure. Furthermore, there may be other different internal fine tunings, such as the minimum number of queries in leaf nodes, the minimum error reduction to suppress the recursive partitioning, etc.

The modified version runs as fast as the original version due to the same algorithm complexity shared by both, but the estimation by the modified version is, in general, better with various frequency distributions we have experimented with than the estimation by the original version. This is due to the fine tunings we have made so that the modified version particularly suits the problem of query size estimation.

The idea to construct a model tree through the **Partition** algorithm is to minimise intra-subset variation of class values, i.e., query result sizes in our case. (In fact such an idea is similar to building a decision tree by C4.5 [Quinlan 1993*a*].) The minimisation of intra-subset variation in the algorithm (see lines 13 and 25) is implemented by:

$$\sum_{i=1}^{partition} \frac{|\mathcal{Q}_i|}{|\mathcal{Q}|} * sd(\mathcal{Q}_i)$$

where $\mathcal{Q}$ is a training query set and $\mathcal{Q}_i$ is the ith query subset of $\mathcal{Q}$, where $i = 1, 2, \ldots, partition$ and $partition$ is the number of all query subsets. All the query subsets $\mathcal{Q}_i$'s are *disjoint* in the queries they contain. $sd(\mathcal{Q})$ is the standard deviation

```
Procedure: Partition($\mathcal{Q}$)
input:     $\mathcal{Q}$, a simple predicate query set
output:    query disjoint subsets of $\mathcal{Q}$

if $|\mathcal{Q}| <$ a certain number of queries then
  return
endif
$\mathcal{Q}_{best} = \emptyset$
$\delta_{best} = -\infty$
$partition = 0$
for each field $i = 1$ to 3 do
  if $i$th field is numerical then
    $\mathcal{Q} =$ sort $\mathcal{Q}$ on $i$th field's values in ascending order
    for each sorted value $v$ of $i$th field in $\mathcal{Q}$ do
      partition $\mathcal{Q}$ into $\mathcal{Q}_1$ and $\mathcal{Q}_2$ where all values of $i$th field in $\mathcal{Q}_1 \leq v$ and in $\mathcal{Q}_2 > v$
      compute an expected error reduction $\delta$:
            $\delta = sd(\mathcal{Q}) - \sum_{i=1}^{2} \frac{|\mathcal{Q}_i|}{|\mathcal{Q}|} * sd(\mathcal{Q}_i)$
      if $\delta > \delta_{best}$ then
        $\mathcal{Q}_{best} = \{\mathcal{Q}_1, \mathcal{Q}_2\}$
        $\delta_{best} = \delta$
        $partition = 2$
      endif
    endfor
  else
    partition $\mathcal{Q}$ into $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_d$ where $d$ is the number of distinct values
       of $i$th field; namely, in each query subset $\mathcal{Q}_i$ after partitioning,
       the values of $i$th field now will be the same
    compute an expected error reduction $\delta$:
          $\delta = sd(\mathcal{Q}) - \sum_{i=1}^{d} \frac{|\mathcal{Q}_i|}{|\mathcal{Q}|} * sd(\mathcal{Q}_i)$
    if $\delta > \delta_{best}$ then
      $\mathcal{Q}_{best} = \{\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_d\}$
      $\delta_{best} = \delta$
      $partition = d$
    endif
  endif
endfor
if $(\frac{sd(\mathcal{Q}) - \sum_{i=1}^{partition} \frac{|\mathcal{Q}_i|}{|\mathcal{Q}|} * sd(\mathcal{Q}_i)}{sd(\mathcal{Q})}) * 100 <$ a certain percentage where each $\mathcal{Q}_i \in \mathcal{Q}_{best}$ then
  return
endif
for each query subset $\mathcal{Q}_i \in \mathcal{Q}_{best}$ do
  Partition($\mathcal{Q}_i$)
endfor
```

Figure 2.10: Partitioning algorithm

of all the result sizes of queries in the query set $\mathcal{Q}$ calculated by:

$$sd(\mathcal{Q}) = \sqrt{\frac{\sum_{i=1}^{|\mathcal{Q}|}(s_{q_i} - \bar{s})}{|\mathcal{Q}| - 1}}$$

where $\bar{s} = \frac{\sum_{i=1}^{|\mathcal{Q}|} s_{q_i}}{|\mathcal{Q}|}$.

The fundamental rationale behind for the intra-subset variation is that the query result sizes in each partitioned query subset $\mathcal{Q}_i$ will be most similar to one another; in other words, the variation of the sizes in the same query subset will be small.

Note that this rationale is similar to that in building any kind of serial histograms by grouping distinct values with their frequencies into histogram buckets such that the variation of frequencies – frequencies can be viewed as result sizes of the queries of form $(b_{\#}, =, x)$ – in the same bucket is low, thereby assisting in improving the quality of query size estimation.

The algorithm recursively partitions the training set of queries $\mathcal{Q}$ into query subsets $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_{partition}$ (see the details of the algorithm in Figure 2.10). The recursion stops in 2 cases (see lines 1 and 33). The first case is when there are not enough a number of queries in $\mathcal{Q}$ — less than a certain minimum number of queries in the set. The second is when the variation of result sizes in all the partitioned query subsets $\mathcal{Q}_i$'s is similar to one another, i.e.,

$$(\frac{sd(\mathcal{Q}) - \sum_{i=1}^{partition} \frac{|\mathcal{Q}_i|}{|\mathcal{Q}|} * sd(\mathcal{Q}_i)}{sd(\mathcal{Q})}) * 100 < \text{a certain percentage}$$

which relatively makes no significant difference from the overall variation of all the result sizes in the entire query set $\mathcal{Q}$.

Recall that simple predicate queries consist of three fields: (an attribute name, a relational operator, a constant value). The partitioning of $\mathcal{Q}$ into its query subsets depends upon whether the current ith field is numerical or discrete. If numerical, then do a binary partitioning on $\mathcal{Q}$ into $\mathcal{Q}_1$ and $\mathcal{Q}_2$ (see line 11). If not, then do a multi-way partitioning on $\mathcal{Q}$ into $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_d$ (see line 21), where d is the number of distinct values of the ith field in the query set $\mathcal{Q}$.

Figure 2.11 shows an example of a model tree for attribute b_1 constructed by the **Partition** algorithm. All the query subsets after the algorithm has terminated reside in the leaf (rightmost) nodes of the tree. The labels **A, B, C, . . ., L** show all the leaf nodes of the tree.

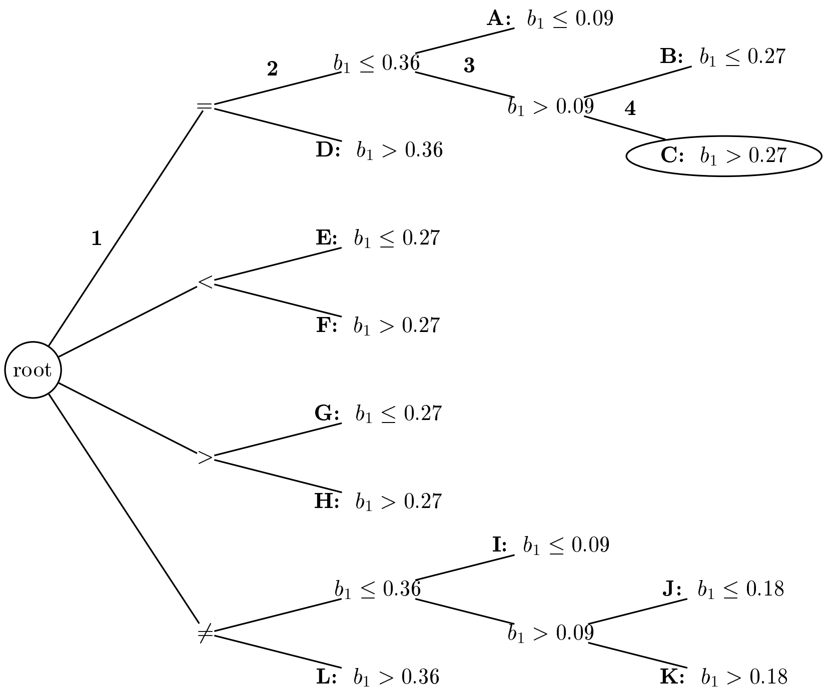


Figure 2.11: A model tree for attribute b_1 constructed by the algorithm **Partition**

Here is the description of how the model tree in Figure 2.11 was constructed. Recall that the query set $\mathcal{Q}$ contains all queries of the form ($b_{\#}$ *relopt* x). Starting from the root node, the entire query set $\mathcal{Q}$ was first multi-way partitioned into 4 query subsets, i.e., the query subset with "=" only as its relational operator, the query subset with "<" only as its operator, and so on. This is a multi-way partitioning.

In the next level (after the relational operator level), those 4 query subsets were then binary partitioned on their constant values, i.e., values of x. For instance, after the "<" level, at node **E** the constant values of b_1 must be less than or equal to 0.27 while at node **F** those of b_1 must be more than 0.27. This is a binary partitioning.

It is possible that any query subset at this stage can still be recursively partitioned further until either of the two stopping conditions of the algorithm becomes true. For example, at node $b_1 \leq 0.36$ the query subset at this node was binary partitioned into 2 query subsets – the subsets with $b_1 \leq 0.09$ and with $b_1 > 0.09$.

2.7.3.2 Result size estimation function in a leaf node

Suppose we have a query:

$$q: \quad b_1, =, 0.29$$

and we want to estimate its result size. The query will be parsed down the constructed model tree towards a leaf node. Shown in Figure 2.11, the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ terminated in the oval leaf node **C** is the one through which the query q traverses. The traversal proceeds as follows: path 1 stems from the fact that this model tree is for attribute b_1 and the query has "=" as its relational operator, path 2 is due to the fact that the constant value 0.29 of b_1 is less than 0.36, path 3 is due to the fact that the constant 0.29 of b_1 is more than 0.09 and path 4 is due to the fact that the constant 0.29 of b_1 is more than 0.27.

The oval and other leaf nodes (**A, B, C, . . ., L**) with their own query subsets have their own linear regression functions for query result size estimation. That is, each leaf node with a query subset $\mathcal{Q}_i$ has a function of the form:

$$g(q) = a_0 + a_1 x_{q,1} + a_2 x_{q,2} + a_3 x_{q,3} \tag{2.44}$$

where $x_{q,i},\ i = 1, 2, 3$ takes on either (1) a numerical scaled value if the ith field of query q is numerical or (2) a rank of the discrete value if the ith field of query q is discrete. (Recall that all values of a numerical field in the training query set must be scaled to have values between 0 and 1.) The one-to-one correspondence between a query q and $x_{q,i}$'s values are: $(b_1, =, 0.29) \equiv (x_{q,1}, x_{q,2}, x_{q,3})$.

With a query subset $\mathcal{Q}_i$ in a leaf node, the coefficients $a_k, k = 0, 1, 2, 3$ in (2.44) are ones of the least square error found by minimising the following sum of squares:

$$\sum_{j=1}^{|\mathcal{Q}_i|} (g(q_j) - s_{q_j})^2 \tag{2.45}$$

where s_{q_j} is the actual result size of the jth query in query subset $\mathcal{Q}_i$. Transform the sum of squares above to a solution of least square error as follows:

$$\widehat{A} = (X^T X)^{-1} X^T Y \tag{2.46}$$

where X^T is the transpose of X. The following are the definitions for all the matrices in (2.46).

$$X = \begin{bmatrix} 1 & x_{q_1,1} & x_{q_1,2} & x_{q_1,3} \\ 1 & x_{q_2,1} & x_{q_2,2} & x_{q_2,3} \\ 1 & \dots & \dots & \dots \\ 1 & x_{q_{|\mathcal{Q}_i|},1} & x_{q_{|\mathcal{Q}_i|},2} & x_{q_{|\mathcal{Q}_i|},3} \end{bmatrix}, \ A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}, \ Y = \begin{bmatrix} s_{q_1} \\ s_{q_2} \\ \dots \\ s_{q_{|\mathcal{Q}_i|}} \end{bmatrix} \tag{2.47}$$

where X is a $(|\mathcal{Q}_i| \times 4)$ matrix, A is a (4×1) matrix and Y is a $(|\mathcal{Q}_i| \times 1)$ matrix.

After solving the ordinary least squares in (2.45) for the coefficients a_k's $k = 0, 1, 2, 3$, given the solved coefficients, $g(q)$ in (2.44) as the linear regression function in the leaf node to which query q belongs, can be used to approximate the query size of q.

2.7.3.3 Selecting most similar queries

Given an unseen query q_u, we show how to compute a similarity value *simval* between the unseen query q_u and a query q_j in the training query set $\mathcal{Q}$ by the algorithm shown in Figure 2.12.

```
Procedure: Compute_Simval($q_u$, $q_j \in \mathcal{Q}$)
input:     $\mathcal{Q}$, a simple predicate query set,
           $q_u$, an unseen query,
           $q_j$, a query in the training set $\mathcal{Q}$
output:    simval similarity value of query $q_j$ to $q_u$

distance = 0
for each field $i = 1$ to 3 do
  if $i$th field is discrete then
    if $i$th field's value of the unseen query $q_u$ is different from that of query $q_j$ then
      $distance = distance + 1$
    endif
  else
    $\Delta = (x_{q_u,i} - x_{q_j,i})$, i.e., the difference between $i$th field's values of the unseen query $q_u$ and query $q_j$
    $distance = distance + \Delta^2$
  endif
endfor
$simval = \frac{1}{\sqrt{distance}}$
```

Figure 2.12: An algorithm to compute a similarity

Note that in the algorithm, if the resulting *distance* after the **for** loop is zero or approaches zero, the value of *simval* will be ∞ in which case the queries q_u and q_j are either the same query or very similar. As a result, this query q_j will be selected as one of the most similar queries to q_u.

By iterating through the whole query set $\mathcal{Q}$ query by query, three queries from $\mathcal{Q}$ with the highest similarity values will be chosen as the most similar to the unseen query. The reason in choosing 3 queries instead of other numbers is twofold [Quinlan 1996]:

- Choosing any number has a tradeoff between bias and variance (see the CART book [Breiman et al. 1984] for the relationship between bias and variance); namely, higher numbers have higher bias but lower variance.
- Generally, the instance-based learning as a KNN (K Nearest Neighbor) method [Duda and Hart 1974; Dasarathy 1991] avoid even numbers, since that number is more likely to lead to ties. Using 1 nearest neighbour (query) is generally considered to have too high variance.

2.7.3.4 Combining leaf node functions and most similar queries

We then adjust the actual result sizes s_{q_i}'s of the three most similar queries to the unseen query q_u by:

$$\tilde{s}_{q_i} = s_{q_i} - (g(q_i) - g(q_u)) \qquad ; \; i = 1, 2, 3$$

prior to combining them to produce the estimated result size $\hat{s}_{q_u}$ of q_u. The combination of the adjusted values $\tilde{s}_{q_i}$'s is done by:

$$\hat{s}_{q_u} = \sum_{i=1}^{3} \tilde{s}_{q_i} * w_{q_i} \qquad ; \; w_{q_i} = \frac{simval_{q_i}}{\sum_{i=1}^{3} simval_{q_i}}. \tag{2.48}$$

which basically averages the estimated result size by taking the weights of the most similar queries into consideration. Hence the selectivities $sel(b_{\#} < x)$ and $sel(b_{\#} = x)$ are calculated by $\frac{\hat{s}_{q_u}}{N}$, where q_u is either $(b_{\#}, <, x)$ or $(b_{\#}, =, x)$.

At this point, it is appropriate for one to ask:

- Why don't we use $g(q_u)$ in (2.44) as the estimated query size $\hat{s}_{q_u}$? This estimation is the pure model tree learning technique.

- Or why don't we use:

$$\hat{s}_{q_u} = \sum_{i=1}^{3} s_{q_i} * w_{q_i} \qquad ; \; w_{q_i} = \frac{simval_{q_i}}{\sum_{i=1}^{3} simval_{q_i}}$$

  as the estimated query size $\hat{s}_{q_u}$? This estimation is the pure instance-based learning technique.

The answer is twofold:

- Quinlan [1993*b*] noted that the function $g(q)$ allows taking into account the difference between the unseen query q_u and a most similar query q_i, i.e.:

$$g(q_i) - g(q_u)$$

  and if $g(q)$ is correct, then the adjusted value $\tilde{s}_{q_i}$:

$$\tilde{s}_{q_i} = s_{q_i} - (g(q_i) - g(q_u))$$

  should be a better value in favour of the unseen query than the quantity s_{q_i} itself. However, since each most similar query q_i is not the unseen query q_u itself, the combination of their adjusted values in (2.48) based on their weights (this is the main principle of KNN) would produce a good estimate for the size of the unseen query.

- The second answer is pragmatic: both the experiments in [Quinlan 1993*b*] and our own experiments in [Harangsri et al. 1997, 1996*c*] show that formula (2.48) for the combined learning technique (instance-based + model tree) yields better results than its individual learning techniques.

2.7.3.5 Algorithm and storage complexity for selections

Algorithm

In the worst case of constructing a model tree, the constructed tree will be a binary tree, i.e., without any multi-way partitioning in the internal nodes. This will make the height of the tree higher than usual, as compared to a model tree with multi-way partitioning internal nodes. Alternatively, multi-way partitioning nodes generally reduce the height of a model tree. Below, the time complexity analysis is based on the worst case binary tree as a model tree.

Let q_u be either $(b_{\#} = x)$ or $(b_{\#} < x)$. To calculate $sel(q_u)$, consider (2.48). $sel(q_u)$ can be found by the following two steps.

1. selecting the three most similar queries to q_u from the training query set $\mathcal{Q}$. This requires $|\mathcal{Q}|$ similarity calculations to find the most similar queries, giving $O(|\mathcal{Q}|)$.

2. parsing each of the most similar queries $q_i,\ i = 1, 2, 3$ and the unseen query q_u down the model tree to a leaf node for a leaf node function $g(q)$.

   Let $leaf\_nodes$ be the number of leaf nodes in a model tree which is calculated by:

   $$leaf\_nodes = \frac{|\mathcal{Q}|}{\text{the average number of queries in each leaf node}}$$

   Using the worst case binary tree as a model tree for the analysis here, since a model tree constructed by the **Partition** algorithm in Figure 2.10 will be reasonably balanced, this then requires h steps:

   $$h \;=\; \log_2\ (leaf\_nodes) \tag{2.49}$$

   to traverse down to a leaf node of the balanced binary model tree [Manber 1989*a*]. h is also the height of the tree.

   Thus, the time requirement to parse the three most similar queries and the unseen query is $4 * h = 4 * \log_2\ (leaf\_nodes)$. By ignoring the constant factor 4, $\log_2\ (leaf\_nodes)$ is bounded by $\log_2 |\mathcal{Q}|$.

Since the growth rate of $|\mathcal{Q}|$ in step 1 is more than $\log_2 |\mathcal{Q}|$ in step 2, the time complexity in calculating $sel(q_u)$ is bounded by $O(|\mathcal{Q}|)$.

Thus, to calculate the query result size for a complex predicate query on R with a certain number of attributes $(\leq\ u)$ involved in the query, the worst case time complexity would be $O(u * |\mathcal{Q}|)$, where u is the number of attributes of R.

Storage

The analysis for storage requirement is based on the balanced binary tree above as the worst case model tree. The total storage requirement is:

$$\overbrace{\text{storage size for a balanced binary tree}}^{\text{model tree}} + \overbrace{\text{storage size for } |\mathcal{Q}| \text{ queries}}^{\text{instance-based}} \tag{2.50}$$

Storage size for a balanced binary tree

For a balanced binary tree, the number of internal and leaf nodes is calculated by Table 2.4.

level	nodes
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
...	...
h	$2^h = 2^h$

Table 2.4: Tree nodes

Using the geometric series [Manber 1989*b*], the total of Table 2.4 is equal to:

$$\begin{aligned} 1 + 2 + 4 + \ldots + 2^h &= 2^{h+1} - 1 \\ &= 2^h * 2 - 1 \end{aligned} \tag{2.51}$$

Substituting the height h of the balanced tree in (2.49) to (2.51), we obtain:

$$leaf\_nodes * 2 - 1 \tag{2.52}$$

as the total number of internal and leaf nodes for the tree. Each of these nodes in h levels of the tree would require a fixed size, say c_1, of storage for storing a cutting point (i.e., one node with one cutting point). Hence the storage size for all the cutting points is: $c_1 * (leaf\_nodes * 2 - 1)$, assuming that each leaf node also stores a cutting point.

Leaf nodes need to store information for the linear regression functions. This simply requires storing the four coefficients a_0, a_1, a_2 and a_3 in each leaf node. The total storage requirement for all leaf nodes is thus $4 * leaf\_nodes$.

Taking all of the preceding into account, the total storage size for a balanced binary model tree is:

$$c_1 * (leaf\_nodes * 2 - 1) + 4 * leaf\_nodes$$

Storage size for $|\mathcal{Q}|$ queries

Each query in $|\mathcal{Q}|$ queries requires a fixed storage size, say c_2, to store a simple predicate query. Hence all $|\mathcal{Q}|$ queries would require: $c_2 * |\mathcal{Q}|$.

Thus the total storage requirement in (2.50) is:

$$c_1 * (leaf\_nodes * 2 - 1) + 4 * leaf\_nodes + c_2 * |\mathcal{Q}|$$

which gives an $O(|\mathcal{Q}|)$ storage complexity.

Both the calculation for $sel(b_{\#} = x)$ and $sel(b_{\#} < x)$ shares a common model tree for attribute $b_{\#}$ with the storage complexity $O(|\mathcal{Q}|)$. Hence, the storage complexity for all u attributes of relation R is $O(u * |\mathcal{Q}|)$.

2.7.4 Query size estimation for joins using M5

The following is an approach to using existing model trees to estimate the selectivity $sel(R_1.b_x \bowtie R_2.b_y)$ of a natural join between attributes $R_1.b_x$ and $R_2.b_y$.

Assume that two model trees are existing for the join attributes $R_1.b_x$ and $R_2.b_y$. We can also imagine that a model tree encompasses an overall function which can approximate the frequency distribution of an attribute. Thus the two model trees have two such functions $g_x(x)$ and $g_y(y)$ for the join attributes $R_1.b_x$ and $R_2.b_y$, respectively. Figure 2.13 shows the two functions.

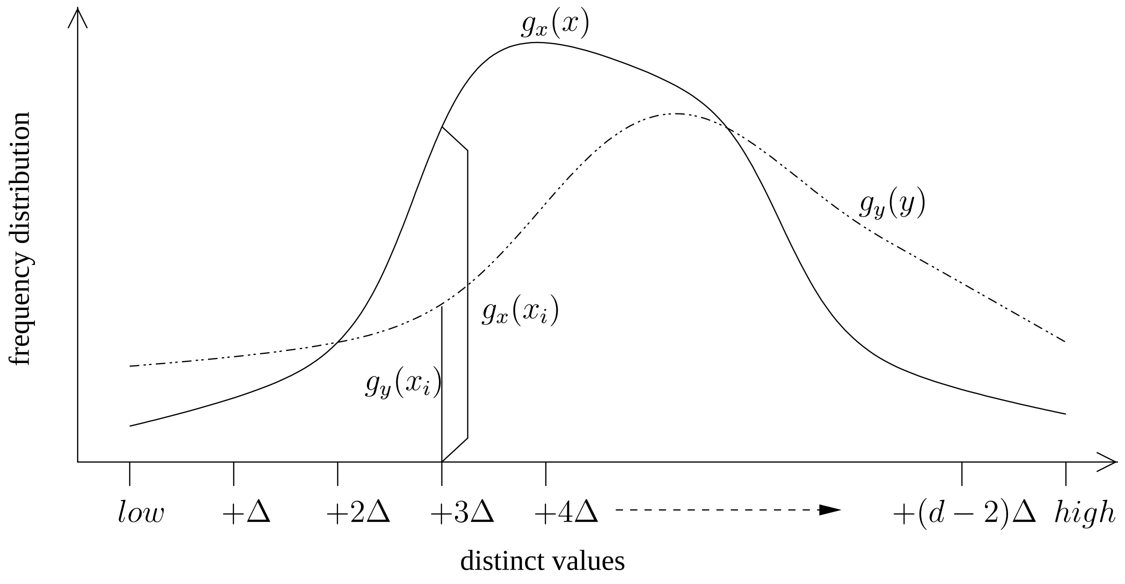


Figure 2.13: Join selectivity calculation

Recall that x_{max} and x_{min} are the maximum and minimum values of attribute $R_1.b_x$, respectively. Likewise, y_{max} and y_{min} are the respective maximum and minimum values of attribute $R_2.b_y$. low in Figure 2.13 is the larger value between x_{min} and y_{min} and $high$ is the smaller value between x_{max} and y_{max}. d_x is the number of distinct values of attribute $R_1.b_x$ and d_y is the number of distinct values of attribute $R_2.b_y$. d is the smaller value between d_x and d_y. N_{R_1} and N_{R_2} are the number of tuples of R_1 and R_2, respectively. The increment value Δ shown in the figure is calculated by: $\frac{(high-low)}{(d-1)}$.

$sel(R_1.b_x \bowtie R_2.b_y)$, the selectivity of the natural join between $R_1.b_x \bowtie R_2.b_y$ is thus calculated by:

$$sel(R_1.b_x \bowtie R_2.b_y) = \frac{\sum_{i=1}^{d} g_x(x_i) * g_y(x_i)}{(N_{R_1} * N_{R_2})} \tag{2.53}$$

where $x_i = low + (i-1) * \Delta$. Figure 2.13 helps to understand how the formula (2.53) works. The height of the two graphs – denoting the approximate frequency distributions of the two join attributes – basically denotes the number of tuples from $R_1.b_x$ and that from $R_2.b_y$ which are joinable, i.e., having a common value. In the figure, we use the notations $g_x(x_i)$ and $g_y(x_i)$ as the heights of the two graphs. The two graphs start from the low value and increment by Δ to the $high$ value — i.e., $low, (low + 1 * \Delta), (low + 2 * \Delta), \ldots, high$, where $high = (low + (d-1) * \Delta)$. The height of a graph, say $g_x(x_i)$, the approximate number of tuples with value x_i can always be obtained by posing an unseen query against its model tree $q_u : (b_x, =, x_i)$, where $i = 1, 2, \ldots, d$. If x_i is numerical, then x_i must be scaled to a value between 0 and 1 before its use.

2.7.4.1 Algorithm and storage complexity for joins

Algorithm

For $sel(R_1.b_x \bowtie R_2.b_y)$, consider (2.53). To obtain $g_x(x_i)$ (and $g_y(x_i)$), the unseen query $q_u : (b_x, =, x_i)$ (and $q_u : (b_y, =, x_i)$) must be posed against the model tree for $R_1.b_x$ (and $R_2.b_y$). This requires $O(|\mathcal{Q}|)$ which is the same as the analysis for the time complexity of $sel(b_\# < x)$ and $sel(b_\# = x)$.

By taking into account the repetition of d times in (2.53) for all distinct values in the common join domain, the time complexity for $sel(R_1.b_x \bowtie R_2.b_y)$ would be $O(d * |\mathcal{Q}|)$.

Storage

The storage cost for $sel(R_1.b_x \bowtie R_2.b_y)$ shares the same cost for the model tree for selections on $R_1.b_x$ (and on $R_2.b_y$), which is $O(|\mathcal{Q}|)$. See the analysis in Section 2.7.3.5.

As per relation in the join $(R_1.b_x \bowtie R_2.b_y)$, the storage complexity per relation in the join is $O(|\mathcal{Q}|)$, if only one attribute in R_1 (and R_2) can be a join attribute.

2.8 Sampling methods

2.8.1 Review on sampling methods

Over the last decade, sampling methods have received considerable attention from many researchers. Compared with other size estimation methods described earlier, there are a larger number of studies, e.g., [Hou et al. 1988, 1989; Lipton et al. 1990; Haas and Swami 1992, 1995] done in support of these sampling-based methods. Some advantages of sampling-based methods are:

- They use the least amount of storage to store statistical parameters in the database profile catalog.
- They can handle dependence among attributes due to the fact that whole tuples are considered. Other multi-dimensional techniques (e.g., multi-dimensional histograms [Muralikrishna and DeWitt 1988; Poosala and Ioannidis 1997], the multi-dimensional curve-fitting method [Sun et al. 1993]) can also deal with dependence among attributes of a relation but require significant storage in the catalog for statistical parameters.
- They do not rely on the underlying data model used by a database system.
- Compared with many other approaches, sampling-based algorithms for selectivity estimation are very simple to implement – perform sampling over a number of relations, i.e., one relation or more and use the resulting sample relation(s) to estimate a selectivity.

- They are supported by a well-developed statistical theory which allows the determination of the amount of sampling required to achieve a certain error bound in the size estimate.

Most previous work [Hou et al. 1988, 1989; Lipton et al. 1990; Hou et al. 1991*b*; Haas and Swami 1992, 1995] on sampling-based methods has focused on simple random sampling (SRS) whereby each tuple in a relation has an equal probability to be selected into a sample. Simple random sampling can be performed under two distinct regimes. The first is with replacement; that is, the tuple which has already been selected from the population can subsequently be selected again. We call this scheme of sampling SRSWR. Most previous SRS work uses this scheme due to (1) the aim of SRS that each tuple must be selected with the same probability and (2) its simpler implementation. The second scheme does not allow replacement; any tuple already selected can not be selected again. This scheme which we call SRSWOR requires a more sophisticated data structure to do sampling. The simple random sampling methods proposed so far mainly differ from one another primarily in their stopping conditions, i.e., when to stop sampling.

Towards the end of this section, we review 4 variants of sampling methods: fixed-step, time-constrained, double and sequential sampling. Generally any sampling method can be used for both selection and join query size estimation problems. In what follows, we will, however, review such 4 variants in terms of selection size estimation.

We begin by defining some notations. N is the cardinality (size) of a relation R and n is the size of a sample relation R' from R. Let $\overline{Y}$ be the selectivity of a complex predicate query defined by $\overline{Y} = \frac{\sum_{i=1}^{N} y_i}{N}$ where $y_i = 1$ if the ith tuple in R satisfies the query; otherwise $y_i = 0$. $\widehat{\overline{Y}}$, the estimated selectivity of the query, is defined similarly to $\overline{Y}$, namely, $\widehat{\overline{Y}} = \frac{\sum_{i=1}^{n} y_i}{n}$. In the sampling literature, $\overline{Y}$ and $\widehat{\overline{Y}}$ are called the *population mean* and *sample mean*, respectively.

Let S^2 be the population variance defined by: $S^2 = \frac{\sum_{i=1}^{N}(y_i - \overline{Y})^2}{(N-1)}$. With a sample size n, let S_n^2 be a sample variance defined by: $S_n^2 = \frac{\sum_{i=1}^{n}(y_i - \widehat{\overline{Y}})^2}{(n-1)}$. Let ϵ be a relative error, typically whose value provided is between 0 and 1, such that with a certain confidence level, the error of the estimated $\widehat{\overline{Y}}$ yielded by a sampling method is within the relative error desired, i.e., the sampling method should guarantee $\frac{abs(\overline{Y} - \widehat{\overline{Y}})}{\overline{Y}} \leq \epsilon$, where *abs*() is the absolute value of the value given.

Fixed-step sampling

Hou and Ozsoyoglu wrote a series of papers on applying sampling techniques, more particularly simple random sampling techniques, to the query size estimation problem. In [Hou et al. 1988; Hou and Ozsoyoglu 1991], the goal of the work is to use simple random sampling to estimate COUNT(E) queries where E is an arbitrary relational algebra expression. Apparently, the stopping condition is not the main focus of the work and thus a sample size n selected is left to the discretion of the database system administrator or a privileged user. Hence, sometimes the size of n selected can be too large (oversampling) or too small (undersampling) and this affects the accuracy of estimated query result sizes. This way of sampling by fixing the number of steps (the size of n) in advance is called *fixed-step* sampling [Haas et al. 1996].

The authors also proposed an enhancement called *cluster sampling* to the simple random sampling proposed. That is, the unit of sampling is *disk page* or *disk block*, instead of a *tuple* in a relation. The former is called *page-level* sampling and the latter is called *tuple-level* sampling. The improvement is because of the fact that when the database I/O system interacts with the file system, the disk page, not the tuple, is fetched into main memory for processing and thus all tuples in a disk page would be exploited, instead of using only one tuple in the disk page and probably ignoring the remaining tuples if the tuple-level sampling is used. As a consequence, a substantial amount of the I/O time can be saved.

Time-constrained sampling

Hou et al. [1989] extended their work (fixed-step sampling) to the real-time environment where time constraints are an important factor to query processing in real-time database systems. The sample size n, which is not the main focus of the earlier work, now is the main one of this work and is controlled by a time quota given. The time-control algorithm will iteratively perform sampling until no time remains. This way of sampling can be called *time-constrained* sampling.

Double sampling

After the time-constrained query size estimation, Hou et al. [1991*a*] also proposed an *error-constrained* size estimation algorithm. Given a query and a relative error ϵ, the proposed algorithm which is a *double sampling* [Cox 1952] does a two-stage sampling. In the first stage, use a small sample size n_1 to approximate the selectivity of the query, i.e., $\widehat{\overline{Y}}_{n_1}$ and calculate the sample size n required by:

$$n = (\frac{(\frac{t}{\epsilon})^2(1-\widehat{\overline{Y}}_{n_1})}{\widehat{\overline{Y}}_{n_1}} + \frac{3}{\widehat{\overline{Y}}_{n_1}(1-\widehat{\overline{Y}}_{n_1}}) + \frac{t^2}{\epsilon^2\widehat{\overline{Y}}_{n_1}n_1}$$

where t is the abscissa of the normal curve that cuts off an area α at the tail and α is a risk of error not within the relative error ϵ given.

In the second stage, if $n \leq n_1$, then there is no more sampling to be done and the algorithm will terminate. That is, the sample size n_1 is sufficient for this query. At this point, substituting n in equation (2.54) below by n_1 would give the estimated selectivity for the query. If $n > n_1$, sample the remaining tuples $n - n_1$ which are not yet taken. The selectivity of the query is then estimated by:

$$\widehat{\overline{Y}} = (\frac{s}{n} - (\frac{\epsilon}{t})^2\frac{s}{(n-s)}) \tag{2.54}$$

where s ($\leq n$) is the total number of tuples in the sample which satisfy the query. One problem with this algorithm is that it does not not address an *oversampling* problem – too much sampling can occur when the selectivity of a query is very small. The sequential sampling algorithms below do address this problem.

Sequential sampling

Lipton and Naughton [1989]; Lipton et al. [1990]; Lipton and Naughton [1990] proposed what they call *adaptive sampling*. In fact, the adaptive sampling is a *sequential sampling* — one decides whether to continue or stop sampling after each unit of sampling is obtained. All sequential sampling algorithms described below are error-constrained, i.e., by a given relative error ϵ.

It is well known that sequential sampling algorithms excels fixed-step sampling algorithms in terms of the sample size required [Olken 1993]. This is as a consequence of the fact that in the derivation for a formula for a sample size n required, sequential sampling algorithms primarily take into account the population parameters, e.g., variance and mean. The stopping condition of the adaptive sampling is defined by:

$$\overbrace{s \geq k_1 b(\frac{1}{\epsilon}+1)\frac{1}{\epsilon}}^{\text{subcond. 1}} \quad \text{or} \quad \overbrace{n \geq k_2 h^2}^{\text{subcond. 2}} \tag{2.55}$$

where $b > \frac{S^2}{\overline{Y}}$ (the upper bound value of b is 1), $h = 100 * \epsilon$ as well as k_1 and k_2 are a constant which can be found in [Lipton et al. 1990]. Roughly, both constants k_1 and k_2 were derived from a number of sample relations under two cases. The first case is when the distribution of $\widehat{\overline{Y}}$'s calculated from the samples around $\overline{Y}$ is normally distributed. The second case is when such a distribution is not.

Whenever either of the two subconditions in (2.55) is satisfied, the algorithm will terminate. The first subcondition is to deal with the number of tuples that satisfy the given query while the second subcondition is to deal with the number of tuples sampled so far. The reason one needs the second subcondition is that in case the selectivity of a query is very small, the right hand side of the first subcondition, i.e., $k_1 b(\frac{1}{\epsilon}+1)\frac{1}{\epsilon}$ will yield a very high value perhaps even more than the size of the relation itself, hence leading to the oversampling problem. The second subcondition serves as a *sanity bound* – bound to assure that the oversampling problem will not arise. When the problem does occur – hence, the first subcondition will always be false –, the second subcondition will take effect. That is, this subcondition will control the termination of the algorithm.

The main disadvantage of the adaptive sampling algorithm is that the value of b must be computed *a priori* via a pilot sample in the relation on which the query is posed. The precomputed value b is then used in the stopping condition (2.55). Since the relation may be dynamically changed over time, the precomputed b may become outdated and no longer fit nicely with the current data in the relation. As a result, the sampling for selectivity estimation of queries on the relation which is based on the precomputed b can sometimes be too large (oversampling) or too small (undersampling).

Haas and Swami in [Haas and Swami 1992; Haas et al. 1993; Haas and Swami 1995; Haas et al. 1996] noticed the problem with the *a priori* bound b and proposed an improved stopping condition by taking into consideration the information in the sample obtained so far, i.e., the mean and variance. Their stopping condition is:

$$\overbrace{n \geq 1}^{\text{subcond. 1}} \quad \text{and} \quad \overbrace{S_n^2 > 0}^{\text{subcond. 2}} \quad \text{and} \quad \overbrace{\epsilon \max(s, n\psi) \geq t(nS_n^2)^{\frac{1}{2}}}^{\text{subcond. 3}} \tag{2.56}$$

When all of the three subconditions are true, then the sampling algorithm will terminate. When the selectivity of a given query is very small, then the value of s, the number of tuples thus far which satisfy the query, in the third subcondition will grow very slowly. Given the very small selectivity, supposing that the given third subcondition is just:

$$\epsilon s \geq t(nS_n^2)^{\frac{1}{2}}$$

(i.e., without the sanity bound term $n\psi$) then the algorithm will cause the oversampling problem because the rate of growth to s is so slow that even if the sample size n reaches a certain maximum size limit which is allowed for sampling, the algorithm still does not terminate.

The term $n\psi$ introduced in $\max(s, n\psi)$ in the third subcondition will solve the oversampling problem. That is, $n\psi$ will become more than s (as the value of n is always incremented every time a new tuple is sampled, while the value of s may or may not be incremented) and thus make the whole subcondition become true and the algorithm terminates.

2.8.2 Query size estimation for selections using SS

Ling and Sun [1995] compared three representative simple random sampling algorithms with replacement. Their analytical and experimental work demonstrated that the most effective algorithm is the sequential algorithm proposed in the series of papers by Haas and Swami [Haas and Swami 1992; Haas et al. 1993; Haas and Swami 1995; Haas et al. 1996]. This sequential sampling algorithm for complex predicate queries is shown in Figure 2.14.

Given a complex predicate query Q, the algorithm in Figure 2.14 will sample tuples of the relation specified in the query tuple by tuple until the stopping condition becomes true. The stopping condition in line 16 is the one in equation (2.56). The variance S_n^2 calculated from the information in the sample so far is equal to

```
Procedure: Selection_Selectivity_Estimation
input:   $Q$ = a complex predicate query,
         $0 < \beta \leq 1$ = maximum sampling fraction,
         $\epsilon$ = relative estimation error,
         $\psi \geq 0$ = sanity bound
var:     $s$ = number of tuples in the sample relation which satisfy $Q$ $(s \leq n)$,
         $n$ = total number of tuples in the sample relation so far,
         $y = \{0, 1\}$,  1 if the tuple sampled satisfies $Q$; 0 otherwise
output:  the estimated selectivity of Q

1   $n = 0;$  $s = 0;$  $w = 0$
2   repeat
3     $y = 0$
4     obtain an SRS tuple from R
5     if the tuple satisfies $Q$ then
6       $s = s + 1$
7       $y = 1$
8     endif
9     $n = n + 1$
10    $w = \frac{(n-1)}{n} w + \frac{(s-ny)^2}{(n+1)n}$
11    if the sample size $n > 1$ then
12      $S_n^2 = \frac{w}{(n-1)}$
13    else
14      $S_n^2 = 0$
15    endif
16    if $S_n^2 > 0$ and $\epsilon \max(s, n\psi) \geq t(nS_n^2)^{\frac{1}{2}}$ then
17      return $\frac{s}{n}$
18    endif
19  until $n \geq \lceil(\beta * N)\rceil$
20  return $\frac{s}{n}$
```

Figure 2.14: Algorithm for selectivity estimation for selections

$S_n^2 = \frac{w}{(n-1)}$ where $w := \frac{(n-1)}{n} w + \frac{(s-ny)^2}{(n+1)n}$ in line 10. That is, every time a new tuple is sampled, the new variance S_n^2 will be updated by the new w and n.

Note that line 11 has the condition to protect the division by zero by $S_n^2 = \frac{w}{(n-1)}$ when the sample size n is equal to 1. When n is equal to 1, i.e., in the first iteration of the repeat-until loop, then S_n^2 would be zero.

For a practical reason, we added another stopping condition in line 19 of the algorithm to stop sampling whenever the sample size so far exceeds the maximum sample size limit ($\lceil(\beta * N)\rceil$) where $\lceil\rceil$ is the ceiling of a value, regardless of what the stopping condition in line 16 is like.

2.8.2.1 Algorithm and storage complexity for selections

Algorithm

Consider the repeat-until loop in Figure 2.14. In the worst case, the algorithm will terminate at the maximum sample size limit ($\lceil(\beta * N)\rceil$). R' is a sample relation of R. Let $N_{R'} = \lceil(\beta * N)\rceil$.

To calculate the query result size for a complex predicate query on R with a certain number of attributes ($\leq u$) involved in the query, the time complexity would be $O(N_{R'})$, where u is the number of attributes of R.

Note that the time complexity does not rely on the number of attributes of R, unlike the time complexities for all the other non-sampling-based methods considered earlier.

Storage

For a relation in a database, it is reasonable that three parameters: β (maximum sampling fraction), ϵ (relative error) and ψ (sanity bound) can be used by the sampling algorithm to estimate selectivities of any complex predicate queries on R.

Hence, these three are needed to be stored in the database profile catalog and the storage complexity would be $O(1)$.

The storage complexity for sampling-based methods does not depend upon u, the number of attributes of R, while the storage complexities for all the other non-sampling-based methods considered earlier do depend.

2.8.3 Query size estimation for joins using SS

In this thesis, star joins are considered. The reason is justified in Section 1.4.1 of Chapter 1. A star join is a join in which any join attribute of the two or more participating relations can join one another on a common join domain. A star join is a join whose qualification is of the form:

$$R_1.a_1 = R_2.a_2 = R_3.a_3 \cdots = R_m.a_m$$

where $R_1, R_2, \ldots, R_m$ are the relations participating in the join and $a_1, a_2, \ldots, a_m$ are the join attributes of $R_1, R_2, \ldots, R_m$, respectively and m is the number of relations participating in the star join. Let R'_i be a sample relation of R_i, where $i = 1, 2, \ldots, m$ and let $N_{R'_i}$ be the cardinality of R'_i.

Similar to the sequential sampling algorithm for selections in Figure 2.14, the algorithm for star join selectivity estimation is shown in Figure 2.15.

```
Procedure: Join_Selectivity_Estimation
input:   $Q$ = a star join query,
         $0 < \beta \leq 1$ = maximum sampling fraction,
         $\epsilon$ = relative estimation error,
         $\psi \geq 0$ = sanity bound
var:     $s$ = number of all tuples in the output of the join among the sample relations ($s \leq n$),
         $n$ = number of all combinations of the tuples in the sample relations obtained so far
output:  the estimated selectivity of Q

1  repeat
2    obtain an SRS tuple from each $R_i$ $i = 1, 2, \ldots, m$ and append it into $R'_i$
3    $s = \sum_{i_1=1}^{N_{R'_1}} \sum_{i_2=1}^{N_{R'_2}} \cdots \sum_{i_m=1}^{N_{R'_m}} (t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$
4    $n = \prod_{j=1}^{m} N_{R'_j}$
5    $\hat{\mu} = \frac{s}{n}$
6    $S_n^2 = \frac{n}{n-1}\hat{\mu}(1 - \hat{\mu})$
7    if $S_n^2 > 0$ and $\epsilon \max(s, n\psi) \geq t(nS_n^2)^{\frac{1}{2}}$ then
8      return $\hat{\mu}$
9    endif
10 until any of $N_{R'_i} \geq \lceil(\beta * N_{R_i})\rceil$ becomes true, where $i = 1, 2, \ldots, m$
11 return $\hat{\mu}$
```

Figure 2.15: Algorithm for selectivity estimation for star joins

The respective stopping conditions of the two algorithms in lines 16 and 7 are the same; the difference between the two is the way we overload S_n^2, s and n. The details of the overloading are below.

Let μ be the selectivity of a given star join and $\hat{\mu}$ an estimated selectivity of the join. The estimated selectivity $\hat{\mu}$ stems from:

$$\hat{\mu} = \frac{\sum_{i_1=1}^{N_{R'_1}} \sum_{i_2=1}^{N_{R'_2}} \cdots \sum_{i_m=1}^{N_{R'_m}} (t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})}{\prod_{i=1}^{m} N_{R'_i}} \tag{2.57}$$

where $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ is a join among tuples i_1th, i_2th, …, i_mth of the participating sample relations. $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ will give value 1 if all the tuples are joinable, i.e., having a common attribute value. Otherwise it will give value 0.

For a complex predicate query, its selectivity is $\frac{s}{n}$. The numerator s is the number of all tuples in the sample relation which satisfy the complex predicate. The denominator n is the number of all tuples in the sample relation.

The selectivity for a star join is defined similarly: $\frac{s}{n}$. The numerator in (2.57) is the number of tuples which satisfy the join predicate – i.e., the number of all tuples in the output of the join among the sample relations – and thus is equivalent to s. The denominator $\prod_{i=1}^{m} N_{R'_i}$ is the size of the cross-product of all the participating sample relations and thus is equivalent to n. Hence, (2.57) is equal to $\frac{s}{n}$.

$S^2 = \frac{N}{N-1}\mu(1-\mu)$. We show the derivation for this formula in Chapter 3. In the algorithm, $S_n^2 = \frac{n}{n-1}\hat{\mu}(1-\hat{\mu})$ whose derivation is similar to the derivation for S^2.

Like the sequential sampling algorithm for selections, we added another stopping condition in line 10 of the algorithm for star joins to stop sampling whenever any sample size so far exceeds the maximum sample size limit ($\lceil(\beta * N_{R_i})\rceil$) where $i = 1, 2, \ldots, m$, regardless of what other subconditions are like.

2.8.3.1 Algorithm and storage complexity for joins

Algorithm

Since the time complexities considered earlier are all for $sel(R_1.b_x \bowtie R_2.b_y)$, the time complexity considered here would also be for the binary join.

$N_{R'_1}$ and $N_{R'_2}$ are the sizes of two sample relations R'_1 and R'_2. Let R'_{12} be the output relation of the join $(R'_1.b_x \bowtie R'_2.b_y)$ and let $N_{R'_{12}}$ be the size of the output relation.

Suppose that a hash-join algorithm [DeWitt et al. 1984] is used as a method to join between two relations. The reason we choose the hash-join algorithm is that the algorithm has a low algorithm complexity, $O(|R'_1| + |R'_2|)$, i.e., only needs a single pass through R'_1 and R'_2 to join them on the matching join attribute values.

The computation of the join $(R'_1.b_x \bowtie R'_2.b_y)$ proceeds as:

1. read in the two sample relations R'_1 and R'_2, which requires $O(N_{R'_1} + N_{R'_2})$.

2. write out the matching tuples of the two relations, which requires $O(N_{R'_{12}})$.

As a result, the time complexity to calculate $sel(R_1.b_x \bowtie R_2.b_y)$ would be $O(N_{R'_1} + N_{R'_2} + N_{R'_{12}})$.

Storage

The **input** part of the algorithm in Figure 2.15 indicates that there are only 3 parameters: β, ϵ and ψ that need to be stored in the database profile catalog for an m-relation star join, where m (≥ 2) is the number of relations in the star join. A binary star join like ($R_1.b_x \bowtie R_2.b_y$) is a specific case of the m-relation star join.

Hence, the storage complexity for selectivity estimation for an m-relation star join would be $O(1)$.

As per relation in the join ($R_1.b_x \bowtie R_2.b_y$), the storage complexity per relation in the join is $O(1)$, if only one attribute in R_1 (and R_2) can be a join attribute.

Note that the storage requirement $O(1)$ above would more likely be introduced separately for the computation of the join selectivity, i.e., separate from the storage requirement $O(1)$ for selections. That is, the three parameters will not be shared between selections and the join – selections and the join would have their own version of the three parameters.

2.9 Summary of algorithm and storage complexities

Table 2.5 shows the summary of algorithm and storage complexities consumed by each estimation method discussed earlier.

Here is a conclusion from the table.

- For selections and joins, UNF has the least algorithm complexity and except SS, UNF also uses the least amount of storage.

- For selections and joins, SS requires the least storage. SS will, however, most likely require the most time in selection and join selectivity estimation.

- For selections, (equi-width and -height) histograms typically use $I = 10 - 15$ buckets. IASE and ASE use p, normally no more than 10. M5 typically uses the number of queries $|\mathcal{Q}|$ in the training set more than 15 queries. By the four methods, the algorithm and storage complexities for selections can be ordered from least to most as follows: IASE and ASE, histograms and M5.

- For joins, IASE and ASE will require less time in calculating $sel(R_1.b_x \bowtie R_2.b_y)$ than M5 if d, the number of distinct values, is more than 7.

method	algorithm	storage
UNF	$O(u)$	$O(u)$
hist*	$O(u * I)$	$O(u * I)$
IASE	$O(u * p)$	$O(u * p)$
ASE	$O(u * p)$	$O(u * p)$
M5	$O(u * \|\mathcal{Q}\|)$	$O(u * \|\mathcal{Q}\|)$
SS	$O(N_{R'})$	$O(1)$

(a) selections

method	algorithm	storage
UNF	$O(1)$	$O(1)$
hist*	$O(1)$	$O(1)$
IASE	$O(p_x * p_y)$	$O(1)$
ASE	$O(p_x * p_y)$	$O(1)$
M5	$O(d * \|\mathcal{Q}\|)$	$O(\|\mathcal{Q}\|)$
SS	$O(N_{R'_1} + N_{R'_2} + N_{R'_{12}})$	$O(1)$

(b) joins

method	storage sharing
UNF	share storage
hist*	half-share storage
IASE	have separate storage
ASE	have separate storage
M5	share storage
SS	have separate storage

(c) storage sharing between selections and joins

Table 2.5: The summary of algorithm and storage complexities
hist* = equi-width and equi-height histograms

The reason is that given that $p_x, p_y \leq 10$ and $|\mathcal{Q}| \geq 15$, $p_x * p_y$ will be bounded by $d * |\mathcal{Q}|$.

- For joins, M5 requires the most storage but this storage cost is also shared between selections and joins. (Recall that the same model tree is used for both selections and joins.)

- The storage sharing between selections and joins is summarised in Table 2.5(c). Note that the half sharing of storage for histograms is because one part of the total storage for joins is the one commonly used by both selections and joins, while the other part is particularly introduced for joins.

2.10 Cost model derivation for multidatabase systems

In this section we review the previous studies on the cost model derivation for multidatabase systems which fundamentally relies on curve-fitting techniques.

Although the review in this section is not much related as a query size estimation method itself, the main purpose of the review below is to raise the significance of query size estimation methods to the cost model derivation problem. That is, in order to obtain an accurate cost model, a good query size estimation method is required.

A number of studies [Dayal 1984; Zhu 1992; Ngu et al. 1993; Evrendilek et al. 1995; Du et al. 1995; Ozcan et al. 1996] proposed various search algorithms and architectures for query processing and optimisation in multidatabase systems. All such studies raise a significant problem of how to obtain the *unknown cost model* of each local database which participates in a multidatabase system. This problem is very severe as a result of the *local autonomy* maintained by the local systems. That is, a local system may or may not provide any statistical information which forms a cost model for the local database to the global query optimiser. Without sufficient statistical information from all local databases, the cost calculation for query execution plans by the global query optimiser is very difficult to be carried out or may not be able to be done at all.

Du et al. [1992]; Zhu [1993]; Zhu and Larson [1994]; Harangsri et al. [1996*a*] studied the derivation of the unknown cost model for a local database. The cost model is a model for cost estimation which can be used to approximate the cost of database operations, e.g., joins and selections in queries (and thus the cost of query execution plans). The cost of a database operation can be measured as *elapsed time* which is calculated by: (the ending time when the operation is completed - the starting time when the operation is received) [Du et al. 1992].

The calculation for an elapsed time of an operation, however, relies on the estimated size(s) of (intermediate) relation(s) in the operation (the formulas related to the sizes will be shown shortly below). In turn, obtaining estimated sizes of intermediate relations is the query size estimation or selectivity estimation problem. Hence, we feel that the crucial and perhaps more important problem of the cost model derivation is the problem of how to obtain a good size estimation method so that an accurate elapsed time of an operation can be obtained. This is because without reliable estimates for the sizes of intermediate relations, how can an accurate elapsed time of an operation be obtained ?

The following is a summary of how an estimated elapsed time of a join or selection operation is carried out, in chronological order of the previous studies done.

• By the studies of Du et al. [1992]; Zhu [1993]; Zhu and Larson [1994], selection operations are manually classified into a number of classes. The same was done for join operations. Each class will then have a cost formula for the estimation of an elapsed time taken by a selection (or join) operation falling into this class. The manual classification requires to know *a priori* the local storage structures of a local database system, which in several cases is impossible due to the local autonomy.

• Use a multiple regression, a curve-fitting technique [Du et al. 1992; Zhu 1993; Zhu and Larson 1994], to derive the *cost parameters* (defined after the formula) for a local database. The cost formula to estimate an elapsed time taken by a selection operation belonging to a class is given by:

$$g_\sigma(N_{R_1}, N_{R_1'}) = a_0 + a_1 N_{R_1} + a_2 N_{R_1'}$$

where N_{R_1} is the cardinality of the relation in the selection operation, $N_{R_1'} =$ (selectivity of the selection $* N_{R_1}$) and a_i's, $i = 0, 1, 2$ are the cost parameters, namely, coefficients of least square error for the polynomial used. Likewise, the cost formula to estimate an elapsed time taken by a join operation belonging to a class is given by:

$$g_{\bowtie}(N_{R_1}, N_{R_2}, N_{R_{12}}) = a_0 + a_1 N_{R_1} + a_2 N_{R_2} + a_3 N_{R_{12}}$$

where N_{R_1} and N_{R_2} are the cardinalities of the two (possibly intermediate) relations in the join operation, $N_{R_{12}} =$ (selectivity of the join$*N_{R_1}*N_{R_2}$) and a_i's, $i = 0, 1, 2, 3$ are the cost parameters.

• Sample queries with selection operations from each class and run them against a database to observe the elapsed times of the queries in the class. Those observed times would be the left-hand side of the $g_\sigma(N_{R_1}, N_{R_1'})$ function above. By the principle of least square error, the function can be solved for the cost parameters. The same process is done for join operations.

• To observe the elapsed times of the sample queries, Du et al. [1992] proposed to use the *synthetic database* (as opposed to the *actual database*) to act as a local database and hence, to derive the cost parameters from the synthetic database. Note that the cost formulas proposed by this work, although different, are very similar to the ones given above — they really give the same feeling that the multiple regression is used to solve this estimation problem.

• Instead of using the synthetic database, Zhu [1993]; Zhu and Larson [1994] improves the above work by using an actual local database to derive the cost parameters.

• Harangsri et al. [1996*a*,*b*] improves the above two manual classifications by an automatic classification and uses an actual local database. Given (sampled queries with) selection operations, they are automatically categorised into a number of classes where each class has a cost formula like above. (The same process is done for join operations.) The authors proposed to use a clustering algorithm so called *hierarchical clustering algorithm* [Anderberg 1973] to perform the automatic classification. This approach does not require to know *a priori* the local storage structures and join methods used of a local system. The algorithm attempts to group selection operations (the same for join operations) into classes such that the error in elapsed time estimation is reduced for the selection operations placed in the same class.

CHAPTER 3

Query size estimation using systematic sampling

Abstract

We develop a theoretical foundation for systematic sampling which suggests that the method gives a more representative sample than the traditional simple random sampling. Subsequent experimental analysis on a range of synthetic relations confirms that the quality of sample relations yielded by systematic sampling is higher than those produced by the traditional simple random sampling.

To ensure that the sample relations produced by the systematic sampling indeed assist in computation for more accurate query result sizes, we compare the systematic sampling with the most efficient simple random sampling called SS using a variety of relation configurations. The results obtained are impressive in that the systematic sampling uses an equal or less amount of sampling but still can provide more accurate query result sizes than SS. Furthermore, the overhead cost incurred by systematic sampling is no more than by SS.

3.1 Introduction

In this chapter, we propose a new sampling-based method called *systematic sampling* (SYSSMP) for the result size estimation of two main database operations: *join* and *selection*. The systematic sampling was earlier proposed by Zhu [1993] with no exploitation of sortedness of data. Systematic sampling on unsorted data (which is called a *random population*) proposed by the work has been statistically known to be as efficient as SRSWOR (see the proof in [Cochran 1963; Murthy and Rao 1988]). In addition, we have also done a preliminary study on the quality of query result sizes approximated by the systematic sampling on unsorted data and the results we obtained correspond with the theoretical proof.

Subsequently, Harangsri et al. [1996*c*] have done a preliminary investigation on systematic sampling with exploitation of sortedness of data and compared SYSSMP on sorted data with 5 other query size estimation methods — a parametric method based on a uniform distribution, a curve-fitting method ASE, a machine learning method M5 and 2 sampling methods, i.e., SRSWR and SRSWOR. The results obtained from the preliminary investigation led us to a conclusion that SYSSMP is apparently the most robust query size estimator, which can deal very well with either join or selection queries.

The preliminary investigation, however, has three main disadvantages: (1) the relation of interest must be sorted usually on more than one attribute, (2) the method requires storage to store summary relations, (3) a statistical ground of the quality of sample relations yielded by the systematic sampling is not given. The systematic sampling proposed here utilises sorted data (if this is the case), requires no extra storage and has a statistical ground of the quality of sample relations obtained.

We have discovered both theoretically and empirically that when data are in order (*ordered population*), the quality of sample relations obtained via SYSSMP is more likely to be superior to the quality of sample relations obtained via SRSWOR and SRSWR. For joins, the quality of a sample relation is determined by a total variance of estimated selectivities for all distinct values[1] in a common join domain whereas for selections, the quality is determined by a variance of an estimated selec-

[1]The selectivity of a distinct value is a ratio, defined by the total number of tuples in a relation having the distinct value divided by the cardinality of the relation.

tivity for the selection. This discovery fundamentally implies that a sample relation obtained via SYSSMP can, in general, well represent the underlying joint frequency distribution of the attributes in the original relation. Figure 3.1 shows a "conceptual" example of how a sample relation represents the underlying actual frequency distribution of an attribute in the original relation. The bottom curve in the figure represents the frequency distribution of the attribute in the sample relation against the actual frequency distribution (shown in the top curve) of the attribute in the original relation.

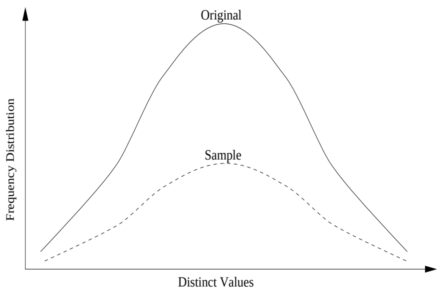


Figure 3.1: A frequency distribution of a sample against an original relation

The overall idea behind systematic sampling is: start with a relation R with cardinality N whose tuples can be accessed in ascending or descending order of an attribute of R; decide on the size n of a sample relation; to produce the sample relation, select a tuple at random from the first $k = \lceil \frac{N}{n} \rceil$ tuples of R and every kth tuple thereafter, where $\lceil\rceil$ is the ceiling of a value.

In order to create a sample relation through SYSSMP, the first two alternatives below can be used:

- If the relation of interest has an index (such as B$^+$-tree [Bayer and McCreight 1972] on any of its attribute, then we can use such as index to obtain a sample relation. (Salzberg [1988] emphasizes the importance of B$^+$-trees in page 143 that virtually no other indexing schemes are currently in use for large files except B$^+$-trees and hashing.) When a B$^+$-tree is created for an attribute, it can then be traversed systematically, for example, in every jth branch of the tree (recall that B$^+$-tree has several branches called *fan-out*) down to its leaf nodes to retrieve tuples for a sample relation. The proposal by Lipton and Naughton [1990] for simple random sampling also requires using such an index in order to obtain a sample relation.

- If the relation does not contain any index but is sorted on one of its attribute, then a systematic sample relation can also be obtained. Figure 3.2 shows a relation with 5 disk blocks and each block can accommodate 5 fixed-length records.

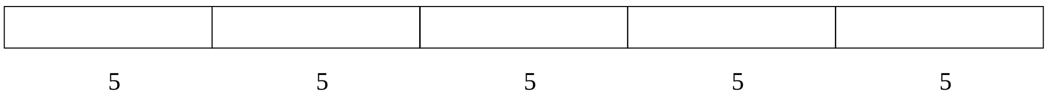


Figure 3.2: A fixed-length record relation

If the tuple required is *tup_required*, then the *i*th disk block where the tuple stays is simply calculated by:

```
for each disk block $i = 1$ to $num\_diskblocks$ do
    if $tup\_required \leq i \cdot block\_factor$ then
        return disk block $i$
    endif
endfor
```

where $block\_factor = 5$ (the number of records per block is 5) and $num\_diskblocks = 5$.

- If the relation contains no index or sorted attribute, then we can simply resort to any of the traditional SRS methods proposed in the literature in order to create a sample relation.

  Note however, that in several database schemas, an attachment of an index into an attribute of interest or the sort on an attribute occurs frequently (they are not something unusual) and as a result, we expect that the method developed here can often be employed in database systems as an alternative (i.e., whenever data are sorted) together with the traditional sampling methods.

In the remainder of this chapter, the term *index* used denotes any of the two alternative above, namely, (1) a physical index on an attribute or (2) a sorted attribute on a relation. Whichever is available on the relation is the index for the relation.

Tuple-level sampling is the sampling whose sampling unit is *tuple* as opposed to another *page-level* sampling whose sampling unit is *disk page* or *disk block.* The most efficient SRS named SS (S̲equential S̲ampling) was proposed in a series of papers by Haas and Swami [1992]; Haas et al. [1993]; Haas and Swami [1995]; Haas et al. [1996]. To justify the superiority of our sampling method for both join and selection queries, we compare SS and SYSSMP via tuple-level sampling using a variety of relation configurations. In regard to joins, we compare the two methods using a variety of star joins — a join in which any join attribute of the two or more participating relations can join one another on a common join domain.

For join queries, in an extreme case of SYSSMP, each join attribute participating in a star join requires an index on it (one index for one join attribute). This requirement is most likely impossible in practice to occur because some of the participating relations may not have any index on the join attributes. To get around the problem, we propose a hybrid sampling scheme between SYSSMP and SS. The scheme works as follows: samples of the relations with the indices on the join attributes are created via SYSSMP and samples of the relations with no indices on the join attributes are created via SS. (It is generally known that standard simple random sampling schemes do not require any index as a must for the creation of sample relations although they can also use indices for the creation.) All the samples are then joined together to produce an estimated join selectivity of the star join.

We conduct 2 sets of experiments for join queries. The first is done for the extreme case, by assuming that each relation has an index on its join attribute. The results obtained from this first set are that with the same amount of sampling, SYSSMP in the extreme case is far more superior in approximating join result sizes than the SS procedure. With the second set of experiments, namely, some relations participating in a star join do not have indices on their join attributes, while the rest do have, the results obtained are that with the same amount of sampling, despite a decreased performance to a certain degree in approximating join result sizes, the hybrid sampling scheme between SYSSMP and SS still outperforms the pure SS procedure.

We conduct one more set of experiments for selection queries. The results obtained are impressive in that SYSSMP uses a less amount of sampling but still can provide more accurate query result sizes.

The overhead cost (e.g., storage maintenance or sampling algorithm complexity) incurred by SS and SYSSMP is the same because first, both of them do not require building any extra storage structure in order to use the methods and second, both sampling algorithms are pretty much the same. However, the major advantage SYSSMP has over SS is that it exploits the "freely available" sortedness of data via the index available while SS does not.

The structure of presentation can be separated into two main parts. The first part is for joins between Sections 3.3–3.5 and the second is for selections between Sections 3.6–3.8. Since the two parts have some common structure of presentation, whenever they do, we put the section numbers for selections in parentheses (see below).

Section 3.2 defines a *proportion* model [Cochran 1963] from which the selectivity for joins and selections can be calculated. Section 3.3 (3.6) gives the basic idea of SYSSMP for joins (selections) and an intuitive explanation of why SYSSMP would be more efficient than SRS with/without replacement. The question of "how many tuples in a relation would be required in a sampling ?" is very important for any sampling-based method. We provide the answer in Section 3.3.1 (3.6.1). Next in Section 3.4 (3.7) we give a theoretical foundation that when a relation can be accessed via an index to obtain systematic tuples in ascending or descending order of an attribute of the relation of interest, SYSSMP can in general yield more efficient sample relations than SRSWOR and SRSWR. Following that, in Section 3.5.2 (3.8.2) we do an analytical study on the quality of sample relations yielded by SYSSMP, SRSWOR and SRSWR in order to corroborate the foundation laid before. The detailed experimental results to demonstrate the performance between SS and SYSSMP are shown in Section 3.5.3 (3.8.3). Finally, we give a conclusion in Section 3.9.

3.2 Proportion model

Let the term *point* denote an element in a population of interest. Assume that there are a certain number of classes in the population. A *proportion* in a population is the number of points which fall into each class in the population, e.g., the proportion of people with false teeth, the proportion of people who watch a particular TV program, the proportion of students who prefer to learn the JAVA programming language and etc.

For the problem of selectivity estimation for joins and selections, the number of classes is 2. Let y be a point in a population. If a point y in the population satisfies a condition, then the point value y would be 1; the point value y would be 0 otherwise. For example, for a selection if a point (in this case a tuple) satisfies the selection (condition), then $y = 1$; otherwise $y = 0$. As for a star join with m participating relations, if a join among m tuples from the m relations (one tuple from each relation) in the star join produces an output tuple in the output relation; i.e., all the values are in common of the join attributes in the m tuples, then $y = 1$; otherwise $y = 0$. Note that the phrase "produces an output tuple in the output relation" is equivalent to satisfying the condition (join predicate).

3.3 Systematic sampling for joins

Table 3.1 shows most common notations used in the main sections for joins, namely, Sections 3.3 and 3.4 (the main sections for selections are 3.6 and 3.7). The notations in Table 3.1(a) are used in Section 3.3 while the ones in Table 3.1(b) are used in Section 3.4. More details of each notation are described later in the corresponding sections.

Given a star join on

$$R_1.a_1 = R_2.a_2 = R_3.a_3 \cdots = R_m.a_m$$

where $R_1, R_2, \ldots, R_m$ are the relations participating in the join and $a_1, a_2, \ldots, a_m$ are the join attributes of relations $R_1, R_2, \ldots, R_m$, respectively, the basic idea of systematic sampling is as follows: consider a relation R_i with cardinality N participating in the star join; obtain a sample relation of R_i by the algorithm in Figure 3.3.

Repeat the algorithm in Figure 3.3 to take sample relations for all the relations in the star join.

Let R'_i be a sample relation of relation $R_i, i = 1, 2, \ldots, m$. All the sample relations R'_i's obtained above are then joined together to yield an output relation. We denote the output relation by $R'_{123\ldots m}$. We then calculate an estimated selectivity $\hat{\mu}$ of the

$\tilde{y}_i$	a point in a population of a star join $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m$ whose value can be either 0 or 1
$\tilde{S}^2$	a population variance over the population of the join $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m$
$\tilde{N}$	$\prod_{i=1}^{m} \|R_i\|$, a population size for the join
$\tilde{n}$	$\prod_{i=1}^{m} \|R'_i\|$, a sample size for the join where R'_i is a sample relation of R_i
μ	a star join selectivity among $R_1, R_2, \ldots, R_m$
$\hat{\mu}$	an estimated star join selectivity of μ
$V(\hat{\mu})$	the variance of the estimated $\hat{\mu}$

(a)

y_i, y_{ij}	a point in relation R, whose value can be either 0 or 1
S^2, S_i^2	a population variance over relation R of a selectivity of a distinct value
S^2_{wsmp}, $S^2_{wsmp_i}$	a variance within systematic sample relations
N	a cardinality of R, a population size of R
n	a cardinality of R', a sample size on R where R' is a sample relation of R
$\overline{Y}$	the selectivity of a distinct value on R
$\hat{\theta}$	the selectivity of a distinct value on R' could be $\widehat{\overline{Y}}$ obtained via SYSSMP, $\widehat{\overline{Y}}_{swr}$ obtained via SRSWR or $\widehat{\overline{Y}}_{swor}$ obtained via SRSWOR
$V(\hat{\theta})$	the variance of the estimated $\hat{\theta}$
d	the number of distinct values in a common join attribute domain
T	$\sum_{i=1}^{d} S_i^2$, the total variance of each population variance S_i^2 in a common join domain
$\overline{T}$	$\frac{T}{d}$, the average of the total variance T
T_{wsmp}	$\sum_{i=1}^{d} S^2_{wsmp_i}$, the total variance of each $S^2_{wsmp_i}$
$\overline{T}_{wsmp}$	$\frac{T_{wsmp}}{d}$, the average of the total variance T_{wsmp}

(b)

Table 3.1: Symbol definitions

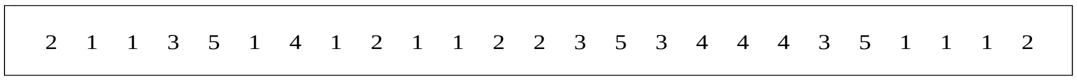
STEP 1. calculate a sample size $n = \beta * N$ where β is a *sampling fraction* $(0 < \beta \leq 1)$ required for each relation participating in the star join. (We describe details of how to obtain β in Section 3.3.1).

STEP 2. calculate a step size $k = \lceil \frac{N}{n} \rceil$ to step through relation R_i.

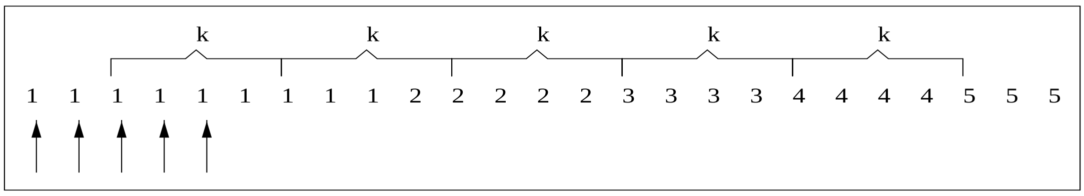
STEP 3. sample a tuple at random from the first k tuples of R_i. For example, In Figure 3.4(b) (showing a relation on a join attribute), $N = 25$ and $n = 5$ so $k = \frac{25}{5} = 5$. A starting random tuple, i.e., the first random tuple, could be any of the tuples marked by the arrow signs. In the figure, the third tuple is chosen as the starting random tuple.

STEP 4. repeat taking a tuple, k tuples away from the current one until the sample size obtained so far is equal to $\beta * N$ and terminate sampling. Figure 3.4(b) shows all the next tuples taken from the starting random tuple.

Figure 3.3: Sample a relation



(a) Unsorted values of an attribute



(b) Values after sorting

Figure 3.4: Unsorted and sorted values and stepping through the sorted values

star join by:

$$\hat{\mu} = \frac{|R'_{123\ldots m}|}{|R'_1||R'_2|\cdots|R'_m|}$$

Hence, an estimated size of the star join is calculated by $\hat{\mu} * |R_1||R_2|\cdots|R_m|$. In order to obtain an accurate estimated join selectivity for a star join, we need a method which can yield "good" sample relations; that is, each of the sample relations obtained can well represent the frequency distribution of the join attribute in the original relation.

Here is an intuitive description of why a sample relation yielded by SYSSMP would be more efficient (i.e., giving a better and more representative sample relation) than a sample relation yielded by SRSWOR or SRSWR.

In view of Figure 3.4, one can easily see that, after all values of the attribute have been sorted in Figure 3.4(b), we will be able to step through all the sorted values and get the values that can well represent the underlying distribution of the attribute in the original relation. On the other hand, with the same relation which perhaps is unordered as shown in Figure 3.4(a), the method of SRS, which simply picks out tuples at random to create a sample relation, would not guarantee to form any good sample relation. The reason could be twofold as follows.

First, with the SRSWR scheme (which most previous work uses), the same tuples may get selected again and again, thereby wasting the sampling being done. Second, with SYSSMP some "redundant" tuples (that do not assist in forming a good representative distribution of the original relation) after the relation got sorted, will never be selected again because with SYSSMP, the next tuple to be selected into a sample relation is the next kth tuple from the current one. For example, in Figure 3.4(b), the tuples with value 1 are selected twice and this kind of selection would not be guaranteed to occur in the case of the simple random sampling with/without replacement.

3.3.1 How many tuples to sample

Based on the theory of simple random sampling, we derive a "conservative" sample size $\beta * |R_i|$ for a systematic sample relation to be taken from each relation R_i participating in a star join. $0 < \beta \leq 1$ is a sampling fraction, a fraction of a total number of tuples to be sampled from each R_i. The reason of being conservative is that if a population of interest is arranged in some order, then the number of units in a sample obtained via simple random sampling will be extra large as a sample size for a systematic sample [Scheaffer et al. 1990]. That is, the number of units needed in a systematic sample should be less than that needed by a simple random sample.

Recall that R'_i is a sample relation of relation R_i, $i = 1, 2, \ldots, m$ participating in a star join. Let a sample size $\tilde{n} = \prod_{i=1}^{m} |R'_i|$ and likewise, a population size $\tilde{N} = \prod_{i=1}^{m} |R_i|$. $\tilde{n}$ would be equivalent to $\prod_{i=1}^{m} \beta * |R_i|$ if each relation R_i is sampled with the same sampling fraction β. We denote an actual star join selectivity by μ (and an estimated join selectivity of the actual μ by $\hat{\mu}$).

Towards the end of Section 3.3.1.2 is the derivation for a sample size $\tilde{n}$ and thus a sampling fraction β. We start by defining a population mean and population variance $\tilde{S}^2$ of a star join in Section 3.3.1.1. The population variance $\tilde{S}^2$ will then be substituted into the derivation for $\tilde{n}$. We then show the derivation for a formula for $\tilde{n}$ for SRSWOR in Section 3.3.1.2 which can also be used for SYSSMP. The formula for $\tilde{n}$ requires a substitution of an actual join selectivity μ. We can use $\hat{\mu}$ obtained from a previous sampling as an estimate (or substitute) for the actual μ. The details of how to initialise the initial value of $\hat{\mu}$ are described in Section 3.3.1.4.

There are times that a star join selectivity can be very small such that a calculated β can be too large, i.e., larger than a maximum sampling fraction limit that one would want to occur in a database system. We propose a solution to this problem in Section 3.3.1.3.

$\tilde{n}$ was first derived by Haas and Swami [1995]. There are two differences, however, between the derivation in [Haas and Swami 1995] and our derivation hereafter.

First, $\tilde{n}$ in [Haas and Swami 1995] was derived based on the simple random sampling *with replacement* (as the sampling procedure SS is sampling with replacement) while here we show the derivation for sampling *without replacement* as it can be used for SYSSMP — this is due to the fact that SYSSMP is, in fact, a sampling without replacement.

The second difference is that in [Haas and Swami 1995], the number of tuples to be sampled from each relation R_i in a star join is an absolute fixed number, say $\mathcal{X}$ and thus $\tilde{n} = \mathcal{X}^m$, while in our case $\tilde{n} = \prod_{i=1}^{m} \beta * |R_i|$. With the $\mathcal{X}^m$ formula, if two relations with 5 and 100 tuples, for example, are in a star join, and $\mathcal{X} = 5$, then the entire relation with 5 tuples will be sampled!, while only 5 tuples will be sampled from the second relation with 100 tuples. We feel that a more natural way of doing sampling over relations in a star join should be done using the same sampling fraction over each relation in the star join.

3.3.1.1 Population mean and variance

The join selectivity μ of a star join with m participating relations $R_1, R_2, \ldots, R_m$, is calculated by:

$$\mu = \frac{|R_{123\ldots m}|}{|R_1||R_2|\ldots|R_m|} = \frac{|R_{123\ldots m}|}{\prod_{i=1}^{m} |R_i|} \tag{3.1}$$

where $R_{123\ldots k}$ is the output relation of the star join.

The actual selectivity μ stems from:

$$\mu = \frac{|R_{123\ldots m}|}{\prod_{i=1}^{m}|R_i|} = \frac{\sum_{i_1=1}^{|R_1|}\sum_{i_2=1}^{|R_2|}\ldots\sum_{i_m=1}^{|R_m|}(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})}{\prod_{i=1}^{m}|R_i|} \tag{3.2}$$

where $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ is a join among tuples i_1th, i_2th, …, i_mth of the relations. The value of $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ is 1 if the tuples are joinable — the join attribute values are in common; the value is 0 otherwise.

Likewise an estimated selectivity $\hat{\mu}$ stems from:

$$\hat{\mu} \;=\; \frac{\sum_{i_1=1}^{|R'_1|}\sum_{i_2=1}^{|R'_2|}\ldots\sum_{i_m=1}^{|R'_m|}(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})}{\prod_{i=1}^{m}|R'_i|} \tag{3.3}$$

Using the proportion model, the population of the join $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m$ can be classified into two classes, namely: $\tilde{y}_i = 1$ if $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ is joinable and $\tilde{y}_i = 0$ if $(t_{i_1} \bowtie t_{i_2} \bowtie \cdots \bowtie t_{i_m})$ is not joinable. Thus the total number of tuples in the output relation $R_{123\ldots m}$ can be defined as:

$$|R_{123\ldots m}| = \sum_{i=1}^{\tilde{N}} \tilde{y}_i = \sum_{i=1}^{\prod_{j=1}^{m}|R_j|} \tilde{y}_i \tag{3.4}$$

where $\tilde{N} = \prod_{j=1}^{m}|R_j|$. According to the definition of a *population mean* (namely, the sum of each point $\tilde{y}_i$ in the population divided by the population size $\tilde{N}$), the mean of the population $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_m$ is defined by:

$$\frac{\sum_{i=1}^{\tilde{N}} \tilde{y}_i}{\tilde{N}} = \frac{\sum_{i=1}^{\prod_{j=1}^{m}|R_j|} \tilde{y}_i}{\prod_{j=1}^{m}|R_j|} = \frac{|R_{123\ldots m}|}{\prod_{j=1}^{m}|R_j|} = \mu \tag{3.5}$$

which is, in fact, the selectivity μ of the join $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$. That is, the population mean is equivalent to the star join selectivity μ.

Since $\tilde{y}_i$ is a 0 or 1 value, the following is valid:

$$\sum_{i=1}^{\tilde{N}} \tilde{y}_i^2 = |R_{123\ldots m}| = \mu\tilde{N} \tag{3.6}$$

The definition of a population variance $\tilde{S}^2$ of the mean or join selectivity μ over the entire population $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$ can be stated as follows:

$$\begin{aligned} \tilde{S}^2 &= \frac{\sum_{i=1}^{\tilde{N}}(\tilde{y}_i - \mu)^2}{\tilde{N}-1} \\ &= \frac{\sum_{i=1}^{\tilde{N}}\tilde{y}_i^2 - \tilde{N}\mu^2}{\tilde{N}-1} \end{aligned} \tag{3.7}$$

Substitute $\mu\tilde{N}$ in (3.6) to (3.7). This gives:

$$\begin{aligned} \tilde{S}^2 &= \frac{\mu\tilde{N} - \tilde{N}\mu^2}{\tilde{N}-1} \\ &= \frac{\tilde{N}}{\tilde{N}-1}\mu(1-\mu) \end{aligned} \tag{3.8}$$

3.3.1.2 How many tuples to sample

In this section, we will show the derivation for $\tilde{n}$ and β based on SRSWOR. Since the nature of SYSSMP is a sampling without replacement, although the sample size $\tilde{n}$ and sampling fraction β derived here are, in fact, for SYSWOR, they can also be used for SYSSMP.

With simple random sampling without replacement, the variance $V(\hat{\mu})$ of an estimated selectivity $\hat{\mu}$ of a sample $R'_1 \bowtie R'_2 \bowtie \ldots \bowtie R'_m$ with size $\tilde{n}$ is defined as:

$$V(\hat{\mu}) = E(\hat{\mu} - \mu)^2 = \frac{\tilde{N} - \tilde{n}}{\tilde{N}}\frac{\tilde{S}^2}{\tilde{n}}$$

where $\tilde{N} = \prod_{i=1}^{m}|R_i|$ and $\tilde{n} = \prod_{i=1}^{m}|R'_i|$. Using $\tilde{S}^2$ in (3.8), we get:

$$V(\hat{\mu}) = \frac{\mu(1-\mu)}{\tilde{n}}\left(\frac{\tilde{N}-\tilde{n}}{\tilde{N}-1}\right) \tag{3.9}$$

Let B be a bound on error, i.e., the difference allowed between the actual selectivity (mean) μ and an estimated selectivity $\hat{\mu}$. Let α be a probability that the actual error is larger than B. In estimating the actual selectivity μ, we wish:

$$Pr(|\hat{\mu} - \mu| \geq B) = \alpha$$

With simple random sampling, if $\hat{\mu}$ is an unbiased, normally distributed estimator of μ, then the error bound B is given by:

$$B = t\sqrt{V(\hat{\mu})} \tag{3.10}$$

where t is the abscissa of the normal curve that cuts off an area α at the tail. Substituting $V(\hat{\mu})$ in (3.9) to (3.10) and solving (3.10) for $\tilde{n}$, we obtain:

$$\tilde{n} = \frac{\frac{t^2(\mu)(1-\mu)}{B^2}}{1 + \frac{1}{\tilde{N}}(\frac{t^2(\mu)(1-\mu)}{B^2} - 1)} \tag{3.11}$$

If $\tilde{N}$ is large, then the denominator of equation (3.11) will be equal to 1 and the approximation of $\tilde{n}$ is:

$$\tilde{n}_0 = \frac{t^2(\mu)(1-\mu)}{B^2} \tag{3.12}$$

Otherwise, $\tilde{n}$ is defined as:

$$\tilde{n} = \frac{\tilde{n}_0}{1 + \frac{\tilde{n}_0 - 1}{\tilde{N}}} \tag{3.13}$$

As mentioned earlier, formula (3.13) requires the actual join selectivity μ and an estimated $\hat{\mu}$ can be used as a substitute for the actual μ and this estimated $\hat{\mu}$ can be obtained from a previous sampling.

Given that $abs()$ is the absolute value of the given parameter, if one is concerned about the relative error $\epsilon = \frac{abs(\mu-\hat{\mu})}{\mu} = \frac{B}{\mu}$ rather than the absolute error B, then B in equations (3.11) and (3.12) can be replaced by $\epsilon\mu$, where the ϵ given should be between 0 and 1.

The sample size $\tilde{n}$ in (3.13) and the sampling fraction β have the following relationship:

$$\boxed{\begin{aligned} \tilde{n} &= (\beta * |R_1|)\ (\beta * |R_2|)\ \cdots\ (\beta * |R_m|) \;=\; \beta^m \prod_{i=1}^{m} |R_i| \\ \beta^m &= \frac{\tilde{n}}{\prod_{i=1}^{m} |R_i|} \\ \beta &= (\frac{\tilde{n}}{\prod_{i=1}^{m} |R_i|})^{\frac{1}{m}} \qquad (3.14) \end{aligned}}$$

Thus, the number of tuples to be sampled from each relation R_i participating in a star join is $\beta * |R_i|$, where β is calculated by (3.14).

3.3.1.3 Oversampling problem with star joins

When a star join selectivity is very small, the calculated β in (3.14) can be too large, i.e., larger than a maximum sampling fraction limit β_{max} that one would wish to occur in a database system, where typically $0 < \beta \leq \beta_{max} \leq 1$. This problem is called the *oversampling* problem. Table 3.2 shows a proposal for a *sanity bound*, a bound to forbid the problem to occur.

We also make sure that a value of ψ (≥ 0) used will produce a reasonable value of $\tilde{n}$ which is to be substituted into (3.14) to yield another reasonable size β.

```
**if** $\beta > \lceil\beta_{max})\rceil$ **then**
    $B = \epsilon\psi$          [ recall that $B = \epsilon\mu$ ]
    $\tilde{n} = \frac{t^2\psi(1-\psi)}{B^2}$
**endif**
```

Table 3.2: A proposal for a sanity bound

3.3.1.4 How to obtain an initial estimated selectivity $\hat{\mu}$

We first describe how we initialise an estimated selectivity $\hat{\mu}$ of a given star join which can be substituted into the formula (3.13) in order to obtain a sample size $\tilde{n}$ and thus a sampling fraction β. Next we show that the total number of all possible star join selectivities that can occur in a database is tractable, i.e., not too many values, so we can possibly maintain them to be reused by a database system.

One way to initialise the initial value of an estimated selectivity $\hat{\mu}$ is that one can always have SS run through the star join. This could be a good choice in obtaining an accurate $\hat{\mu}$ since the method of SS has the "adaptive" stopping condition to terminate sampling whenever the sample size $\tilde{n}$ satisfies the error bound B given.

Using the formula (3.13) requires maintaining $\hat{\mu}$ in the database system. Here is our justification of why all possible estimated star join selectivities $\hat{\mu}$'s can be maintained to be reused by a database system.

Consider a 5-relation database. Suppose that all of the 5 relations can be joined in a star-join like manner on a common join domain. To return a join selectivity consulted by a query optimiser, Table 3.3 shows the total number of all possible selectivities that can occur in any star join queries on the database. C^5_m is the number of all combinations for an m-relation star join, where $m = 2, 3, 4, 5$.

Suppose we have a 10-relation database and all relations can be joined in the star-join like manner. The total number of all possible star join predicates and thus selectivities needed to be maintained by a database system is: $2^m - m - 1 = 2^{10} - 10 - 1 = 1013$ values. This total number of join selectivities is practical to be maintained and reused by a database system.

rels	comb
2	$C_2^5 = 10$
3	$C_3^5 = 10$
4	$C_4^5 = 5$
5	$C_5^5 = 1$
total	26

Table 3.3:

In contrast, for selection queries, the total number of selection selectivities grows too large to be maintained and reused by a database system. The following is the justification.

The formula for a sample size n is defined by:

$$n = \frac{t^2 \text{predicate selectivity}(1 - \text{predicate selectivity})}{B^2}$$

which can be used for selection queries. (Note that it is not hard to see that the derivation for this formula can be done similarly to the derivation for the formula (3.13).) In the light of selection queries, we cannot maintain all possible estimated selectivities (as substitutes to the predicate selectivity in the formula) since the total number of selectivities (predicates) grows exponentially. Let us give an example.

Consider a relation R with a set of attributes. A simple predicate on an attribute, say a1, of R is of the form: (R.a1, *relopt*, *const*). Let $d = 250$ be the number of distinct values of a1 and suppose that all these 250 values can appear as a constant *const* in a simple predicate on R. *relopt* can have 6 choices since a relational operator could be any of $<, >, \neq, =, \geq, \leq$. Thus, the total number of different simple predicates (and thus selectivities) would be 6∗250=1500. This total number is just for a single attribute a1, not yet for other single attributes of R, i.e., a2, a3, and so on. Moreover, we have not yet considered complex predicate (multiple attribute) queries whose predicates are specified upon more than a single attribute. As a result, the total number of possible predicates for selection queries grows too large to be maintained and reused by a database system.

For the selection queries, we propose two solutions in Section 3.6 to the problem with the formula above.

3.4 Theoretical foundation for systematic sampling

Our aim in this section is to show theoretically that systematic sampling can, in general, guarantee to yield efficient sample relations of each relation participating in a star join, thus producing a selectivity estimate of high quality for an actual join selectivity.

The selectivity of a distinct value is a ratio which is defined by the total number of tuples in a relation having the distinct value divided by the cardinality of the relation. SYSSMP is more efficient than SRS with/without replacement if and only if:

1. The variance of the estimated selectivity of a distinct value in the common join domain is lower. This variance indicates the quality of the estimated selectivity for one distinct value in the join attribute domain of the sample relation. Lower variance indicates more accurate estimates.

2. The total variance of estimated selectivities for all distinct values in the common join domain is lower. This total variance indicates the overall quality of a sample relation. This affects the accuracy of an estimated join selectivity, which is calculated by using estimated selectivities from all of the sample relations participating in the join.

The proof in Section 3.4.1 shows when the variance for a single distinct value will be lower for SYSSMP than SRS (point 1). Section 3.4.2 completes the proof by showing that the total variance over the entire join domain will be lower for SYSSMP than SWS (point 2).

3.4.1 Variance of estimated selectivity of a distinct value

In a star join with m participating relations, let d be the number of distinct values in a common join domain of the attributes $a_1, a_2, \ldots, a_m$ in the join.

The following definition for a star join selectivity is the main motivation behind why we do need an accurate estimated selectivity of a distinct value:

$$\mu = \sum_{i=1}^{d} \prod_{j=1}^{m} \text{selectivity of the } i\text{th distinct value on relation } R_j \tag{3.15}$$

In view of a relation, say R_j, the selectivities of each ith distinct value $i = 1, 2, \ldots, d$ in the relation contribute significantly to the calculation of the join selectivity μ of the star join.

Let us consider a relation R which participates in the star join. Let R' be a sample relation from R. Let $\overline{Y}_i$ be the selectivity of the ith distinct value $(i = 1, 2, \ldots d)$ from relation R and let $\widehat{\overline{Y}}_i$ be the selectivity of the same distinct value from sample relation R'. By definition, the *variance* $V(\widehat{\overline{Y}}_i)$

$$V(\widehat{\overline{Y_i}}) = E(\widehat{\overline{Y}}_i - \overline{Y}_i)^2 = \sum(\widehat{\overline{Y}}_i - \overline{Y}_i)^2 Pr(\widehat{\overline{Y}}_i)$$

where $Pr(\widehat{\overline{Y}}_i)$ is a probability that each sample can be selected. If a sample relation R' created by SYSSMP is more efficient than SRS, then the total variance of each $\widehat{\overline{Y}}_i$, which is called *total variance of estimated selectivities for all distinct values*, must produce a lower value, namely:

$$\sum_{i=1}^{d} V(\widehat{\overline{Y}}_i) = V(\widehat{\overline{Y}}_1) + V(\widehat{\overline{Y}}_2) \cdots + V(\widehat{\overline{Y}}_d)$$

where each $V(\widehat{\overline{Y}}_i)$ is the *variance of an estimated selectivity of the ith distinct value on a sample relation.* Towards the end of Section 3.4.2, we will show how to obtain such an efficient sample relation R'.

Given relation R with N tuples, to select n $(\leq N)$ tuples of the relation, the method of systematic sampling is to choose a tuple at random from the first $k = \lceil \frac{N}{n} \rceil$ tuples of R and every kth tuple thereafter. Suppose that relation R given in Figure 3.5 is sorted on its join attribute.

1 1 1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5

Figure 3.5: Relation R sorted on the join attribute

where $N = 25$. Let $n = 5$ so $k = \frac{N}{n} = 5$. Table 3.4(a) shows all possible systematic sample relations R''s which can be taken from R.

Consider Table 3.4(b). Let t_{ij} be the jth tuple of the ith systematic sample

systematic sample no.				
1	2	3	4	5
1	1	1	1	1
1	1	1	1	2
2	2	2	2	3
3	3	3	4	4
4	4	5	5	5

(a)

	1	2	...	i	...	k
	y_{11}	y_{21}		y_{i1}		y_{k1}
	y_{12}	y_{22}		y_{i2}		y_{k2}
	...	...		...		...
	y_{1n}	y_{2n}		y_{in}		y_{kn}
$\sum_{j=1}^{n} y_{ij}$	a_1	a_2		a_i		a_k
Means	$\bar{y}_1 = \frac{a_1}{n}$	$\bar{y}_2 = \frac{a_2}{n}$		$\bar{y}_i = \frac{a_i}{n}$		$\bar{y}_k = \frac{a_k}{n}$

(b)

Table 3.4: Systematic sample relations and notations

relation and $f(t_{ij}, x)$ a function of a [0,1] value, i.e.:

$$y_{ij} = f(t_{ij}, x) = \begin{cases} 1 & \text{if the } j\text{th tuple of the } i\text{th sample contains value } x \\ 0 & \text{if it doesn't} \end{cases}$$

For example, in view of the selectivity of the distinct value equal to x, if tuple j of the sample relation i contains x as its value, then y_{ij} would be 1, otherwise 0. Therefore, using systematic sample relation number 1 in Table 3.4(a) and using the [0,1] value representation, the selectivity when the distinct value is equal to 1 is $\bar{y}_1 = \frac{a_1}{n} = \frac{2}{5}$ (there are two rows which consist of value 1 in the sample relation number 1). Using the same distinct value (=1), the selectivities calculated using systematic sample relations 2, 3, 4 and 5 are equal to $\bar{y}_2 = \frac{a_2}{n} = \frac{2}{5}, \bar{y}_3 = \frac{a_3}{n} = \frac{2}{5}, \bar{y}_4 = \frac{a_4}{n} = \frac{2}{5}$ and $\bar{y}_5 = \frac{a_5}{n} = \frac{1}{5}$, respectively.

By definition, a mean $\overline{Y}$ of a population of size N can be defined as:

$$\text{population mean} = \overline{Y} = \frac{y_1 + y_2 + \ldots y_N}{N} = \frac{A}{N}$$

Using the proportion model (each point y_i in a population can have two values 0 or 1), the population mean defined above is a proportion. x is a distinct value in a common join attribute domain and we want to know how many of tuples of relation R satisfy x, namely, its selectivity. This is, in fact, finding a "proportion" on relation R which contains x in the tuples. Thus the proportion of R with value x is the selectivity $\overline{Y}$ of value x on R, namely, $\overline{Y} = \frac{A}{N}$, where A is the total number of tuples on R having x as a value.

The main theorem which is quoted from page 209 of reference [Cochran 1963] is applicable for a mean and hence a proportion. Since this theorem (and its two corollaries) only state on a selectivity (proportion) of a given distinct value of relation R, i.e., proportion of R with the given distinct value, for simplicity of notation in the theorem and its two corollaries, we will drop the subscript i of $V(\widehat{\overline{Y}}_i)$, the variance of the selectivity of the ith distinct value on a sample relation.

Let $V(\widehat{\overline{Y}})$, $V(\widehat{\overline{Y}}_{swr})$ and $V(\widehat{\overline{Y}}_{swor})$ be such a variance obtained via SYSSMP, SRSWR and SRSWOR, respectively, where $\widehat{\overline{Y}}, \widehat{\overline{Y}}_{swr}$ and $\widehat{\overline{Y}}_{swor}$ are the selectivities of the ith distinct value calculated from each individual R' yielded by SYSSMP, SRSWR, and SRSWOR respectively. Let the value of the ith distinct value equal x.

Before proceeding to the theorem, let us define one more symbol S^2, a population variance. Let S^2 be the variance of the selectivity of the distinct value x over the entire relation R. S^2 is thus defined as:

$$S^2 = \frac{\sum_{i=1}^{N}(y_i - \overline{Y})^2}{N-1} = \frac{\sum_{i=1}^{N} y_i^2 - N\overline{Y}^2}{N-1} \tag{3.16}$$

where each y_i corresponds to an element y_{rj} in Table 3.4(b) ($r = 1, 2, \ldots, k,\ j = 1, 2, \ldots, n$). It is straightforward to see that $\sum_{i=1}^{N} y_i^2 = A$ since each y_i in the table is a [0,1] value. Thus, the variance S^2 is:

$$S^2 = \frac{A - N\overline{Y}^2}{N-1} \tag{3.17}$$

Since $A = N\overline{Y}$, the substitution of this in (3.17) gives:

$$\begin{aligned} S^2 &= \frac{N\overline{Y} - N\overline{Y}^2}{N-1} \\ &= \frac{N}{N-1}\overline{Y}(1-\overline{Y}) \end{aligned} \tag{3.18}$$

The theorem can be stated as follows:

Theorem 3.4.1 The variance of the selectivity $\widehat{\overline{Y}}$ for distinct value x on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{N-1}{N}S^2 - \frac{k(n-1)}{N}S^2_{wsmp} \tag{3.19}$$

where

$$S^2_{wsmp} = \frac{1}{k(n-1)} \sum_{i=1}^{k} \sum_{j=1}^{n} (y_{ij} - \bar{y}_i)^2 \tag{3.20}$$

is the variance among tuples that lie within the same systematic sample relation (subscript $wsmp$ = within a systematic sample relation). Recall that y_{ij} as shown in Table 3.4(b) is a function $f(t_{ij}, x)$ of a [0,1] value of tuple j of systematic sample relation i. $\bar{y}_i = \frac{a_i}{n}$ is the selectivity of distinct value x calculated from the ith systematic sample relation. The proof is not given here since it was already shown in [Cochran 1963]. The first important result from Theorem 3.4.1 is:

Corollary 3.4.1 The selectivity for distinct value x obtained from a systematic sample relation is more accurate than the selectivity for the same distinct value from a simple random sample relation without replacement if and only if

$$S^2_{wsmp} > S^2 \tag{3.21}$$

The proof was also given in the same reference. The main result from this corollary is that SYSSMP is more efficient than SRSWOR if the variance within systematic sample relations is greater than the variance of the actual selectivity over the entire relation. To ensure this with a high possibility, a sort over the population in our case, on the join attribute of the entire relation, in ascending or descending order must be done [Murthy and Rao 1988; Scheaffer et al. 1990], which will then result in heterogeneous tuples within the same systematic sample relation.

Next is the second result which we have derived based also on Theorem 3.4.1.

Corollary 3.4.2 The selectivity for distinct value x obtained from a systematic sample relation is more accurate than the selectivity for the same distinct value from a simple random sample relation with replacement if and only if

$$S^2_{wsmp} > \frac{N-1}{N} S^2 \tag{3.22}$$

Proof. According to the theory of simple random sampling with replacement (see in general sampling books such as [Cochran 1963; Thomson 1992; Scheaffer et al. 1990; Murthy and Rao 1988]), it is well-known that the variance of $\widehat{\overline{Y}}_{swr}$, the selectivity obtained from a simple random sample relation of cardinality n with replacement, is given by:

$$V(\hat{\overline{Y}}_{swr}) = \frac{N-1}{N}\frac{S^2}{n} \tag{3.23}$$

From Theorem 3.4.1, $V(\hat{\overline{Y}}) < V(\hat{\overline{Y}}_{swr})$ if and only if

$$\begin{aligned}
\frac{N-1}{N}S^2 - \frac{k(n-1)}{N}S^2_{wsmp} &< \frac{N-1}{N}\frac{S^2}{n} \\
k(n-1)S^2_{wsmp} &> (N-1)S^2 - \frac{N-1}{n}S^2 \\
k(n-1)S^2_{wsmp} &> \frac{nN-n-N+1}{n}S^2 \\
k(n-1)S^2_{wsmp} &> \frac{N(n-1)-(n-1)}{n}S^2 \\
S^2_{wsmp} &> \frac{N-1}{nk}S^2 \qquad (3.24)
\end{aligned}$$

The substitution of $N = nk$ to the inequality (3.24) gives:

$$S^2_{wsmp} > \frac{N-1}{N}S^2$$

If N is large, then the result above would be the same as the the result in Corollary 3.4.1, namely:

$$S^2_{wsmp} > S^2$$

Hence, from the main two results in Corollaries 3.4.1 and 3.4.2, we can conclude that if the cardinality N of relation R is large, then systematic sampling is more efficient than simple random sampling with/without replacement if and only if the variance within systematic sample relations is larger than the variance of the actual selectivity of distinct value x over the entire relation. It is intuitively clear that if there is little variation within a systematic sample relation, relative to the variation in the entire relation, some (perhaps a lot of) tuples in the sample relation are repeating the same information, which implies that those repetitive tuples (together with the remaining tuples in the sample relation) will not be able to form a good representative sample relation.

The following is an example to show how to calculate variances of a selectivity of a distinct value, namely, $V(\widehat{\overline{Y}})$, $V(\widehat{\overline{Y}}_{swr})$ and $V(\widehat{\overline{Y}}_{swor})$. The example also serves to add an understanding in Section 3.4.2 of how we calculated a total variance of estimated selectivities for all distinct values.

Example 1: Using relation R on the join attribute as shown in Figure 3.5 and given a distinct value x of the join attribute equal to 1, calculate variances $V(\widehat{\overline{Y}})$, $V(\widehat{\overline{Y}}_{swr})$ and $V(\widehat{\overline{Y}}_{swor})$.

The selectivity $\overline{Y}$ for distinct value $x = 1$ on R is $\frac{A}{N}$ which is $\frac{9}{25}$. From Theorem 3.4.1, the variance $V(\widehat{\overline{Y}})$ on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{N-1}{N}S^2 - \frac{k(n-1)}{N}S^2_{wsmp}$$

where the variance S^2 is:

$$S^2 = \frac{N}{N-1}\overline{Y}(1-\overline{Y})$$

from equation (3.18). The variance S^2_{wsmp} within systematic sample relations in equation (3.20) is:

$$\begin{aligned} S^2_{wsmp} &= \frac{1}{k(n-1)}\sum_{i=1}^{k}\sum_{j=1}^{n}(y_{ij} - \bar{y}_i)^2 \\ &= \frac{1}{k(n-1)}\sum_{i=1}^{k}(\sum_{j=1}^{n} y_{ij}^2 - n\bar{y}_i^2) \qquad (3.25) \end{aligned}$$

Since y_{ij} is a [0,1] value, $\sum_{j=1}^{n} y_{ij}^2 = a_i$, the number of tuples in the ith systematic sample relation with the distinct value equal to 1. $\bar{y}_i = \frac{a_i}{n}$ is the selectivity for distinct value $x = 1$ calculated from the ith systematic sample relation. Substitute a_i to equation (3.25). This gives:

$$S^2_{wsmp} = \frac{1}{k(n-1)}\sum_{i=1}^{k}(a_i - n\bar{y}_i^2) \qquad (3.26)$$

Table 3.5(b) shows how to calculate the variance in (3.26) step-by-step.

Thus,

$$S^2_{wsmp} = \frac{1}{5(5-1)}(\frac{28}{5}) = 0.28$$

systematic sample no.					
i	1	2	3	4	5
	1	1	1	1	1
	1	1	1	1	2
	2	2	2	2	3
	3	3	3	4	4
	4	4	5	5	5
a_i	2	2	2	2	1
$\bar{y}_i$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$

(a) Sample selectivities

sample no.	$a_i - n\bar{y}_i^2$
$i = 1$	$2 - 5(\frac{2}{5})^2$
$i = 2$	$2 - 5(\frac{2}{5})^2$
$i = 3$	$2 - 5(\frac{2}{5})^2$
$i = 4$	$2 - 5(\frac{2}{5})^2$
$i = 5$	$1 - 5(\frac{1}{5})^2$
$\sum_{i=1}^{5}(a_i - n\bar{y}_i^2)$	$\frac{28}{5} = 5.6$

(b) S^2_{wsmp}, variance within sample relations

Table 3.5: Selectivities and a variance, when a distinct value = 1

$$S^2 = \frac{25}{25-1}(\frac{9}{25})(1-\frac{9}{25}) = 0.24.$$

The variance $V(\widehat{\overline{Y}})$ of the selectivity of $x = 1$ on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{25-1}{25}0.24 - \frac{5(5-1)}{25}0.28 = 0.0064.$$

The following two formulas for $V(\widehat{\overline{Y}}_{swor})$ and $V(\widehat{\overline{Y}}_{swr})$ can be found in general sampling books (see [Cochran 1963; Thomson 1992; Scheaffer et al. 1990; Murthy and Rao 1988], for examples). The variance $V(\widehat{\overline{Y}}_{swor})$ of the selectivity of $x = 1$ on a simple random sample relation without replacement is:

$$V(\widehat{\overline{Y}}_{swor}) = \frac{N-n}{N}\frac{S^2}{n} = \frac{25-5}{25} * \frac{0.24}{5} = 0.0384.$$

The variance $V(\widehat{\overline{Y}}_{swr})$ of the selectivity of $x = 1$ on a simple random sample relation with replacement is:

$$V(\widehat{\overline{Y}}_{swr}) = \frac{N-1}{N}\frac{S^2}{n} = \frac{25-1}{25} * \frac{0.24}{5} = 0.04608.$$

Since $S^2_{wsmp} > S^2$, $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swor})$ (following Corollary 3.4.1), so is $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swr})$ (following Corollary 3.4.2). As one might have expected, $V(\widehat{\overline{Y}}_{swor}) < V(\widehat{\overline{Y}}_{swr})$.

For other distinct values of the join attribute $x = 2, 3, 4$ and 5, follow the same procedure as shown above and Table 3.6 shows the results for all the distinct values.

dist val x	$\overline{Y}$	S^2	S^2_{wsmp}	$V(\widehat{\overline{Y}})$	$V(\widehat{\overline{Y}}_{swor})$	$V(\widehat{\overline{Y}}_{swr})$
1	0.36	0.24	0.28	0.0064	0.038	0.046
2	0.20	0.17	0.20	2.7756e-17	0.027	0.032
3	0.16	0.14	0.16	0.0064	0.022	0.027
4	0.16	0.14	0.16	0.0064	0.022	0.027
5	0.12	0.11	0.12	0.0096	0.018	0.021

Table 3.6: Means and variances on different distinct values

3.4.2 Total variance of estimated selectivities for all distinct values

In Section 3.4.1, we have shown the variance solely for a single distinct value in a common join attribute domain obtained via SYSSMP, SRSWOR and SRSWR. In this section, we will show a total variance of estimated selectivities for all distinct values in the common join domain under the three sampling methods.

Let S_i^2 be the variance of the actual selectivity of the ith distinct value over the entire relation R, where $i = 1, 2, \ldots, d$. Let $S^2_{wsmp_i}$ be the variance within systematic sample relations of the ith distinct value. In this section, there are four main results which are also based on Theorem 3.4.1. Generally, they are to claim that the systematic sampling is more efficient (in producing a sample relation) than the simple random sampling with/without replacement if and only if the total/average of $S^2_{wsmp_i}$'s must be larger than the total/average of S_i^2's for all $i = 1, 2, \ldots, d$. Let T be the total variance of S_i^2's, i.e.,

$$T = S_1^2 + S_2^2 + \ldots + S_d^2 = \sum_{i=1}^{d} S_i^2$$

and let T_{wsmp} be the total variance within systematic sample relations, defined as:

$$T_{wsmp} = S^2_{wsmp_1} + S^2_{wsmp_2} + \ldots + S^2_{wsmp_d} = \sum_{i=1}^{d} S^2_{wsmp_i}$$

Denote the average of T by $\overline{T} = \frac{T}{d}$ and the average of T_{wsmp} by $\overline{T}_{wsmp} = \frac{T_{wsmp}}{d}$. Recall that $\widehat{\overline{Y}}_i, \widehat{\overline{Y}}_{swr_i}$ and $\widehat{\overline{Y}}_{swor_i}$ are the selectivities of the ith distinct value calculated from a sample relation obtained via SYSSMP, SRSWR, and SRSWOR respectively.

Corollary 3.4.3 SYSSMP yields a more efficient sample relation than SRSWOR if and only if

$$T_{wsmp} > T \tag{3.27}$$

Proof. If SYSSMP is more efficient than SRSWOR in yielding a sample relation, it is obvious to claim:

$$\begin{aligned} V(\widehat{\overline{Y}}_1) + V(\widehat{\overline{Y}}_2) + \ldots + V(\widehat{\overline{Y}}_d) &< V(\widehat{\overline{Y}}_{swor_1}) + V(\widehat{\overline{Y}}_{swor_2}) + \ldots + V(\widehat{\overline{Y}}_{swor_d}) \\ \sum_{i=1}^{d} V(\widehat{\overline{Y}}_i) &< \sum_{i=1}^{d} V(\widehat{\overline{Y}}_{swor_i}) \end{aligned} \tag{3.28}$$

According to the theory of simple random sampling without replacement, it is well-known that the variance of $\widehat{\overline{Y}}_{swor_i}$, the selectivity obtained from a simple random sample relation of cardinality n without replacement, is given by:

$$V(\widehat{\overline{Y}}_{swor_i}) = \frac{N-n}{N}\frac{S_i^2}{n} \tag{3.29}$$

The substitutions of $V(\widehat{\overline{Y}}_i)$ in equation (3.19) from Theorem 3.4.1 and of $V(\widehat{\overline{Y}}_{swor_i})$ in (3.29) to (3.28) give:

$$\begin{aligned} \sum_{i=1}^{d}(\frac{N-1}{N}S_i^2 - \frac{k(n-1)}{N}S_{wsmp_i}^2) &< \sum_{i=1}^{d}\frac{N-n}{N}\frac{S_i^2}{n} \\ \frac{N-1}{N}\sum_{i=1}^{d}S_i^2 - \frac{k(n-1)}{N}\sum_{i=1}^{d}S_{wsmp_i}^2 &< \frac{N-n}{Nn}\sum_{i=1}^{d}S_i^2 \\ k(n-1)\sum_{i=1}^{d}S_{wsmp_i}^2 &> ((N-1) - \frac{N-n}{n})\sum_{i=1}^{d}S_i^2 \\ k(n-1)\sum_{i=1}^{d}S_{wsmp_i}^2 &> \frac{N(n-1)}{n}\sum_{i=1}^{d}S_i^2 \end{aligned} \tag{3.30}$$

$\frac{N}{n} = k$ so substitute k to the right hand side of (3.30). This gives:

$$\begin{aligned} k(n-1)\sum_{i=1}^{d}S_{wsmp_i}^2 &> k(n-1)\sum_{i=1}^{d}S_i^2 \\ T_{wsmp} &> T \end{aligned}$$

The second result basically relies on Corollary 3.4.3.

Corollary 3.4.4 SYSSMP yields a more efficient sample relation than SRSWOR if and only if

$$\overline{T}_{wsmp} > \overline{T} \tag{3.31}$$

Proof. From Corollary 3.4.3, we have

$$T_{wsmp} > T$$

It is valid to divide both sides of $T_{wsmp} > T$ by d if $d > 0$, namely:

$$\frac{T_{wsmp}}{d} > \frac{T}{d}$$

which is equal to:

$$\overline{T}_{wsmp} > \overline{T}$$

Again, to ascertain $T_{wsmp} > T$ and $\overline{T}_{wsmp} > \overline{T}$, relation R must be sorted in ascending or descending order [Murthy and Rao 1988; Scheaffer et al. 1990].

Corollary 3.4.5 SYSSMP yields a more efficient sample relation than SRSWR if and only if

$$T_{wsmp} > \frac{N-1}{N}T \tag{3.32}$$

Proof. If SYSSMP is more efficient than SRSWR in yielding a sample relation, it is obvious to claim:

$$\begin{aligned} V(\widehat{\overline{Y}}_1) + V(\widehat{\overline{Y}}_2) + \ldots + V(\widehat{\overline{Y}}_d) &< V(\widehat{\overline{Y}}_{swr_1}) + V(\widehat{\overline{Y}}_{swr_2}) + \ldots + V(\widehat{\overline{Y}}_{swr_d}) \\ \sum_{i=1}^{d} V(\widehat{\overline{Y}}_i) &< \sum_{i=1}^{d} V(\widehat{\overline{Y}}_{swr_i}) \end{aligned} \tag{3.33}$$

The substitutions of $V(\widehat{\overline{Y}}_i)$ in equation (3.19) from Theorem 3.4.1 and of $V(\widehat{\overline{Y}}_{swr_i})$ in equation (3.23) to (3.33) give:

$$\begin{aligned} \sum_{i=1}^{d}\left(\frac{N-1}{N}S_i^2 - \frac{k(n-1)}{N}S_{wsmp_i}^2\right) &< \sum_{i=1}^{d}\frac{N-1}{N}\frac{S_i^2}{n} \\ \frac{N-1}{N}\sum_{i=1}^{d}S_i^2 - \frac{k(n-1)}{N}\sum_{i=1}^{d}S_{wsmp_i}^2 &< \frac{N-1}{Nn}\sum_{i=1}^{d}S_i^2 \end{aligned}$$

$$\begin{aligned}
k(n-1)\sum_{i=1}^{d} S^2_{wsmp_i} &> ((N-1) - \frac{N-1}{n})\sum_{i=1}^{d} S_i^2 \\
k(n-1)\sum_{i=1}^{d} S^2_{wsmp_i} &> \frac{nN-n-N+1}{n}\sum_{i=1}^{d} S_i^2 \\
k(n-1)\sum_{i=1}^{d} S^2_{wsmp_i} &> \frac{N(n-1)-(n-1)}{n}\sum_{i=1}^{d} S_i^2 \\
\sum_{i=1}^{d} S^2_{wsmp_i} &> \frac{N-1}{nk}\sum_{i=1}^{d} S_i^2 \qquad (3.34)
\end{aligned}$$

Since $N = nk$, substitute this to (3.34), giving:

$$\begin{aligned}
\sum_{i=1}^{d} S^2_{wsmp_i} &> \frac{N-1}{N}\sum_{i=1}^{d} S_i^2 \\
T_{wsmp} &> \frac{N-1}{N}T
\end{aligned}$$

If N is large, then

$$\sum_{i=1}^{d} S^2_{wsmp_i} > \sum_{i=1}^{d} S_i^2$$

$$T_{wsmp} > T$$

which is the same as the result in Corollary 3.4.3.

The last result relies on Corollary 3.4.5.

Corollary 3.4.6 SYSSMP yields a more efficient sample relation than SRSWR if and only if

$$\overline{T}_{wsmp} > \frac{N-1}{N}\overline{T} \qquad (3.35)$$

Proof. From Corollary 3.4.5, we have

$$T_{wsmp} > \frac{N-1}{N}T$$

Divide both sides of $T_{wsmp} > \frac{N-1}{N}T$ by d if $d > 0$, namely:

$$\frac{T_{wsmp}}{d} > \frac{N-1}{N}\frac{T}{d}$$

which is equal to:

$$\overline{T}_{wsmp} > \frac{N-1}{N}\overline{T}$$

if N is large, then

$$\overline{T}_{wsmp} > \overline{T}$$

Therefore, if N is large, we can then conclude that a sample relation produced by the systematic sampling is more efficient than that by the simple random sampling with/without replacement if the total variance T_{wsmp} within systematic sample relations is larger than the total population variance T.

The following is an example demonstrating how to calculate T_{wsmp} and T. Recall that $T_{wsmp} = \sum_{i=1}^{d} S^2_{wsmp_i}$ and $T = \sum_{i=1}^{d} S_i^2$. The purpose of the demonstration is merely to assist in understanding how we obtained the results in Section 3.5.2. The results in Table 3.10 of Section 3.5.2 are calculated and tabulated in the same fashion as the example below.

Example 2: Using relation R on the join attribute as shown in Figure 3.5 and reproducing (from the original table 3.6 for ease in reference here) Table 3.7(a) which consists of $S^2_{wsmp_i}, S_i^2, V(\widehat{\overline{Y}}_i), V(\widehat{\overline{Y}}_{swor_i})$ and $V(\widehat{\overline{Y}}_{swr_i})$ for each ith distinct value $i = 1, 2, \ldots, 5$, Table 3.7 illustrates how to calculate total variances and sample relation variances.

i-th dist val	S_i^2	$S^2_{wsmp_i}$	$V(\widehat{\overline{Y}}_i)$	$V(\widehat{\overline{Y}}_{swor_i})$	$V(\widehat{\overline{Y}}_{swr_i})$
1	0.24	0.28	0.0064	0.038	0.046
2	0.17	0.20	2.7756e-17	0.027	0.032
3	0.14	0.16	0.0064	0.022	0.027
4	0.14	0.16	0.0064	0.022	0.027
5	0.11	0.12	0.0096	0.018	0.021

(a) Variances of each distinct value

$T = \sum_{i=1}^{5} S_i^2$	$\frac{N-1}{N}T$	$T_{wsmp} = \sum_{i=1}^{5} S^2_{wsmp_i}$	$\sum_{i=1}^{5} V(\widehat{\overline{Y}}_i)$	$\sum_{i=1}^{5} V(\widehat{\overline{Y}}_{swor_i})$	$\sum_{i=1}^{5} V(\widehat{\overline{Y}}_{swr_i})$
0.797	0.765	0.920	0.029	0.127	0.153

(b) Total variances and sample relation variances

Table 3.7: Total variance calculation

The results in Table 3.7(b) correspond to Corollary 3.4.3 ($T_{wsmp} > T$) and and to Corollary 3.4.5 ($T_{wsmp} > \frac{N-1}{N}T$). We can then conclude that SYSSMP for relation R sorted on the join attribute produces a sample relation of higher quality than SRSWOR and SRSWR.

3.5 Experimental results for joins

A method is first described in Section 3.5.1 for generating synthetic data. The same method is used to generate data in subsequent Sections 3.5.2 and 3.5.3.

Next we conduct the first 20 experiments in Section 3.5.2 in order to substantiate the theoretical claim we have developed in Sections 3.4.1 and 3.4.2. The aim of the experiments is to demonstrate that when relations are sorted on their join attributes, the sample relations yielded by SYSSMP are, in general, of higher quality than those by SRSWOR and SRSWR.

The theoretical and experimental results above are just to claim that a better quality of sample relations can be obtained by SYSSMP. However, the claim that sample relations with higher quality can indeed assist in computation for more accurate query result size estimates for star joins still remains to be justified. We conduct two more sets of experiments in Section 3.5.3 so as to demonstrate that we can indeed obtain more accurate query size estimates for join queries.

3.5.1 Experimental setup

Table 3.8 shows all the parameters and their meanings for different kinds of data distributions used in all experiments in this thesis.

Notation	Meaning
norm($mean, \delta$)	normal distribution with mean $mean$ and standard deviation δ
unf($low, high$)	uniform distribution with
	low the lowest random value to be generated
	$high$ the highest random value to be generated
fdist(f_1, f_2)	F distribution with
	degrees of freedom for numerator f_1
	degrees of freedom for denominator f_2
zipf($NumDist$,z)	$NumDist$ the expected number of distinct values
	z, the values between 0 to 1
semizipf($NumDist$,$z = 0.5$)	$NumDist$ the expected number of distinct values
	this is a special case of the zipf distribution where $z = 0.5$
exp(av)	exponential distribution with mean av

Table 3.8: Parameters for each distribution

The data generated in Section 3.5.2 and Section 3.5.3 for join attributes attempt to follow work in the literature [Lipton et al. 1990; Haas and Swami 1992]; that is, in a common join attribute domain, use different kinds of the data distributions as shown in Table 3.8 to generate values in the domain. Randomly generated are the parameters, e.g., $low, high, mean, NumDist$ and so on for the distributions.

All join attribute values, except values generated by the zipf [Zipf 1949] and semizipf distributions, are first generated at random in accordance with a given distribution, i.e. either norm(), unf() or exp(). Then those generated values are shifted using the following scheme:

$$\begin{aligned} ScaleVal &= \frac{val - minval}{maxval - minval} \\ x &= LowVal + \lceil ScaleVal * NumDist \rceil \end{aligned}$$

where val, $minval$ and $maxval$ are a value, the maximum and minimum values, respectively generated from the given distribution. x is a join attribute value which appears in the relation being created. $ScaleVal$ is a value scaled to the range between 0 and 1. The $LowVal$ is the lowest possible value on a common join attribute domain, the $NumDist$ is the expected number of distinct values on the common domain. Note that the value of $NumDist$ given for relation creation is higher[2] than or equal to the value of d, which is the actual number of distinct values obtained after the relation has been created. The reason for the shifting is to ensure that join attribute values of two or more relations in any star join will then stay on the same common domain.

For the zipf and semizipf distributions, join attribute values are generated by:

$$x = i + (LowVal - 1)$$

where $i = 1, 2, \ldots, NumDist$ and each x value generated has a frequency computed by: $N * \frac{c}{i^z}$ where $c = \frac{1}{\sum_{j=1}^{NumDist} \frac{1}{j^z}}$ and N is the cardinality of the relation of interest.

3.5.2 Quality of sample relations yielded by SYSSMP, SRSWOR and SRSWR

Table 3.9 shows all relations used for the first set of experiments with $N = 10,000$ tuples each. The number of distinct values of join attributes of the relations ranges approximately from 10, 20, 30 and 40. Four experiments (namely R1, R2, R3 and

[2]This is dependent upon the distribution of $ScaleVal$'s which may cause the relation being created to/not to have the same number of distinct values as the value supplied.

R4) are conducted for the exponential data distribution, another four experiments for the zipf data distribution, ... and so on (see the table).

One may argue that since the range d of the number of distinct values used (namely, between 10-40) is so small, the results obtained here can be biased in favor of the systematic sampling proposed. Here are two reasons to justify the experiments here.

- First, the experiments developed here are to provide an observation that when data are sorted, it is more likely that SYSSMP will produce higher quality sample relations. But such an observation is still not entirely conclusive, i.e., there is still also room for SRS to be better than SYSSMP and this can be seen from the results obtained for selections in Table 3.24 of Section 3.8.2. (However, from the table, we still find that when data are sorted, most of the times we will be able to obtain higher quality sample relations. The symbol in the table $\%(1 > 2)$ means the percentage that SYSSMP is superior to SRS which is significantly better in most of the cases.)

  The more solid and extensive set of experiments whose number of distinct values is upto 900 is shown in Section 3.5.3 (see also Table 3.11 in the section for the number of distinct values used.). It is this latter set of experiments, not the former, to prove whether SYSSMP is really good or not in producing better join selectivities.

- Second, despite the small range d (10-40) used, this range is still practical enough to occur in real-world databases. According to [Turney and Jankulak 1993], in 64 real-world attribute domains whose cardinalities range from 15 to 48842 tuples, the range d of the number of distinct values is in between 2-100, which is quite close to the one used here.

Using the 10k relations, we conducted 20 experiments using 20 different relation configurations as shown in Table 3.9. A sample size n used in all the experiments in this section is $n = 10\%$ of the original sizes of the relations, which is equal to 1000 tuples. Thus the step size k would be equal to $\frac{N}{n} = \frac{10000}{1000} = 10$ in all the experiments. k also represents the total number of all possible unique systematic sample relations which can be obtained from an original relation.

reln.	dist mode	d	dist mode	d
R1	exp(87.346)	10	zipf(10,0.156)	10
R2	exp(123.976)	20	zipf(20,0.857)	20
R3	exp(145.866)	30	zipf(30,0.699)	30
R4	exp(104.164)	43	zipf(40,0.448)	40

(a) Relation configurations

reln.	dist mode	d	dist mode	d	dist mode	d
R1	unf(3847.290,3856.290)	10	semizipf(10,0.5)	10	norm(2393.555,148.650)	10
R2	unf(2604.370,2623.370)	20	semizipf(20,0.5)	20	norm(3965.942,141.974)	20
R3	unf(2527.100,2556.100)	30	semizipf(30,0.5)	30	norm(3533.447,207.898)	30
R4	unf(3765.259,3804.259)	40	semizipf(40,0.5)	40	norm(3318.726,188.213)	41

(b) Relation configurations

Table 3.9: 5 data distributions of Exponential, Zipf, Uniform, Semizipf and Normal

Table 3.10 shows all results for the relations shown in Table 3.9. The results are calculated and can be interpreted in the same fashion as the results in example 2 in Section 3.4.2. That is, the total variance of estimated selectivities for all distinct values on a sample relation yielded by SYSSMP is generally lower than the total variance by SRSWOR and SRSWR. Hence, the quality of sample relations yielded by SYSSMP would be higher than by SRSWOR or SRSWR. Alternatively, each sample relation created by SYSSMP would well represent the underlying frequency distribution of the join attribute in the original relation.

3.5.3 Query size estimation

Now that the quality of sample relations generated by SYSSMP is higher than that by SRSWOR and SRSWR, in this section we will claim that sample relations with higher quality can indeed assist in computation for more accurate query result size estimates for star joins.

We start with a demonstration in Section 3.5.3.1 that given an index per join attribute (the extreme case of SYSSMP) participating in a star join, the quality of query result size estimates obtained via SYSSMP is far superior to the SS procedure. Recall that an index on an attribute is implemented via either a physical index or a sorted attribute on the relation.

In Section 3.5.3.2 we will show that when there are only some (not all) join attributes indexed, the quality of query result size estimates, although degraded to some certain extent, via the hybrid sampling scheme between SYSSMP and SS is

exp.	$T = \sum S_i^2$	$\frac{N-1}{N}T$	$T_{wsmp} = \sum S^2_{wsmp_i}$	$\sum V(\widehat{\overline{Y}}_i)$	$\sum V(\widehat{\overline{Y}}_{swor_i})$	$\sum V(\widehat{\overline{Y}}_{swr_i})$
exp,R1	0.59996930	0.59990930	0.60050851	0.00000130	0.00053997	0.00059991
exp,R2	0.78517142	0.78509290	0.78587548	0.00000330	0.00070665	0.00078509
exp,R3	0.86270787	0.86262160	0.86347968	0.00000540	0.00077644	0.00086262
exp,R4	0.90259352	0.90250326	0.90339920	0.00000746	0.00081233	0.00090250
zipf,R1	0.89877102	0.89868114	0.89957918	0.00000154	0.00080889	0.00089868
zipf,R2	0.90089257	0.90080248	0.90170070	0.00000348	0.00081080	0.00090080
zipf,R3	0.94172603	0.94163186	0.94256957	0.00000486	0.00084755	0.00094163
zipf,R4	0.96837578	0.96827894	0.96924204	0.00000614	0.00087154	0.00096828
unf,R1	0.89351981	0.89343046	0.89432292	0.00000186	0.00080417	0.00089343
unf,R2	0.94858458	0.94848972	0.94943604	0.00000312	0.00085373	0.00094849
unf,R3	0.96575104	0.96565446	0.96661562	0.00000546	0.00086918	0.00096565
unf,R4	0.97458626	0.97448880	0.97545766	0.00000660	0.00087713	0.00097449
semi,R1	0.88378740	0.88369902	0.88458198	0.00000162	0.00079541	0.00088370
semi,R2	0.93757084	0.93747708	0.93841261	0.00000288	0.00084381	0.00093748
semi,R3	0.95636568	0.95627004	0.95722262	0.00000464	0.00086073	0.00095627
semi,R4	0.96615440	0.96605778	0.96701782	0.00000698	0.00086954	0.00096606
norm,R1	0.76024058	0.76016456	0.76092392	0.00000156	0.00068422	0.00076016
norm,R2	0.89588161	0.89579202	0.89668509	0.00000362	0.00080629	0.00089579
norm,R3	0.93090947	0.93081638	0.93174334	0.00000478	0.00083782	0.00093082
norm,R4	0.95149797	0.95140282	0.95234875	0.00000642	0.00085635	0.00095140

Table 3.10: Total variances and sample relation variances

still superior to the pure SS procedure.

3.5.3.1 Index per join attribute (extreme case)

We conducted 24 experiments to demonstrate the performance between SS versus SYSSMP. We performed each 6 experiments for star joins among 2, 3, 4, and 5 relations, respectively. For all the star joins with $m = 2, 3, 4, 5$ relations, a sampling fraction β is set to 10% for each relation participating in the star joins.

If we allow for (1) a bound on error B, (2) a probability α that the error is not within the bound given and (3) have a large population size N $(= \prod_{j=1}^{m} |R_j|)$ (which is the case in our experiments), then we can calculate a sample size $\tilde{n}$ $(= \prod_{i=1}^{m} |R_i'|)$ for both SS and SYSSMP by using equation (3.12) in page 110 (see the derivation of $\tilde{n}$ for SS, namely, SRSWR in Appendix A in page 165). In the equation, an estimated join selectivity $\hat{\mu}$ can be used as a substitute for the actual join selectivity μ and can be obtained from a previous sampling. In equation (3.14), if a sample size $\tilde{n}$ is the same for the two sampling methods SS and SYSSMP (which is true, assuming that both use the same previous estimated selectivity), then the sampling fraction β of tuples to be sampled from each relation would also be the same.

Here we are trying to claim that with the same amount of sampling β, SYSSMP would be superior to SS in producing better join selectivities (and thus result sizes). To fulfil such a claim, we picked $\beta = 10\%$ sampling for all the experiments here. $\beta = 10\%$ picked is thus justified under the same fair basis of comparison between the two sampling methods. By knowing that the claim is trustworthy or reliable, the sample size formula $\tilde{n}$ above together with SYSSMP on sorted data should be used on-line, instead of SS, to approximate join selectivities.

Table 3.11(a) shows 6 configurations for 2 relations in 2-relation star joins, Table 3.11(b) shows another 6 configurations for 3 relations in 3-relation star joins, $\ldots$, and Table 3.11(d) shows the last 6 configurations for 5 relations in 5-relation star joins. All the relations are of 10,000 tuple cardinality. Although these 10,000 tuple relations that we used throughout all experiments here (Sections 3.5.3.1 and 3.5.3.2) are of a small size, as compared to real world relations, they can produce join result sizes upto 6.028e+08 tuples, which is extremely large.

Before proceeding to experimental results, let us define three error measures which we have used to compare the results both for joins and selections. The three measures are shown in Figure 3.6. Fundamentally all the three error measures aim to gauge an average error between actual result sizes and their estimates.

All the graphs in Figures 3.7, 3.8, 3.9 and 3.10 show 30 runs (see the X-axis) of SS against SYSSMP. The Y-axis represents the join result size. Let us clarify one of them, e.g., graph in Figure 3.7(a). This graph is for a star join with 2 relations. For the ith run ($i = 1, 2, \ldots, 30$) of either SS or SYSSMP, two sample relations are created and joined together to yield an estimated cardinality $\hat{\mu}_i \tilde{N}$ as plotted in the graph. There are 3 lines in the graph; the solid line is for the join result size estimates for SS, the dashed line for the join result size estimates for SYSSMP and the straight dotted line for the actual join result size. The aim of the graph is "the closer the graph lies in respect to the graph of the actual value, the better the estimation method".

One can clearly see from all those graphs that generally SYSSMP has a significantly lower fluctuation of its estimates against the actual join result size than SS. In other words, many peaks (or spikes) have taken place in the graphs but a lot more peaks with high error belong to SS, rather than to SYSSMP.

	R1		R2	
exp.	distrib	*d*	distrib	*d*
SJ1	semizipf(400,0.5)	400	unf(2968,3019)	52
SJ2	zipf(500,0.237)	500	norm(3893.188,196.320)	384
SJ3	semizipf(600,0.5)	600	zipf(600,0.317)	600
SJ4	exp(106.714)	376	norm(2966.064,217.159)	548
SJ5	exp(109.277)	378	semizipf(800,0.5)	800
SJ6	norm(3227.783,257.053)	664	unf(3701.172,3798.470)	76

(a) 2 relations

	R1		R2		R3	
exp.	distrib	*d*	distrib	*d*	distrib	*d*
SJ1	unf(3827.271,3888.726)	62	zipf(400,0.048)	400	exp(144.960)	235
SJ2	zipf(500,0.734)	500	norm(3726.318,145.969)	376	exp(98.996)	320
SJ3	unf(3928.223,3990.802)	63	norm(3401.245,290.111)	468	exp(94.514)	383
SJ4	zipf(700,0.315)	700	unf(3585.327,3700.789)	116	norm(3092.773,268.599)	557
SJ5	zipf(800, 0.945)	800	semizipf(800,0.5)	800	norm(3097.412,171.849)	559
SJ6	semizipf(900, 0.5)	900	zipf(900,0.194)	900	exp(86.307)	454

(b) 3 relations

	R1		R2		R3		R4	
exp.	distrib	*d*	distrib	*d*	distrib	*d*	distrib	*d*
SJ1	zipf(500,0.775)	500	exp(99.371)	282	norm(3206.055,276.952)	407	norm(2008.545,137.766)	395
SJ2	norm(1678.223,110.155)	433	unf(2802.612,2857.233)	56	norm(2071.411,236.481)	426	zipf(600,0.682)	600
SJ3	semizipf(700,0.5)	700	unf(3015.991,3116.359)	101	norm(2284.180,208.998)	525	unf(3750.000,3923.284)	174
SJ4	semizipf(800,0.5)	800	zipf(800,0.807)	800	norm(3650.391,3724.013)	579	exp(142.209)	437
SJ5	semizipf(900,0.5)	900	exp(86.490)	464	semizipf(900,0.5)	900	unf(3366.577,3419.262)	54
SJ6	zipf(400,0.781)	400	unf(3913.940,3970.148)	58	semizipf(400,0.5)	400	unf(3842.529,3894.625)	53

(c) 4 relations

	R1		R2		R3		R4		R5	
exp.	distrib	*d*	distrib	*d*	distrib	*d*	distrib	*d*	distrib	*d*
SJ1	norm(3735.107,160.175)	403	exp(108.229)	313	unf(3501.831,3613.560)	113	norm(3826.293,244.107)	383	unf(2838.867,3036.646)	199
SJ2	exp(128.801)	400	exp(138.199)	343	unf(2904.297,2963.424)	60	norm(3613.647,223.311	450	norm(1719.971,329.877)	476
SJ3	norm(1684.448,286.298)	489	exp(136.253)	363	unf(3510.742,3653.385)	144	zipf(600,0.892)	600	unf(3774.170,3950.456)	177
SJ4	zipf(700,0.440)	700	exp(83.258)	413	norm(2743.530,226.974)	555	unf(3248.291,3328.311)	81	semizipf(700,0.5)	700
SJ5	zipf(800,0.654)	800	zipf(800,0.143)	800	norm(3112.183,189.890)	619	unf(3075.562,3217.562)	143	unf(2572.266,2744.348)	173
SJ6	exp(134.624)	465	exp(132.294)	497	norm(3117.065,118.102)	663	zipf(900,0.053)	900	zipf(900,0.858)	900

(d) 5 relations

Table 3.11: Relation configurations for star joins

Root Mean Square Error is defined as follows:

$$\sqrt{\sum_{i=1}^{\mathcal{Z}} \frac{(\hat{\mu}_i \tilde{N} - \mu \tilde{N})^2}{\mathcal{Z}}}$$

where $\mathcal{Z}$ is the number of samplings (we used 30 in all experiments), μ an actual join selectivity, $\hat{\mu}_i$ an estimated join selectivity which results from the ith sampling, $\tilde{N} = \prod_{j=1}^{m} |R_j|$, $\hat{\mu}_i \tilde{N}$ a result size estimate of a given star join in the ith sampling and $\mu \tilde{N}$ an actual result size of the star join.

Mean Residual Error is defined as follows:

$$\frac{\sum_{i=1}^{\mathcal{Z}} abs(\hat{\mu}_i \tilde{N} - \mu \tilde{N})}{\mathcal{Z}}$$

Mean Relative Error is defined as follows:

$$\frac{\sum_{i=1}^{\mathcal{Z}} 100 * \frac{abs(\hat{\mu}_i \tilde{N} - \mu \tilde{N})}{\mu \tilde{N}}}{\mathcal{Z}}$$

For selection queries, we also used the three error measures above where μ is substituted by an actual selection selectivity $\overline{Y}$, $\hat{\mu}$ by an estimated selection selectivity $\widehat{\overline{Y}}$ and $\mathcal{Z}$ by a total number of queries used $|\mathcal{Q}|$, $\tilde{N}$ by a cardinality N of the relation of interest.

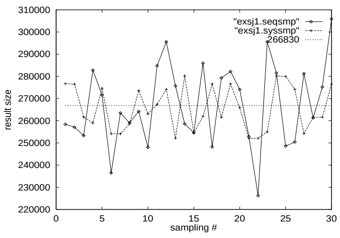
The reason we chose more than one measure – that is, one can use only one of the 3 measures to do the evaluation – is that if one is suspicious of one error measure used, he/she can look at the other two. At least two of the 3 should consistently give a good evaluation to a method in the same direction if the method is to be good.

Figure 3.6: Three error measures

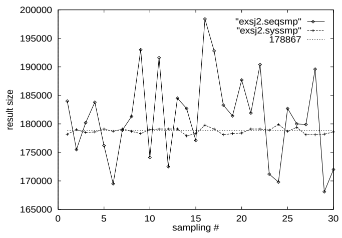
Apart from the graphs, in each experiment (24 experiments), we have also summarised three kinds of error in estimation, i.e., root mean square, mean residual and mean relative errors as shown in Table 3.12 for the 2-relation star joins, Table 3.13 for the 3-relation star joins, Table 3.14 for the 4-relation star joins and Table 3.15 for the 5-relation star joins. Note that when considering the root mean square and mean residual errors in the tables, to compare the errors against their actual values, see the corresponding actual values labelled in the right corner of the graphs.

3.5.3.2 Some indices among join attributes

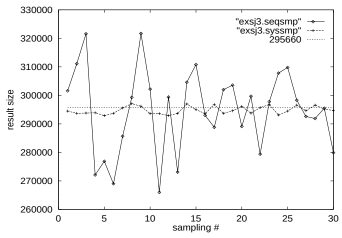
In this section, we will demonstrate that when some (not all) join attributes participating in a star join have indices (via either a physical index or a sorted attribute) and the rest do not, the quality of join result size estimates obtained via the hybrid sampling scheme between SYSSMP and SS is still superior to the pure SS procedure.



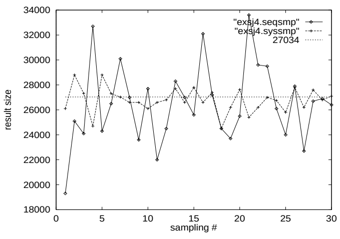
(a) SJ1



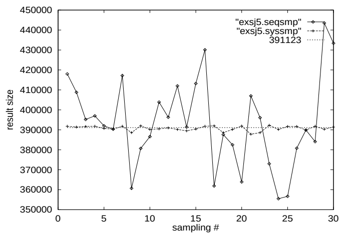
(b) SJ2



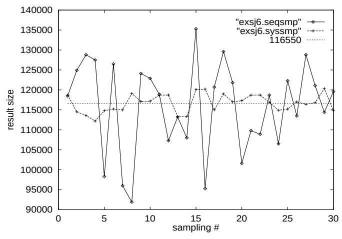
(c) SJ3



(d) SJ4

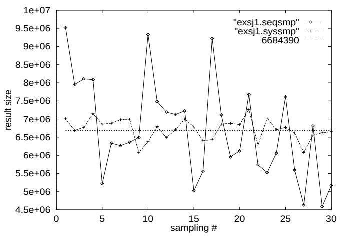


(e) SJ5

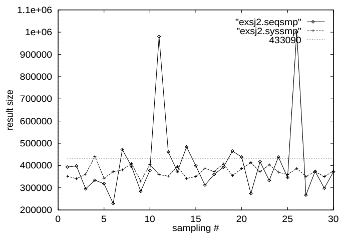


(f) SJ6

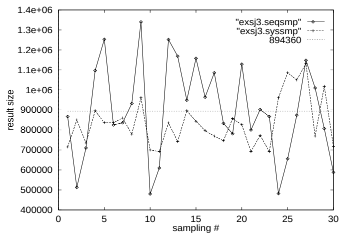
Figure 3.7: Star joins with 2 relations (binary joins)



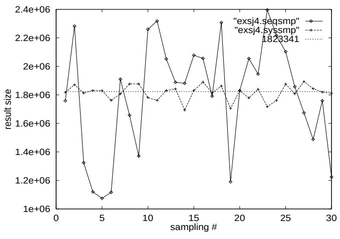
(a) SJ1



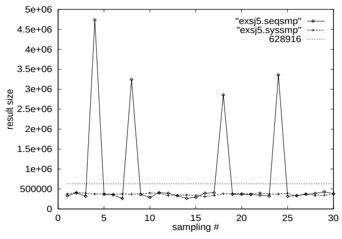
(b) SJ2



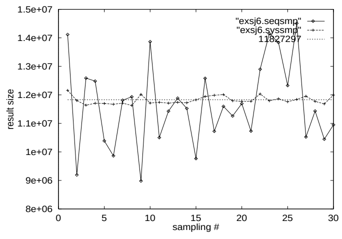
(c) SJ3



(d) SJ4

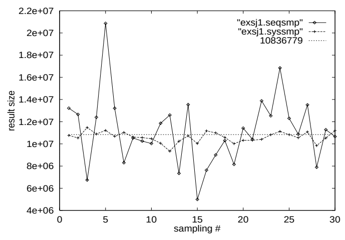


(e) SJ5

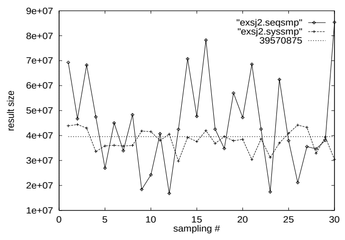


(f) SJ6

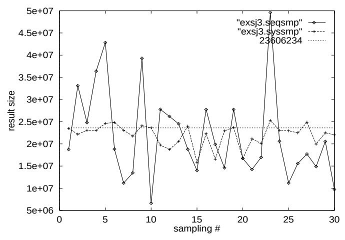
Figure 3.8: Star joins with 3 relations



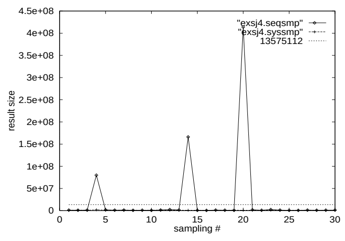
(a) SJ1



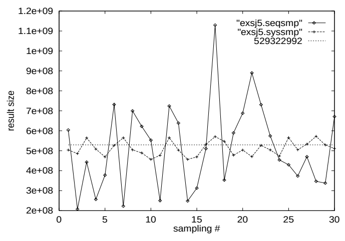
(b) SJ2



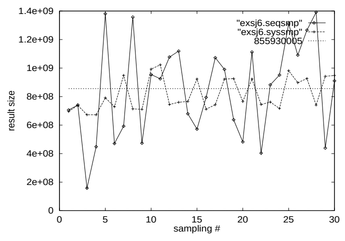
(c) SJ3



(d) SJ4

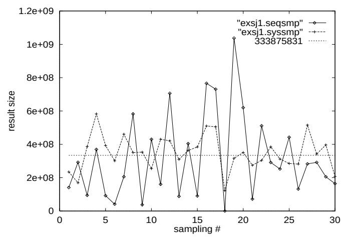


(e) SJ5

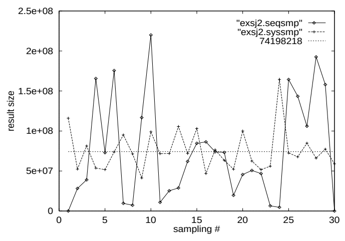


(f) SJ6

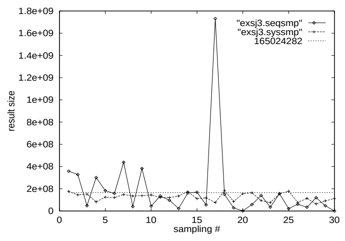
Figure 3.9: Star joins with 4 relations



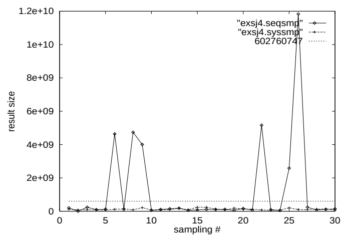
(a) SJ1



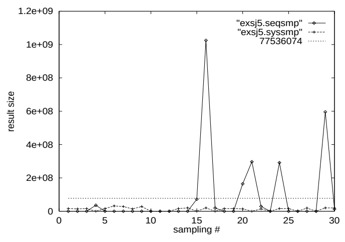
(b) SJ2



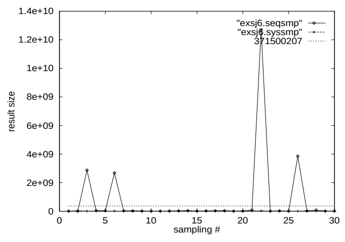
(c) SJ3



(d) SJ4



(e) SJ5



(f) SJ6

Figure 3.10: Star joins with 5 relations

	2 Rels	
exp.	SS	SYSSMP
SJ1	1.817e+04	9.875e+03
SJ2	7.953e+03	5.110e+02
SJ3	1.440e+04	1.583e+03
SJ4	3.165e+03	9.950e+02
SJ5	2.227e+04	1.253e+03
SJ6	1.119e+04	2.219e+03

(a) Root mean square error

	2 Rels	
exp.	SS	SYSSMP
SJ1	1.547e+04	9.048e+03
SJ2	6.448e+03	4.290e+02
SJ3	1.157e+04	1.405e+03
SJ4	2.451e+03	7.640e+02
SJ5	1.729e+04	1.011e+03
SJ6	9.303e+03	1.903e+03

(b) Mean residual error

	2 Rels	
exp.	SS	SYSSMP
SJ1	6	3
SJ2	4	0
SJ3	4	0
SJ4	9	3
SJ5	4	0
SJ6	8	2

(c) Mean relative error

Table 3.12: Estimation errors for 2-relation star joins

	3 Rels	
exp.	SS	SYSSMP
SJ1	1.325e+06	2.869e+05
SJ2	1.698e+05	6.526e+04
SJ3	2.321e+05	1.325e+05
SJ4	3.932e+05	5.207e+04
SJ5	1.128e+06	2.612e+05
SJ6	1.456e+06	1.341e+05

(a) Root mean square error

	3 Rels	
exp.	SS	SYSSMP
SJ1	1.106e+06	2.308e+05
SJ2	1.087e+05	6.053e+04
SJ3	1.883e+05	1.152e+05
SJ4	3.199e+05	3.909e+04
SJ5	6.291e+05	2.600e+05
SJ6	1.177e+06	1.140e+05

(b) Mean residual error

	3 Rels	
exp.	SS	SYSSMP
SJ1	17	3
SJ2	25	14
SJ3	21	13
SJ4	18	2
SJ5	100	41
SJ6	10	1

(c) Mean relative error

Table 3.13: Estimation errors for 3-relation star joins

	4 Rels	
exp.	SS	SYSSMP
SJ1	3.099e+06	5.118e+05
SJ2	1.880e+07	4.468e+06
SJ3	1.022e+07	2.942e+06
SJ4	8.002e+07	1.256e+07
SJ5	2.135e+08	3.955e+07
SJ6	3.294e+08	1.164e+08

(a) Root mean square error

	4 Rels	
exp.	SS	SYSSMP
SJ1	2.304e+06	3.987e+05
SJ2	1.445e+07	3.549e+06
SJ3	8.489e+06	2.046e+06
SJ4	3.181e+07	1.256e+07
SJ5	1.770e+08	3.384e+07
SJ6	2.828e+08	1.105e+08

(b) Mean residual error

	4 Rels	
exp.	SS	SYSSMP
SJ1	21	4
SJ2	37	9
SJ3	36	9
SJ4	234	93
SJ5	33	6
SJ6	33	13

(c) Mean relative error

Table 3.14: Estimation errors for 4-relation star joins

	5 Rels	
exp.	SS	SYSSMP
SJ1	2.552e+08	1.035e+08
SJ2	6.407e+07	2.502e+07
SJ3	3.113e+08	5.174e+07
SJ4	2.593e+09	4.853e+08
SJ5	2.160e+08	6.587e+07
SJ6	2.442e+09	3.503e+08

(a) Root mean square error

	5 Rels	
exp.	SS	SYSSMP
SJ1	2.113e+08	8.076e+07
SJ2	5.341e+07	1.796e+07
SJ3	1.529e+08	4.286e+07
SJ4	1.368e+09	4.825e+08
SJ5	1.252e+08	6.516e+07
SJ6	9.904e+08	3.502e+08

(b) Mean residual error

	5 Rels	
exp.	SS	SYSSMP
SJ1	63	24
SJ2	72	24
SJ3	93	26
SJ4	227	80
SJ5	161	84
SJ6	267	94

(c) Mean relative error

Table 3.15: Estimation errors for 5-relation star joins

To obtain a result size estimate of a star join, the samples of the relations with the indices are obtained via SYSSMP whereas the samples of the relations without any index are obtained via the SS procedure. All the samples are then joined together to produce a result size estimate of the star join.

The objective of all experiments done in this section is to verify the hypothesis that:

when the number of the indexed attributes increases, the quality of join result size estimates should be better.

We used the 5-relation configurations shown in Table 3.11(d) to conduct experiments here. We experimented upon star joins over the 5 participating relations by having 2, 4 and 5 (i.e., all) attributes indexed over the 5 relations.

Table 3.16 shows 6 experiments each of which has 2 and 4 indexed attributes. Assuming that each relation has one join attribute on it, the relations with the indices on them are shown in the table. For example, in experiment SJ1, in the light of 2 indexed attributes, R2 and R3 have the indices on them and thus the rest of the participating relations, i.e., R1, R4, and R5 do not.

exp.	2 attrs indexed	4 attrs indexed
SJ1	R2, R3	R2, R3, R1, R5
SJ2	R5, R3	R5, R3, R4, R2
SJ3	R1, R5	R1, R5, R3, R2
SJ4	R2, R1	R2, R1, R5, R4
SJ5	R3, R5	R3, R5, R4, R1
SJ6	R4, R5	R4, R5, R2, R3

Table 3.16: Relations with 2 and 4 indices on them

All indexed relations shown in the table are generated at random. Note however, that for 4 indexed attributes, two more indexed attributes are randomly generated and added to the set of the previous two indexed attributes. For instance, in experiment SJ1, R2 and R3 are randomly chosen first as the first two indexed attributes, then R1 and R5 are added later to the first two attributes to form the 4 indexed attribute set.

The graphs in Figures 3.11 and 3.12 can be interpreted in the same fashion as the graphs in Section 3.5.3.1. Each graph shows a comparison between the hybrid sampling scheme and pure SS procedure with i indexed attributes, where $i = 2, 4$ or 5. From left to right, the number of indexed attributes are gradually increased, i.e., 2, 4 and all (5) indexed attributes, respectively. Note that the graphs (in the rightmost column) for all attributes indexed are in fact reproduced from the graphs for the 5-relation star joins presented in Section 3.5.3.1. This is for ease and convenience in comparison and to demonstrate a trend when the number of indexed attributes increases to the extreme case.

A clear trend that one can perceive is that the fluctuation of the graphs via the hybrid sampling scheme (which represents the quality of result size estimates) gradually reduces from left to right in all the graphs shown. This should be attributed to the fact that when increasing more indexed attributes, we will have more sample relations with high quality. Furthermore, three kinds of error measures that we have used in Section 3.5.3.1, namely, root mean square, mean residual and mean relative errors are also shown in Table 3.17(a),3.17(b) and 3.17(c), respectively. Both the graphs and tables of error measures indicate the same trend that SYSSMP still promises to work well when there are only some join attributes indexed.

exp.	SS	2 inds	4 inds	all inds
SJ1	2.552e+08	2.045e+08	1.465e+08	1.035e+08
SJ2	6.407e+07	4.789e+07	3.558e+07	2.502e+07
SJ3	3.113e+08	1.670e+08	9.727e+07	5.174e+07
SJ4	2.593e+09	1.571e+09	4.968e+08	4.853e+08
SJ5	2.160e+08	2.219e+08	1.111e+08	6.587e+07
SJ6	2.442e+09	1.630e+09	8.539e+08	3.503e+08

(a) Root mean square error

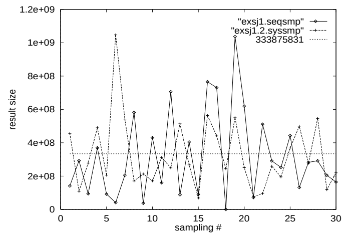
exp.	SS	2 inds	4 inds	all inds
SJ1	2.113e+08	1.636e+08	1.205e+08	8.076e+07
SJ2	5.341e+07	4.259e+07	2.960e+07	1.796e+07
SJ3	1.529e+08	1.167e+08	6.437e+07	4.286e+07
SJ4	1.368e+09	9.473e+08	4.959e+08	4.825e+08
SJ5	1.252e+08	1.133e+08	8.343e+07	6.516e+07
SJ6	9.904e+08	6.997e+08	5.601e+08	3.502e+08

(b) Mean residual error

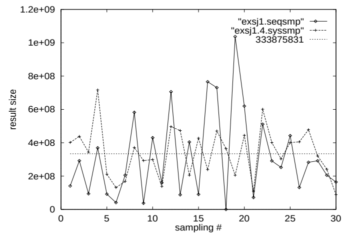
exp.	SS	2 inds	4 inds	all inds
SJ1	63	49	36	24
SJ2	72	57	40	24
SJ3	93	71	39	26
SJ4	227	157	82	80
SJ5	161	146	108	84
SJ6	267	188	151	94

(c) Mean relative error

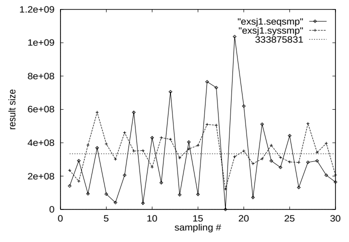
Table 3.17: Error trend with increment of indexed attributes



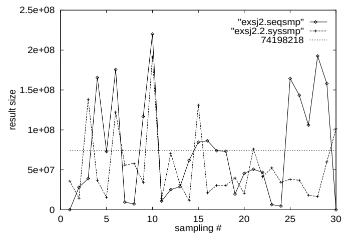
(a) SJ1, 2 indexed attrs



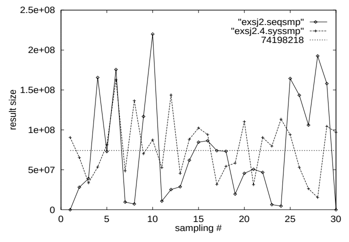
(b) SJ1, 4 indexed attrs



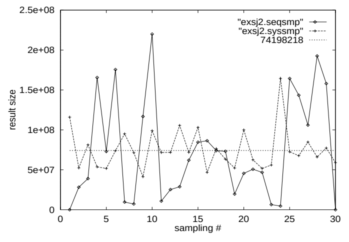
(c) SJ1, all indexed attrs



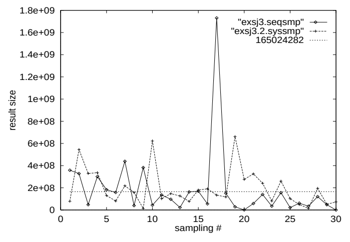
(d) SJ2, 2 indexed attrs



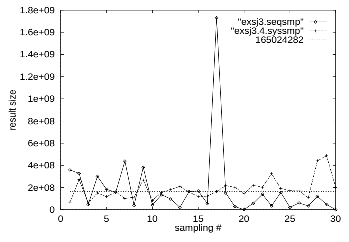
(e) SJ2, 4 indexed attrs



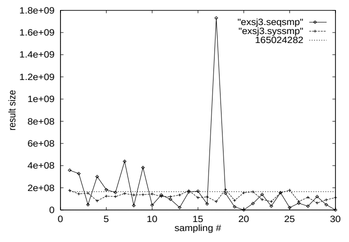
(f) SJ2, all indexed attrs



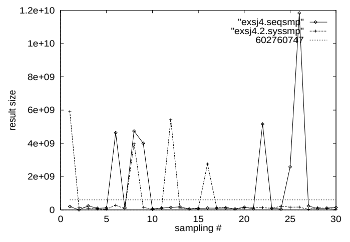
(g) SJ3, 2 indexed attrs



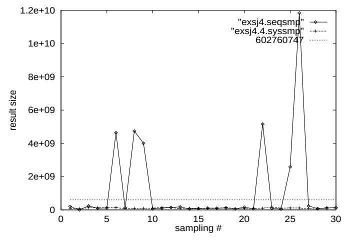
(h) SJ3, 4 indexed attrs



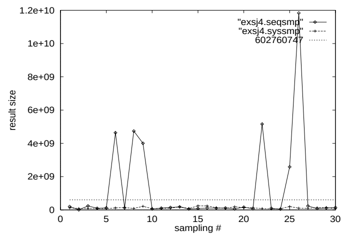
(i) SJ3, all indexed attrs



(j) SJ4, 2 indexed attrs

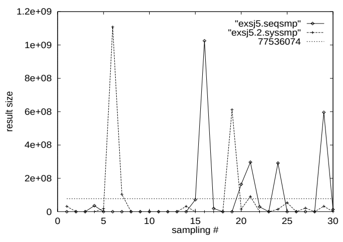


(k) SJ4, 4 indexed attrs

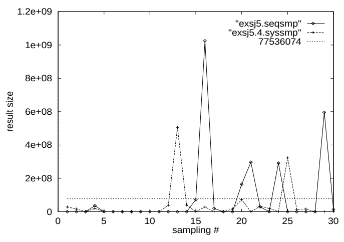


(l) SJ4, all indexed attrs

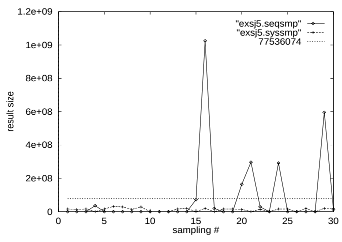
Figure 3.11: Star joins with 5 relations with 2, 4, and all indexed attributes



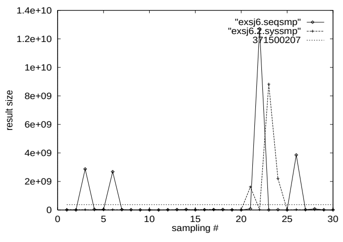
(a) SJ5, 2 indexed attrs



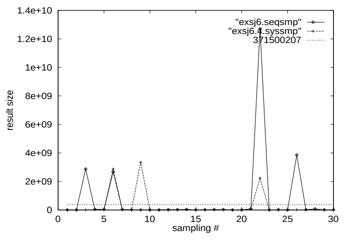
(b) SJ5, 4 indexed attrs



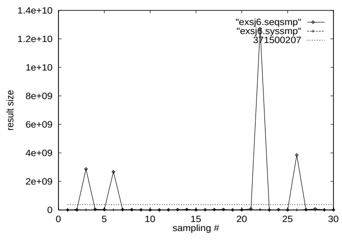
(c) SJ5, all indexed attrs



(d) SJ6, 2 indexed attrs



(e) SJ6, 4 indexed attrs



(f) SJ6, all indexed attrs

Figure 3.12: Star joins with 5 relations with 2, 4, and all indexed attributes

3.6 Systematic sampling for selections

S^2	a population variance over relation R of a selectivity of a complex predicate
S^2_{wsmp}	a variance within systematic sample relations due to the complex predicate
$\overline{Y}$	the selectivity of the complex predicate on R
$\hat{\theta}$	the selectivity of the complex predicate on R' could be $\widehat{\overline{Y}}$ obtained via SYSSMP, $\widehat{\overline{Y}}_{swr}$ obtained via SRSWR or $\widehat{\overline{Y}}_{swor}$ obtained via SRSWOR
$V(\hat{\theta})$	the variance of the estimated $\hat{\theta}$

Table 3.18: Symbol redefinitions

Table 3.18 shows all the notations that we redefine from their original definitions in Table 3.1(b) for the use of selections. The details of each notation are described in this section 3.6 and Section 3.7. The main point of the redefinitions is that instead of being the original definitions which phrase upon "a selectivity of a distinct value" as used for joins, now the new definitions are with respect to "a selectivity of a complex predicate". In spite of the redefinitions, the notations defined and descriptions given in the main sections for joins (Sections 3.3 and 3.4) and selections (Sections 3.6 and 3.7) are self-contained so the reader should not be confused by the redefinitions here.

Given a complex predicate query Q on R with cardinality N, the sampling algorithm in Figure 3.13 is to obtain a systematic sample relation R' and thus an estimated selectivity $\widehat{\overline{Y}}$ of Q calculated from R'.

We then calculate an estimated size of the query Q by $\widehat{\overline{Y}} * N$. Again in order to obtain an accurate estimated selectivity $\widehat{\overline{Y}}$ for a query Q, we need a method which can yield "good" sample relations; that is, the sample relation obtained can well represent the joint frequency distribution of the attributes (which are specified in the complex predicate of query Q) in the original relation.

In what follows, we first describe the problems about the sample size n in Section 3.6.1. Then we propose two solutions to the sample size in Sections 3.6.2 and 3.6.3, respectively.

```
STEP 1. let $n$ ($\le N$) be a sample size of relation $R'$ to be sampled from $R$. We describe
        details of problems about $n$ for selections and how to work it out in Section 3.6.1.
STEP 2. calculate a step size $k = \frac{N}{n}$ to step through relation $R$ to obtain $R'$.
STEP 3. $\check{n} = 0$;  $s = 0$.
STEP 4. sample a tuple at random from the first $k$ tuples of $R$.
STEP 5. $\check{n} = \check{n} + 1$.
STEP 6. check if the query $Q$ satisfies the tuple obtained and if so, then $s = s + 1$.
STEP 7. take a tuple, $k$ tuples away from the current one if there is any and go to step 5.
STEP 8. return $\widehat{\overline{Y}} = \frac{s}{\check{n}}$ as the estimated selectivity of $Q$.
```

Figure 3.13: Selectivity estimation by systematic sampling

3.6.1 How many tuples to sample

The derivation for a sample size n required for a query Q can be done in a similar fashion as the derivation we have shown in Section 3.3.1.2. More precisely, if the size of relation R is large, then the formula (3.12), namely, $\tilde{n}_0 = \frac{t^2(\mu)(1-\mu)}{B^2}$ we have derived for joins can be reused for selections. The substitutions are shown in Table 3.19(a).

μ = join selectivity	$\longrightarrow \overline{Y}$ = selection selectivity
$\tilde{N} = \prod_{j=1}^{m} \lvert R_j \rvert$	$\longrightarrow N$ = cardinality of R
$\tilde{n} = \prod_{j=1}^{m} \lvert R'_j \rvert$	$\longrightarrow n$ = cardinality of R'

(a) Substitutions

Example
Let size of R, $N = 10,000$ tuples
selectivity $\overline{Y} = 0.01$
relative error $\epsilon = 0.1$ (10%)
prob. of out-of-bound error $\alpha = 10\%$
confidence level $= 100 - \alpha = 90\%$
thus $t^2 = 2.71$

(b) Oversampling problem

Table 3.19: Substitutions and the oversampling problem

Hence a sample size n of R' required for SYSSMP would be:

$$n = \frac{t^2\overline{Y}(1-\overline{Y})}{B^2} \tag{3.36}$$

where t = the abscissa of the normal curve that cuts off an area α at the tail, $B = abs(\overline{Y} - \widehat{\overline{Y}})$ = a bound on error desired, $\overline{Y}$ = a selectivity of a complex predicate in query Q, $\widehat{\overline{Y}}$ = an estimated selectivity of $\overline{Y}$ (the actual selectivity), and α = a probability that the error is not within the bound given.

The above formula can be rephrased in terms of a relative error ϵ if desired. Since an error bound $B = abs(\overline{Y} - \widehat{\overline{Y}})$ and so a relative error $\epsilon = \frac{abs(\overline{Y} - \widehat{\overline{Y}})}{\overline{Y}}$, the error bound B would be equal to $\overline{Y}\epsilon$. Substituting $B = \overline{Y}\epsilon$ into formula (3.36), we get:

$$n = \frac{t^2 \overline{Y}(1 - \overline{Y})}{\overline{Y}^2 \epsilon^2} = \frac{t^2(1 - \overline{Y})}{\overline{Y}\epsilon^2} \tag{3.37}$$

There are two principal problems in using formula (3.37). The first is that $\overline{Y}$ is unknown and we want to estimate it. $\overline{Y}$ also varies from query to query and therefore we can't work out its value in advance, as opposed to joins (see also the description of the problem with the exponential number of selections in Section 3.3.1.4).

The second problem is the oversampling problem. Given an example in Table 3.19(b) and using formula (3.37), the sample size n required would be $\frac{2.71(1-0.01)}{0.01(0.1)^2} = 26,829$, which is larger than the cardinality of R itself.

In order to make the formula usable for systematic sampling, we propose two solutions as follows:

1. do a double sampling [Cox 1952] to find the estimator of $\overline{Y}$, i.e., $\widehat{\overline{Y}}$ in the first sampling and use $\widehat{\overline{Y}}$ instead of $\overline{Y}$ to calculate a sample size n in formula (3.37). This method is called *double systematic sampling*. The double sampling algorithm was first proposed by Hou et al. [1991*b*] mainly for simple random sampling but the main problem against the algorithm proposed is that it may cause the oversampling to occur when a selectivity of a selection query is very small. The details of our extension to the original algorithm are described in Section 3.6.2.

2. find an average size $\bar{n}$ of samples needed by the SS procedure, namely:

$$\bar{n} = \frac{\sum_{i=1}^{|\mathcal{Q}|} n_i}{|\mathcal{Q}|} \tag{3.38}$$

   where n_i is the sample size of the ith previous user query in a query set $\mathcal{Q}$ and use $\bar{n}$ or slightly less tuples as n to terminate SYSSMP. The drawback of this method is that we would require to run SS through the set of queries $\mathcal{Q}$ before this method will work. This method is called *feedback systematic sampling*. The details of the method are described in Section 3.6.3.

3.6.2 Double systematic sampling

Given a query Q on R, the following is the sketch of the main idea of the algorithm for double systematic sampling:

1. use a small systematic sample size n_1 in order to estimate $\overline{Y}$ in the formula $\frac{t^2\overline{Y}(1-\overline{Y})}{B^2}$.

   Let $\widehat{\overline{Y}}$ be the selectivity due to the sample size n_1. Thus a systematic sample size n_2 required would be:

$$n_2 = \frac{t^2\widehat{\overline{Y}}(1-\widehat{\overline{Y}})}{B^2}$$

2. if $n_2 \leq n_1$, then return $\widehat{\overline{Y}}$ as the selectivity of Q (no more sampling needed).

3. sample the remaining tuples, i.e., $n_2 \leftarrow n_2 - n_1$ and return the estimated selectivity $\widehat{\overline{Y}}$ of Q:

$$\widehat{\overline{Y}} = \frac{(s_1 + s_2)}{(n_1 + n_2)}$$

   where s_1 and s_2 are each the total number of tuples which satisfy Q in the first and second samples, respectively and $(n_1 + n_2)$ is, in fact, equal to the sample size n required.

The sketch of the algorithm above has the oversampling problem — the algorithm does not have any mechanism to stop sampling when a selectivity of Q is very small, thus incurring too much sampling done as described in Section 3.6.1. Many modern sampling algorithms, e.g., adaptive [Lipton and Naughton 1990] and sequential [Haas and Swami 1992] algorithms, have known such a problem and introduced, in a different manner, a sanity bound to cure the problem. A sanity bound is a bound to make sure that a sampling algorithm will terminate before the problem occurs. Table 3.20 shows our proposal for a sanity bound. $0 < \beta_{max} \leq 1$ is a maximum sampling fraction on relation R from which the maximum number of tuples sampled from R would be $\beta_{max} * N$.

We also make sure that a value of ψ ($\geq$ 0) used in Table 3.20 will produce a reasonable value of n_2 which is considerably less than N. Figure 3.14 shows all complete details of the algorithm, including the sanity bound we have just proposed.

```
if $n_2 > \lceil(\beta_{max} * N)\rceil$ then
    $B_2 = \epsilon\psi$          [ recall that $B = \epsilon\overline{Y}$ ]
    $n_2 = \frac{t^2\psi(1-\psi)}{B_2^2}$
endif
```

Table 3.20: A proposal for a sanity bound

```
Procedure: double_syssmp
input:   $Q$ = a selection query,
         $n_1$ = minimum # of tuples in the first sample,
         $\epsilon$ = relative error,
         $\psi \geq 0$ = sanity bound,
output:  an estimated selectivity of Q

sample the first $n_1$ tuples from R
(let $s_1$ be the total number of tuples which satisfy $Q$)
$\widehat{\overline{Y}} = \frac{s_1}{n_1}$
$B = \epsilon\widehat{\overline{Y}}$
if $B$ is not zero then
    $n_2 = \frac{t^2\widehat{\overline{Y}}(1-\widehat{\overline{Y}})}{B^2}$
    if $n_2 > \lceil(\beta_{max} * N)\rceil$ then
        $B_2 = \epsilon\psi$
        $n_2 = \frac{t^2\psi(1-\psi)}{B_2^2}$
    endif
else
    $B_2 = \epsilon\psi$
    $n_2 = \frac{t^2\psi(1-\psi)}{B_2^2}$
endif
if $n_2 \leq n_1$ then
    return $\widehat{\overline{Y}} * N$
endif
# sample the remaining tuples
$n_2 = n_2 - n_1$
sample the remaining tuples $n_2$ from R
(let $s_2$ be the total number of tuples which satisfy $Q$)
return $\frac{(s_1+s_2)}{(n_1+n_2)}$
```

Figure 3.14: Double systematic sampling

A couple of issues should be noted about the algorithm. First, when $\widehat{\widehat{Y}} = 0$, then B (error bound) would also be zero. Then the **else** block between lines 11 and 14 will get fired and a sample size n_2 will be calculated using the specified sanity bound. Likewise, when $\widehat{\widehat{Y}}$ approaches zero (not zero but a very small value), then a sample size n_2 in line 6 will produce a large value perhaps greater than the maximum sampling size $\lceil(\beta_{max} * N)\rceil$. In this case, the **if** block between lines 7 and 10 will get fired. Note that this **if** block for a sanity bound has the same rationale as the **else** block explained above.

3.6.3 Feedback systematic sampling

Using sample size feedback as a consequence of having run SS through a set of queries $\mathcal{Q}$, feedback systematic sampling works as follows:

STEP 1. calculate an average sample size $\bar{n}$ required for any new incoming query Q by:

$$\bar{n} = \frac{\sum_{i=1}^{|\mathcal{Q}|} n_i}{|\mathcal{Q}|}$$

where n_i = the sample size of the ith previous user query in $\mathcal{Q}$.

STEP 2. calculate a step size $k = \frac{N}{\bar{n}}$ to step through relation R to obtain R'. $\bar{n}$ used could be slightly smaller than the calculated value in step 1.

STEP 3. $\check{n} = 0;\ s = 0$.

STEP 4. sample a tuple at random from the first k tuples of R.

STEP 5. $\check{n} = \check{n} + 1$.

STEP 6. check if the query Q satisfies the tuple obtained and if so, then $s = s + 1$.

STEP 7. take a tuple, k tuples away from the current one if there is any and go to step 5.

STEP 8. return $\widehat{\widehat{Y}} = \frac{s}{\check{n}}$ as the estimated selectivity of Q.

Figure 3.15: Feedback systematic sampling

The drawback of this method is that we need to run SS through a set of queries $\mathcal{Q}$ before this method will work.

3.7 Application of theoretical foundation of SYSSMP to selections

Recall that the main theorem (3.4.1) for SYSSMP in Section 3.4 states on a proportion (which is a mean), defined by a ratio of a total number of points which satisfy a criterion of interest divided by the total number of points in the population. Selections, in essence, are a proportion of:

$$\frac{\text{the total number of tuples which satisfy the selection predicate}}{\text{the total number of tuples in a relation}}$$

Our aim in this section is simply to apply the foundation we have developed earlier for joins to selections. That is, we will show that systematic sampling can, in general, guarantee to yield an efficient sample relation, thus producing an estimate of high quality for an actual selectivity of a complex predicate.

Like in the join case, for selections, SYSSMP would be more efficient in yielding a sample relation than SRS with/without replacement if and only if a variance of an estimated selectivity of a complex predicate produced by SYSSMP is lower. In Section 3.7.1, we will show when this would occur.

3.7.1 Variance of estimated selectivity of a complex predicate

A complex predicate on relation R can be specified upon more than one attribute of R. To simplify our description but without loss of generality, the example relation R with respect to a single attribute of R in Figure 3.5 is reused for the description in this section. The analogy to more than one attribute can be done in the same manner.

Let x be a complex predicate in a query Q. Using the notations in Table 3.4(b), we redefine the function $f(t_{ij}, x)$ of a [0,1] value which we have used in Section 3.4.1 as follows:

$$y_{ij} = f(t_{ij}, x) = \begin{cases} 1 & \text{if the } j\text{th tuple of the } i\text{th sample satisfies predicate } x \text{ of } Q \\ 0 & \text{if it doesn't} \end{cases}$$

In view of the selectivity of predicate x, if the jth tuple of the ith sample relation satisfies predicate x, then y_{ij} would be 1, otherwise 0. Using systematic sample relation number 1 in Table 3.4(a) and using the [0,1] value representation, a selectivity given predicate $x = (\text{a1} \geq 3)$ is $\bar{y}_1 = \frac{a_1}{n} = \frac{2}{5}$ (there are two rows which satisfy predicate x in the sample relation number 1). With the same predicate x, the selectivities calculated using systematic sample relations 2, 3, 4 and 5 are equal to $\bar{y}_2 = \frac{a_2}{n} = \frac{2}{5}, \bar{y}_3 = \frac{a_3}{n} = \frac{2}{5}, \bar{y}_4 = \frac{a_4}{n} = \frac{2}{5}$ and $\bar{y}_5 = \frac{a_5}{n} = \frac{3}{5}$, respectively.

We redefine the following symbols that we have used in Section 3.4. $\overline{Y}$ is used as the actual selectivity of a distinct value in a common join attribute domain. In this section, we use $\overline{Y}$ as the actual selectivity of a complex predicate x in query Q. Let $V(\widehat{\overline{Y}})$, $V(\widehat{\overline{Y}}_{swr})$ and $V(\widehat{\overline{Y}}_{swor})$ be a variance of an estimated selectivity of predicate x on a sample relation obtained via SYSSMP, SRSWR and SRSWOR, respectively, where $\widehat{\overline{Y}}, \widehat{\overline{Y}}_{swr}$ and $\widehat{\overline{Y}}_{swor}$ are the selectivities of predicate x calculated from each individual R' obtained via SYSSMP, SRSWR, and SRSWOR respectively. S^2, a population variance, is the variance of the selectivity of predicate x over the entire relation R which is defined the same as the ones in equations (3.17) and (3.18). A is the total number of tuples on R that satisfy predicate x; consequently, $\overline{Y} = \frac{A}{N}$.

The main theorem (3.4.1) in Section 3.4 is applicable for a proportion and the actual selectivity $\overline{Y}$ of a complex predicate x is a proportion of tuples in R which satisfy predicate x, namely, $\frac{A}{N}$. The main theorem can be rephrased for a complex predicate x as follows:

Corollary 3.7.1 The variance of the selectivity $\widehat{\overline{Y}}$ of predicate x on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{N-1}{N} S^2 - \frac{k(n-1)}{N} S^2_{wsmp} \tag{3.39}$$

where

$$S^2_{wsmp} = \frac{1}{k(n-1)} \sum_{i=1}^{k} \sum_{j=1}^{n} (y_{ij} - \bar{y}_i)^2 \tag{3.40}$$

is the variance among tuples that lie within the same systematic sample relation. y_{ij} as shown in Table 3.4(b) is a function $f(t_{ij}, x)$ of a [0,1] value of tuple j of systematic sample relation i. $\bar{y}_i = \frac{a_i}{n}$ is the selectivity of predicate x calculated from the ith systematic sample relation.

Like the two results for joins in Corollaries (3.4.1) and (3.4.2) in Section 3.4, such two results can be rephrased for a complex predicate x as follows:

Corollary 3.7.2 The selectivity of predicate x obtained from a systematic sample relation is more accurate than the selectivity of the same predicate from a simple random sample relation without replacement if and only if

$$S^2_{wsmp} > S^2 \tag{3.41}$$

Corollary 3.7.3 The selectivity of predicate x obtained from a systematic sample relation is more accurate than the selectivity of the same predicate from a simple random sample relation with replacement if and only if

$$S^2_{wsmp} > \frac{N-1}{N} S^2 \tag{3.42}$$

The first result in equation (3.41) above indicates that if $S^2_{wsmp} > S^2$, then $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swor})$ while the second result in equation (3.42) indicates that if $S^2_{wsmp} > \frac{N-1}{N} S^2$, then $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swr})$.

To ensure with a high possibility that the two results above would occur, a sort over the entire relation, in ascending or descending order must be done, which will then result in heterogeneous tuples within the same systematic sample relation.

The following is an example demonstrating how to calculate different kinds of the variances we have just described above. The experimental results obtained in Section 3.8.2 need an understanding of how S^2_{wsmp} and S^2 are calculated and thus it is important for the reader to work out the example below.

Example 3: Using relation R in Figure 3.5 and given predicate $x = (\text{a1} \geq 3)$, calculate variances $V(\widehat{\overline{Y}})$, $V(\widehat{\overline{Y}}_{swr})$ and $V(\widehat{\overline{Y}}_{swor})$.

The selectivity $\overline{Y}$ of predicate x on R is $\frac{A}{N}$ which is $\frac{11}{25}$. From the theorem given in equation (3.39), the variance $V(\widehat{\overline{Y}})$ on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{N-1}{N} S^2 - \frac{k(n-1)}{N} S^2_{wsmp}$$

where the variance S^2 is:

$$S^2 = \frac{N}{N-1} \overline{Y}(1 - \overline{Y})$$

from equation (3.18). The variance S^2_{wsmp} within systematic sample relations in

systematic sample no.					
i	1	2	3	4	5
	1	1	1	1	1
	1	1	1	1	2
	2	2	2	2	3
	3	3	3	4	4
	4	4	5	5	5
a_i	2	2	2	2	3
$\bar{y}_i$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{3}{5}$

(a) Sample selectivities

sample no.	$a_i - n\bar{y}_i^2$
$i = 1$	$2 - 5(\frac{2}{5})^2$
$i = 2$	$2 - 5(\frac{2}{5})^2$
$i = 3$	$2 - 5(\frac{2}{5})^2$
$i = 4$	$2 - 5(\frac{2}{5})^2$
$i = 5$	$3 - 5(\frac{3}{5})^2$
$\sum_{i=1}^{5}(a_i - n\bar{y}_i^2)$	6

(b) Variance within sample relations

Table 3.21: Selectivities and variances of systematic sample relations

equation (3.40) is:

$$\begin{aligned} S_{wsmp}^2 &= \frac{1}{k(n-1)}\sum_{i=1}^{k}\sum_{j=1}^{n}(y_{ij} - \bar{y}_i)^2 \\ &= \frac{1}{k(n-1)}\sum_{i=1}^{k}(\sum_{j=1}^{n} y_{ij}^2 - n\bar{y}_i^2) \qquad (3.43) \end{aligned}$$

Since y_{ij} is a [0,1] value, $\sum_{j=1}^{n} y_{ij}^2 = a_i$, the number of tuples in the ith systematic sample relation satisfying predicate x. $\bar{y}_i = \frac{a_i}{n}$ is the selectivity of predicate x on the ith systematic sample relation. Substitute a_i to equation (3.43). This gives:

$$S_{wsmp}^2 = \frac{1}{k(n-1)}\sum_{i=1}^{k}(a_i - n\bar{y}_i^2)$$

Table 3.21(b) shows how to calculate the variance S_{wsmp}^2 in the equation above step-by-step.

Thus,

$$\begin{aligned} S_{wsmp}^2 &= \frac{1}{5(5-1)}6 = 0.3 \\ S^2 &= \frac{25}{25-1}(\frac{11}{25})(1 - \frac{11}{25}) = 0.257. \end{aligned}$$

The variance $V(\widehat{\overline{Y}})$ of the selectivity of predicate x on a systematic sample relation is:

$$V(\widehat{\overline{Y}}) = \frac{25-1}{25}0.257 - \frac{5(5-1)}{25}0.3 = 0.00672.$$

The variance $V(\widehat{\overline{Y}}_{swor})$ of the selectivity of predicate x on a simple random sample relation without replacement is:

$$V(\widehat{\overline{Y}}_{swor}) = \frac{N-n}{N}\frac{S^2}{n} = \frac{25-5}{25} * \frac{0.257}{5} = 0.04112.$$

The variance $V(\widehat{\overline{Y}}_{swr})$ of the selectivity of predicate x on a simple random sample relation with replacement is:

$$V(\widehat{\overline{Y}}_{swr}) = \frac{N-1}{N}\frac{S^2}{n} = \frac{25-1}{25} * \frac{0.257}{5} = 0.049344.$$

Since $S^2_{wsmp} > S^2$, $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swor})$ (following the result in equation (3.41)), so is $V(\widehat{\overline{Y}}) < V(\widehat{\overline{Y}}_{swr})$ (following the result in equation (3.42)). As one might have expected, $V(\widehat{\overline{Y}}_{swor}) < V(\widehat{\overline{Y}}_{swr})$.

3.8 Experimental results for selections

A method is first described in Section 3.8.1 for generating synthetic data for selection queries. These synthetic relations and sets of queries are used subsequently in Sections 3.8.2 and 3.8.3.

Next in Section 3.8.2, we conduct the first set of experiments for an analytical study of variances of estimated selectivities of complex predicates which compares among SYSSMP, SRSWOR and SRSWR. This is to support the application of the theoretical foundation we have shown in Section 3.7 — the variance of an estimated selectivity of a complex predicate by SYSSMP, in general, is lower than the variance by SRSWOR and SRSWR.

Then in Section 3.8.3, we conduct the second set of experiments to demonstrate that the lower variance can indeed assist in more accurate computation for result sizes of selection queries.

3.8.1 Experimental setup

Functional dependency (FD) as defined by Korth and Silberschatz [1991] is a set of constraints defined on some relations of a database and each constraint expresses one fact about the database application we are modelling. Given that ν and γ are a (non-strict) subset of the attributes of relation R, a functional dependency $\nu \rightarrow \gamma$ which holds on R can be defined as: for all pairs of tuples t_1 and t_2, if the attribute values on ν of t_1 are equal to the ones on ν of t_2, then the attribute values on γ of t_1 must be equal to the ones on γ of t_2.

Real world data may have some form of FDs which imply "attribute dependence" of some attributes on a relation. Christodoulakis [1984] also commented that due to the nature of database environments, attribute dependence are typical rather than unusual. For example, attributes "degree" and "salary" of employees in a company may have a relationship, i.e., a functional dependency: degree → salary, as shown in Table 3.22.

degree	salary
Bachelor	10,000
Master	15,000
PhD	20,000

Table 3.22: an FD

Christodoulakis [1984] also showed that large errors in query size estimates may occur if the attribute independence assumption is used to calculate query result size estimates. This exactly corresponds to the results obtained in [Harangsri et al. 1996*c*; Poosala and Ioannidis 1997] in relation to the attribute independence and dependence assumptions.

To our best knowledge, the synthetic relations generated in this thesis are the first attempt to emulate real world data. The idea is similar to the data set generator [Melli 1997] employed in the machine learning community, which attempts to generate relations based on rules. Rules can be regarded as functional dependencies. Furthermore, in line with our belief, the author also said that real world data sets are likely to have some implicit structure. We believe that synthetic data generators based on this idea should be more robust than the data randomly generated whose values among attributes of the relation are independent of one another — no any kind of correlation among the attributes of the relation at all.

We generated several various relations with different cardinalities, FDs and numbers of distinct values as shown in Table 3.23. The table comprises 12 relations of 10 attributes in each relation with 10k, 50k and 100k cardinalities, each of which has a

	10k		50k		100k	
reln	funcdep	dist mode	funcdep	dist mode	funcdep	dist mode
R1	a2 → a3 a4 a5	unf(3959,4159)	a3 → a4 a5 a7	fdist(30,15)	a1 a6 → a4	exp(117)
	a9 → a8 a7	fdist(30,11)	a2 → a6 a8	norm(129,244)	a5 → a10	chisq(8)
R2	a3 a9 → a7	exp(80)	a3 a5 → a8 a9	unf(3099,4099)	a3 a6 → a8 a10	norm(103,308)
	a1 a4 → a6 a8	chisq(5)	a10 → a1 a2	norm(92,347)	a4 → a5 a7	chisq(9)
R3	a3 a5 → a7	norm(101,142)	a9 a2 → a5	fdist(30,15)	a8 → a9	fdist(30,13)
	a2 a6 a4 → a9	chisq(19)	a1 a3 a6 → a4	exp(88)	a1 a3 a4 → a5	chisq(12)
R4	a3 a7 → a9	exp(138)	a7 → a8 a6 a5	chisq(5)	a7 → a8 a10	exp(98)
	a5 → a10	unf(3177,3377)	a1 a3 → a2	unf(3573,4573)	a1 a2 → a6	norm(146,177)

(a) Functional dependencies

reln	numbers of distinct values of a1, a2, ..., a10
R1-10k	494, 201, 159, 159, 165, 536, 10, 10, 10, 854
R2-10k	26, 504, 435, 26, 474, 26, 334, 25, 435, 854
R3-10k	494, 47, 801, 47, 801, 47, 489, 488, 45, 854
R4-10k	494, 504, 677, 492, 201, 536, 677, 488, 448, 182
R1-50k	494, 1523, 10, 10, 10, 505, 10, 466, 707, 854
R2-50k	487, 497, 1001, 492, 1001, 536, 724, 412, 546, 2085
R3-50k	606, 8, 606, 342, 8, 606, 724, 488, 8, 854
R4-50k	1001, 429, 1001, 492, 28, 28, 28, 24, 707, 854
R1-100k	858, 504, 474, 417, 35, 858, 724, 488, 707, 33
R2-100k	494, 504, 2022, 38, 37, 2022, 36, 478, 707, 773
R3-100k	45, 504, 45, 45, 43, 536, 724, 12, 12, 854
R4-100k	1215, 1215, 474, 492, 474, 480, 730, 385, 707, 506

(b) numbers of distinct values

Table 3.23: Relations with different cardinalities, FDs, distributions and distinct values

different set of FDs with various kinds of data distributions and different numbers of distinct values in it.

For example, relation R1 with cardinality 10k (see Table 3.23(a)) has two FDs, the first of which is a2 → a3 a4 a5 with uniform distribution unf(3959,4159) and the second is a9 → a8 a7 with F distribution fdist(30,11).

For the rest of attributes of R1 with 10k, i.e., a1, a6 and a10, they are all independent of any others (and thereby suit the attribute independence assumption) and each attribute has its own single-attribute frequency distribution which is randomly picked from any of the data distributions shown in Table 3.8. For all these attributes, we ignore to show their frequency distributions here.

3.8.2 Variance of estimated selectivity of a complex predicate

In this section, we use only the 10k relations in Table 3.23 to study selectivity variances by SYSSMP compared against those by SRSWR and SRSWOR. These 10k and the other remaining relations together with their queries will be used again later in Section 3.8.3.

Using the 10k relations with 10 attributes on each relation, we have conducted 40 experiments on 4 relations, R1, R2, R3 and R4 — 10 experiments are for one relation and therefore one experiment is for one attribute. A sample size n used in all experiments in this section is $n = 10\%$ of the original sizes of the relations, which is equal to 1000 tuples. Thus the step size k would be equal to $\frac{N}{n} = \frac{10000}{1000} = 10$ in all the experiments. k also represents the total number of all possible unique systematic sample relations which can be obtained from an original relation.

For an attribute of relation R, an index (via either a physical index or a sorted attribute on R) is created on the attribute. Then apply the procedure in Figure 3.16 in order to find:

1. when SYSSMP is more efficient than SRS with and without replacement, namely, $S^2_{wsmp} > S^2$,
2. when the latter is better, namely, $S^2 > S^2_{wsmp}$,
3. and when they are as good, namely, $S^2_{wsmp} = S^2$.

```
input:   $\mathcal{Q}$ = a set of queries,
output:  percentages for three cases

$count_{S^2} = 0$;  $count_{S^2_{wsmp}} = 0$;  $count_{eqi} = 0$
for each query $Q \in \mathcal{Q}$ do
  run $Q$ through $k$ systematic samples of relation $R$ to find $S^2_{wsmp}$
  run $Q$ through relation R to find $S^2$
  if $S^2_{wsmp} > S^2$ then
    $count_{S^2_{wsmp}} = count_{S^2_{wsmp}} + 1$
  else
    if $S^2 > S^2_{wsmp}$ then
      $count_{S^2} = count_{S^2} + 1$
    else
      $count_{equi} = count_{equi} + 1$
    endif
  endif
endfor
return $100 * \frac{count_{S^2_{wsmp}}}{|\mathcal{Q}|}$, $100 * \frac{count_{S^2}}{|\mathcal{Q}|}$, $100 * \frac{count_{equi}}{|\mathcal{Q}|}$
```

Figure 3.16: Procedure to count the number of times that SYSSMP/SRS outperforms

Repeat creating an index on another attribute of relation R and applying the same procedure in Figure 3.16 until all the attributes of R are exhausted. Table 3.24 shows by percentage in three columns when (1) $S^2_{wsmp} > S^2$, (2) $S^2 > S^2_{wsmp}$ and (3) $S^2_{wsmp} = S^2$, respectively.

attr	% $(1 > 2)$	% $(2 > 1)$	% $(1 = 2)$
a1	42.25	48.25	9.5
a2	79.75	10.75	9.5
a3	75.75	14.75	9.5
a4	75	15.5	9.5
a5	81.25	9.25	9.5
a6	57.75	32.75	9.5
a7	72	18.5	9.5
a8	73.25	17.25	9.5
a9	74	16.5	9.5
a10	44.75	45.75	9.5

(a) R1-10k

attr	% $(1 > 2)$	% $(2 > 1)$	% $(1 = 2)$
a1	76.25	20.75	3
a2	52.5	44.5	3
a3	86.5	10.5	3
a4	76.5	20.5	3
a5	60.5	36.5	3
a6	76.5	20.5	3
a7	85.5	11.5	3
a8	79.5	17.5	3
a9	84.75	12.25	3
a10	56	41	3

(b) R2-10k

attr	% $(1 > 2)$	% $(2 > 1)$	% $(1 = 2)$
a1	62	36	2
a2	77.75	20.25	2
a3	73.25	24.75	2
a4	73	25	2
a5	82.75	15.25	2
a6	73	25	2
a7	84.5	13.5	2
a8	53.75	44.25	2
a9	74	24	2
a10	63.5	34.5	2

(c) R3-10k

attr	% $(1 > 2)$	% $(2 > 1)$	% $(1 = 2)$
a1	70.25	29.25	0.5
a2	56.75	42.75	0.5
a3	78	21.5	0.5
a4	60	39.5	0.5
a5	81	18.5	0.5
a6	64.75	34.75	0.5
a7	87.25	12.25	0.5
a8	58.25	41.25	0.5
a9	77	22.5	0.5
a10	79.5	20	0.5

(d) R4-10k

*** $1 = S^2_{wsmp}$, $2 = S^2$.

Table 3.24: Selectivity variances on each indexed attribute over 400 queries

For example, with a relation, say R1, a set $\mathcal{Q}$ of 400 queries is used to perform experiments and this same set of queries is also used when repeating experiments with other indexed attributes a2, a3, ..., a10. The same process is also employed for the other relations R2, R3 and R4. All the queries used in $\mathcal{Q}$ are in conjunctive normal form as follows:

$$\bigwedge_{\text{some } \mathrm{a}i \,\in\, \{\mathrm{a1},\, \mathrm{a2}, \ldots, \mathrm{a10}\}} \mathrm{a}i \ \mathit{relopt}\ x_{\mathrm{a}i} \tag{3.44}$$

According to Table 3.24, one can clearly see that over 400 queries used, the probability (percentage) that SYSSMP is more efficient than SRS in most of the cases is significantly higher. Let us emphasise here again that when $S^2_{wsmp} > S^2$, the variances of estimated selectivities of queries due to SYSSMP would be lower than SRS with/without replacement and, hence, the quality of sample relations yielded by SYSSMP would be higher. We also note in the table that when SRS and SYSSMP are as efficient, i.e., $S^2_{wsmp} = S^2$, the selectivities of the queries used are zero. When query selectivities are zero, whatever sampling method used will always give the same variance of zero. Therefore, it is insignificant for this case and can be disregarded.

Furthermore, Table 3.24 gives a feeling that SYSSMP will be at least as efficient as SRS with/without replacement. For instance, in Table 3.24(a) at attributes a1 and a10, the performance of the two is as good, i.e., about 50% by 50%.

3.8.3 Query size estimation

As for the join queries, for selection queries we also used the three error measures as defined in Figure 3.6 in page 133, where μ is substituted by an actual selection selectivity $\overline{Y}$, $\hat{\mu}$ by an estimated selection selectivity $\widehat{\overline{Y}}$ and $\mathcal{Z}$ by a total number of queries used $|\mathcal{Q}|$ (we used 400 in all experiments), $\tilde{N}$ by a cardinality N of the relation of interest.

We used all the relations shown in Table 3.23 and the same sets of 400 queries as used in Section 3.8.2 to do experiments here. There are three issues for comparison to demonstrate in this section, namely, (1) a scatter plot between actual and estimated query result sizes, (2) the three error measures as defined above and (3) a total number of tuples accessed by each sampling method. The aim of a scatter plot is that the closer to the diagonal line the points, the better the method. A total number of tuples accessed is defined by the sum of the sample sizes used by a set of 400 queries, i.e., $\sum_{i=1}^{|\mathcal{Q}|} n_i$, where n_i is the sample size of a query. In the graphs and tables following, we denote double systematic sampling by DSYS and feedback systematic sampling by FSYS.

With regard to the small sample size n_1 required by DSYS to estimate the actual selectivity $\overline{Y}$ of a query in the first sampling, we used $n_1 = 300$ tuples for all experiments. Table 3.25 shows all sampling parameters used in all experiments both for SS and DSYS. (FSYS does not require any of the parameters in the table. It simply requires $\bar{n}$.)

For each relation, we created 4 indices on the 4 most frequently used attributes of the relation. The reason in choosing the top most frequently used attributes to create an index on is that normally an attribute on which the DBA would consider to create a physical index or which the DBA would consider to sort[3], is usually

[3]In SQL, queries may contain **ORDER BY** *attribute_name* which calls for the sort on the attribute. Sorting is normally expensive and time-consuming and if this sorting must be done very

accessed/appears most frequently in user queries. In addition, the reason we chose 4 is just to ensure that we perform sufficient experiments (i.e., enough variety to the different attributes of the same relation) for one relation since a physical index or a sorted attribute could possibly be created on any one of the attributes of the relation. Figure 3.17 is the procedure we used for comparison among the three sampling methods: SS, DSYS and FSYS.

```
**for** each attribute $a$ in the 4 most frequently used attributes of $R$ **do**
  attach an index to attribute $a$ of $R$
  run 400 queries to find the actual result sizes $\overline{Y}_iN$ of the queries, $i = 1, 2, \ldots, 400$
  with the same index on attribute $a$
    run 400 queries via SS to find the estimated query result sizes $\widehat{Y_iN}$ of the queries, $i = 1, 2, \ldots, 400$
    run 400 queries via DSYS to find the estimated query result sizes $\widehat{\widehat{Y}}_iN$ of the queries, $i = 1, 2, \ldots, 400$
    calculate average sample size $\bar{n}$ by formula (3.38)
    run 400 queries via FSYS using a slightly smaller value than $\bar{n}$ as the sample size for all the queries
  compare the results obtained using the three comparison issues
**endfor**
```

Figure 3.17: Procedure to compare three sampling methods

α=0.05, prob. of out-of-bound error
$\psi = 0.1$, sanity bound
$\epsilon = 0.1$, relative estimation error
$\beta_{max} = 0.1$, maximum sampling fraction

Table 3.25: Sampling parameters

Since we have 12 different configuration relations (4 for 10k, 4 for 50k and 4 for 100k) and each relation has 4 different indices on it, the total number of experiments performed is 48.

Since the scatter plots between actual and estimated query result sizes for all 48 experiments give the same "look and feel", we show only the ones for R1-100k, R2-100k, R3-100k, R4-100k. The plots are shown in Figures 3.18, 3.19, 3.20 and 3.21.

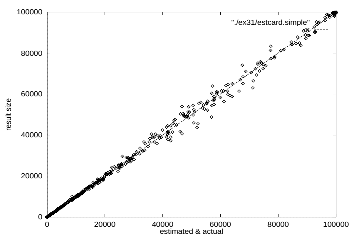
The three different error measures are shown in Tables 3.26, 3.27, 3.28, 3.29, 3.30, 3.31, 3.32, 3.33, 3.34, 3.35, 3.36 and 3.37.

The total number of tuples accessed by each sampling method is shown in Tables 3.38, 3.39, 3.40 and 3.41.

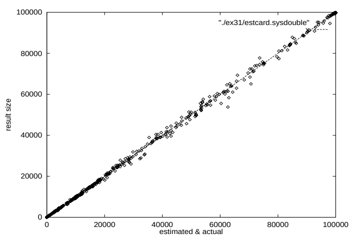
3.9 Conclusion

The main novel achievement in this chapter is that compared with SRSWOR and SRSWR, with regard to joins, SYSSMP using sorted data proves to provide a lower

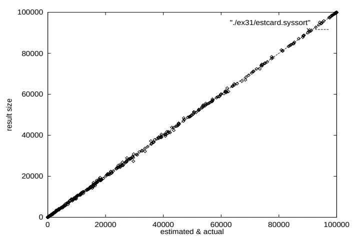
often, then DBA should consider to sort the relation on this attribute in advance so as to improve the speed of query processing.



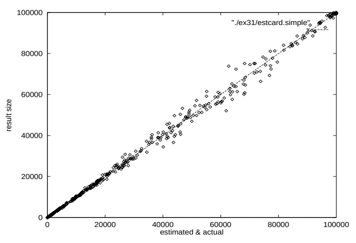
(a) SS, attr#1



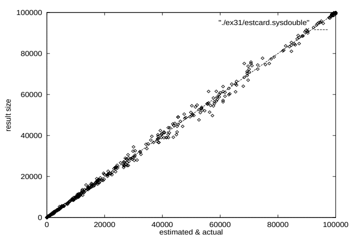
(b) DSYS, attr#1



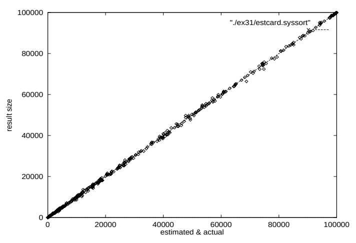
(c) FSYS, attr#1



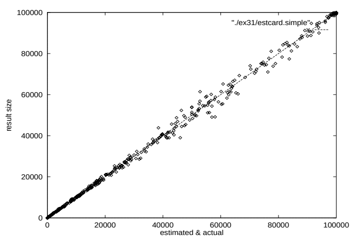
(d) SS, attr#2



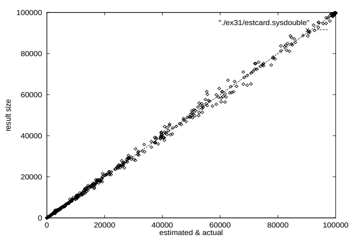
(e) DSYS, attr#2



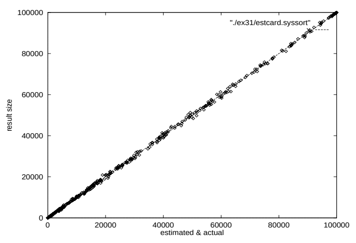
(f) FSYS, attr#2



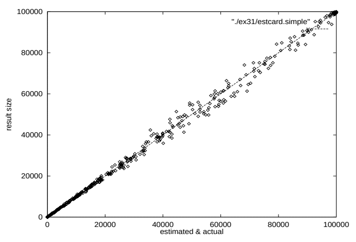
(g) SS, attr#3



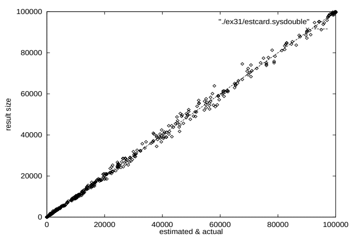
(h) DSYS, attr#3



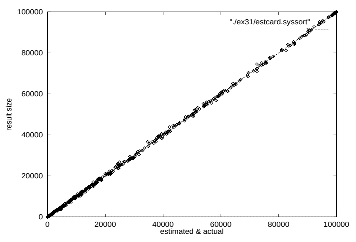
(i) FSYS, attr#3



(j) SS, attr#4



(k) DSYS, attr#4



(l) FSYS, attr#4

Figure 3.18: R1-100k, estimated & actual

R1-10k

attr#	SS	DSYS	FSYS
1	183	160	130
2	193	153	121
3	187	128	81
4	188	117	88

(a) Root mean square error

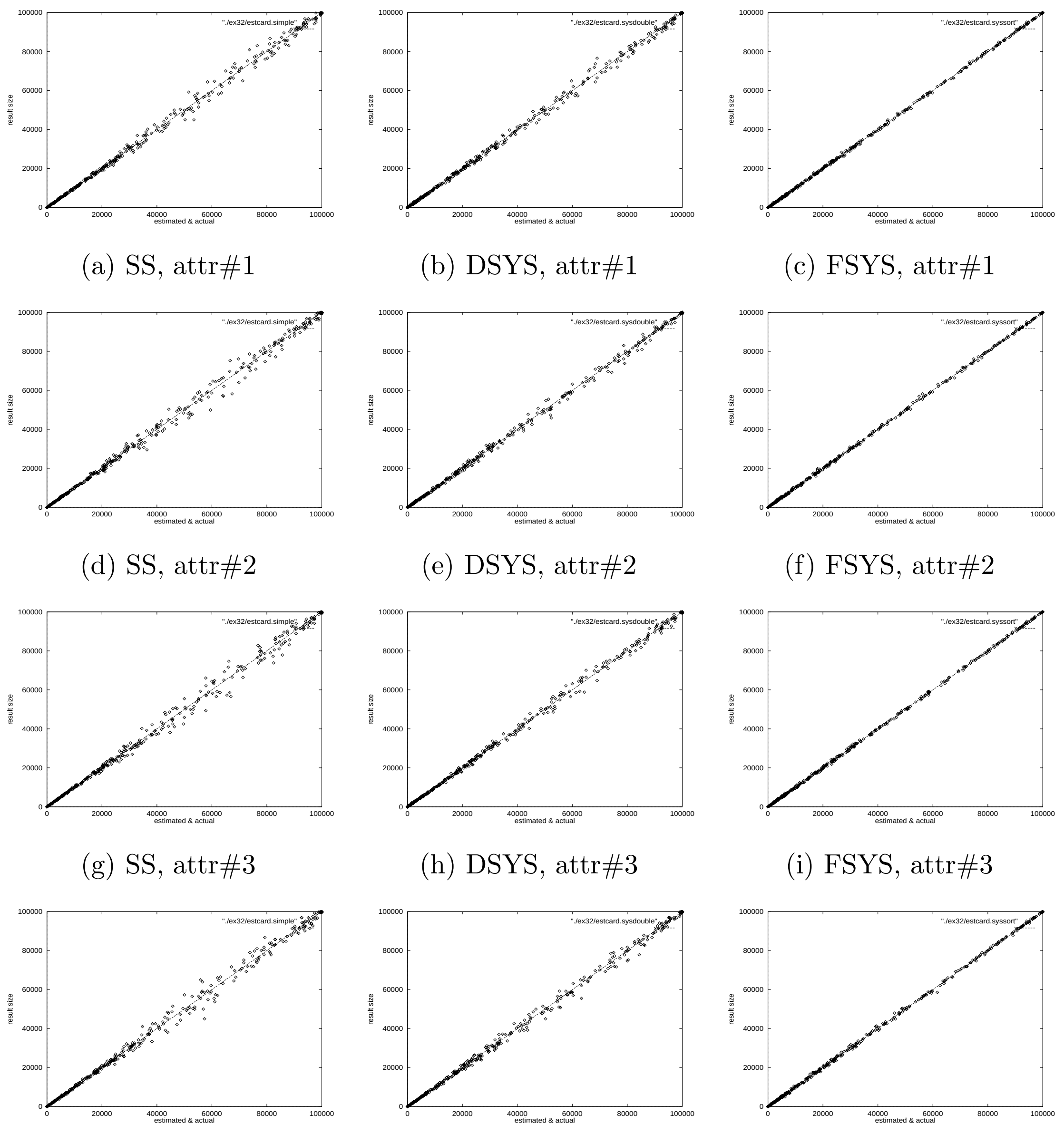
attr#	SS	DSYS	FSYS
1	111	104	83
2	114	95	80
3	113	76	47
4	114	73	53

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	22.69	17.07	16.88
2	13.39	14.28	17.47
3	15.26	11.14	15.07
4	17.21	15.07	12.00

(c) Mean relative error

Table 3.26: Estimation errors



(a) SS, attr#1 (b) DSYS, attr#1 (c) FSYS, attr#1

(d) SS, attr#2 (e) DSYS, attr#2 (f) FSYS, attr#2

(g) SS, attr#3 (h) DSYS, attr#3 (i) FSYS, attr#3

(j) SS, attr#4 (k) DSYS, attr#4 (l) FSYS, attr#4

Figure 3.19: R2-100k, estimated & actual

R2-10k

attr#	SS	DSYS	FSYS
1	206	143	114
2	202	101	75
3	182	124	96
4	192	123	78

(a) Root mean square error

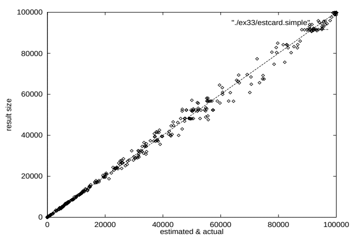
attr#	SS	DSYS	FSYS
1	131	96	78
2	125	69	48
3	116	85	59
4	122	80	49

(b) Mean residual error

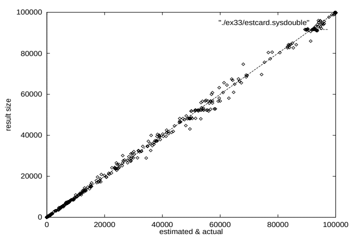
attr#	SS	DSYS	FSYS
1	11.71	8.56	11.77
2	10.53	6.31	9.38
3	12.34	8.39	9.06
4	9.81	9.38	8.50

(c) Mean relative error

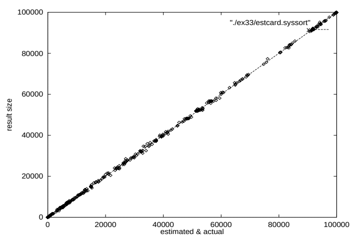
Table 3.27: Estimation errors



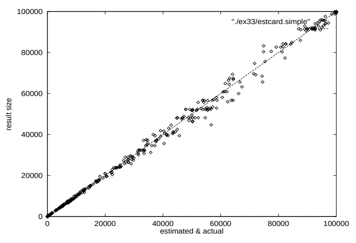
(a) SS, attr#1



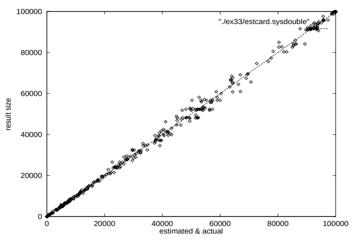
(b) DSYS, attr#1



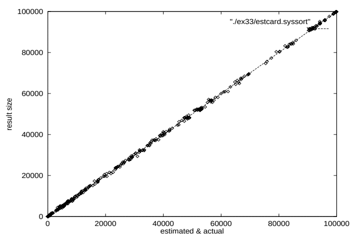
(c) FSYS, attr#1



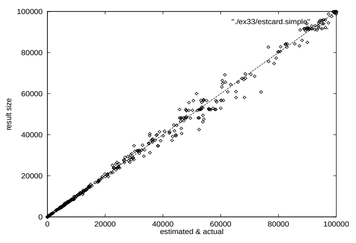
(d) SS, attr#2



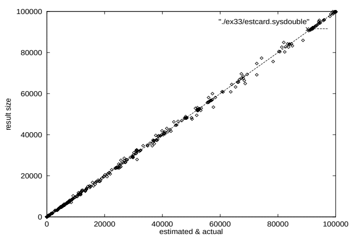
(e) DSYS, attr#2



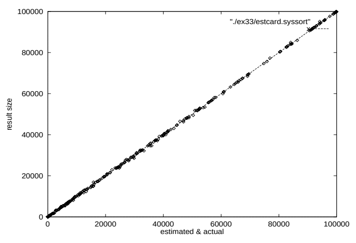
(f) FSYS, attr#2



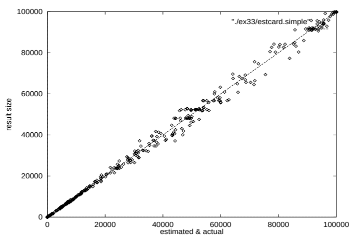
(g) SS, attr#3



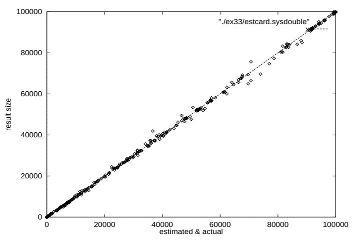
(h) DSYS, attr#3



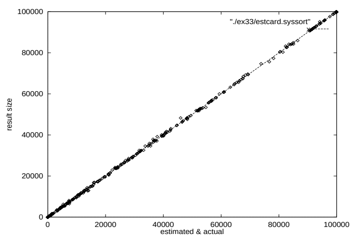
(i) FSYS, attr#3



(j) SS, attr#4



(k) DSYS, attr#4



(l) FSYS, attr#4

Figure 3.20: R3-100k, estimated & actual

R3-10k

attr#	SS	DSYS	FSYS
1	215	141	123
2	207	132	103
3	210	164	135
4	216	126	94

(a) Root mean square error

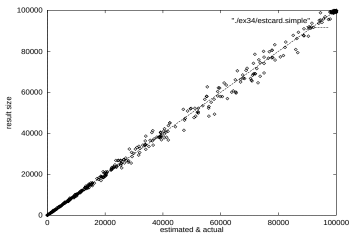
attr#	SS	DSYS	FSYS
1	133	103	88
2	134	88	69
3	132	112	92
4	137	88	60

(b) Mean residual error

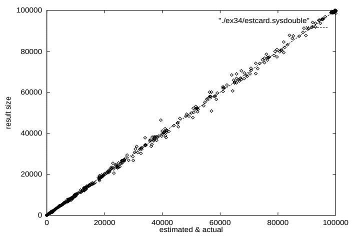
attr#	SS	DSYS	FSYS
1	7.91	9.61	12.25
2	9.30	6.35	6.55
3	7.78	11.45	9.81
4	9.60	6.69	7.18

(c) Mean relative error

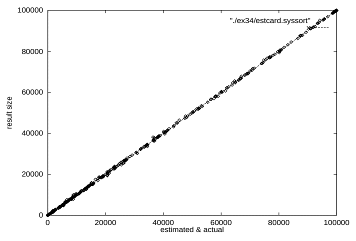
Table 3.28: Estimation errors



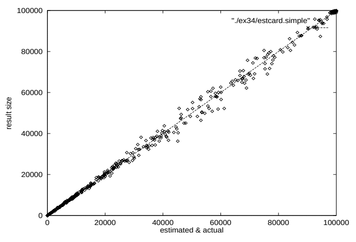
(a) SS, attr#1



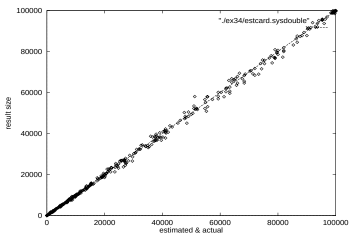
(b) DSYS, attr#1



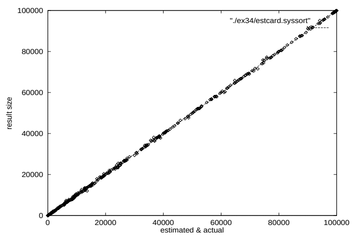
(c) FSYS, attr#1



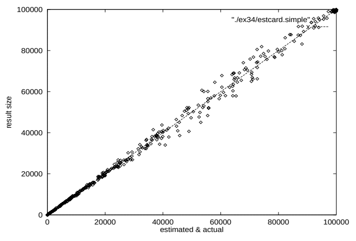
(d) SS, attr#2



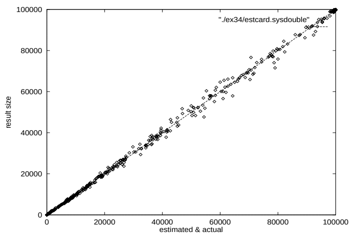
(e) DSYS, attr#2



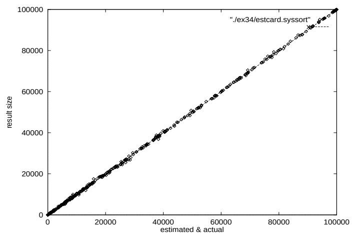
(f) FSYS, attr#2



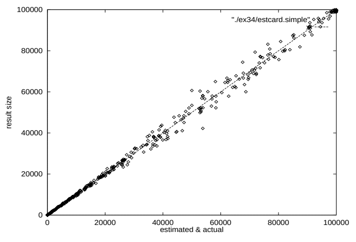
(g) SS, attr#3



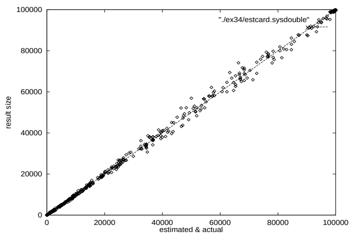
(h) DSYS, attr#3



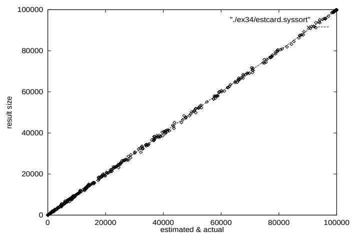
(i) FSYS, attr#3



(j) SS, attr#4



(k) DSYS, attr#4



(l) FSYS, attr#4

Figure 3.21: R4-100k, estimated & actual

R4-10k

attr#	SS	DSYS	FSYS
1	182	135	101
2	182	128	98
3	208	153	124
4	195	143	95

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	113	91	71
2	117	88	68
3	132	107	87
4	125	96	64

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	8.73	8.28	11.16
2	8.60	6.63	7.94
3	9.66	9.16	10.41
4	9.28	7.21	9.46

(c) Mean relative error

Table 3.29: Estimation errors

R1-50k

attr#	SS	DSYS	FSYS
1	962	682	286
2	903	569	220
3	960	555	234
4	902	608	267

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	511	370	172
2	469	327	137
3	491	313	151
4	471	350	169

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	9.11	7.79	6.99
2	8.97	7.02	5.61
3	8.01	9.56	6.75
4	6.55	6.99	8.95

(c) Mean relative error

Table 3.30: Estimation errors

R2-50k

attr#	SS	DSYS	FSYS
1	1007	613	238
2	915	620	277
3	1004	626	227
4	896	640	230

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	584	395	158
2	526	395	172
3	589	378	149
4	523	396	147

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	5.35	4.81	4.10
2	5.01	4.49	3.80
3	5.46	4.92	4.27
4	5.51	4.62	4.48

(c) Mean relative error

Table 3.31: Estimation errors

R3-50k

attr#	SS	DSYS	FSYS
1	1103	720	347
2	1021	759	379
3	972	758	381
4	1022	435	193

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	638	455	237
2	577	469	256
3	541	459	258
4	576	252	109

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	8.14	8.26	9.49
2	8.16	7.30	7.90
3	8.18	8.37	9.63
4	7.64	5.69	5.58

(c) Mean relative error

Table 3.32: Estimation errors

R4-50k

attr#	SS	DSYS	FSYS
1	968	693	331
2	968	530	262
3	1022	523	244
4	979	675	392

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	592	450	234
2	580	302	143
3	611	309	137
4	606	420	273

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	5.28	5.39	6.08
2	4.89	4.20	3.71
3	5.11	4.50	3.15
4	5.38	5.36	8.08

(c) Mean relative error

Table 3.33: Estimation errors

R1-100k

attr#	SS	DSYS	FSYS
1	1750	1178	397
2	2001	1466	512
3	1677	1282	572
4	1871	1354	516

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	990	662	240
2	1109	892	347
3	941	805	394
4	1098	854	357

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	5.19	4.74	3.44
2	5.71	6.27	4.79
3	5.35	5.53	4.25
4	5.63	6.12	5.24

(c) Mean relative error

Table 3.34: Estimation errors

R2-100k

attr#	SS	DSYS	FSYS
1	1652	1286	393
2	1858	1173	426
3	2091	1193	385
4	1948	1406	525

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	958	785	268
2	1068	695	264
3	1188	732	268
4	1104	866	345

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	4.98	5.18	3.59
2	5.18	5.53	3.90
3	4.74	5.43	3.59
4	4.87	5.65	4.25

(c) Mean relative error

Table 3.35: Estimation errors

R3-100k

attr#	SS	DSYS	FSYS
1	1969	1402	407
2	1938	1482	444
3	2195	776	274
4	1924	795	317

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	1103	808	259
2	1074	858	298
3	1182	414	140
4	1117	392	151

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	5.28	9.84	5.14
2	5.69	7.88	5.35
3	6.56	5.18	4.06
4	6.77	4.91	3.37

(c) Mean relative error

Table 3.36: Estimation errors

R4-100k

attr#	SS	DSYS	FSYS
1	1857	1067	324
2	1896	1154	355
3	2027	1370	369
4	1982	1455	461

(a) Root mean square error

attr#	SS	DSYS	FSYS
1	1087	617	203
2	1093	705	218
3	1144	760	227
4	1120	883	321

(b) Mean residual error

attr#	SS	DSYS	FSYS
1	4.94	4.63	3.93
2	5.03	5.61	3.56
3	5.36	4.85	3.94
4	5.33	6.71	5.26

(c) Mean relative error

Table 3.37: Estimation errors

attr#	SS	DSYS	FSYS
1	273954	301827	250000
2	273158	301198	250000
3	273547	301522	250000
4	273891	301166	250000

(a) R1-10k

attr#	SS	DSYS	FSYS
1	260768	289797	250000
2	259654	289905	250000
3	259411	290003	250000
4	260233	289722	250000

(b) R2-10k

attr#	SS	DSYS	FSYS
1	251911	282153	250000
2	252210	282335	250000
3	253288	281093	250000
4	252331	281720	250000

(c) R3-10k

Table 3.38: Total numbers of tuples accessed

attr#	SS	DSYS	FSYS
1	258934	288738	250000
2	260286	286403	250000
3	259188	287811	250000
4	259871	286599	250000

(a) R4-10k

attr#	SS	DSYS	FSYS
1	1149769	831232	1000000
2	1149523	829678	1000000
3	1148202	831731	1000000
4	1147160	839036	1000000

(b) R1-50k

attr#	SS	DSYS	FSYS
1	877373	631867	800000
2	877989	630832	800000
3	875147	632065	800000
4	881242	630768	800000

(c) R2-50k

Table 3.39: Total numbers of tuples accessed

attr#	SS	DSYS	FSYS
1	988038	714880	800000
2	985112	725092	800000
3	984954	725387	800000
4	987376	717957	800000

(a) R3-50k

attr#	SS	DSYS	FSYS
1	892626	647554	800000
2	892142	644204	800000
3	891579	637310	800000
4	890797	645282	800000

(b) R4-50k

attr#	SS	DSYS	FSYS
1	1681264	783015	1600000
2	1688559	781884	1600000
3	1689155	772681	1600000
4	1686694	790580	1600000

(c) R1-100k

Table 3.40: Total numbers of tuples accessed

attr#	SS	DSYS	FSYS
1	1647615	809140	1600000
2	1640351	803205	1600000
3	1639484	824567	1600000
4	1647529	829081	1600000

(a) R2-100k

attr#	SS	DSYS	FSYS
1	1816384	839129	1600000
2	1811852	832546	1600000
3	1811673	825897	1600000
4	1814197	852198	1600000

(b) R3-100k

attr#	SS	DSYS	FSYS
1	1635813	768138	1600000
2	1635073	763627	1600000
3	1634745	764981	1600000
4	1634976	765399	1600000

(c) R4-100k

Table 3.41: Total numbers of tuples accessed

total variance of estimated selectivities for all distinct values in a common join attribute domain. With regard to selections, SYSSMP using sorted data proves to provide a lower variance of an estimated selectivity of a complex predicate. Thus, the quality of a sample relation yielded by SYSSMP for both join and selection queries would be higher than that by SRSWOR and SRSWR.

For joins, the more we have sample relations of high quality, the more accurate the estimated join selectivity we will attain. This has been verified by the experiments done for SYSSMP and the hybrid sampling scheme between SYSSMP and SS. SYSSMP for join queries uses the same amount of sampling as SS but still can provide more accurate query result size estimates.

For selections, the double and feedback systematic sampling algorithms proposed also demonstrate that SYSSMP uses a less amount of sampling than SS while still giving higher accuracy of result sizes of selection queries.

Appendix

A How many tuples needed by SRSWR

With simple random sampling with replacement, the variance $V(\hat{\mu})$ of an estimated selectivity $\hat{\mu}$ of a sample $R'_1 \bowtie R'_2 \bowtie \ldots \bowtie R'_m$ with size $\tilde{n}$ is defined as:

$$V(\hat{\mu}) = E(\hat{\mu} - \mu)^2 = \frac{\tilde{N} - 1}{\tilde{N}} \frac{\tilde{S}^2}{\tilde{n}} \tag{3.45}$$

where $\tilde{N} = \prod_{i=1}^{m} |R_i|$ and $\tilde{n} = \prod_{i=1}^{m} |R'_i|$. Using $\tilde{S}^2$ in (3.8), we get:

$$V(\hat{\mu}) \;=\; \frac{\mu(1-\mu)}{\tilde{n}} \tag{3.46}$$

The error bound B is defined in the same fashion as in (3.10), i.e.: $B = t\sqrt{V(\hat{\mu})}$. Substituting $V(\hat{\mu})$ in (3.45) to the error bound B given above and solving B for $\tilde{n}$, we obtain:

$$\tilde{n} = \frac{t^2(\mu)(1-\mu)}{B^2} \tag{3.47}$$

If the relative error ϵ is required instead of the absolute error B, then B in equation (3.47) can be replaced by $\epsilon\mu$. The number of tuples to be sampled from each relation participating in a star join is $\beta * |R_i|$, where β is calculated by (3.14) and $\tilde{n}$ in (3.14) is replaced by $\tilde{n}$ above in (3.47).

CHAPTER 4

Improving join selectivities by bootstrap method

"A new technique that involves powerful computer calculations is greatly enhancing the statistical analysis of problems in virtually all fields of science. The method, which is now surging into practical use after a decade of refinement, allows statisticians to determine more accurately the reliability of data analysis in subjects ranging from politics to medicine to particle physics."

Quoted from page 4 of [Simon and Bruce 1997].

In this chapter, we describe a new method called *bootstrap* which can be used to improve join selectivities obtained from the earlier Chapter 3. We will first give the motivation in Section 4.1 for the improvement before we begin on the major section of the chapter which is the introduction.

4.1 Motivation in improving join selectivities

Let μ be the selectivity of a star join and $\hat{\mu}$ be an estimated selectivity of the star join. A *bias* is defined by $|E(\hat{\mu}) - \mu|$, where $E(\hat{\mu})$, an expected value of the $\hat{\mu}$, is defined by:

$$E(\hat{\mu}) = \sum_{\hat{\mu}} \hat{\mu} p(\hat{\mu}) \tag{4.1}$$

where $p(\hat{\mu})$ is the probability associated with $\hat{\mu}$. We can also call $E(\hat{\mu})$ a weighted average or simply an *average* for short.

A desirable property of an estimator (here an estimated selectivity) is *unbiased.* An estimator $\hat{\mu}$ is unbiased if $|E(\hat{\mu}) - \mu| = 0$. This statement basically suggests that an obtained estimated join selectivity would be in the vicinity of the actual join selectivity μ, but perhaps not the actual join selectivity itself.

On the other hand, an estimator $\hat{\mu}$ is *biased* if $|E(\hat{\mu}) - \mu| \neq 0$. It is considered that such an estimator is not desirable.

By the bias definition $|E(\hat{\mu}) - \mu|$, any *one-time sampling* which creates a single sample will naturally produce a bias. That is, normally the average $E(\hat{\mu})$ of a single sample obtained will not be equal to the actual join selectivity μ. In other words, an estimated join selectivity of a sample would be very difficult to be equal to the actual join selectivity. This further suggests that any sampling methods proposed in the literature for selectivity estimation, including systematic sampling, will normally produce a bias because they all do only one-time sampling for an estimated selectivity.

The unbiasedness definition, $|E(\hat{\mu}) - \mu| = 0$, also implies that a collection of samples obtained by *resampling* – sampling is done in a number of times – will result in the convergence of $|E(\hat{\mu}) - \mu|$ to 0. Typically, we would prefer our samples obtained to satisfy the unbiasedness as this implies that the average (expected value) by the samples would be equal to the actual join selectivity.

The bootstrap method is known to reduce the bias $|E(\hat{\mu}) - \mu|$ [Smith and Belle 1984]. By the nature of the bootstrap sampling which entails resampling many times, the statement of bias reduction basically implies that the more samples taken (more resampling done), the smaller the difference between $E(\hat{\mu})$ and μ would be. Alternatively, the more samples taken, the better the convergence to zero of $|E(\hat{\mu}) - \mu|$ would be.

Assuming that (1) resampling has been done η (≥ 1) times, thus taking η samples from the population and that (2) each of the η samples is picked from the population with the same probability, the expected value $E(\hat{\mu})$ above in (4.1) would be reduced to: $E(\hat{\mu}) = \frac{\sum_{\hat{\mu}} \hat{\mu}}{\eta}$. This expected value is, in fact, the average value used by the bootstrap method as the estimator of μ.

In the join selectivity estimation problem, the knowledge of bias reduction by the bootstrap method suggests us that the more we increase the number of times in sampling for a star join selectivity, the more accurate to the actual join selectivity, the average of all estimated join selectivities for the star join we will be able to obtain.

4.2 Introduction

Loosely, the bootstrap method [Efron 1979] is the method which involves resampling sample data with replacement many and many times in order to produce an empirical estimate of the entire sampling distribution of a parameter of interest [Grichting 1995]. The technique has been successfully and widely applied to many fields of science and a number of its applications are mentioned in [Efron 1979]. In the literature of query size estimation, the method was proposed for the estimation of the number of distinct values in an attribute domain by Haas et al. [1995]. To our best knowledge, none of any previous studies has proposed this method for the join selectivity estimation problem.

Following from the previous Chapter 3, the formula for calculating a sample size $\tilde{n}$ (and thus a sampling fraction β with which all relations participating in a star join are sampled) calls for the substitution of an estimated join selectivity $\hat{\mu}$ into the formula. For ease in reference, let us reproduce the formula here again. Given a star join with m participating relations, a sampling fraction β is calculated by:

$$\beta = (\frac{\tilde{n}}{\prod_{i=1}^{m} |R_i|})^{\frac{1}{m}} \tag{4.2}$$

where $\tilde{n}$ is defined by:

$$\tilde{n} = \frac{t^2(\mu)(1-\mu)}{B^2} \tag{4.3}$$

(B is the desired error bound and t is the abscissa of the normal curve.) μ in (4.3) can be substituted by an estimated $\hat{\mu}$. The formula for β implies that the more accurate a value of $\hat{\mu}$, the more precise the values of $\tilde{n}$ and β we will gain. Moreover, such an accurate value $\hat{\mu}$ will also be in favor of the cost calculation for query execution plans in order to select the optimal plan in a search space.

Recall that it is possible to store all possible estimated star join selectivities in a database profile catalog. (It is, however, not possible to store all possible selection selectivities as a result of the exponential number of selections which can occur in queries.) Substitute an estimated stored value for the selectivity of a star join into formula (4.3) and as a consequence, apply the resulting sampling fraction calculated by formula (4.2) to all the participating relations; a new estimated selectivity for the star join will be calculated as the outcome.

The main aim in this chapter is that we are not just to obtain a new estimated selectivity of a star join, but we are also attempting to improve the new estimated selectivity by the bootstrap method. The general concept of the method is that given a star join, sampling would be done in a number of times, say η (≥ 1) times; in each time an estimated join selectivity $\hat{\mu}_i, i = 1, 2, \ldots, \eta$ will be produced; and the average $\hat{\mu}$ of all the obtained estimated join selectivities in η times will be used as a better selectivity estimate for the star join than the individual $\hat{\mu}_i$'s. That is, the better estimate is $\hat{\mu} = \frac{\sum_{i=1}^{\eta} \hat{\mu}_i}{\eta}$.

An obvious question anyone can see here is: How are we going to implement the method which apparently looks expensive due to the *η-time sampling* policy, as opposed to the *one-time sampling* policy used by all the sampling methods proposed earlier, including the systematic sampling proposed in this thesis ? The answer is the delay of resampling – we do not do the η-time sampling all at once. The query size estimator (as part of the query optimiser) will wait until a query which incorporates the star join comes to the database system and do only one-time sampling for the star join included in the query in order to obtain an estimated selectivity, i.e., one of those $\hat{\mu}_i$'s. In this fashion, the initial $\hat{\mu}$'s obtained for the star join will perhaps be less accurate but then when more times of resampling have been done, the $\hat{\mu}$'s will gain better and better values. This is as a result of the convergence $|E(\hat{\mu}) - \mu| \to 0$ of the average estimated join selectivity.

Database updates will not much affect the accuracy of the bootstrap method. The reason is that whenever any update takes place, especially one which substantially changes the contents of the database, the current average estimated join selectivities stored in the profile catalog will be reset back to the beginning again. That is, resampling will be restarted at $\eta = 0$ again.

In all experiments, we use only a small number of times, i.e., $\eta = 15$ times in resampling, before any update will take place to the database, to study the improvement of join selectivities by the bootstrap method. The reason we do not use higher numbers than $\eta = 15$ is that the database at that point may possibly undergo significant changes on it and resampling will then be reset back to start at $\eta = 0$ again. All the results from the experiments give the same impression that the average estimated selectivity $\hat{\mu}$ by the bootstrap method for $i = 1, 2, \ldots, \eta = 15$, is generally more accurate than the individual estimated selectivity $\hat{\mu}_i$ by the one-time sampling.

There is no other overhead cost incurred by the bootstrap method, except the already-known storage cost which is required to keep all possible estimated join selectivities in the database profile catalog. However, since we have already accepted such a cost in Chapter 3 in order to use systematic sampling (one-time sampling) regardless of whether or not we will introduce the bootstrap method here, we do not really make any other extra overhead cost in order to use the bootstrap method proposed here.

Using the bootstrap method, one can benefit from the following:

- The bootstrap method is well-known in reducing the bias of an estimator, therefore being able to improve the quality of a join selectivity over time.
- For retrieval-intensive databases[1], in each one-time sampling to obtain an estimated selectivity of a star join, one can use a small sampling fraction for all the relations participating in the star join, i.e., even smaller than the calculated sampling fraction by the formula (4.2). Thus at runtime, when the query optimiser consults for the selectivity of the star join, the sampling for it can be done more quickly than ever before. This is due to the smaller sample sizes used by those relations.

Here is the structure of presentation in this chapter. In Section 4.3, we describe the bootstrap method and apply it to the join selectivity estimation problem. Section 4.4 contains 4 sets of experimental results comparing between:

- one-time SRSWR and bootstrap SRSWR.

[1]A database to which updates do not occur as frequently as queries to the database.

- one-time SRSWOR and bootstrap SRSWOR.
- one-time SYSSMP and bootstrap SYSSMP.
- one-time HYBRID and bootstrap HYBRID.

Suppose that x is one of the sampling schemes: SRSWR, SRSWOR, SYSSMP and HYBRID (the combination between SS and SYSSMP). One-time sampling with x scheme is sampling via the x scheme which is done only one time to obtain an estimated join selectivity. Bootstrap sampling with x scheme is sampling via the x scheme which is done η times to obtain an average estimated join selectivity. The results obtained are convincing and in great favor of the bootstrap method for all the different sampling schemes studied.

Although in the original bootstrap work, sampling is done with replacement, there are also ongoing studies investigating other sampling schemes [Grichting 1995]. Here the last three schemes, namely, SRSWOR, SYSSMP and HYBRID, are our experimental work in support of the bootstrap method via other sampling schemes.

We summarise the work in Section 4.5.

4.3 Bootstrap method

Denote by $R_{123\ldots m}$ the output relation of a star join among m participating relations, namely, $R_i, i = 1, 2, \ldots, m$. μ, the selectivity of a star join, is calculated by: $\mu = \frac{|R_{123\ldots m}|}{|R_1||R_2|\cdots|R_m|}$.

R'_i is a sample relation taken from relation $R_i, i = 1, 2, \ldots, m$. $R'_{123\ldots m}$ is the output relation of a star join among m sample relations, $R'_i, i = 1, 2, \ldots, m$. $\hat{\mu}$, the estimated star join selectivity, as a result of the join among the m sample relations, is calculated by: $\hat{\mu} = \frac{|R'_{123\ldots m}|}{|R'_1||R'_2|\cdots|R'_m|}$.

Suppose we are interested in approximating a parameter of interest θ by using an estimator $\hat{\theta} = f(y_1, y_2, y_3, \ldots, y_{\tilde{n}})$. $(y_1, y_2, y_3, \ldots, y_{\tilde{n}})$ is a sample of size $\tilde{n}$ in the population. The parameter θ could be a population mean, a population proportion, a population median, an error rate in discriminant analysis and so on. The basic idea of the bootstrap method is as follows:

1. produce a sample with replacement from the population. This sample is called a *bootstrap sample*.

2. from the bootstrap sample obtained, calculate an estimator of θ.

3. repeat steps 1 and 2 η times so as to obtain η bootstrap samples and thus η estimators of θ. Let $\hat{\theta}_j$ be an estimator of θ by the jth resampling number, where $j = 1, 2, \ldots, \eta$. Then calculate the final average estimator $\hat{\theta}$ by:

$$\hat{\theta} = \frac{\sum_{j=1}^{\eta} \hat{\theta}_j}{\eta}$$

Using the bootstrap method above with no modification to calculate an average estimated selectivity $\hat{\mu}$ ($\hat{\theta}$) of a star join is probably too expensive. The reason is that given a star join whose $\hat{\mu}$ the query optimiser wants to know, the query size estimator has to do η-time sampling all at once for η selectivities, i.e., $\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_\eta$, as opposed to only one-time sampling (which all previous sampling methods proposed so far for selectivity estimation need to do), in order to return an average estimated join selectivity $\hat{\mu}$ to the optimiser.

We utilise the fact that a common star join will appear again and again in many user queries. Each time the common star join occurs in a query, the query size estimator will do only one-time sampling for the star join. By accumulation of sampling on many processed queries with the star join, this process is equivalent to resampling many times all at once.

Given a star join with m participating relations, let $\hat{\mu}_{old}$ be an average estimated selectivity of the star join stored in the profile catalog and let j be the current resampling number of the star join. Initially, when there is no sampling done yet for the star join, set $j = 1$ and set the initial value of $\hat{\mu}_{old}$ by SS (recall that initialising the initial value of an estimated join selectivity can be done through having SS run through the star join). Figure 4.1 shows our modified version of the original bootstrap method.

After the procedure in Figure 4.1 terminates, the old estimated selectivity $\hat{\mu}_{old}$ stored in the profile catalog will now be replaced by the new $\hat{\mu}$, instead of $\hat{\mu}_j$ since it is believed that overall, $\hat{\mu}$ is a better estimate than $\hat{\mu}_j$. Subsequently, when a new query comes to the database system which incorporates the star join, this current $\hat{\mu}$ will proceed to become $\hat{\mu}_{old}$ in order to calculate a new $\hat{\mu}$ for the query optimiser.

STEP 1. obtain each sample relation R'_i from $R_i, i = 1, 2, \ldots, m$ by a sampling fraction β:

$$\beta = \left(\frac{\tilde{n}}{\prod_{i=1}^{m} |R_i|}\right)^{\frac{1}{m}} \qquad \text{where } \tilde{n} = \frac{t^2(\hat{\mu}_{old})(1-\hat{\mu}_{old})}{B^2}$$

STEP 2. join together all the sample relations R'_i's obtained to produce an output relation and calculate an individual estimated selectivity $\hat{\mu}_j$ of the star join by:

$$\hat{\mu}_j = \frac{|R'_{123\ldots m}|}{|R'_1||R'_2|\cdots|R'_m|}$$

STEP 3. calculate the final average join selectivity estimator $\hat{\mu}$ by:

$$\hat{\mu} = \frac{(j-1)\hat{\mu}_{old} + \hat{\mu}_j}{j} \tag{4.4}$$

where $\hat{\mu}_{old}$ is, in fact, the average estimated selectivity which was returned to the query optimiser in the last time resampling $(j-1)$, where $j \geq 2$. Like $\hat{\mu}_{old}$, $\hat{\mu}$ is the current average estimated selectivity which is being returned to the optimiser. We can also write that:

$$\hat{\mu}_{old} = \frac{\hat{\mu}_1 + \hat{\mu}_2 + \ldots + \hat{\mu}_{(j-1)}}{(j-1)}$$

STEP 4. increment j by 1.

STEP 5. return $\hat{\mu}$ to the query optimiser.

Figure 4.1: Join selectivity estimation by the bootstrap method

4.4 Experimental results

The database configurations for all experiments here are reused from Table 3.11(d) in Chapter 3. See also the table near page 131. The table consists of 6 different database configurations each of which comprises 5 relations. All the relations are of 10k cardinality.

Like all experiments in Chapter 3, the sampling fraction $\beta = 10\%$ is used throughout all experiments in this chapter. That is, each time a resampling is done, it is done with the 10% sampling fraction. Resampling is done with $\eta = 15$ times in all experiments.

For each database configuration SJi $i = 1, 2, 3, 4, 5, 6$, we experiment with star joins among 2 relations (2rels), 3 relations (3rels), 4 relations (4rels) and 5 relations (5rels) in the database. With star joins among $j$rels from a database where $j =$

$2, 3, 4, 5$, the j relations are randomly picked from the database prior to proceeding to do experiments.

In the experiments for SYSSMP, all the j relations in $j$rels star joins are sorted, where $j = 2, 3, 4, 5$ (recall that this is the extreme case of SYSSMP). In the experiments for HYBRID (the combination between SS and SYSSMP), only 2 of all the relations in a star join are sorted on their join attributes, irrespective of 2rels, 3rels, 4rels or 5rels star joins considered. The 2 sorted relations are randomly picked from the database. Sample relations of the two are thus created via SYSSMP. The remaining relations in the star join are unsorted and hence, their sample relations are created via SS.

There are 4 pages of graphs, i.e., in Figures 4.3, 4.4, 4.5 and 4.6, respectively. Each page is a comparison between one-time sampling with sampling scheme x and bootstrap sampling with scheme x (η-time sampling), where x is one of the sampling schemes: SRSWR, SRSWOR, SYSSMP and HYBRID.

Let us consider one page, e.g., in Figure 4.3. The graphs 4.3(a), 4.3(b), 4.3(c) and 4.3(d) show the experiments using the database configuration SJ1 for star joins among 2rels, 3rels, 4rels and 5rels, respectively from left to right. Now consider one of the graphs, say graph 4.3(a). We reproduce this graph in Figure 4.2 because we want to magnify it (like using a magnifying glass) to make it easy to see and understand the description below.

The X-axis of the graph in Figure 4.2 labels the resampling number from 1,2,3,... till 15 to the rightmost of the axis. The Y-axis is for the estimated join selectivity. There are 3 lines in the graph; the solid line is for the join selectivity estimates produced by the one-time sampling via SRSWR scheme; the dashed line is for the join selectivity estimates produced by the bootstrap sampling via SRSWR; and the straight dotted line from left to right is for the actual join selectivity. The aim of the graph is "the closer a line by a method lies in respect to the line of the actual join selectivity, the better the estimation method".

In the line by the bootstrap sampling, each point $\hat{\mu}$ (an average estimated selectivity) is calculated by:

$$\hat{\mu} = \frac{(j-1)\hat{\mu}_{old} + \hat{\mu}_j}{j}$$

(reproduced from equation (4.4)) where $j = 1, 2, 3, \ldots, \eta$. That is, once a resampling

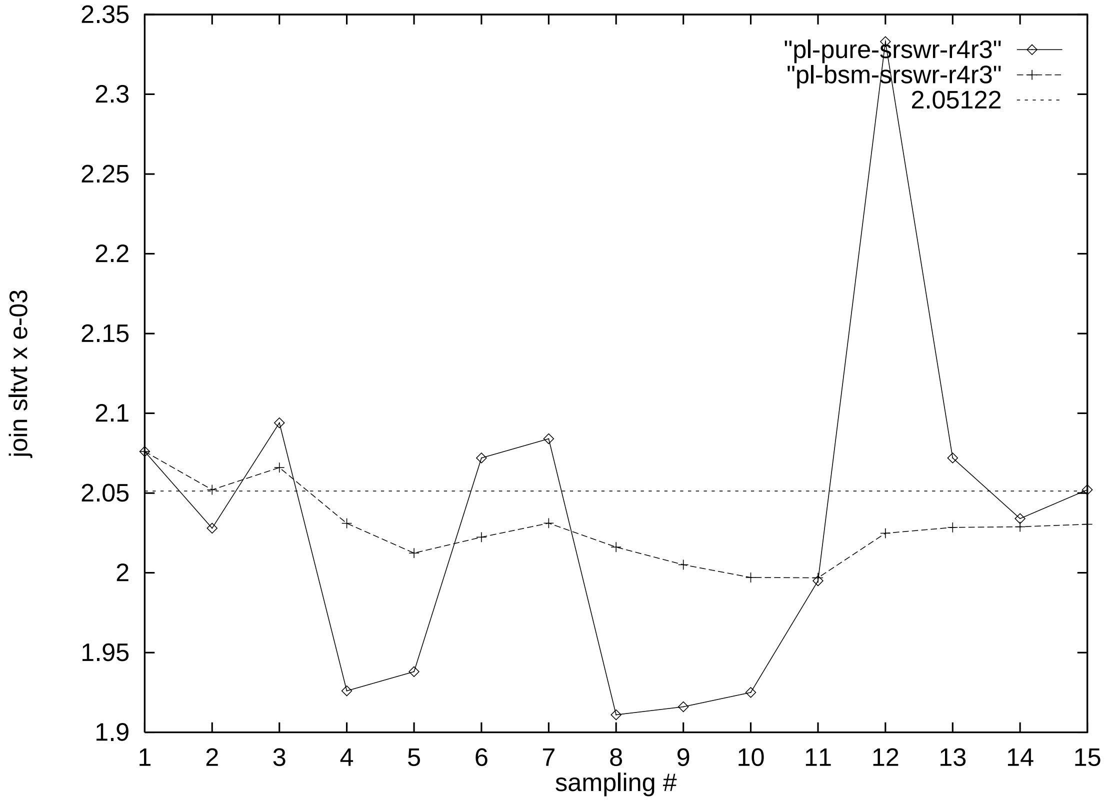


Figure 4.2: SJ1, 2rels, SRSWR

has been done, the current $\hat{\mu}$ will proceed to become $\hat{\mu}_{old}$ and the value of j will be incremented by 1.

Note also that for any graph when $j = 1$, the $\hat{\mu}$ for the first time resampling is equal to $\hat{\mu}_1$ (see in the graph 4.2 for an example).

For all graphs, one can also see that the fluctuation by the bootstrap sampling is clearly a lot lower than that by the one-time sampling. (The fluctuation represents the quality of the method.) Generally the line in a graph by the bootstrap sampling tends to be smoother than the line by the one-time sampling which tends to be spike-like.

The spike-likeness of a line can be ascribed to the fact that the one-time sampling will principally produce a high bias $|E(\hat{\mu}) - \mu|$ which is equal to $|\hat{\mu}_j - \mu|$ for any j from 1,2, ... till 15.

The claim that the bootstrap method can reduce the bias of an estimator is also fulfilled. Given an example in graph 4.2, as more sampling has been done from resampling number 1,2,3, ... till 15, the line by the bootstrap sampling tends to converge to the line of the actual join selectivity, whereas the line by the one-time sampling tends to converge nowhere.

As usual, apart from the graphs, three kinds of error are summarised in Table 4.1 for SRSWR, Table 4.2 for SRSWOR, Table 4.3 for SYSSMP and Table 4.4 for HYBRID. In the tables, we use the notations 1-x and b-x. The former represents one-time sampling with sampling scheme x and the latter represents bootstrap sampling with sampling scheme x.

However, we slightly changed the 3 error measures defined in Figure 3.6 (see the figure in page 133). The changes are made basically to leave out the sample size $\tilde{N}$ (see the original figure for comparison) because here we compare the join selectivity, not the join result size. Here are 3 slightly modified error measures:

Root Mean Square Error is defined by: $\sqrt{\sum_{i=1}^{\mathcal{Z}} \frac{(est\_val_i - \mu)^2}{\mathcal{Z}}}$ where $\mathcal{Z}$ is the number of resamplings which is 15 in all experiments, μ an actual join selectivity, est_val_i is either (1) an estimated join selectivity which results from the ith sampling, i.e., $\hat{\mu}_i$ by the one-time sampling or (2) an estimated join selectivity $\hat{\mu}$ by the bootstrap sampling.

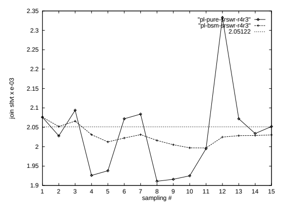
Mean Residual Error is defined by: $\frac{\sum_{i=1}^{\mathcal{Z}} abs(est\_val_i - \mu)}{\mathcal{Z}}$.

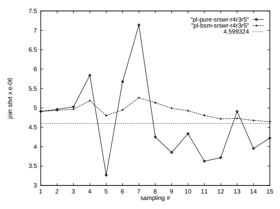
Mean Relative Error is defined by: $\frac{\sum_{i=1}^{\mathcal{Z}} 100 * \frac{abs(est\_val_i - \mu)}{\mu}}{\mathcal{Z}}$.

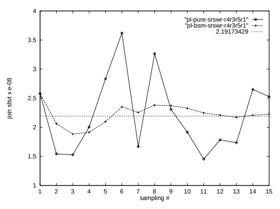
All the graphs and tables demonstrate in the same direction that the bootstrap sampling with any sampling scheme outperforms, in join selectivity estimation, the one-time sampling with the same sampling scheme.

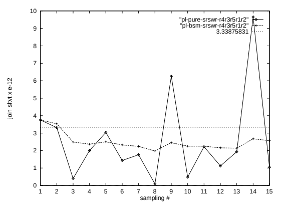
4.5 Conclusion

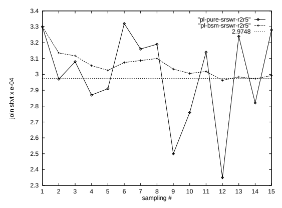
In this chapter we have achieved applying the bootstrap sampling method to the join selectivity estimation problem. The theoretical studies of the bootstrap method, e.g., by [Singh 1981; Bickel and Freedman 1981] show the accuracy of the bootstrap estimation in many problem domains. Our experimental study here is one in support of the effectiveness and efficiency of the bootstrap method to a database problem.

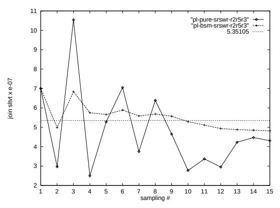

(a) SJ1, 2rels

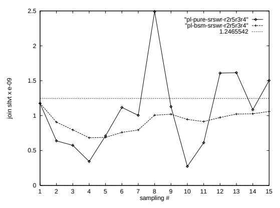

(b) SJ1, 3rels

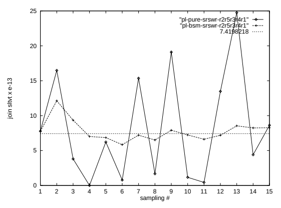

(c) SJ1, 4rels

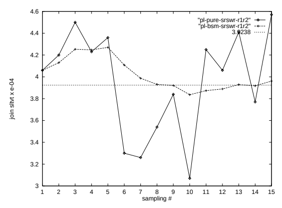

(d) SJ1, 5rels

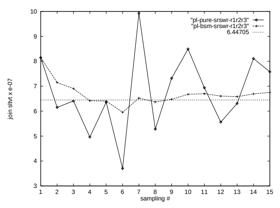

(e) SJ2, 2rels

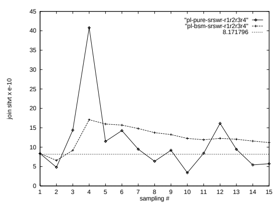

(f) SJ2, 3rels

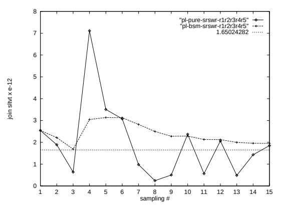

(g) SJ2, 4rels

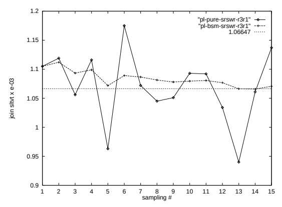

(h) SJ2, 5rels

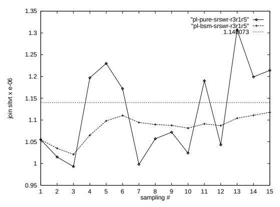

(i) SJ3, 2rels

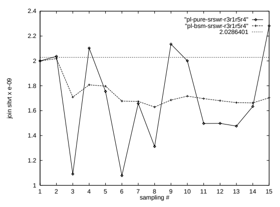

(j) SJ3, 3rels

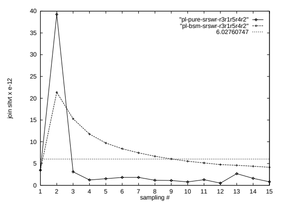

(k) SJ3, 4rels

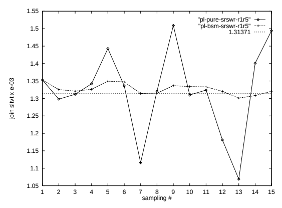

(l) SJ3, 5rels

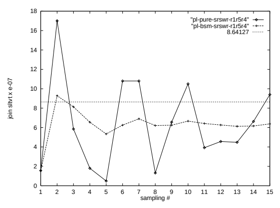

(m) SJ4, 2rels

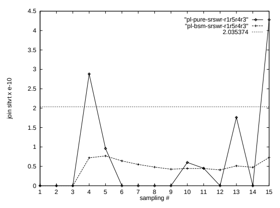

(n) SJ4, 3rels

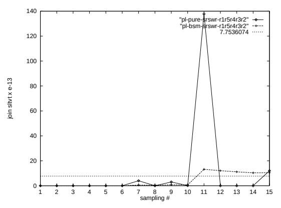

(o) SJ4, 4rels

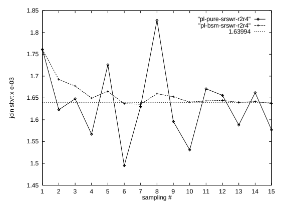

(p) SJ4, 5rels

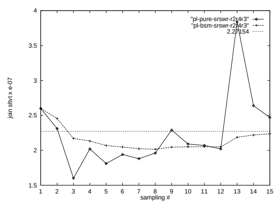

(q) SJ5, 2rels

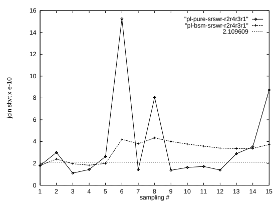

(r) SJ5, 3rels


(s) SJ5, 4rels


(t) SJ5, 5rels


(u) SJ6, 2rels


(v) SJ6, 3rels


(w) SJ6, 4rels

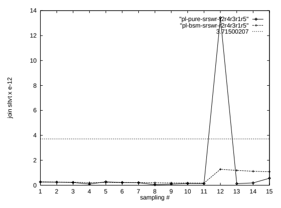
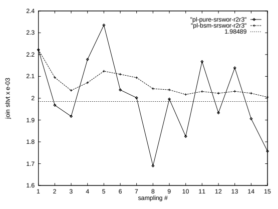
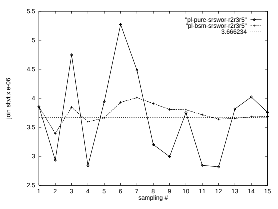

(x) SJ6, 5rels

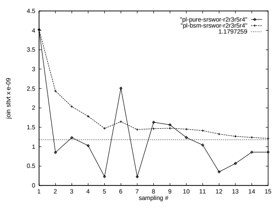
Figure 4.3: One-time SRSWR against bootstrap SRSWR



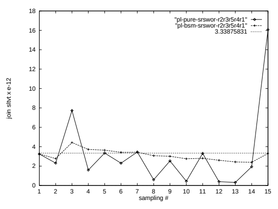
(a) SJ1, 2rels



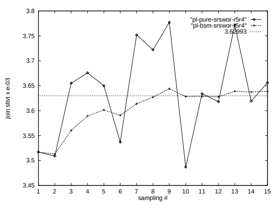
(b) SJ1, 3rels



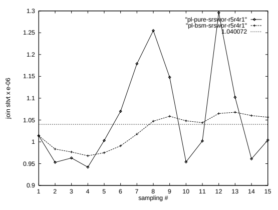
(c) SJ1, 4rels



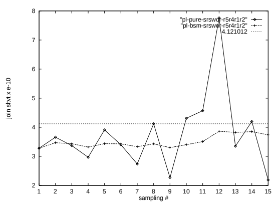
(d) SJ1, 5rels



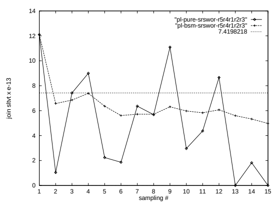
(e) SJ2, 2rels



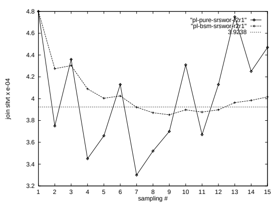
(f) SJ2, 3rels



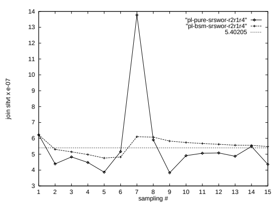
(g) SJ2, 4rels



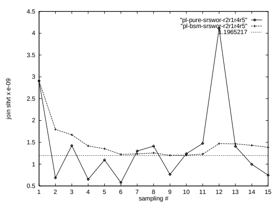
(h) SJ2, 5rels



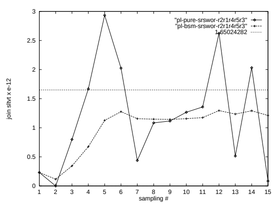
(i) SJ3, 2rels



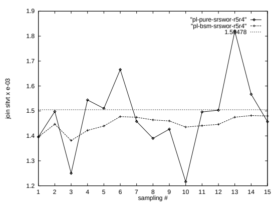
(j) SJ3, 3rels



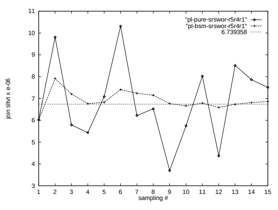
(k) SJ3, 4rels



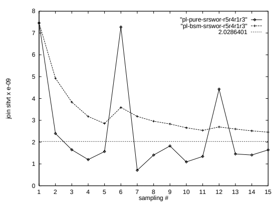
(l) SJ3, 5rels



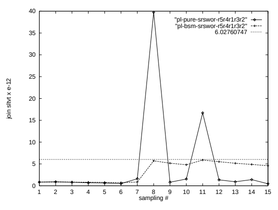
(m) SJ4, 2rels



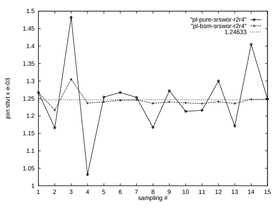
(n) SJ4, 3rels



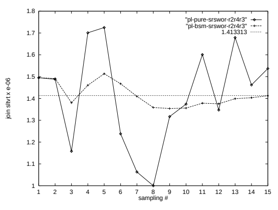
(o) SJ4, 4rels



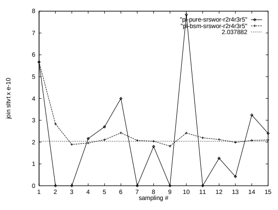
(p) SJ4, 5rels



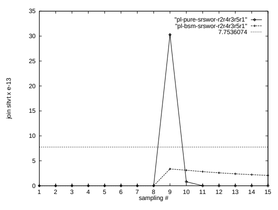
(q) SJ5, 2rels



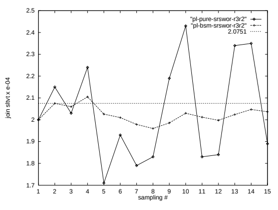
(r) SJ5, 3rels



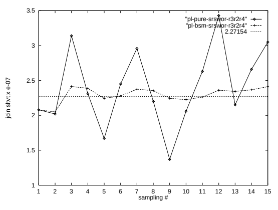
(s) SJ5, 4rels



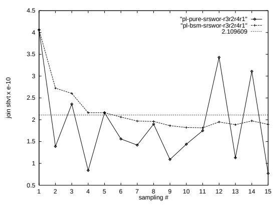
(t) SJ5, 5rels



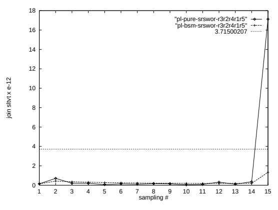
(u) SJ6, 2rels



(v) SJ6, 3rels

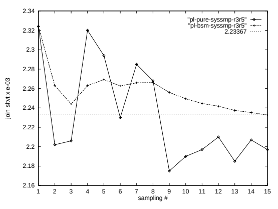


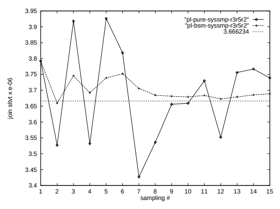
(w) SJ6, 4rels

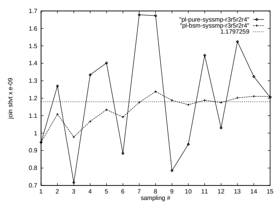


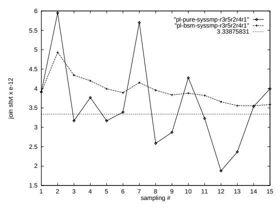
(x) SJ6, 5rels

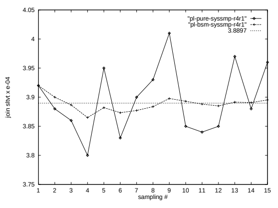
Figure 4.4: One-time SRSWOR against bootstrap SRSWOR

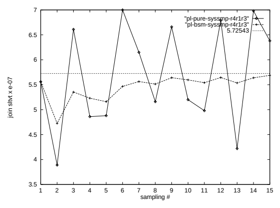

(a) SJ1, 2rels

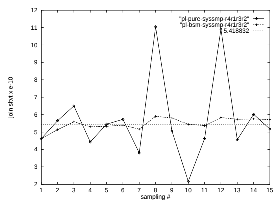

(b) SJ1, 3rels

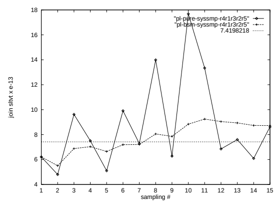

(c) SJ1, 4rels

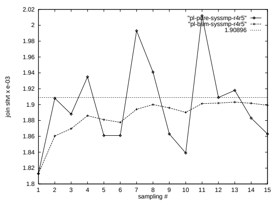

(d) SJ1, 5rels

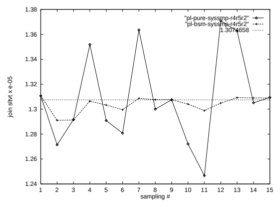

(e) SJ2, 2rels

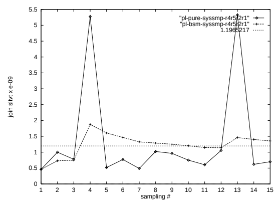

(f) SJ2, 3rels

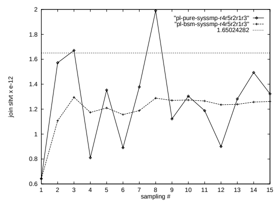

(g) SJ2, 4rels

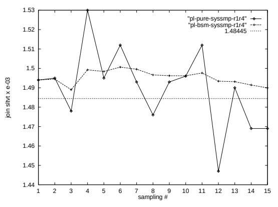

(h) SJ2, 5rels

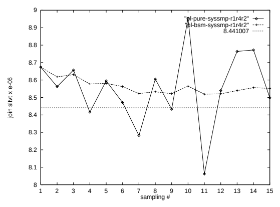

(i) SJ3, 2rels

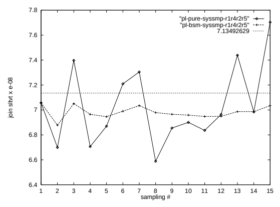

(j) SJ3, 3rels

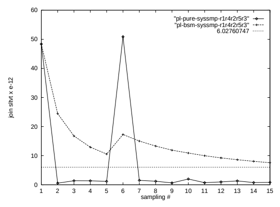

(k) SJ3, 4rels

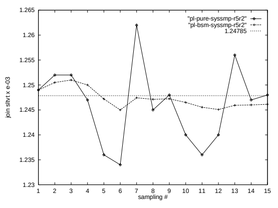

(l) SJ3, 5rels

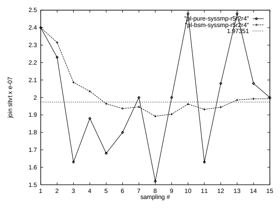

(m) SJ4, 2rels

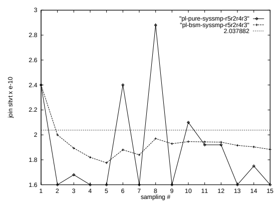

(n) SJ4, 3rels

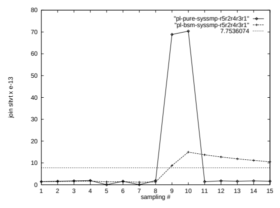

(o) SJ4, 4rels

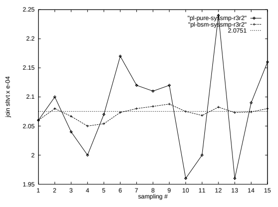

(p) SJ4, 5rels

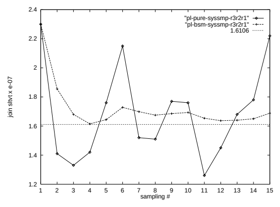

(q) SJ5, 2rels

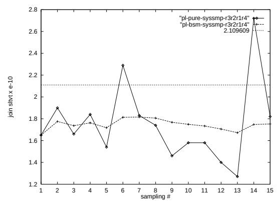

(r) SJ5, 3rels


(s) SJ5, 4rels


(t) SJ5, 5rels


(u) SJ6, 2rels


(v) SJ6, 3rels


(w) SJ6, 4rels

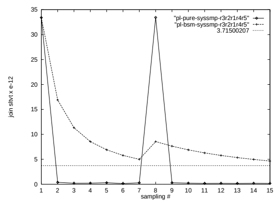
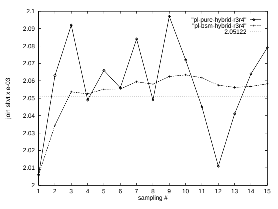
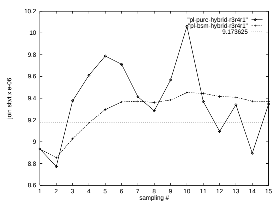
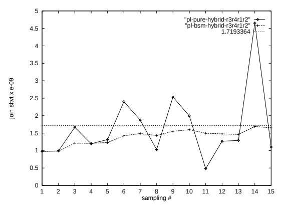
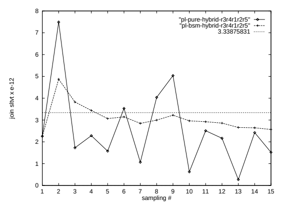

(x) SJ6, 5rels

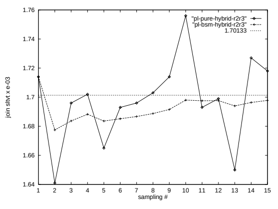
Figure 4.5: One-time SYSSMP against bootstrap SYSSMP

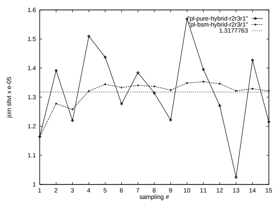

(a) SJ1, 2rels

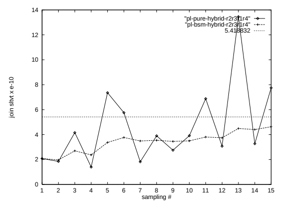

(b) SJ1, 3rels

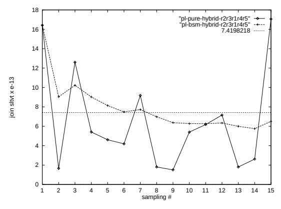

(c) SJ1, 4rels

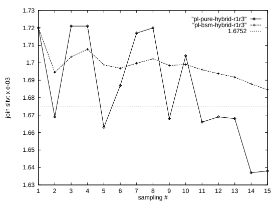

(d) SJ1, 5rels

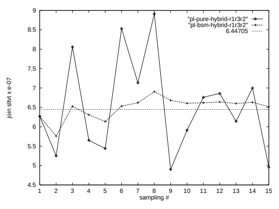

(e) SJ2, 2rels

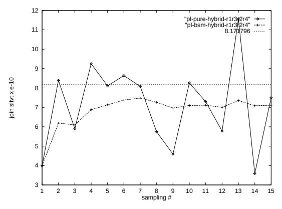

(f) SJ2, 3rels

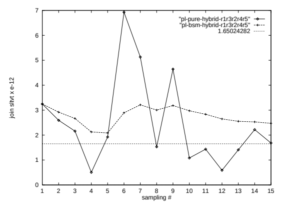

(g) SJ2, 4rels

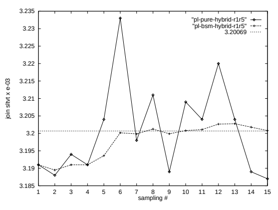

(h) SJ2, 5rels

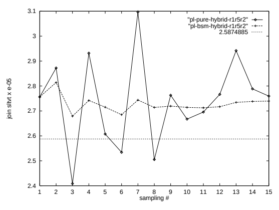

(i) SJ3, 2rels

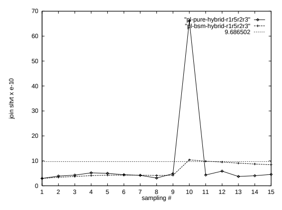

(j) SJ3, 3rels

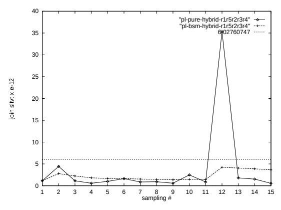

(k) SJ3, 4rels

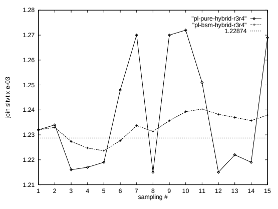

(l) SJ3, 5rels

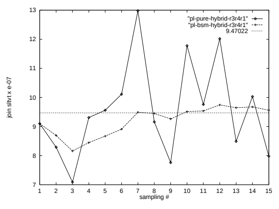

(m) SJ4, 2rels

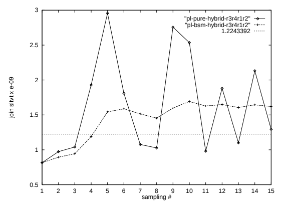

(n) SJ4, 3rels

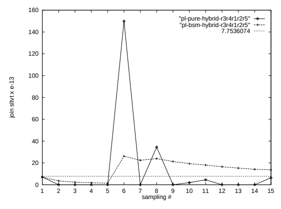

(o) SJ4, 4rels

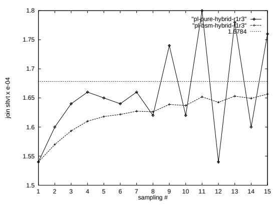

(p) SJ4, 5rels

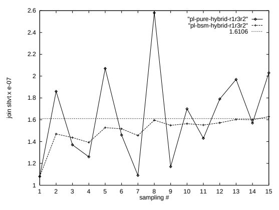

(q) SJ5, 2rels

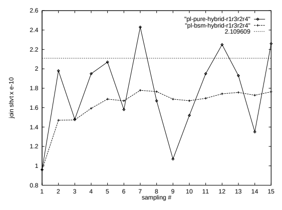

(r) SJ5, 3rels


(s) SJ5, 4rels


(t) SJ5, 5rels


(u) SJ6, 2rels


(v) SJ6, 3rels


(w) SJ6, 4rels

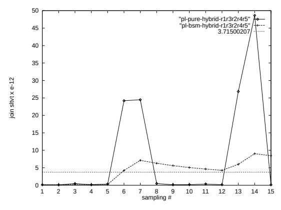

(x) SJ6, 5rels

Figure 4.6: One-time HYBRID against bootstrap HYBRID

	2rels		3rels		4rels		5rels	
exp.	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr
SJ1	1.065e-04	3.204e-05	9.818e-07	3.575e-07	6.461e-09	1.771e-09	2.548e-12	9.558e-13
SJ2	2.838e-05	1.171e-05	2.159e-07	6.738e-08	5.931e-10	3.565e-10	7.590e-13	1.495e-13
SJ3	4.638e-05	1.723e-05	1.555e-07	5.272e-08	9.196e-10	5.057e-10	1.718e-12	8.600e-13
SJ4	5.968e-05	2.180e-05	1.003e-07	6.294e-08	4.910e-10	3.089e-10	9.610e-12	5.120e-12
SJ5	1.197e-04	1.985e-05	4.984e-07	2.812e-07	1.804e-10	1.615e-10	3.427e-12	6.515e-13
SJ6	8.421e-05	3.662e-05	5.144e-08	2.000e-08	4.157e-10	1.380e-10	4.225e-12	3.277e-12

(a) Root mean square error

	2rels		3rels		4rels		5rels	
exp.	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr
SJ1	7.741e-05	2.871e-05	7.892e-07	3.099e-07	5.563e-09	1.380e-09	2.059e-12	9.084e-13
SJ2	2.370e-05	8.476e-06	1.812e-07	5.168e-08	4.855e-10	3.279e-10	6.301e-13	1.012e-13
SJ3	4.059e-05	1.214e-05	1.216e-07	3.283e-08	5.023e-10	4.429e-10	1.195e-12	7.359e-13
SJ4	4.617e-05	1.735e-05	9.280e-08	5.686e-08	3.838e-10	2.870e-10	6.318e-12	3.245e-12
SJ5	8.639e-05	1.578e-05	4.302e-07	2.428e-07	1.719e-10	1.596e-10	1.519e-12	6.227e-13
SJ6	6.566e-05	1.987e-05	3.716e-08	1.832e-08	2.287e-10	1.175e-10	3.931e-12	3.249e-12

(b) Mean residual error

	2rels		3rels		4rels		5rels	
exp.	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr	1-srswr	b-srswr
SJ1	3.774	1.400	17.158	6.738	25.382	6.296	61.657	27.209
SJ2	7.967	2.849	33.872	9.658	38.944	26.303	84.917	13.637
SJ3	10.345	3.095	18.858	5.092	61.472	54.204	72.400	44.592
SJ4	4.329	1.627	8.140	4.987	18.921	14.147	104.810	53.841
SJ5	6.576	1.202	49.789	28.095	84.437	78.400	195.952	80.308
SJ6	4.004	1.212	16.361	8.066	108.392	55.696	105.803	87.457

(c) Mean relative error

Table 4.1: One-time SRSWR against bootstrap SRSWR

	2rels		3rels		4rels		5rels	
exp.	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor
SJ1	1.741e-04	9.667e-05	7.290e-07	1.702e-07	9.457e-10	8.728e-10	3.854e-12	5.675e-13
SJ2	9.156e-05	4.897e-05	1.122e-07	3.897e-08	1.330e-10	6.436e-11	4.566e-13	1.907e-13
SJ3	4.669e-05	2.714e-05	2.307e-07	4.541e-08	9.311e-10	5.050e-10	9.794e-13	7.864e-13
SJ4	1.449e-04	6.388e-05	1.794e-06	4.500e-07	2.134e-09	1.830e-09	1.029e-11	3.624e-12
SJ5	1.005e-04	1.896e-05	2.191e-07	5.283e-08	2.262e-10	9.744e-11	9.441e-13	6.656e-13
SJ6	2.267e-05	6.374e-06	5.673e-08	1.103e-08	9.639e-11	5.683e-11	4.853e-12	3.416e-12

(a) Root mean square error

	2rels		3rels		4rels		5rels	
exp.	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor
SJ1	1.398e-04	7.937e-05	6.008e-07	1.307e-07	6.479e-10	5.325e-10	2.333e-12	4.623e-13
SJ2	7.448e-05	3.141e-05	9.161e-08	3.209e-08	9.618e-11	6.122e-11	3.936e-13	1.618e-13
SJ3	4.151e-05	1.581e-05	1.224e-07	3.899e-08	5.706e-10	2.866e-10	8.426e-13	6.779e-13
SJ4	1.027e-04	5.673e-05	1.470e-06	3.034e-07	1.364e-09	1.322e-09	7.355e-12	2.867e-12
SJ5	6.960e-05	1.217e-05	1.852e-07	4.451e-08	1.771e-10	4.096e-11	8.684e-13	6.516e-13
SJ6	2.050e-05	5.595e-06	4.539e-08	9.127e-09	8.252e-11	3.374e-11	4.176e-12	3.404e-12

(b) Mean residual error

	2rels		3rels		4rels		5rels	
exp.	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor	1-srswor	b-srswor
SJ1	7.044	3.999	16.386	3.566	54.918	45.135	69.867	13.848
SJ2	2.052	0.865	8.808	3.086	23.339	14.855	53.046	21.802
SJ3	10.579	4.030	22.651	7.218	47.685	23.956	51.060	41.081
SJ4	6.822	3.770	21.818	4.502	67.219	65.190	122.025	47.562
SJ5	5.584	0.977	13.102	3.149	86.913	20.102	112.005	84.038
SJ6	9.880	2.696	19.982	4.018	39.116	15.993	112.420	91.640

(c) Mean relative error

Table 4.2: One-time SRSWOR against bootstrap SRSWOR

	2rels		3rels		4rels		5rels	
exp.	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp
SJ1	4.935e-05	3.176e-05	1.465e-07	5.113e-08	3.030e-10	9.260e-11	1.102e-12	7.030e-13
SJ2	5.799e-06	1.238e-06	9.958e-08	3.632e-08	2.306e-10	3.359e-11	3.769e-13	1.171e-13
SJ3	5.386e-05	3.317e-05	3.627e-07	6.960e-08	1.566e-09	3.502e-10	5.181e-13	4.851e-13
SJ4	2.035e-05	1.141e-05	2.347e-07	1.315e-07	3.208e-09	1.598e-09	1.657e-11	1.335e-11
SJ5	7.780e-06	1.956e-06	3.002e-08	1.488e-08	4.128e-11	1.700e-11	2.337e-12	5.601e-13
SJ6	7.692e-06	1.093e-06	3.229e-08	1.983e-08	5.000e-11	3.655e-11	1.133e-11	9.009e-12

(a) Root mean square error

	2rels		3rels		4rels		5rels	
exp.	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp
SJ1	4.402e-05	2.346e-05	1.261e-07	3.742e-08	2.679e-10	6.239e-11	7.949e-13	6.125e-13
SJ2	4.931e-06	9.133e-07	9.030e-08	2.682e-08	1.486e-10	2.695e-11	2.549e-13	1.023e-13
SJ3	4.373e-05	2.394e-05	2.835e-07	4.668e-08	9.354e-10	2.688e-10	4.370e-13	4.595e-13
SJ4	1.658e-05	1.088e-05	1.874e-07	1.239e-07	2.847e-09	1.531e-09	1.008e-11	8.963e-12
SJ5	5.990e-06	1.753e-06	2.460e-08	8.660e-09	3.717e-11	1.500e-11	1.375e-12	5.341e-13
SJ6	6.299e-06	8.319e-07	2.606e-08	1.123e-08	4.630e-11	3.626e-11	6.990e-12	5.479e-12

(b) Mean residual error

	2rels		3rels		4rels		5rels	
exp.	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp	1-syssmp	b-syssmp
SJ1	1.971	1.050	3.438	1.021	22.705	5.289	23.810	18.346
SJ2	1.268	0.235	15.772	4.684	27.421	4.973	34.350	13.787
SJ3	2.291	1.254	2.168	0.357	78.175	22.467	26.480	27.844
SJ4	1.117	0.733	2.220	1.468	3.990	2.146	167.210	148.691
SJ5	0.480	0.140	12.463	4.388	18.238	7.363	177.362	68.880
SJ6	3.036	0.401	16.182	6.970	21.949	17.188	188.161	147.479

(c) Mean relative error

Table 4.3: One-time SYSSMP against bootstrap SYSSMP

	2rels		3rels		4rels		5rels	
exp.	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid
SJ1	2.633e-05	1.417e-05	3.925e-07	2.154e-07	9.703e-10	3.908e-10	1.945e-12	6.499e-13
SJ2	2.834e-05	1.261e-05	1.379e-06	4.751e-07	3.187e-10	2.151e-10	5.074e-13	2.651e-13
SJ3	3.075e-05	2.449e-05	1.215e-07	2.686e-08	2.342e-10	1.602e-10	1.931e-12	1.160e-12
SJ4	1.288e-05	5.590e-06	2.318e-06	1.442e-06	1.549e-09	4.512e-10	8.835e-12	3.887e-12
SJ5	2.410e-05	6.794e-06	1.600e-07	5.574e-08	7.986e-10	3.565e-10	3.789e-12	1.023e-12
SJ6	8.059e-06	6.443e-06	4.117e-08	1.694e-08	5.443e-11	5.208e-11	1.536e-11	3.093e-12

(a) Root mean square error

	2rels		3rels		4rels		5rels	
exp.	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid
SJ1	2.122e-05	9.779e-06	3.305e-07	2.028e-07	7.154e-10	3.268e-10	1.669e-12	5.356e-13
SJ2	2.016e-05	1.098e-05	1.151e-06	3.076e-07	2.672e-10	1.998e-10	4.324e-13	1.665e-13
SJ3	2.581e-05	2.310e-05	1.012e-07	2.184e-08	1.759e-10	1.363e-10	1.269e-12	1.106e-12
SJ4	1.058e-05	3.723e-06	1.941e-06	1.409e-06	8.790e-10	3.725e-10	6.275e-12	3.734e-12
SJ5	1.962e-05	5.986e-06	1.235e-07	3.962e-08	6.036e-10	3.415e-10	1.666e-12	9.055e-13
SJ6	7.059e-06	5.602e-06	3.454e-08	1.101e-08	4.278e-11	4.801e-11	9.846e-12	2.746e-12

(b) Mean residual error

	2rels		3rels		4rels		5rels	
exp.	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid	1-hybrid	b-hybrid
SJ1	1.035	0.477	3.603	2.211	41.608	19.008	49.993	16.043
SJ2	1.185	0.645	8.731	2.334	49.315	36.874	58.275	22.437
SJ3	1.541	1.379	15.694	3.388	21.529	16.682	76.879	67.015
SJ4	0.331	0.116	7.501	5.444	90.741	38.457	104.112	61.951
SJ5	1.597	0.487	13.045	4.184	49.298	27.894	214.928	116.786
SJ6	4.206	3.338	21.444	6.838	20.277	22.756	265.042	73.929

(c) Mean relative error

Table 4.4: One-time HYBRID against bootstrap HYBRID

CHAPTER 5

Query size estimation using local regression

Abstract

Local regression proves to provide very robust query size estimation due to its adaptive features of polynomials and windows used. The simple but effective idea of local regression is to select a number of local windows to work on and in each local window, fit the data points falling into the window by a polynomial with a low degree (2 or 3, normally). It is easy to see why such a small degree would normally suffice to handle a small curve in the local window (i.e., a partition of the whole graph). Hence, this is the main strength of why local regression works well in many real-life problem domains.

Many previous techniques such as uniform distribution based, histogram, curve-fitting and machine learning, are shown to be subsumed under local regression. In order to make use of different strengths of the previous techniques, we propose a data structure in order to gracefully and generically implement together all the previous techniques in a single framework.

To our best knowledge, neural networks are new in the setting of query size estimation. Query optimisation researchers may like to know how neural networks perform in this scenario. We show how to apply them to the query size estimation problem.

With a variety of relation configurations, a number of experiments using both simple and complex predicate queries have been conducted to see the efficiency of 7 estimation methods. Many of them are proposed earlier and the rest are proposed here.

Since the introduction section 5.1 below is very long, to make it easy to get all the points we want to make, we briefly describe them here:

• Notations and definitions define some notations and definitions to be used throughtout the chapter.

• Concept of local regression describe the basic underlying idea of local regression.

• Global regression and its problem Global regression is a specific case of local regression. Here we identify a main problem against global regression.

• Overview of local regression variants give an overview of many variants of local regression.

• Multi-dimensional regression models and their storage problem identify the storage requirement problem against multi-dimensional models. This prevents the use in practice of multi-dimensional models provided by local regression methods. Hence, only are single-dimensional regression models in use in practice for query size estimation.

• Background for neural networks give a background for neural networks. We will also use them for query size estimation.

• Background for a machine learning method M5 give a background for a machine learning method called M5. M5 is also a variant of local regression. We will modify the original M5 for query size estimation here.

• Query size estimation by local regression variants and neural networks describe the overall idea of how to apply local regression variants and neural networks to the query size estimation problem.

• Single implementation framework for local regression variants describe an implementation for all local regression variants that can be done under a single framework.

• Method evaluation evaluate those local regression variants and neural networks.

• Local regression with join selectivity is the the idea of how to use local regression to approximate join selectivities.

5.1 Introduction

For systems where sampling-based query size estimation methods are not appropriate, e.g., distributed database systems (either multidatabase systems or homogeneous distributed systems) or any system which wants an instant and quick way of query size estimation, we propose that a novel curve-fitting method called *local regression* be used for size estimation of selection queries.

After decades of refinement by many researchers, e.g., by Cleveland [1979]; Cleveland and Devlin [1988]; Cleveland and Grosse [1991]; Cleveland and Loader [1996]), local regression has now surged into practical use. It has been applied to many fields of science and many applications since late 19th century. Work by Cleveland [1979] is the one which makes local regression become very popular.

• Notations and definitions

Let us first define some notations and definitions that we will use throughout the chapter. Let R be a relation in a database of interest with N as its cardinality (the number of tuples). Suppose that relation R has u attributes, namely, $b_1, b_2, \ldots, b_u$. Let $f(x)$ be either a function of a *frequency distribution* of x – how many times the x value appears in relation R – or a function of a *cumulative frequency distribution* of x which does not include the frequency of x itself – how many times any value less than x appears in R. We will use $f(x)$ in one of the two contexts. Note that we slightly abuse the term "cumulative frequency distribution" which is supposed to include the frequency of x as well.

Let $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_d, f(x_d))$ be a data set. We call it "data set" because we want to generalise the term over the two contexts of $f(x)$. For all the *data points* $(x_j, f(x_j))$ $j = 1, 2, \ldots, d$ in the data set, $f(x)$ used must be either a frequency distribution function or a cumulative frequency distribution function but not both at the same time. Each x_j, $j = 1, 2, \ldots, d$ in the data set is a distinct value in the domain of an attribute, say $b_{\#}$, which is one of the attributes of R. d is the number of distinct values which appear in R under $b_{\#}$. Assume also that each x_j in the sequence defined in the data set is in ascending order.

• Concept of local regression

The underlying idea of local regression can be roughly described as follows:

- select a number of windows *numwin* to work on. ("A number of windows" is equivalent to "a number of buckets" used by histograms. In the literature of local regression, the former is used rather than the latter.)
- define upper bound values for the windows, i.e., upper boundaries for each window. One can also imagine similarly of upper bound values of histograms.
- partition all the data points in the data set into their corresponding windows based on x_i's values.
- locally fit the data points which fall into a local window by a polynomial with a low degree, normally 2 or 3. Repeat this step with the remaining windows.

Although the attribute domains considered here are all numerical which could be either discrete or continuous, the application to alphanumeric domains (string value domains) can also be done. Prior to applying the local regression methods proposed in this chapter, the only additional process involved and needed to be done is a conversion by a function from alphanumeric values to their corresponding floating point numbers. A database system that uses such a conversion is IBM's DB2-6000 system, for example [Poosala 1997].

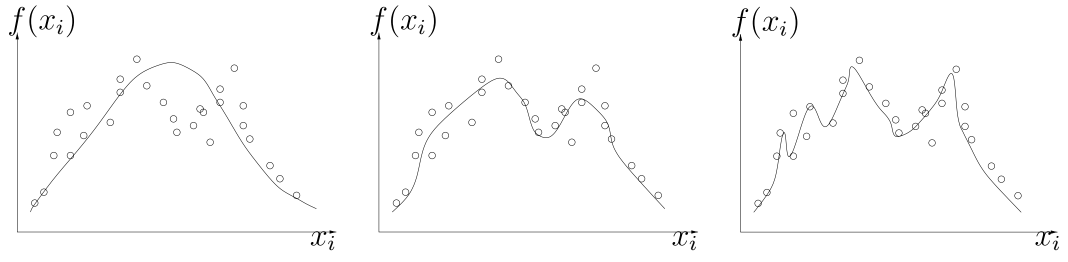
• Global regression and its problem

Two curve-fitting methods earlier proposed by Sun et al. [1993]; Chen and Roussopoulos [1994] are a special form of local regression proposed in this chapter. They are called *global regression.* The methods build only a single window (as opposed to multiple windows by local regression) which holds all the data points in the data set defined above in the single window and fit them all with a single polynomial with a high degree, namely, degree 10 for [Sun et al. 1993] and degree 6 for [Chen and Roussopoulos 1994].

The main drawback against the global curve-fitting methods is that many times they cannot fit data nicely, e.g., *multimodal* data (data with many curves), hence missing many points in the data set. This drawback is known as *oversmoothing* problem (see Figure 5.1(a) for a "conceptual" example). In the figure, we introduce a new term *bandwidth* which simply means a window width. The more windows used for fitting, the smaller the bandwidth for the windows.

The oversmoothing problem is mollified when more windows are used to fit the same data set. Figure 5.1(b) shows a local regression model with a good number of windows used.

However, fitting the same data set with too many windows (too small bandwidth) can also yield too noisy results and thus poor results as shown in Figure 5.1(c). The reason is that there are insufficient data points which fall into local windows and hence, the fit is too sensitive to the individual data points in the windows. See also Figure 2.3 in page 14 of Chapter 2 in [Loader 1997*d*] for the bandwidth problem in which the data used are real and the graphs plotted are from the real data.



(a) A single window, too large bandwidth, missing many points

(b) A good number of windows, good bandwidth, good results

(c) Too many windows, too small bandwidth, too noisy results

Figure 5.1: Local regression with different numbers of windows used

• Overview of local regression variants

We describe a general method for local regression in Section 5.2. Three variants of local regression are proposed here; one is called *Locally Weighted Regression* (LWR) [Cleveland and Loader 1996]; the second is called *Instant and Accurate Size Estimation* (IASE) [Sun et al. 1993]; and the last is called *Adaptive Size Estimation* (ASE) [Chen and Roussopoulos 1994]. All the three variants are in general a polynomial regression; as a result, the principle of least square error is used to find the best-fit coefficients of the polynomial used. While IASE and ASE do not consider any weight for data points used, LWR does consider. That is, points have higher weights if they are closer to a *fitting point* – a fixed point in a local window; points have lower weights if they are further away from the fitting point. The least square error by LWR is so called *weighted least square error* while the one by IASE and ASE is so called *ordinary least square error.*

We will show in this chapter that equi-height histograms (HIST) [Piatetsky-Shapiro and Connell 1984], the most popular type of histograms used by commercial database systems (see Table 1 in Chapter 3 of [Poosala 1997]), are also a special form of local regression which employs *local constant fitting* [Cleveland and Loader 1996] where a polynomial degree 0 is used in each local window (bucket). The local constant fitting is, in fact, the uniform distribution assumption used inside each bucket for query size estimation. That is, any distinct values that are grouped into a bucket would have an identical frequency which is averaged from the total of frequencies of all the distinct values inside the bucket. According to [Cleveland and Loader 1996], the local constant fitting very infrequently proves to be the best choice in practice and was only widely appreciated in the early smoothing literature.

In [Poosala et al. 1996; Poosala and Ioannidis 1997], a new and large taxonomy of histograms were studied. Among many types of histograms in the taxonomy, V-Optimal(V,A) and MaxDiff(V,A) histograms are the most attractive and the best choice of all, i.e., producing less errors in query result size estimation compared with others considered. These two types of histograms are also a form of local regression with the local constant fitting.

For all such histograms: HIST, V-Optimal(V,A) and MaxDiff(V,A), they are the same in (1) a window type used which is called *adaptive* – the window width varies from bucket to bucket – and (2) the local constant fitting used. They are different in grouping of distinct values into buckets (windows). However, they all fall short commonly in the same problem with the local constant fitting. That is, in spite of any kind of grouping employed, there may still exist some small curves (a small curve is a small partition of the entire graph of a data distribution) remaining in some buckets and fitting a small curve, such as a straight line with a slope (which is not the straight horizontal line), a hill-like curve, etc, by the polynomial degree 0 (i.e., the average frequency in the bucket) would not basically be as efficient as fitting it by a polynomial with a slightly higher degree, such as 1 or 2.

We will also show that a parametric method [Selinger et al. 1979*a*] based on the uniform distribution (UNF) is actually a single-window histogram method or a global constant fitting method.

• Multi-dimensional regression models and their storage problem

Multi-dimensional local regression models are, most probably, more appropriate to deal with size estimation of complex predicate queries than single-dimensional local regression models. The reason is that multi-dimensional models can deal with both an *attribute dependence* assumption (some correlation among attributes in a relation) if there is any and an *attribute independence* assumption (no correlation among attributes in the relation), whereas single-dimensional models basically can deal only with the attribute independence assumption. The rationale behind is that building multi-dimensional models takes into account the joint frequency distribution among a number of attributes whether or not those attributes really depend on one another – they may even be independent of one another. However, the main drawback of such models is their storage complexity, namely, the storage requirement. Building multi-dimensional models even for a global regression (single window) requires an exponential amount of storage to the number of attributes involved. We analyse their storage complexity below.

Let us take a look at an example of a multi-dimensional global regression model. For 2-attribute conjunctive queries of the form (some lower bound $\leq R.b_1 \leq$ some upper bound) $\wedge$ (some lower bound $\leq R.b_2 \leq$ some upper bound), to estimate sizes of the queries, a 2-dimensional regression model can be used, as proposed by Sun et al. [1993]. The 2-dimensional model can be simplified to: $\sum_{i=0}^{p_{b_1}} \sum_{j=0}^{p_{b_2}} a_{ij} x_{b_1}^i x_{b_2}^j$. $x_{b_1}^i$ together with $x_{b_2}^j$ is a pair of joint values of attributes b_1 and b_2 that appear together in relation R. a_{ij} is a coefficient and p_{b_1} and p_{b_2} are the polynomial degrees of attributes b_1 and b_2 respectively used by the model. Suppose that a degree for both p_{b_1} and p_{b_2} is equal to 10 (as used in the paper). The total number of coefficients a_{ij}'s needed to be maintained in the database profile catalog would be $(10+1) \times (10+1)$ or 11^2.

For 2-attribute disjunctive queries of the form (some lower bound $\leq R.b_1 \leq$ some upper bound) $\vee$ (some lower bound $\leq R.b_2 \leq$ some upper bound), the storage requirement would be the same, i.e., 11^2. As a result, for $K-$attribute conjunctive (disjunctive) queries, a K-dimensional regression model would require to maintain

11^K coefficients in order to do size estimation for the K-attribute queries.

A multi-dimensional histogram is a multi-dimensional model for size estimation of complex predicate queries. The derivation for the storage complexity for multi-dimensional histograms [Muralikrishna and DeWitt 1988; Muralikrishna 1988; Poosala and Ioannidis 1997; Poosala 1997] is similar to that for the multi-dimensional regression model shown above. Let K be the number of attributes considered to build a K-dimensional histogram. If *bucket* is the number of buckets allowed for each of the K attributes, then the total number of buckets required to build the multi-dimensional histogram is $bucket^K$. To calculated this, one can think of K nested loops each of which has *bucket* iterations, hence giving such a value $bucket^K$. This storage complexity could be the main reason prohibiting the use of multi-dimensional histograms in commercial database systems (e.g., INGRES, Sybase, DB2, Informix, MS-sqlserver, Oracle and Teradata) and only single-dimensional histograms which call for a linear size of storage are currently in use in those commercial database systems. The same reason could also apply to multi-dimensional regression models.

As a consequence, here we will focus only on building single-dimensional regression models for individual attributes of relation R and use those models together to estimate sizes of complex predicate queries. The storage size required by a single-dimensional regression model is linear, proportional to the number of windows used. That is, for a model, the storage size for all necessary parameters comprises 1) all upper bound values of the local windows used and 2) a fixed number of polynomial coefficients used by each local window.

• Background for neural networks

Backpropagation neural networks (NNs) [Rumelhart et al. 1986] are a nonlinear regression function approximator that has been successfully used in many fields of science and applications. To our best knowledge, NNs have never been used in the query optimisation literature, more particularly to the query size estimation. In Section 5.3, we describe a general method for NNs whereby many regression problems are solved.

We also describe the most well-known and popular training algorithm called *backpropagation* which is used to train neural networks so that a network can learn how to handle the regression problem at hand, basically an estimation problem.

• Background for a machine learning method M5

Harangsri et al. [1997] proposed a machine learning method called M5 [Quinlan 1992, 1993*b*]. M5 is also a form of local regression with a linear regression in each leaf node (window) – a polynomial degree 1 is used in each local window, instead of using a higher degree, such as 2 or higher. M5 combines two distinguished learning techniques: *model tree learning* [Quinlan 1992] and *instance-based learning* [Kibler et al. 1989; Aha 1990; Aha et al. 1991] into a new combined learning technique. Using feedback from already-processed queries, M5 creates a model tree whose leaf nodes consist of linear regression functions.

To approximate the size of a new unseen query, the most similar queries to the unseen query are selected from the stored already-processed queries and the result size of the unseen query is calculated based on 1) some linear regression functions in the model tree built and 2) the result sizes of the most similar queries.

M5 in its original implementation [Harangsri et al. 1997] basically uses a great deal of storage to maintain query feedback. Since a main and important aim in this thesis is to make any method proposed most practical to commercial database systems – use a small amount of storage, M5 in this chapter does not maintain any user feedback but merely creates model trees from which query size estimation can be carried out. The complete details of M5 can be found in Section 2.7 of Chapter 2.

• Query size estimation by local regression variants and neural networks

Section 5.4 illustrates how the general methods for local regression and neural networks can be applied to our problem domain which is to estimate query result sizes. Here we show how to build single-dimensional regression models from which query size estimation can be carried out. To build a model say for an attribute $b_{\#}$ of R, it is sufficient to consider two simple predicates of the forms $b_{\#} = x$ and $b_{\#} < x$. For all 7 methods: IASE, ASE, LWR, M5, HIST, NN and UNF, a single-dimensional model is built for the two simple predicates. In general, using single-dimensional regression models to estimate query result sizes implies that the attribute independence assumption is relied on. (Perhaps, this may not entirely be the case if the SVD technique proposed in [Poosala and Ioannidis 1997; Poosala 1997] is used to create single-dimensional models. But the severe disadvantage against this technique is that it cannot be extended to deal with the attribute dependence with more than two attributes involved.)

Like building histograms with a sample size n ($\leq N$), a single-dimensional model created by any of local regression variants and a neural network can also be built from a sample size n. That is, a data set with frequencies: $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_d, f(x_d))$ which is the source of creating a single-dimensional model, can be created from a sample size n, in place of the size N of an entire relation.

• Single implementation framework for local regression variants

Just like the claim in Poosala's thesis (see page 94 of [Poosala 1997]), it is our experience that there is, at least so far, no single *universal* estimation method which can universally handle all kinds of data distributions. That is, among the 7 available methods: IASE, ASE, LWR, M5, HIST, NN and UNF although in most of the times with the data experimented with, LWR seems to perform the best, there are times that others (one of those) perform better than LWR. With (1) the unknown nature of data distributions in future perhaps difficult to fit and (2) different strengths of the local regression methods, instead of proposing LWR as the best single method for any database system, we propose a data structure in Section 5.5 which makes it possible to gracefully and generically implement together all the methods IASE, ASE, LWR, M5, HIST and UNF (or even V-Optimal(V,A) as well as MaxDiff(V,A) histograms because both are only different from HIST in the grouping of distinct values into buckets) in a single framework. The implementation would not be difficult as all of these methods are a form of local regression.

• Method evaluation

Section 5.6 contains three extensive sets of experiments with a variety of relation configurations. The aim of the first set is to compare local regression models against global models. The aim of the second set is to compare the efficiency among the 7 methods using simple predicate queries. The aim of the last set is similar to that of the second set except the queries used. Complex predicate queries are used here.

Since a significant aim in this chapter is the amount of storage required by a method, all the comparisons among the 7 methods have been made basically using the same number of parameters. We summarise the main achievements of this chapter in Section 5.8.

• Local regression with join selectivity

For the join selectivity estimation problem, since the experimental work is not yet done, we will give very sound justifications in Section 5.7 of why local regression would perform for joins equally as well as when it performs for selections.

5.2 Method for local regression

In this section, we describe a general method for local regression. Three variants of local regression are described here; one is Locally Weighted Regression (LWR) [Cleveland and Loader 1996]; the second is Instant and Accurate Size Estimation (IASE) [Sun et al. 1993]; and the last is Adaptive Size Estimation (ASE) [Chen and Roussopoulos 1994].

Loader [1997*b*,*d*] has proposed 3 partitioning schemes to partition x's values in an attribute domain into a number of windows. Recall that the width of windows is called *bandwidth*. The most simple scheme is called *fixed-width* bandwidth. The next more sophisticated one is called *nearest neighbour* bandwidth and the most sophisticated one is *adaptive* bandwidth. For our current work, we adopt the most simple scheme fixed-width bandwidth.

Let us define a fixed-width bandwidth h by $\frac{(x_{max}-x_{min})}{nunwin}$ where x_{max} is the maximum value in the domain of an attribute $b_{\#}$, x_{min} is the minimum value of attribute $b_{\#}$ and $numwin$ is the number of windows one wants to use for fitting. The location of a window j is thus defined by $(x_{min} + (j-1) * h, x_{min} + j * h]$, where $j = 1, 2, \ldots, numwin$. $x_{min} + (j-1) * h$ is the lower bound value of window j and $x_{min} + j * h$ is the upper bound value of window j. The data points which fall into a local window are fitted by a polynomial with a low degree p, normally $p = 2$ or 3 [Loader 1997*b*]. Intuitively, with a small number of data points in a local window, it is sufficient to fit them with a polynomial with the low degree.

d is the number of distinct values of attribute $b_{\#}$. A given entire data set is: $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_d, f(x_d))$. Recall that all x_i's in the given data set are already in ascending order. Let data points $(x_k, f(x_k)), (x_{k+1}, f(x_{k+1})), \ldots, (x_{l-1}, f(x_{l-1}), (x_l, f(x_l))$, i.e., a partition in the entire data set, fall into a fitting window, say j, where $1 \leq k \leq d$ and $1 \leq l \leq d$ but $k \leq l$. As a result, all of $x_k, x_{k+1}, \ldots, x_{l-1}, x_l$ must be in the range $(x_{min} + (j-1) * h, x_{min} + j * h]$ of window j.

Let m be the number of data points in window j which is equal to $(l - k + 1)$. Let $fitpnt$ be the fitting point in window j, which is the point in the middle of the window, defined by $fitpnt = x_{min} + (j-1) * h + \frac{h}{2}$. A polynomial $g(x)$ that one can use to fit the m data points in window j could be:

$$g(x) = \sum_{i=0}^{p} \frac{a_i (x - fitpnt)^i}{i!} \tag{5.1}$$

which is used by LWR, or:

$$g(x) = \sum_{i=0}^{p} a_i x^i \tag{5.2}$$

which is used by ASE and IASE. p in (5.1) and (5.2) is a degree of the polynomial used. In fact, IASE uses a polynomial of the form $g(x) = \sum_{i=-p1}^{p2} a_i x^i$, where $p = p_1 + p_2$ and $p_1, p_2 \geq 0$. Thus if $p_1 = 0$, then the form of the polynomial is reduced to the same polynomial as ASE.

With LWR, a data point at x_i in window j is weighted by $w(\frac{x_i - fitpnt}{h})$, where $w(x)$ is a weight function. There are a number of weight functions that one can use such as Gaussian function $w(x) = exp(-(2.5x)^2/2)$, Triweight function $w(x) = (1 - abs(x)^2)^3$ and etc. ($abs(\cdot)$ is the absolute value of the parameter given.) See various functions in [Loader 1997*d*]. A weight function used must be symmetrical and decreasing in the range $[0, 1]$ and satisfy two conditions, $w(0) = 1$ and $w(1) = 0$. The weight function used in our implementation is called *Tricube*, defined by $w(x) = (1 - abs(x)^3)^3$, where $abs(x) < 1$.

A common rationale of the different weight functions is that by assigning different weights to different data points, the points closer to the fitting point $fitpnt$ have more weights and the points further away have lower weights.

The coefficients a_i's in equation (5.1) or (5.2) can be found by locally weighted least squares. That is, choose some a_i's values to minimise:

$$\sum_{i=k}^{l} w(x_i)(f(x_i) - g(x_i))^2 \tag{5.3}$$

Transform the weighted sum of squares (5.3) to matrix form as follows:

$$(Y - XA)^T W (Y - XA) \tag{5.4}$$

Solve equation (5.4) for A:

$$\widehat{A} = (X^T W X)^{-1} X^T W Y \tag{5.5}$$

where X^T is the transpose of X. Note that $\widehat{A}$ is an approximate value of A. This is due to the fact that the resulting coefficients a_i's obtained may or may not be able to make the least squares in (5.3) completely equal to zero. The following are the definitions for all the matrices in (5.5).

X is an $m \times (p+1)$ matrix and one of the following:

$$X = \begin{bmatrix} \frac{(x_k - fitpnt)^0}{0!} & \frac{(x_k - fitpnt)^1}{1!} & \cdots & \frac{(x_k - fitpnt)^p}{p!} \\ \frac{(x_{k+1} - fitpnt)^0}{0!} & \frac{(x_{k+1} - fitpnt)^1}{1!} & \cdots & \frac{(x_{k+1} - fitpnt)^p}{p!} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{(x_{l-1} - fitpnt)^0}{0!} & \frac{(x_{l-1} - fitpnt)^1}{1!} & \cdots & \frac{(x_{l-1} - fitpnt)^p}{p!} \\ \frac{(x_l - fitpnt)^0}{0!} & \frac{(x_l - fitpnt)^1}{1!} & \cdots & \frac{(x_l - fitpnt)^p}{p!} \end{bmatrix}, \ X = \begin{bmatrix} x_k^0 & x_k^1 & \cdots & x_k^p \\ x_{k+1}^0 & x_{k+1}^1 & \cdots & x_{k+1}^p \\ \cdots & \cdots & \cdots & \cdots \\ x_{l-1}^0 & x_{l-1}^1 & \cdots & x_{l-1}^p \\ x_l^0 & x_l^1 & \cdots & x_l^p \end{bmatrix} \tag{5.6}$$

The first X matrix on the left which is used by LWR consists of elements derived from polynomial (5.1) while the second X on the right which is used by ASE and IASE consists of elements derived from polynomial (5.2).

W is an $m \times m$ diagonal matrix, which has weights along the diagonal and 0 elsewhere. A is a $(p+1) \times 1$ matrix of the coefficients of the polynomial used and Y is an $m \times 1$ response matrix. The following are all the matrices:

$$W = \begin{bmatrix} w(x_k) & 0 & 0 & \cdots & \cdots & 0 \\ 0 & w(x_{k+1}) & 0 & \cdots & \cdots & 0 \\ 0 & 0 & w(x_{k+2}) & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & w(x_{l-1}) & 0 \\ 0 & 0 & 0 & 0 & 0 & w(x_l) \end{bmatrix}, \ A = \begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_{p-1} \\ a_p \end{bmatrix}, \ Y = \begin{bmatrix} f(x_k) \\ f(x_{k+1}) \\ \cdots \\ f(x_{l-1}) \\ f(x_l) \end{bmatrix} \tag{5.7}$$

If we do not take any weight into consideration (in other words, we treat W as the identity matrix) and use the right side matrix X in (5.6) (employed by ASE and IASE) – implying that this is the unweighted or ordinary least squares problem –, then the least square problem here would be the same as that of ASE and IASE. Hence, the least squares in (5.5) would then reduce to:

$$\widehat{A} = (X^T X)^{-1} X^T Y \tag{5.8}$$

The main differences from previous work is that we introduce (1) local regression in place of global regression (more windows used for fitting), (2) an approach to solve the query size estimation by the weighted least squares in addition to the unweighted least squares and (3) a new type of polynomial (other types can also be used).

In the remainder of this chapter, for ASE and IASE, the solution $\widehat{A}$ for coefficients is the solution to the least squares by (5.8) where the matrix X used is the right side one of (5.6). For LWR, the solution $\widehat{A}$ is by (5.5) where the matrix X used is the left side one of (5.6).

Overall, LWR outperforms ASE and IASE in estimation [Cleveland and Loader 1996]. This is due to 1) the closed form of the polynomial (5.1) which is easier to fit than the normal polynomial (5.2) and 2) different weights assigned to data points. We will also show by experiments that among the three variants: LWR, ASE and IASE, the most superior one is LWR.

5.3 Method for backpropagation NN

Figure 5.2 shows a typical architecture of the well-known multi-layer feed-forward network. The network is unidirectional; the flow of the network is only in a single direction (in the figure from left to right). The network dimension is $numinp \times numout$ which maps *numinp* inputs to *numout* outputs. Circles in the network are each a *neuron* which serves as a computing unit.

The first layer (the first column of neurons) is the input layer which performs no computation but feeds input patterns to the network. Neurons in other layers do perform computation by receiving their input from the neurons in the previous layer. Hidden layers are any layer after the input layer but not the output layer.

The output layer is the last layer (last column of neurons) in the network which typically produces *approximate output patterns*, namely, computed output patterns of the form $(\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_{numout})$. An *input pattern* is a vector of inputs of the form

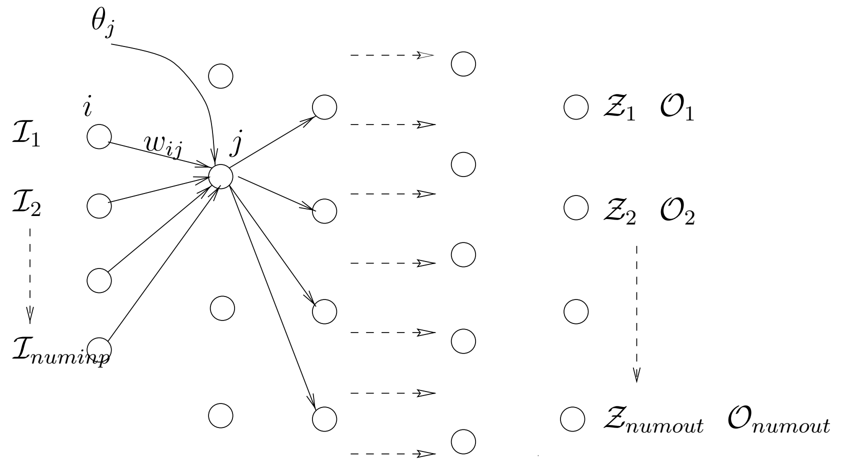

θ_j
i
$\mathcal{I}_1$
w_{ij}
j
$\mathcal{I}_2$
$\mathcal{I}_{numinp}$
$\mathcal{Z}_1$ $\mathcal{O}_1$
$\mathcal{Z}_2$ $\mathcal{O}_2$
$\mathcal{Z}_{numout}$ $\mathcal{O}_{numout}$

Figure 5.2: A typical multi-layer neural network architecture

$(\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{numinp})$. A *desired output pattern* is a vector of outputs of the form $(\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{numout})$. A desired output pattern consists of actual output values $\mathcal{O}_i$'s which occur in reality and normally can be observed from the application, whereas an approximate output pattern consists of approximate output values to the actual ones computed by the neural network.

A neuron j (see also Figure 5.2) is activated to compute its output activation value $\mathcal{A}_j$ by:

$$\mathcal{A}_j = \mathcal{F}(\mathcal{I}_j) \tag{5.9}$$

where:

$$\mathcal{I}_j = \sum_i w_{ij}\mathcal{A}_i + \theta_j \tag{5.10}$$

where w_{ij} is the weight from neuron i to j whose value is between 0 and 1, θ_j is a bias for neuron j and $\mathcal{F}(\mathcal{I}_j)$ is a sigmoid activation function, e.g., $\mathcal{F}(\mathcal{I}_j) = \frac{1}{1+exp^{-\mathcal{I}_j}}$ which is used in our implementation.

In the first hidden layer where actual computation commences, the activation values $\mathcal{A}_i$'s in equation (5.10) are, in fact, the input values $\mathcal{I}_i$'s, where $i = 1, 2, \ldots,$ *numinp* from the input layer. Equations (5.9) and (5.10) are recursive in their computation through the activation values. These activation values propagate forward toward the output layer of the network. At the output layer, each $\mathcal{A}_i$ computed is each $\mathcal{Z}_i$ as shown in the figure, where $i = 1, 2, \ldots, numout$. This is actually how the network operates on-line to compute approximate output patterns. Note that when a neural network is in operation on-line, all the weights among connections must be fixed. This mode of operation is called *working mode.*

The other mode of network operation is called *training mode.* As the name implies, the network will not be in actual use while being trained. The network is fed by a number of input patterns and will attempt to adjust the weights over the network by the well-known *backpropagation* learning algorithm [Rumelhart et al. 1986] so that errors are minimised, due to the differences between the computed (approximate) output patterns and the desired (actual) output patterns.

The backpropagation algorithm proceeds to reduce the errors as follows. First, an input pattern consisting of $(\mathcal{I}_i, \mathcal{I}_2, \ldots, \mathcal{I}_{numinp})$ is given to the network and propagates forward by the recursive computation in (5.9) and (5.10) through the network to the output layer, yielding a computed output pattern $(\mathcal{Z}_1, \mathcal{Z}_2, \ldots, \mathcal{Z}_{numout})$ in the output layer. This is called *forward propagation* which is exactly the same as the working mode described above.

Next is *error correction* due to the pairwise difference between the desired value $\mathcal{O}_i$ and the computed value $\mathcal{Z}_i$ at the output neuron i, where $i = 1, 2, \ldots, numout$. The error δ_i by neuron i $(= 1, 2, \ldots, numout)$ in the output layer is computed by:

$$\delta_i = \mathcal{Z}_i \; (1 - \mathcal{Z}_i) \; (\mathcal{O}_i - \mathcal{Z}_i) \tag{5.11}$$

and likewise the error δ_i by neuron i in a hidden layer is computed by:

$$\delta_i = \mathcal{A}_i \; (1 - \mathcal{A}_i) \; \sum_j w_{ij} \; \delta_j \tag{5.12}$$

where $\mathcal{A}_i$'s value is computed by the forward propagation.

Figure 5.3 best explains how to compute δ_i in a hidden layer. Note the direction of error correction in the figure which is from rear to front – propagate errors back in the network for weight adjustment. This is why the algorithm is called backpropagation.

Using the errors defined in (5.11) and (5.12), the network then adjusts a weight w_{ij} by:

$$w_{ij} \leftarrow w_{ij} + \gamma \; \mathcal{A}_i \; \delta_j$$

where γ is a learning rate used by the network. A learning rate used in each layer of the network could be different.

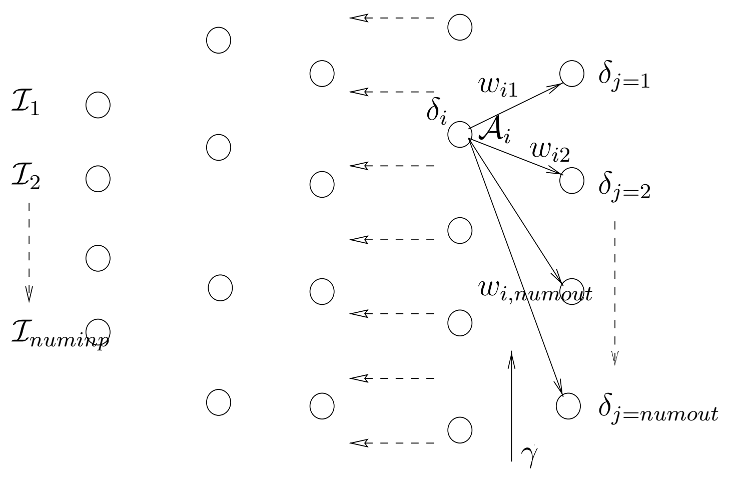


Figure 5.3: Backpropagation of error (rear to front) to adjust network weights

Similar to the weight adjustment, the bias to neuron i, i.e., θ_i is adjusted by:

$$\theta_i \leftarrow \theta_i + \gamma\ \delta_i$$

For a next input pattern, the same process as described above is repeated of 1) forward propagation and 2) backpropagation to reduce the pattern error and adjust the network's weights. Repeat until all input and desired output patterns are exhausted. We call one pass of the network through all the input and desired output patterns *one iteration*.

The method described above is that of the standard backpropagation neural network which can be applied to many problem domains. In our problem domain, our input patterns have only one dimension, namely, $numinp = 1$, which consists of x's values and our output patterns also have only one dimension, namely, $numout = 1$, which consists of $f(x)$'s values. Recall that a given data set is: $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_d, f(x_d))$. The network architecture used in our implementation is shown in Figure 5.4.

Normally, in many applications, 1 or 2 hidden layers are sufficient. In our case, we decided to use one hidden layer and 4 hidden neurons for the hidden layer. How many neurons used in each hidden layer would be sufficient is hard to determine and still remains an open problem in the neural network community. Although using too small a number of hidden neurons may produce poor results, using too many can also produce poor results. This is because it is more difficult to train the network with more hidden neurons.

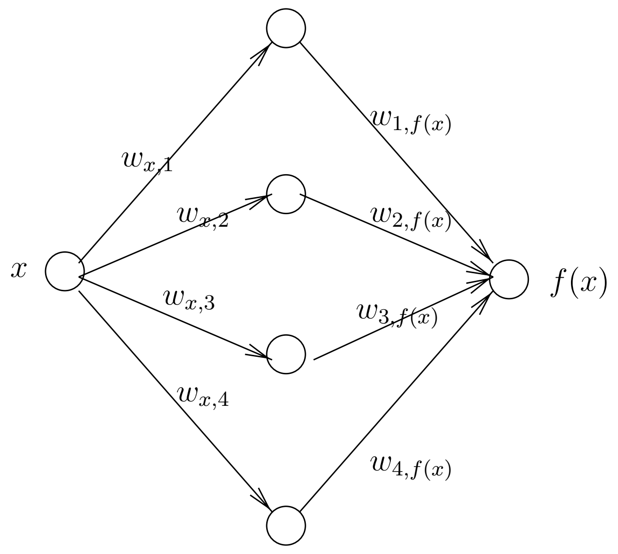


Figure 5.4: 3-layer, 1 input and 1 output network

5.4 Query size estimation

Let Q be a complex query on relation R. Using the independence assumption, the size of Q can be approximated from its composite simple predicates. (Refer to Section 2.2 of Chapter 2 for the calculation of the size.) Recall that we consider two types of simple predicates, namely of the form $b_{\#} = x$ and $b_{\#} < x$, where x is a value in the domain of $b_{\#}$. They are sufficient to cover other types of predicates ($b_{\#}$ *relopt* x) in estimating sizes of any complex predicate queries, where *relopt* is any of the relational operators. Based on the two simple predicates, we build a single-dimensional regression model for selectivity estimation of $sel(b_{\#} = x)$ and $sel(b_{\#} < x)$ in Section 5.4.1.

In Section 5.2, we have proposed 3 different local regression methods: LWR, ASE and IASE each of which uses a fixed-width bandwidth scheme. Any description, whenever referring to these 3 local regression methods, hence implies that windows built for local regressions by these methods are fixed-width.

M5 proposed in [Harangsri et al. 1997] is a form of local regression with a linear regression (polynomial degree 1) used in each local window. M5 uses a form of adaptive windows, i.e., variable-width windows and employs a partitioning algorithm to create them. The rationale in building adaptive windows is to have the data points in a local window have similar frequencies (or similar cumulative frequencies), i.e., similar $f(x)$'s values in the window. The partitioning algorithm to create adaptive windows can be found near the page 70 of Chapter 2. Any description, whenever referring to M5, hence implies that windows built for local regressions by M5 are variable-width.

The original version of M5 implemented requires to maintain user query feedback. Feedback by a query is of the form (simple predicate, query result size). M5's accuracy when estimating sizes of new unseen queries, is very high due to the great assistance of the feedback from many already-processed queries maintained but perhaps with the tradeoff of a large amount of storage required. Since the main concern of this thesis is storage requirement used by different methods, the version of M5 here does not maintain any query feedback and simply creates a model tree consisting of a number of linear regression functions for the local windows built (one function for one window).

5.4.1 Building a single-dimensional regression model for $b_{\#}$

In this section, we will show how to build a single-dimensional regression model for attribute $b_{\#}$. This involves deriving two formulas for the selectivities of $sel(b_{\#} = x)$ and $sel(b_{\#} < x)$.

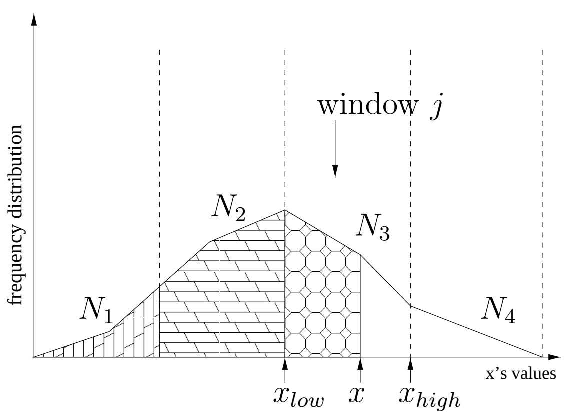
To build a single-dimensional model, one can select to:

- fit the frequency distribution of $b_{\#}$ or
- fit the cumulative frequency distribution of $b_{\#}$.

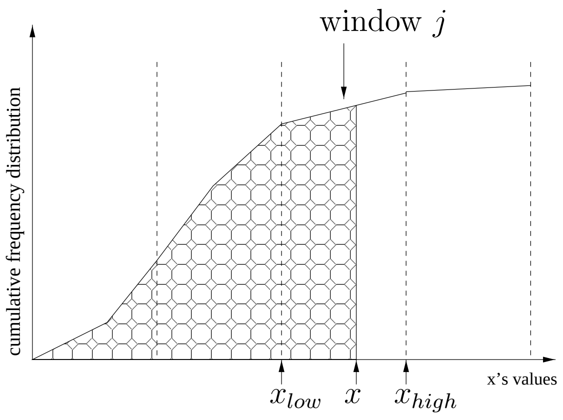
Different methods, e.g., ASE, IASE, LWR, M5, NN and HIST employ either of the two fittings above to build a single-dimensional model. We will shortly describe what kind of fitting a method employs.

Let $(x_1, f_1), (x_2, f_2), \ldots, (x_d, f_d)$ be data points for a frequency distribution of x_i's. Likewise, let $(x_1, F_1), (x_2, F_2), \ldots, (x_d, F_d)$ be data points for a cumulative frequency distribution of x_i's which does not include the frequency of x_i itself. That is, an F_j is equal to the total number of times that any x's value less than x_j occurs in relation R under $b_{\#}$. Mathematically, an F_j is equal to $\sum_{i=1}^{j-1} f_i$, where $j = 2, 3, \ldots, d$ and when $j = 1$, $F_j = 0$.

Figure 5.5(a) shows a frequency distribution of x_i's values while Figure 5.5(b)



(a) A frequency distribution



(b) A cumulative frequency distribution

Figure 5.5: A frequency distribution against its cumulative frequency distribution

shows a corresponding cumulative frequency distribution. Both of the distributions have 4 windows.

Below in Sections 5.4.1.1, 5.4.1.2, 5.4.1.3 5.4.1.4, 5.4.1.5 and 5.4.1.6, we will show how 7 respective methods: IASE, ASE, LWR, M5, NN and HIST (UNF) build a single-dimensional regression model for the selectivities of $sel(b_{\#} = x)$ and $sel(b_{\#} < x)$.

5.4.1.1 IASE

IASE proposed to fit frequency distributions. From a given data set, $(x_1, f_1), (x_2, f_2), \ldots, (x_d, f_d)$, let data points $(x_k, f_k), (x_{k+1}, f_{k+1}), \ldots, (x_{l-1}, f_{l-1}), (x_l, f_l)$ (a partition of the entire data set) fall into a fitting window, say j (see Figure 5.5(a) also). x_{low} in the figure thus is equal to $x_{min} + (j-1) * h$ and x_{high} is equal to $x_{min} + j * h$, where h is a fixed-width bandwidth. N_1, N_2, N_3 and N_4 in the figure each denote the total number of tuples which fall into windows 1, 2, 3 and 4, respectively. Mathematically, $N_j = \sum_{i=k}^{l} f_i$, where $j = 1, 2, \ldots, numwin$. In this case, the number of windows used $numwin$ is equal to 4. The following are all matrices needed to be prepared for the least squares in (5.8).

$$X = \begin{bmatrix} x_k^0 & x_k^1 & x_k^2 \\ x_{k+1}^0 & x_{k+1}^1 & x_{k+1}^2 \\ \dots & \dots & \dots \\ x_{l-1}^0 & x_{l-1}^1 & x_{l-1}^2 \\ x_l^0 & x_l^1 & x_l^2 \end{bmatrix}, \quad A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \quad Y = \begin{bmatrix} f_{x_k} \\ f_{x_{k+1}} \\ \dots \\ f_{x_{l-1}} \\ f_{x_l} \end{bmatrix} \tag{5.13}$$

We then solve the ordinary least squares in (5.8) for the coefficients a_i's. Suppose that x in the two simple predicates $b_{\#} = x$ and $b_{\#} < x$ falls into window j. Given the solved coefficients, $sel(b_{\#} = x)$, the selectivity of predicate $b_{\#} = x$, is calculated by $g(x) = \sum_{i=0}^{2} a_i x^i$, divided by N. $sel(b_{\#} < x)$, the selectivity of predicate $b_{\#} < x$, is calculated by the sum of $\sum_{i=1}^{j-1} \frac{N_i}{N}$ and:

$$\frac{N_j * \frac{\int_{x_{low}}^{x} g(x)d(x)}{\int_{x_{low}}^{x_{high}} g(x)d(x)}}{N} \tag{5.14}$$

In Figure 5.5(a), the shaded area of window $j = 3$ equivalently denotes the selectivity calculated by equation (5.14). All the three shaded areas together in windows 1, 2 and 3 form $sel(b_{\#} < x)$. Recall that $g(x) = \sum_{i=0}^{p} a_i x^i$. With polynomial degree $p = 2$, the indefinite integral of $g(x)$ is defined by:

$$G(x) = \int g(x)d(x) = \int \sum_{i=0}^{2} a_i x^i d(x) = \sum_{i=0}^{2} \frac{a_i x^{(i+1)}}{(i+1)} \tag{5.15}$$

$G(x)$ can be called the *cumulative frequency distribution* of x. The definite integral of $g(x)$ between x' and x'' is defined by:

$$\int_{x'}^{x''} g(x)d(x) = G(x'') - G(x') = \sum_{i=0}^{2} \frac{a_i x''^{(i+1)}}{(i+1)} - \sum_{i=0}^{2} \frac{a_i x'^{(i+1)}}{(i+1)} \tag{5.16}$$

Using (5.16), equation (5.14) can be solved for $sel(b_{\#} < x)$.

With IASE, the number of parameters per window needed to be maintained in the database profile catalog is 4, i.e., a_0, a_1, a_2 and N_i.

5.4.1.2 ASE

ASE proposed to fit cumulative frequency distributions and uses the cumulative frequency distribution function $G(x)$ in (5.15) to fit data points in a fitting window, say j. Let data points $(x_k, F_k), (x_{k+1}, F_{k+1}), \ldots, (x_{l-1}, F_{l-1}), (x_l, F_l)$ fall into window j located between x_{low} and x_{high} (see also Figure 5.5(b)). The following are all matrices needed to be prepared for the least squares in (5.8).

$$X = \begin{bmatrix} \frac{x_k^1}{1} & \frac{x_k^2}{2} & \frac{x_k^3}{3} \\ \frac{x_{k+1}^1}{1} & \frac{x_{k+1}^2}{2} & \frac{x_{k+1}^3}{3} \\ \ldots & \ldots & \ldots \\ \frac{x_{l-1}^1}{1} & \frac{x_{l-1}^2}{2} & \frac{x_{l-1}^3}{3} \\ \frac{x_l^1}{1} & \frac{x_l^2}{2} & \frac{x_l^3}{3} \end{bmatrix}, \quad A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \quad Y = \begin{bmatrix} F_{x_k} \\ F_{x_{k+1}} \\ \ldots \\ F_{x_{l-1}} \\ F_{x_l} \end{bmatrix} \tag{5.17}$$

Note that elements in matrix X above are derived from function $G(x)$ in (5.15).

We solve the ordinary least squares in (5.8) for the coefficients a_i's. Suppose that x in the predicates $b_\# = x$ and $b_\# < x$ falls into window j. Given the solved coefficients, $sel(b_\# < x)$ is calculated by $G(x) = \sum_{i=0}^{2} \frac{a_i x^{(i+1)}}{(i+1)}$, divided by N. $sel(b_\# = x)$ is calculated by $g(x) = \sum_{i=0}^{2} a_i x^i$, divided by N.

With ASE, the number of parameters per window needed to be maintained in the profile catalog is 3, i.e., a_0, a_1 and a_2, which is one parameter less than IASE. As opposed to IASE, ASE does not need to maintain N_i for a local window.

We decided that the following 3 methods below: LWR, M5 and NN, also fit the cumulative frequency distribution. Although LWR, M5 and NN (as well as ASE and IASE) can fit either the frequency distribution or cumulative frequency distribution (by simply changing from f_i's in matrix Y to F_i's or vice versa), we selected to fit the cumulative frequency distribution for the three methods. The main reason is simply that we can save one parameter (which is N_i) per window but without sacrificing for the accuracy obtained.

5.4.1.3 LWR

LWR fits cumulative frequency distributions and uses the polynomial $g(x)$ defined in (5.1), i.e., $g(x) = \sum_{i=0}^{p} \frac{a_i (x - fitpnt)^i}{i!}$. Let data points $(x_k, F_k), (x_{k+1}, F_{k+1}), \ldots, (x_{l-1}, F_{l-1}), (x_l, F_l)$ fall into a fitting window, say j located between x_{low} and x_{high}. The following are all matrices needed to be prepared for the weighted least squares in (5.5).

$$X = \begin{bmatrix} \frac{(x_k - fitpnt)^0}{0!} & \frac{(x_k - fitpnt)^1}{1!} & \frac{(x_k - fitpnt)^2}{2!} \\ \frac{(x_{k+1} - fitpnt)^0}{0!} & \frac{(x_{k+1} - fitpnt)^1}{1!} & \frac{(x_{k+1} - fitpnt)^2}{2!} \\ \dots & \dots & \dots \\ \frac{(x_{l-1} - fitpnt)^0}{0!} & \frac{(x_{l-1} - fitpnt)^1}{1!} & \frac{(x_{l-1} - fitpnt)^2}{2!} \\ \frac{(x_l - fitpnt)^0}{0!} & \frac{(x_l - fitpnt)^1}{1!} & \frac{(x_l - fitpnt)^2}{2!} \end{bmatrix}, \quad A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \quad Y = \begin{bmatrix} F_{x_k} \\ F_{x_{k+1}} \\ \dots \\ F_{x_{l-1}} \\ F_{x_l} \end{bmatrix} \tag{5.18}$$

The weight matrix W is defined in (5.7). We then solve the weighted least squares in (5.5) for the coefficients a_i's. Suppose that x in the predicates $b_\# = x$ and $b_\# < x$ falls into window j. Given the solved coefficients, $sel(b_\# < x)$ is calculated by $g(x) = \sum_{i=0}^{p} \frac{a_i(x-fitpnt)^i}{i!}$, divided by N. $sel(b_\# = x)$ is calculated by $\frac{g(x+\triangle)-g(x)}{N}$, where $\triangle$ is defined by $\frac{x_{max}-x_{min}}{(d-1)}$.

With LWR, the number of parameters per window needed is 3, i.e., a_0, a_1 and a_2.

5.4.1.4 M5

M5 fits cumulative frequency distributions and uses a polynomial degree 1 for fitting, namely, $g(x) = a_0 + a_1 x$. Let data points $(x_k, F_k), (x_{k+1}, F_{k+1}), \dots, (x_{l-1}, F_{l-1}), (x_l, F_l)$ fall into a fitting window, say j located between x_{low} and x_{high}. Using Figure 5.5(b) for the description here, we slightly abuse the fixed-width windows in the figure for the adaptive (variable-width) windows used by M5. That is, the windows shown in the figure are originally meant to be fixed-width. The following are all matrices needed to be prepared for the least squares in (5.8).

$$X = \begin{bmatrix} x_k^0 & x_k^1 \\ x_{k+1}^0 & x_{k+1}^1 \\ \dots & \dots \\ x_{l-1}^0 & x_{l-1}^1 \\ x_l^0 & x_l^1 \end{bmatrix}, \quad A = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, \quad Y = \begin{bmatrix} F_{x_k} \\ F_{x_{k+1}} \\ \dots \\ F_{x_{l-1}} \\ F_{x_l} \end{bmatrix} \tag{5.19}$$

We then solve the ordinary least squares in (5.8) for the coefficients a_i's. The ordinary least squares is the typical way that M5 uses to find best-fit coefficients.

Suppose that x in the predicates $b_{\#} = x$ and $b_{\#} < x$ falls into window j. Given the solved coefficients, $sel(b_{\#} < x)$ is calculated by $g(x) = a_0 + a_1 x$, divided by N. Similar to LWR, $sel(b_{\#} = x)$ is calculated by $\frac{g(x+\triangle)-g(x)}{N}$, where $\triangle$ is defined by $\frac{x_{max}-x_{min}}{(d-1)}$.

With M5, the number of parameters per window needed to be maintained in the database profile catalog is (2+1) = 3. The first 2 parameters come from the two coefficients a_0 and a_1. The last parameter comes from the upper bound value for a local adaptive (variable-width) window.

Note that unlike the fixed-width windows (employed by IASE, ASE and LWR) whose upper bound values (i.e., $x_{min} + j * h$) can be calculated from x_{min} and h, for the variable-width windows one has to keep these upper bound values in the profile catalog. The purpose of the upper bound values is to check in which window among *numwin* windows the x's value in the given predicate ($b_{\#}$ *relopt* x) lies.

In case the reader is aware that one has to store h in the catalog, the value of h can be calculated from $\frac{x_{max}-x_{min}}{numwin}$. Normally x_{min} and x_{max} are always assumed to be in the catalog for any attribute in any database system. For *numwin*, since nowadays the most popular method implemented in commercial database systems is histogram-based, *numwin* is also a usual and regular value in the catalog for any attribute.

5.4.1.5 NN

NN fits cumulative frequency distributions. A neural network for an attribute $b_{\#}$ is rudimentarily a global regression. One can imagine that only a single window is used. Thus all the data points in the data set $(x_1, F_1), (x_2, F_2), \ldots, (x_d, F_d)$ are used by a network, where each x_i, $(i = 1, 2, \ldots, d)$ is an input pattern fed to the input neuron x and F_i is the corresponding desired output pattern at the output neuron $f(x)$ (see Figure 5.4).

There are a few reasons of why we globally fit the entire data set. First, the nature of neural networks themselves is nonlinear. By the backpropagation learning algorithm, a network should be able to adjust itself (weights) to well fit the entire data set given, no matter what kind of data distribution handled is like. Second, the backpropagation network described in Section 5.3 is the standard one which has been successfully used to solve many nonlinear regression problems and which is typically fed with the entire data set. Last, training a neural network which is finding the optimal weights over the network, is exceedingly time-consuming. Finding the optimal weights is analogous to finding the optimal coefficients by the least squares used by any of the local regression methods. However, finding the optimal coefficients can be done in an instant time.

In the training mode of a network, both x_i and F_i are scaled to values between 0 and 1 before being used by the network. x_i is scaled by $\frac{x_i - x_{min}}{x_{max} - x_{min}}$ and F_i is scaled by $\frac{F_i}{N}$ (as F_i is always less than N). Note that $\frac{F_i}{N}$ is a selectivity $sel(b_\# < x)$. Thus in the working mode (from which to approximate query sizes), every output value produced by neuron $f(x)$ is a $sel(b_\# < x)$, where x must be scaled by $\frac{x - x_{min}}{x_{max} - x_{min}}$ before being fed to the input neuron x of the network.

$\triangle$ is defined by $\frac{x_{max} - x_{min}}{(d-1)}$. $sel(b_\# = x)$ is calculated by:

$$\text{(resulting selectivity at neuron } f(x),\text{ given } \frac{x + \triangle - x_{min}}{x_{max} - x_{min}} \text{ to the input neuron } x) -$$
$$\text{(resulting selectivity at neuron } f(x),\text{ given } \frac{x - x_{min}}{x_{max} - x_{min}} \text{ to the input neuron } x)$$

Consider the network architecture in Figure 5.4. The number of parameters needed to be maintained in the profile catalog for a neural network is (1) the number of weights over connections in the network, (2) the number of biases for all neurons and (3) the number of all learning rates for each layer.

The first consists of (4+4) = 8 parameters. 4 parameters come from 4 weights from the input neuron x to the hidden layer and another 4 come from the remaining 4 weights from the hidden layer to the output neuron $f(x)$.

The second consists of (4+1) = 5 parameters. 4 parameters are biases for the neurons in the hidden layer and 1 parameter is the bias for the output neuron $f(x)$.

For the third, the network uses a single learning rate for the whole network. Thus only 1 parameter is required.

Therefore, the total number of parameters per network for an attribute $b_\#$ needed to be maintained in the database profile catalog is (8+5+1) = 14.

5.4.1.6 HIST and UNF

The current and most popular implementation of histograms in commercial database systems [Poosala 1997], HIST fits frequency distributions and uses a polynomial degree 0 for fitting. Each bucket of equi-height histograms is comparable to a fitting window. The width of equi-height histogram buckets varies which is analogous to adaptive windows used by M5. Equi-height histograms are, in fact, a specific form of local regression that employs *local constant fitting* [Cleveland and Loader 1996]. The following is the reason.

HIST uses the polynomial degree 0, namely, $g(x) = c$ to locally fit data points in a fitting window, say j located between x_{low} and x_{high}, where c is the fitting constant in window j (use Figure 5.5(a) to which to relate the following description). The use of a constant c for a local fitting is the reason why the fitting is called local constant fitting.

Let data points $(x_k, f_k), (x_{k+1}, f_{k+1}), \ldots, (x_{l-1}, f_{l-1}), (x_l, f_l)$ fall into window j. Suppose that x in the predicates $b_{\#} = x$ and $b_{\#} < x$ falls into window j. $sel(b_{\#} = x)$ is the fitting constant c in window j, divided by N.

Like IASE (see also Section 5.4.1.1), $sel(b_{\#} < x)$ is calculated by the sum of $\sum_{i=1}^{j-1} \frac{N_i}{N}$ and:

$$\frac{N_j * \frac{\int_{x_{low}}^{x} g(x)d(x)}{\int_{x_{low}}^{x_{high}} g(x)d(x)}}{N} \tag{5.20}$$

Due to the equi-height nature of equi-height histograms, each N_i, $i = 1, 2, \ldots, numwin$ is identical, namely, approximately equal to $\frac{N}{numwin}$. $\sum_{i=1}^{j-1} \frac{N_i}{N}$ is therefore reduced to $(j-1) * \frac{(\frac{N}{numwin})}{N} = \frac{(j-1)}{numwin}$. N_j in (5.20) is also reduced to $\frac{N}{numwin}$. This reduces (5.20) to:

$$\frac{\int_{x_{low}}^{x} g(x)d(x)}{\int_{x_{low}}^{x_{high}} g(x)d(x)} * \frac{1}{numwin} \tag{5.21}$$

In equation (5.21), since $g(x) = c$, the indefinite integral of $g(x)$ is equal to $G(x) = c\ x$. The definite integral of $g(x)$ between x' and x'' is defined by:

$$\int_{x'}^{x''} g(x)d(x) = G(x'') - G(x') = c\ x'' - c\ x' = c(x'' - x') \tag{5.22}$$

Using (5.22), equation (5.21) is reduced to:

$$\frac{c(x - x_{low})}{c(x_{high} - x_{low})} * \frac{1}{numwin} = \frac{(x - x_{low})}{(x_{high} - x_{low})} * \frac{1}{numwin} \tag{5.23}$$

As a result, $sel(b_{\#} < x)$ is equal to:

$$\frac{(j-1)}{numwin} + \frac{(x - x_{low})}{(x_{high} - x_{low})} * \frac{1}{numwin} \tag{5.24}$$

This formula (5.24) is exactly the same as the one (2.11) in page 49. Note that this formula is based on the uniform scheme which is originally proposed by Muralikrishna and DeWitt [1988] and is found superior by Muralikrishna and DeWitt [1988]; Harangsri et al. [1998] to the formula based on the half scheme. (See Section 2.5.2 in Chapter 2 for more details about the uniform and half schemes.)

If one looks closely at formula (5.24), given that the number of windows used is 1 and so $numwin = 1$ and $j = 1$, the formula will reduce to:

$$\frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{5.25}$$

where $x_{high} = x_{max}$ and $x_{low} = x_{min}$. This is exacty the same as the formula for the same predicate $(b_{\#} < x)$ used by UNF.

In fact, UNF is a special case of histograms with only one fitting window. When there is only one window used for a global regression, formula (5.24) will reduce to (5.25).

In addition, if the fitting constant c in the single window is equal to $\frac{N}{d}$ (as a result of the equal number of tuples for any distinct value of $b_{\#}$), then $sel(b_{\#} = x)$ would be $\frac{1}{d}$ which is another formula for the predicate $b_{\#} = x$ used by UNF.

All adaptive *numwin* windows of HIST use a single identical fitting constant, say c, i.e., share the same fitting constant c across all the windows [Piatetsky-Shapiro and Connell 1984]. The single fitting constant c, divided by N, namely $\frac{c}{N}$ (which represents $sel(b_{\#} = x)$) could be either the *attribute density* (defined below), $\frac{1}{d}$, or etc.

Piatetsky-Shapiro and Connell [1984] (the original work for HIST) proposed that the attribute density be used for $sel(b_{\#} = x)$ instead of $\frac{1}{d}$ which is used by UNF. The attribute density is defined by: $\frac{\sum_{j=1}^{d} f_j^2}{N^2}$ where f_j is the frequency of distinct value x_j. The attribute density was used in Piatetsky-Shapiro and Connell's experiments and the authors justified that the attribute density takes an unequal frequency distribution of $b_{\#}$ into consideration (if there is any) while the formula $\frac{1}{d}$, which is based on the uniform distribution, always presumes that each distinct value in the domain of $b_{\#}$ has the same frequency (number of tuples) to appear in relation R. In all experiments in this chapter with HIST, we follow Piatetsky-Shapiro and Connell's proposal.

Due to the adaptiveness of each equi-height histogram bucket, the number of parameters per bucket (window) needed to be maintained in the database profile catalog is 1, which is the upper bound value for a bucket. Like M5, the upper bound value is also maintained in the catalog for a local adaptive window used by M5.

Table 5.1 shows a summary of how many parameters per window are needed to be maintained in the profile catalog for the different methods described above.

method	fitting type	window type	parameters per window	what parameter ?
IASE	fd	fixed-width	4	a_0, a_1, a_2, N_i
ASE	cfd	fixed-width	3	a_0, a_1, a_2
LWR	cfd	fixed-width	3	a_0, a_1, a_2
M5	cfd	adaptive	3	a_0, a_1, an upper bound value
NN*	cfd	one window	8	weights, biases, a learning rate
HIST	fd	adaptive	1	an upper bound value
UNF*	fd	one window	-	-

fd = frequency distribution, cfd = cumulative frequency distribution
* = global regression only

Table 5.1: Fitting and window types and number of parameters required per window.

5.5 Implementation

Figure 5.6 is the data structure we propose for local regression so that all different local regression methods: IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A), MaxDiff(V,A) and etc. can be implemented together in a database system under a single framework. Each record of the data structure keeps information of a local regression method used by an attribute of relation R.

Below we describe that such a structure can be used in order to deal with any unseen, perhaps difficult-to-fit data distributions in future. The main flexibility of the structure is that we can control (1) the degree of the polynomial used, i.e.,

```
struct est_scheme {
  int      est_method;
  int      numwin;
  int      wintype;
  int      poly_deg;
  double   coeff[0 .. numwin-1][0 .. poly_deg];
  double   upbound[0 .. numwin];
  int      base_est_func_equi;
  int      base_est_func_less;
}
```

Figure 5.6: A data structure for the implementation of many local regression methods under a single framework

`poly_deg` in the figure and (2) the window type, i.e, `wintype` (fixed, nearest neighbour or adaptive).

Loader [1997*c*] provides a local regression software package called **locfit** on his home page with the three kinds of user-selected bandwidths: fixed, nearest neighbour and adaptive so that users can experiment it with their own data. Furthermore, he demonstrated with success the strengths of local regression with many real-life data.

The reason for the proposal of the structure is that we have observed that some local regression methods are good for some data distributions and others are good for some other data distributions. In other words, we believe that there is no universal method which is optimal for all kinds of data distributions. However, with the combination of many variants of local regression, we can hope to deal with any unseen data distributions that can occur in future. The reason is similar to that of the **locfit** software package in changing bandwidths and polynomial degrees (2 or 3 normally) in order to suit the data at hand.

Here are a few advantages of why the data structure would be able to cope with any unseen future data distributions:

- Many times we have noticed that polynomial degree 3 can assist in building more accurate single-dimensional regression models but of course, with more storage to pay for.

- Loader [1997*b*] commented on when to use each window type. For example, when data (x's value) are clumped in some of the areas over the entire attribute domain, the nearest neighbour window would normally be the best choice. See also Chapter 9 on Bandwidth Selection on the web [Loader 1997*a*].

- There are times that adaptive windows by M5 contribute to the improvement of single-dimensional regression models. This is perhaps as a result of the nice rationale in building adaptive windows by the partitioning algorithm of M5 (see the algorithm in page 70 of Chapter 2). Such a rationale is similar to the rationale of MaxDiff(V,A) histograms [Poosala et al. 1996; Poosala and Ioannidis 1997] in grouping distinct values into buckets which attempts to avoid grouping the distinct values with vastly different frequencies into the same bucket.

- If the data at hand is of the uniform distribution, then the structure with `numwin`=1 with the global constant fitting can be used. This implies that the UNF method is used for estimation.

- If the data at hand is of the Zipf distribution [Zipf 1949] which was claimed to occur most frequently in real-life databases, by observing from many experiments with different forms of the Zipf distribution, LWR is the most suitable choice.

Here is the description of how one can use the structure. Table 5.2 shows 3 examples of `est_scheme` records in Figures 5.2(a) and 5.2(b) which can be uploaded to the database profile catalog of a database system. Let us describe the records.

The first record is for the LWR method which uses 4 windows for estimation. The window type `wintype` is fixed-width. The degree of a polynomial used is 2 for all local windows. With this estimation method, the field `upbound` for upper bound values is not applicable which is denoted by NA. The base estimation function for the "<" operator is estimation function 3 as shown in Table 5.2(c). The base estimation function for the "=" operator is function 3 as well. We use the term `base` such as in `base_est_func_equi` to imply that for the "=" operator, the estimation can be done by sharing the same base function 3 as the one used by the "<" operator, namely, $\frac{g_2(x+\triangle)-g_2(x)}{N}$. The p value in (where p = `poly_deg`) is the polynomial degree to be passed to the corresponding estimation function.

field	value	value
`est_method`	LWR	M5
`numwin`	4	4
`wintype`	fixed	adaptive
`poly_deg`	2	1
`coeff`	a list of coeffs with degree 2	a list of coefficients with degree 1
`upbound`	NA	a list of upper bound values
`base_est_func_equi`	3 (where p = `poly_deg`)	1 (where p = `poly_deg`)
`base_est_func_less`	3 (where p = `poly_deg`)	1 (where p = `poly_deg`)

(a) est_scheme records 1 and 2

field	value
`est_method`	HIST
`numwin`	12 (buckets)
`wintype`	adaptive
`poly_deg`	0
`coeff`	NA
`upbound`	a list of upper bound values
`base_est_func_equi`	1 (where p = `poly_deg`)
`base_est_func_less`	2 (where p = `poly_deg`)

(b) est_scheme record 3

#	base_est_func_{equi,less}
1	$g_1(x) = \sum_{i=0}^{p} a_i x^i$
2	$G_1(x) = \sum_{i=0}^{p} \frac{a_i x^{(i+1)}}{(i+1)}$
3	$g_2(x) = \sum_{i=0}^{p} a_i \frac{(x-fitpnt)^i}{i!}$

(c) Base estimation functions

Table 5.2: 3 est_scheme records and base estimation functions

The second record is for M5. The window type `wintype` built by M5 is adaptive and the polynomial degree used is 1 for all local windows. With this method, one needs to have a list of upper bound values `upbound` (from which to locate to which window x in a predicate ($b_{\#}$ *relopt* x) belongs) stored in the database profile catalog. The base estimation function for the "=" and "<" operators is the same, sharing the same base function 1.

The third record is for HIST. The number of windows (buckets) used is 12. Equi-height histograms are adaptive so the window type `wintype` is adaptive. Since HIST is local constant fitting, the polynomial degree `poly_deg` used is zero in each local window. Due to the adaptiveness of equi-height histograms, one needs to have a list of upper bound values stored in the catalog. The estimation function for the "=" operator is function 1 where the polynomial degree passed to the function is p = `poly_deg`. The estimation function for the "<" operator is function 2 where the polynomial degree passed to the function is p = `poly_deg`.

Next is the question of how the records are created. Given data sets of the form $(x_i, f_i),\ i = 1, 2, \ldots, d$ for all attributes $\# = 1, 2, \ldots, u$ of relation R, the following is the idea in order to combine variants of local regression into the catalog.

The idea is to select the best method per attribute among IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A) and MaxDiff(V,A). Over all the methods (IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A), MaxDiff(V,A)), repeat one by one to build a single-dimensional model by a method for attribute $b_{\#}$. Then find out which method produces the best model by examining from its average square error [Poosala 1997] which is calculated by:

$$\sum_{i=1}^{d}(f_i - g(x_i))^2$$

where f_i is the frequency of the distinct value x_i and $g(x_i)$ is its estimate by the method. Whichever is the best model by a method, save the record of the method for its subsequent upload to the profile catalog.

For any other attributes of relation R, do the same thing as attribute $b_{\#}$ to build a model. Then all the best combined single-dimensional models will be ready to upload to the profile catalog.

5.6 Experimental results

We describe the generation of two sets of relations in Section 5.6.1. There are a few relations and their query sets newly generated in this chapter. This is for the sake of more variety and extensiveness of data used to test the robustness of each estimation method.

In Section 5.6.2, the aim is to compare local regression against global regression. For each of the methods: LWR, ASE and IASE, we will compare and see how the results are improved when increasing the number of windows used.

In Section 5.6.3, we aim at comparing among 7 available methods: IASE, ASE, LWR, M5, HIST, NN and UNF, based on the same number of parameters used. (Actually, NN and UNF are a global regression, namely, one window methods, and thus the number of parameters used by them is always fixed.) In other words, all the methods will be strictly compared using the same number of parameters. Queries used here for the comparison are all with simple predicates.

In Section 5.6.4, the aim is similar to Section 5.6.3 except the queries used. The queries used here are with complex predicates in conjunctive normal form. All the descriptions made in Section 5.6.3 are hence applicable to this section.

Except the global regression (one window) experiments in Section 5.6.2, the polynomial degree p locally used in each window for methods LWR, ASE and IASE is 2 throughout all experiments.

Basically the experiments for the UNF method are conducted so as to be base results and hence, the results of other estimation methods can be compared with the base checkpoint results.

5.6.1 Experimental setup

We first describe the generation of two sets of synthetic relations. Then at the end of the section, to evaluate the efficiency of each estimation method, we use the three error measures defined previously but with a slight modification for the purpose here.

Table 5.3 shows different kinds of data distributions and their necessary parameters used throughout all experiments. We repeat this table here again as there are a few more new notations needed to be defined and for ease in reference to the notations used in Table 5.4 for relation configurations.

Notation	Meaning
norm($mean, \delta$)	normal distribution with mean $mean$ and standard deviation δ
chisq(df)	chi-square distribution with degrees of freedom df
unf($low, high$)	uniform distribution with
	low the lowest random value to be generated
	$high$ the highest random value to be generated
fdist(df_1, df_2)	F distribution with
	degrees of freedom for numerator df_1
	degrees of freedom for denominator df_2
zipf($NumDist$,z)	$NumDist$ the expected number of distinct values
	z, the values between 0 to 1
semizipf($NumDist$,$z = 0.5$)	$NumDist$ the expected number of distinct values
	this is a special case of the zipf distribution where $z = 0.5$
exp(av)	exponential distribution with mean av
bimod($mean_i, \delta_i,\ i = 1, 2$)	bi-modal distribution by two overlapping normal distributions
trimod($mean_i, \delta_i,\ i = 1, 2, 3$)	tri-modal distribution by three overlapping normal distributions

Table 5.3: Parameters for each distribution.

There are two sets of relations generated. The first set is shown in Table 5.4. The purpose of this set is to test the efficiency of each estimation method on the relations configured with a variety of data distributions (F, normal, exponential and so on) by using simple predicate queries. In the table, some of the relations with *'s are newly generated in this chapter and the rest are simply taken from [Harangsri et al. 1997].

The first set is used in Sections 5.6.2 which is the test between local regression

relation	data distribution parameter	# of distinct values	card N
R-unf*	unf(30, 363)	333	10,000
R-exp*	exp(70)	340	10,000
R-norm	norm(200,150)	539	10,000
R-chi	chisq(10)	38	20,000
R-bimod	bimod((250,150), (450,50))	530	12,500
R-trimod*	trimod((198,43), (348,43), (498,43))	540	10,000
R-semizipf*	semizipf(350,0.5)	350	10,000
R-zipf*	zipf(240,0.6)	240	10,000

Table 5.4: Relations with different configurations.

against global regression and is also used in Section 5.6.3 which is the test for simple predicate queries.

The purpose of the second set of relations is to test the efficiency of each estimation method by using complex predicate queries and this is demonstrated in Section 5.6.4. Created via functional dependencies, the second set of relations together with their query sets (with complex predicates) is reused from Table 3.23 in Chapter 3 (see the table near page 153).

Here we slightly modified the three error measures as follows. The modification is the change of $\hat{\mu}_i$ to sel_ind_i and of $\tilde{N}$ to N (see also the original error measures in Figure 3.6 near page 133 for comparison). The first change is due to the current consideration of selection selectivities based on the independence assumption, in place of the join selectivities. The second change is due to the current consideration of a relation cardinality, in place of the cartesian product of all the cardinalities of the relations which participate in a star join.

Root Mean Square Error (rms) is defined as: $\sqrt{\sum_{i=1}^{|\mathcal{Q}|} \frac{(sel\_ind_i * N - \overline{Y}_i N)^2}{|\mathcal{Q}|}}$ where $\mathcal{Q}$ is a query set, $|\mathcal{Q}|$ is the total number of queries in the set (400 queries used in all experiments), $\overline{Y}_i$ the actual selectivity of a query Q_i, sel_ind_i an estimated selectivity of the query, based on the attribute independence assumption, calculated by the equation in (2.2) in Chapter 2, N the cardinality of the relation on which the query is posed, as well as $sel\_ind_i * N$ and $\overline{Y}_i N$ an estimated result size and the actual result size of the query, respectively.

Mean Residual Error (mrsd) is defined as: $\frac{\sum_{i=1}^{|\mathcal{Q}|} abs(sel\_ind_i * N - \overline{Y}_i N)}{|\mathcal{Q}|}$

Mean Relative Error (mrlt) is defined as: $\frac{\sum_{i=1}^{|\mathcal{Q}|} 100 * \frac{abs(sel\_ind_i * N - \overline{Y}_i N)}{\overline{Y}_i N}}{|\mathcal{Q}|}$

5.6.2 Local against global regression

The aim of this set of experiments is to show that local regression does a better job in fitting data rather than global regression. The relations in Table 5.4 are used to conduct experiments, together with their simple predicate query sets (one set with 400 queries is for one relation).

Three local regression methods, LWR, ASE and IASE are compared. Recall that local regression with one window used, i.e., the largest bandwidth, is the global regression. More windows (smaller bandwidths) used tend to improve query result sizes (i.e., with lower errors). However, too many windows (too small bandwidths) will possibly sometimes make the results too noisy and thus poorer [Cleveland and Loader 1996; Loader 1997*b*]. This is chiefly because there are insufficient numbers of data points falling into the local windows.

In Tables 5.5 and 5.6 we increment by 1 the number of windows used to fit a data set from 1 to 5 windows, namely, from the largest bandwidth (1 window) to the smallest bandwidth (5 windows). Most of the results shown in the tables demonstrate that local regression for all the three methods is, in general, superior to their global regression in fitting the same data set. However, there are some experiments such as the results by LWR for R-norm for 4 and 5 windows, and the ones by LWR and ASE for R-bimod for 4 and 5 windows. The errors for 5 windows of both R-norm and R-bimod are higher than the ones for 4 windows. This could be as the consequence of "too noisy" fitting for the 5 windows, which then results in worse errors (instead of improved errors).

For one window used (global regression case), the errors in Table 5.5(a) are based on global regression by:

- polynomial degree $p = 6$ for ASE and LWR. For ASE, the polynomial is $g(x) = \sum_{i=0}^{p} a_i x^i$ and for LWR, the polynomial is $g(x) = \sum_{i=0}^{p} \frac{a_i(x-fitpnt)^i}{i!}$. This degree of polynomial was proposed and used in the original work [Chen and Roussopoulos 1994].

- polynomial degree $p = 10$ for IASE, where $g(x) = \sum_{i=-p_1}^{p2} a_i x^i$ where $p = p_1 + p_2$ and $p_1 > 0, p_2 > 0$. This degree of polynomial was also used in the original work [Sun et al. 1993].

error	LWR	ASE	IASE
	R-unf		
rms	2403	3078	3214
mrsd	1417	1842	2013
mrlt	88	105	92
	R-exp		
rms	5673	4812	5186
mrsd	3696	3148	3363
mrlt	1767	1525	985
	R-norm		
rms	3516	770	36
mrsd	2213	481	33
mrlt	146	54	58
	R-chi		
rms	7179	6547	1466
mrsd	4469	4711	1381
mrlt	315	138	55
	R-bimod		
rms	2662	324	236
mrsd	1754	211	102
mrlt	70	22	426
	R-trimod		
rms	3738	3297	3533
mrsd	2237	1911	2179
mrlt	3187	3217	450
	R-semizipf		
rms	5182	3831	4209
mrsd	3520	2483	2771
mrlt	444	371	130
	R-zipf		
rms	5282	4000	4666
mrsd	3642	2724	3208
mrlt	373	317	153

(a) 1 window

error	LWR	ASE	IASE
	R-unf		
rms	42	108	70
mrsd	27	68	42
mrlt	8	28	6
	R-exp		
rms	97	355	71
mrsd	46	173	36
mrlt	9	96	10
	R-norm		
rms	87	118	50
mrsd	58	64	35
mrlt	17	15	13
	R-chi		
rms	866	1370	542
mrsd	662	1194	418
mrlt	23	38	29
	R-bimod		
rms	526	452	399
mrsd	303	257	203
mrlt	30	22	17
	R-trimod		
rms	126	144	158
mrsd	78	95	100
mrlt	19	31	26
	R-semizipf		
rms	84	319	149
mrsd	43	192	79
mrlt	7	53	8
	R-zipf		
rms	112	314	171
mrsd	63	201	95
mrlt	9	47	10

(b) 2 windows

error	LWR	ASE	IASE
	R-unf		
rms	35	66	59
mrsd	22	42	38
mrlt	5	26	6
	R-exp		
rms	52	283	54
mrsd	23	161	25
mrlt	6	55	7
	R-norm		
rms	102	72	27
mrsd	63	46	18
mrlt	21	14	13
	R-chi		
rms	738	1290	441
mrsd	530	1109	334
mrlt	22	36	9
	R-bimod		
rms	186	229	122
mrsd	99	120	83
mrlt	14	15	11
	R-trimod		
rms	81	84	91
mrsd	48	59	55
mrlt	14	30	14
	R-semizipf		
rms	71	203	90
mrsd	35	128	44
mrlt	7	35	6
	R-zipf		
rms	92	173	126
mrsd	50	110	64
mrlt	8	36	7

(c) 3 windows

Table 5.5: Three kinds of errors for 1, 2 and 3 windows.

For more than one window (local regression case), all the three methods use polynomial degree $p = 2$. With the same number of windows used, IASE uses one more parameter per window than LWR and ASE for the value of N_i, i.e., the total number of tuples which fall into window i. By the degree $p = 2$, the usual values in a local window used by all the three methods are coefficients, a_0, a_1 and a_2.

5.6.3 Simple predicate queries

The aim of experiments in this section is to demonstrate that with simple predicate queries, which of the 7 available methods gives the best results. The relations and their query sets used in Section 5.6.2 are reused for experiments here. Since in practical database systems, the number of parameters used by a method is significant to the success of the method – such a number reflects how much storage is required for the method, all the comparisons here are strictly based on the same number of parameters used.

Assuming that the number of distinct values, maximum and minimum values for all attributes of all the relations in a database are the basic usual parameters maintained in any database profile catalog of any database system, the UNF method

error	LWR	ASE	IASE
	R-unf		
rms	27	74	70
mrsd	18	47	40
mrlt	6	27	6
	R-exp		
rms	43	213	52
mrsd	19	132	24
mrlt	5	40	5
	R-norm		
rms	17	23	28
mrsd	12	17	21
mrlt	14	12	12
	R-chi		
rms	483	1295	288
mrsd	281	1101	218
mrlt	9	35	7
	R-bimod		
rms	94	90	127
mrsd	58	55	76
mrlt	12	10	10
	R-trimod		
rms	42	94	63
mrsd	26	63	38
mrlt	31	29	12
	R-semizipf		
rms	55	154	70
mrsd	27	99	33
mrlt	7	30	5
	R-zipf		
rms	80	152	92
mrsd	39	103	48
mrlt	8	30	7

(a) 4 windows

error	LWR	ASE	IASE
	R-unf		
rms	27	44	44
mrsd	18	36	27
mrlt	7	26	6
	R-exp		
rms	38	121	51
mrsd	18	86	23
mrlt	6	34	5
	R-norm		
rms	29	22	24
mrsd	18	15	19
mrlt	14	11	12
	R-chi		
rms	228	1291	419
mrsd	172	1094	298
mrlt	6	35	7
	R-bimod		
rms	141	150	56
mrsd	78	83	35
mrlt	12	12	8
	R-trimod		
rms	24	58	41
mrsd	15	42	25
mrlt	19	26	10
	R-semizipf		
rms	50	82	80
mrsd	24	58	39
mrlt	6	27	4
	R-zipf		
rms	77	128	74
mrsd	36	81	38
mrlt	7	28	5

(b) 5 windows

Table 5.6: Three kinds of errors for 4 and 5 windows.

requires no parameter to be maintained.

Table 5.1 in Section 5.4 shows the number of parameters per window used by the different methods. Consult the table also for the calculation below. Given a certain number of parameters we want to maintain in a database profile catalog, here we show how many windows each method can have. For LWR, ASE and M5, they all require 3 parameters per window; if we want 12 parameters to be maintained in the profile catalog, then all the methods can have 4 windows (namely, $\frac{12}{3}$). For IASE, since 4 parameters are needed (one extra parameter is N_i for the window), it can only have 3 windows ($\frac{12}{4}$) in order to have 12 parameters maintained. For HIST, an equi-height histogram requires only 1 parameter per window (or bucket); thus, it can have 12 windows for 12 parameters to be maintained. The calculation for 15 parameters can be done in the same fashion as that for 12 parameters. The errors for 12 and 15 parameters are shown respectively in Tables 5.7 and 5.8.

By the calculation shown above, the results with 12 parameters of IASE in Table 5.7 are the same ones for 3 windows of IASE as shown in Table 5.5(c). Likewise, the results with 15 parameters of IASE in Table 5.8 are the same ones for 4 windows of IASE as shown in Table 5.6(a).

For NN, the architecture of the neural network used is 3-layer network (see Figure 5.4) with 1 input neuron in the input layer for x_i's values and 1 output neuron for $f(x_i)$'s values and 4 neurons in the hidden layer to perform a nonlinear regression in the network. The method is required to maintain 14 parameters for weights, biases and a learning rate as described in Section 5.4.1.5. We trained each network (one network is for one single-dimensional regression model) for 10000 iterations; that is, 1 iteration is one pass through the whole data set. A learning rate of 0.001 is used throughout a network — the connection from the input to hidden layer and the connection from the hidden to output layer both use the same learning rate 0.001. With the very low learning rate, we hope that any network used will learn slowly but gradually improve its error while more iterations are gone through without getting stuck in local minima.

The errors for UNF and NN (both are global methods, i.e., single window methods) shown in Tables 5.7 and 5.8 are reproduced for ease in reference when comparing the errors.

For all the results shown in Tables 5.7 and 5.8, the trend from the best to worst performance is as follows. LWR performs the best. HIST, ASE and IASE perform equally well; each takes turn to be better than the other two but generally they perform quite similarly. M5 performs slightly worse than the three. NN is next from M5 and UNF as expected performs worst.

The reason LWR performs the best could be as a consequence of the polynomial used and different weights assigned to data points. The reason the three: HIST, ASE and IASE performs similarly could be because they use the same number of parameters and same type of polynomial (see more description about HIST in the next paragraph). The reason M5 performs slightly worse than the three could be because of the low fitting power by the polynomial degree 1 used inside windows. NN does not perform very nicely; the reason could be that it is difficult and perhaps time-consuming to train a network to find the optimal weights over the network. UNF is a parametric method which always assumes that any distinct value in the domain would have the same frequency to appear in the relation. Hence, the only relation with which UNF performs satisfactorily is R-unf !!.

Our observation reveals that HIST would have performed much worse than LWR,

error	LWR	HIST	ASE	IASE	M5	NN	UNF
	R-unf						
rms	27	32	74	59	44	228	60
mrsd	18	20	47	38	28	145	38
mrlt	6	5	27	6	7	43	6
	R-exp						
rms	43	96	213	54	126	186	2411
mrsd	19	71	132	25	70	101	1494
mrlt	5	120	40	7	20	24	269
	R-norm						
rms	17	78	23	27	336	134	1165
mrsd	12	36	17	18	166	86	742
mrlt	14	39	12	13	25	17	82
	R-chi						
rms	483	662	1295	441	752	1544	4534
mrsd	281	534	1101	334	489	1045	3062
mrlt	9	54	35	9	22	104	135
	R-bimod						
rms	94	111	90	122	563	682	1795
mrsd	58	59	55	83	270	398	1115
mrlt	12	37	10	11	24	28	66
	R-trimod						
rms	42	92	94	91	103	330	379
mrsd	26	48	63	55	55	206	232
mrlt	31	39	29	14	18	183	83
	R-semizipf						
rms	55	48	154	90	53	227	1218
mrsd	27	36	99	44	30	147	782
mrlt	7	32	30	6	6	30	33
	R-zipf						
rms	80	103	152	126	69	236	1594
mrsd	39	71	103	64	40	162	1061
mrlt	8	50	30	7	7	25	47

Table 5.7: Three kinds of errors for 12 parameters.

ASE and IASE if we had compared them by the number of windows used in place of the number of parameters. This is generally because of the insufficient fitting power of the polynomial degree 0 used by histograms. This also implies that fitting data by histograms really does need more windows in order to make curves inside buckets become straight enough (not much curvy) so that each of the more straight curves can be fitted by the average frequency of the bucket. Compared with methods like LWR, ASE and IASE which can make use of the fitting power by a higher polynomial degree for their fitting perhaps non-straight-line curves inside buckets, histograms much more rely on the number of windows used than do the three methods.

5.6.4 Complex predicate queries

The aim of this last set of experiments is similar to that for simple predicate queries demonstrated in Section 5.6.3 – test the efficiency among 7 available methods. The differences are twofold. The first is that queries used here are complex in conjunctive normal form. The second difference is that relations on which to test these complex queries are the ones created via functional dependencies from Table 3.23 near page 153.

The errors in Tables 5.9 and 5.10 can be interpreted in the same fashion as in

error	LWR	HIST	ASE	IASE	M5	NN	UNF
	R-unf						
rms	27	26	44	70	44	228	60
mrsd	18	16	36	40	28	145	38
mrlt	7	5	26	6	7	43	6
	R-exp						
rms	38	73	121	52	126	186	2411
mrsd	18	57	86	24	70	101	1494
mrlt	6	117	34	5	20	24	269
	R-norm						
rms	29	56	22	28	336	134	1165
mrsd	18	25	15	21	166	86	742
mrlt	14	37	11	12	25	17	82
	R-chi						
rms	228	594	1291	288	752	1544	4534
mrsd	172	496	1094	218	489	1045	3062
mrlt	6	53	35	7	22	104	135
	R-bimod						
rms	141	79	150	127	563	682	1795
mrsd	78	47	83	76	270	398	1115
mrlt	12	35	12	10	24	28	66
	R-trimod						
rms	24	69	58	63	103	330	379
mrsd	15	37	42	38	55	206	232
mrlt	19	34	26	12	18	183	83
	R-semizipf						
rms	50	49	82	70	53	227	1218
mrsd	24	37	58	33	30	147	782
mrlt	6	32	27	5	6	30	33
	R-zipf						
rms	77	106	128	92	69	236	1594
mrsd	36	74	81	48	40	162	1061
mrlt	7	50	28	7	7	25	47

Table 5.8: Three kinds of errors for 15 parameters.

Tables 5.7 and 5.8. Also consult the descriptions in Section 5.6.3 for more details about each method.

For all the results shown in Tables 5.9 and 5.10, the trend from the best to worst performance is similar to the experiments for simple predicate queries (see above also). LWR performs the best. HIST, ASE, IASE and M5 perform equally well; each takes turn to be better than the other two but generally they perform quite similarly. NN is next from the four and UNF as expected performs worst.

5.7 Why local regression would assist in obtaining accurate join selectivities

For join selectivity estimation by local regression, the same method as the one we proposed for M5 in Section 2.7.4 of Chapter 2 can be used to approximate join selectivities. Although we have not conducted any experiments to confirm the effectiveness of local regression in approximating join selectivities – the experiments demonstrated above in Section 5.6 is solely for selection selectivity estimation –, the following is an attempt to show that if an estimation method is accurate for selections in approximating selection selectivities, then it should also very likely be

error	LWR	HIST	ASE	IASE	M5	NN	UNF
	R1-10k						
rms	550	774	547	548	550	582	1552
mrsd	220	333	222	224	229	307	851
mrlt	66	140	69	69	74	98	1691
	R2-10k						
rms	270	353	272	278	281	351	919
mrsd	152	189	159	164	165	247	520
mrlt	77	48	89	59	146	283	106
	R3-10k						
rms	197	217	269	259	246	493	477
mrsd	121	128	175	159	152	354	283
mrlt	82	157	109	152	35	239	206
	R4-10k						
rms	63	65	94	122	125	298	182
mrsd	38	40	59	70	74	225	103
mrlt	7	7	8	9	83	109	10
	R1-50k						
rms	2330	3409	2337	2337	2353	2492	11052
mrsd	804	1282	838	842	850	1173	5616
mrlt	1093	1850	1094	1095	1099	1113	8266
	R2-50k						
rms	169	178	295	377	619	1257	473
mrsd	106	116	189	223	298	910	306
mrlt	9	7	10	8	13	92	9
	R3-50k						
rms	2345	2616	2360	2363	2364	2600	7516
mrsd	1156	1270	1217	1216	1206	1645	4147
mrlt	150	138	151	151	150	177	1740
	R4-50k						
rms	1736	2071	1754	1767	1765	1998	3698
mrsd	961	1106	988	1008	1011	1367	2304
mrlt	222	228	226	225	225	254	454
	R1-100k						
rms	1943	1645	2083	2124	2131	3175	4664
mrsd	1042	863	1165	1181	1279	2318	2435
mrlt	175	88	168	168	193	218	102
	R2-100k						
rms	1855	2268	1906	1935	1925	2796	5152
mrsd	977	1093	1059	1087	1080	2071	2574
mrlt	30	27	30	30	31	69	37
	R3-100k						
rms	3978	5683	4423	4392	4124	4971	16887
mrsd	2218	2510	2663	2549	2406	3285	9606
mrlt	570	473	591	581	591	624	2081
	R4-100k						
rms	556	578	842	1181	1082	3060	2024
mrsd	350	372	517	642	604	2283	1165
mrlt	16	7	15	12	41	195	15

Table 5.9: Three kinds of errors for 12 parameters.

error	LWR	HIST	ASE	IASE	M5	NN	UNF
	R1-10k						
rms	557	668	553	555	558	582	1552
mrsd	230	274	231	234	242	307	851
mrlt	49	154	52	52	57	98	1691
	R2-10k						
rms	306	346	306	308	317	351	919
mrsd	158	185	159	165	172	247	520
mrlt	73	64	92	67	152	283	106
	R3-10k						
rms	190	200	262	253	246	493	477
mrsd	115	115	173	150	152	354	283
mrlt	75	120	105	102	35	239	206
	R4-10k						
rms	54	62	71	102	125	298	182
mrsd	35	39	47	60	74	225	103
mrlt	9	6	7	9	83	109	10
	R1-50k						
rms	2315	3410	2321	2327	2349	2492	11052
mrsd	886	1147	910	929	945	1173	5616
mrlt	1664	1704	1664	1665	1671	1113	8266
	R2-50k						
rms	156	170	198	281	619	1257	473
mrsd	98	111	136	174	298	910	306
mrlt	8	7	9	8	13	92	9
	R3-50k						
rms	2210	2575	2204	2214	2238	2600	7516
mrsd	1021	1113	1045	1067	1091	1645	4147
mrlt	124	117	124	124	124	177	1740
	R4-50k						
rms	1589	1936	1588	1593	1611	1998	3698
mrsd	826	1046	837	848	872	1367	2304
mrlt	240	209	242	241	242	254	454
	R1-100k						
rms	1768	1760	1788	1886	1996	3175	4664
mrsd	948	867	969	1091	1234	2318	2435
mrlt	131	57	132	133	159	218	102
	R2-100k						
rms	1946	2360	1967	1971	2007	2796	5152
mrsd	974	1156	1022	1032	1083	2071	2574
mrlt	24	24	25	25	25	69	37
	R3-100k						
rms	3683	5377	3964	3722	3765	4971	16887
mrsd	1870	2349	2173	1951	2032	3285	9606
mrlt	139	565	234	208	220	624	2081
	R4-100k						
rms	500	534	754	967	1082	3060	2024
mrsd	319	339	457	597	604	2283	1165
mrlt	16	9	15	11	41	195	15

Table 5.10: Three kinds of errors for 15 parameters.

accurate in approximating join selectivities.

We first show to use local regression to estimate join selectivities in Section 5.7.1 and by the end of the section we will justify why local regression would indeed help to obtain accurate join selectivities. We then give an additional evidence in Section 5.7.2 that previous studies, e.g., in [Poosala 1997] also used the similar approach as used by local regression to estimate join selectivities.

5.7.1 How to use local regression to estimate join selectivities

In Figure 5.7 is the reproduction of the graph originally used by M5 from Figure 2.13 of Section 2.7.4 in Chapter 2 for the description here.

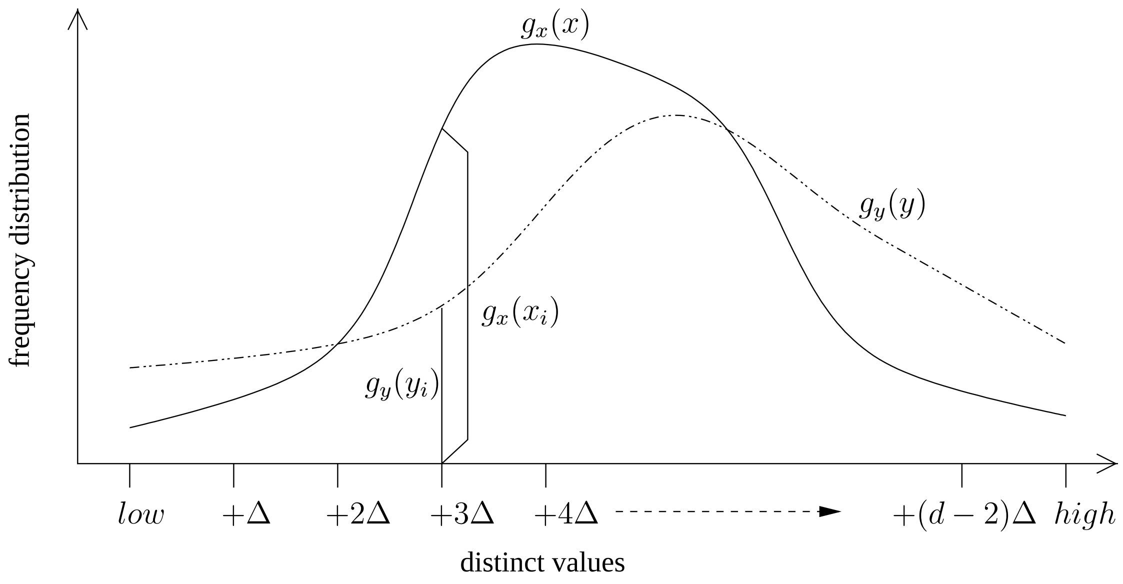


Figure 5.7: Join selectivity calculation

Consider a natural join between attributes $R_1.b_x$ and $R_2.b_y$. Let x_{max} and x_{min} be the maximum and minimum value of attribute $R_1.b_x$, respectively. Likewise, y_{max} and y_{min} are the respective maximum and minimum value of attribute $R_2.b_y$. low in Figure 5.7 is the larger value between x_{min} and y_{min} and $high$ is the smaller value between x_{max} and y_{max}. d_x is the number of distinct values of attribute $R_1.b_x$ and d_y is the number of distinct values of attribute $R_2.b_y$. d is the smaller value between d_x and d_y. N_{R_1} and N_{R_2} are the number of tuples of R_1 and R_2, respectively. The increment value Δ shown in the figure is calculated by: $\frac{(high-low)}{(d-1)}$.

$sel(R_1.b_x \bowtie R_2.b_y)$, the selectivity of a natural join between $R_1.b_x \bowtie R_2.b_y$ is thus calculated by:

$$sel(R_1.b_x \bowtie R_2.b_y) = \frac{\sum_{i=1}^{d} g_x(x_i) * g_y(x_i)}{(N_{R_1} * N_{R_2})} \tag{5.26}$$

where $x_i = low + (i-1) * \Delta$. The height of the two graphs – denoting the approximate frequency distributions of the two join attributes – basically denotes the number of tuples from $R_1.b_x$ and that from $R_2.b_y$ which are joinable — having a common value. In the figure, we use the notations $g_x(x_i)$ and $g_y(y_i)$ as for the height of the two graphs.

As one can see, inherently to obtain an accurate estimated join selectivity in (5.26), one requires two approximating functions $g_x(x_i)$ and $g_y(x_i)$ which very well capture the frequency distributions of attributes $R_1.b_x$ and $R_2.b_y$, respectively. These two functions can be obtained from any of the methods, e.g., IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A), MaxDiff(V,A), and etc, all of which are subsumed under local regression.

In Section 5.5, we have proposed an idea to select the best method among IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A) and MaxDiff(V,A). That is, over all the methods (IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A), MaxDiff(V,A)), repeat one by one to build a function $g_x(x_i)$ by the method for attribute b_x (and likewise a function $g_y(x_i)$ for attribute b_y). Then find out which method produces the best function by examining from its average least square error which is calculated by:

$$\sum_{i=1}^{d_x} (f_x(x_i) - g_x(x_i))^2 \tag{5.27}$$

where $f_x(x_i)$ is the frequency of the distinct value x_i and $g_x(x_i)$ is its estimate by the method. One of those methods with the least square error will give the "optimal" function $g_x(x_i)$ (and likewise the "optimal" function $g_y(x_i)$).

We use the most accurate functions $g_x(x_i)$ and $g_y(x_i)$ as suggested by local regression to compute a join selectivity, on the basis that those functions produce the most accurate selection selectivities. This should then assist in obtaining an accurate estimated join selectivity.

Considered above is a binary join. For other k-way joins where $k \geq 2$, one must also rely on the accuracy of frequency distribution approximating functions $g_{b_j}(x_i)$'s for all attributes b_j's in the k-way join for the estimation of a join selectivity. Thus, if all such functions are accurate, namely, giving the minimum least square error, then the estimated join selectivity would also be accurate.

5.7.2 Sharing the same histograms for selection and join selectivities

Poosala [1997] also used the idea of sharing the same approximating frequency distribution functions for both selection and join selectivities. That is, he used his serial histograms which are equivalent to the approximating frequency distribution functions $g_x(x_i)$ and $g_y(x_i)$ above, to approximate both selection and join selectivities, like the method proposed above in Section 5.7.1. The proof in his thesis is that serial histograms are optimal for natural join and selection selectivities, i.e., compared with many other types of histograms considered, serial histograms give the lowest square error [1] in approximating natural join and selection selectivities. Note that the square error is defined in (5.27) and also that we have already defined the optimality of a histogram by equation 2.8 near page 39 of Chapter 2.

Poosala [1997]'s approach above fits very well with our proposal that one should share the same approximating frequency distribution functions to the estimation of both join and selection selectivities. But since local regression can suggest the best method which produces the "optimal" approximating function with the least square error among all local regression methods considered such as IASE, ASE, LWR, M5, HIST, UNF, V-Optimal(V,A), MaxDiff(V,A), and etc, local regression should then win in the estimation of both join and selection selectivities with higher accuracy than the estimation by serial histograms.

5.8 Conclusion

Local regression has been proposed to improve the quality of query size estimation for selection queries. The following are the main achievements contributed by this chapter:

- Local regression proves to provide more robust quesy size estimation than the global one.

- Never before in the literature of non-sampling methods for query size estima-

[1]This lowest square error is just in comparison with different types of histograms only.

tion have comparisons been as comprehensive as the ones demonstrated in this chapter.

- For the first time, we have applied the standard backpropagation neural network method to the query size estimation problem. This has been done as a case study to manifest its performance to our problem domain.
- A generic data structure has been proposed so as to (1) make use of different strengths of local regression methods in the same place of a database system and (2) prepare in advance for any future unseen, perhaps difficult-to-fit data distributions which can happen in data stored in databases. This is the first and foremost attempt to combine and mix several variants of local regression into a single framework.

CHAPTER 6

Query optimisers that use sampling methods

Abstract

Some query optimisation researchers have suggested that sampling incurs too much run-time overhead to be a feasible technique for selectivity estimation in a query optimiser, even though it can generate more accurate estimates than other approaches. In this chapter, we aim to address these concerns and demonstrate that sampling is not only *feasible*, but can also be a reasonably *efficient* and a very *effective* technique in query size estimaton for real query optimisers.

First for the feasibility issue, it is crucial to address that the total time spent for sampling for selectivity estimaton particularly for joins (as a join is a most time-consuming operation) just slightly interferes in the total time to execute a join query which consists of selection and join operations.

For selections in a join query, although it is not too hard to see that sampling for selection selectivities is likely to be done, we even have an evidence that the sampling is already in practical use by a commercial database system. For joins in the join query we will show by a simplified analysis that a sampling fraction can be selected such that on-line sampling to estimate all necessary join selectivities will just slightly interferes in the total time to execute a join query.

Second for the efficiency and effectiveness issue, we then proceed to demonstrate that our more accurate size estimaton technique HYBRID can generally yield cheaper query execution plans than a less accurate technique SRSWR, by using join queries with 4-5 relations involved in the queries.

The last finding in this chapter is that the cost of the resampling technique in resampling to obtain more accurate join selectivities is acceptable and thus the technique should be introduced for the practical use by database systems.

In this chapter, we describe a query optimiser that uses sampling as selectivity estimation. We first give the overall cost of query execution in Section 6.1 before we begin on the major section of the chapter which is the introduction.

6.1 The overall cost of query execution

A multi-stage process is involved in the execution of a join query, as we described in Chapter 1. The query is first parsed into an internal form, which is itself a naive execution plan. This is then transformed to an "optimal" execution plan which is passed to the database engine to be evaluated. An example of a join query is shown in Figure 1.2 of Chapter 1.

The overall cost of executing a query is then determined by the cost of performing the optimisation and the cost of actually evaluating the query execution plan to obtain the final result. Let us define:

`cost(eval')` = cost of executing query without optimisation
`cost(opt)` = cost of determining the "optimal" query plan
`cost(eval)` = cost of executing the "optimal" query plan

It is generally known that the difference in amount between the two costs `cost(eval')` and `cost(eval)` can be several orders of magnitude. That is, the cost `cost(eval')` in the worst case, namely, the cartesian product, can be very expensive. Thus, it is no doubt that any database system would need a query optimiser in order to quest for at least a plan which is far from the worst-case query plan. As a consequence, there are only `cost(opt)` and `cost(eval)` to be considered for a given query and we will no longer consider `cost(eval')`.

In this chapter, we consider the query optimisation algorithm that employs an *exhaustive search* algorithm as the approach to searching everywhere in a search space to find the "optimal" plan. The reason we consider the exhaustive search instead of other search algorithms is that:

- First in this chapter, one of the issues about which we want to ascertain is that a more accurate query size estimation technique would generally result in selection of cheaper query execution plans than the plans resulting from a less accurate estimation technique.

Unlike the limited search, since the exhaustive search does not cut down any part of the entire search space from consideration, there would be no side effect as a consequence of the cut-down search space. Recall the limited search algorithms (described in Chapter 1) cuts down part of the entire search space and only consider a portion of the entire space.

The view of good and bad plans in the entire space by the two estimation techniques compared in the same query optimiser can be different. The cut-down search space may disregard some good or even the optimal plans from consideration which would otherwise be selected by the query optimiser using the size estimation technique (one of the two).

Thus through the use of the exhaustive search, the comparison results obtained by the two estimation techniques would be without the side effect of the cut-down search space.

- Second, if the number of relations involved in join queries is low (less than or equal to 7), then exhaustive search would be applicable. That is, the query optimiser will not spend excessive time for optimising the queries.

  In addition, all experiments in this chapter are only conducted by using join queries with 4–5 relations involved in the queries.

With regard to the term "optimal" plan, given the use of the exhaustive search by the optimiser, since the optimiser must rely on *estimated* (as opposed to *actual*) selectivities in order to determine the optimal plan, this implies that the "optimal" plan obtained may or may not be truly optimal – depending upon the quality of estimated selectivities produced by the selectivity estimation method used. Recall that the estimation of a plan cost is calculated from the sum of each subplan cost whose calculation in turn relies on estimated selectivities. Clearly we need a good estimation method for an accurate plan cost estimation.

We noted above that the total cost in executing a join query consists of:

$$\texttt{cost(opt)} + \texttt{cost(eval)} \tag{6.1}$$

Let m be the number of relations involved in a join query. The exhaustive search algorithm considers the entire search space to find the "optimal" execution plan for a join query. If the number of relations involved in the join query is not too large ($m \leq 7$), then the entire search space considered by the query optimiser will be small. This small search space will contain a small number of plans, namely $\leq 7!$ plans, considered by the query optimiser.

However, when $m > 7$, then the exhaustive search will not be practical and many more recent limited search algorithms described in Chapter 1 will replace the exhaustive search algorithm and play the role to find a near-optimal plan.

New limited search algorithms like Simulated Annealing [Ioannidis and Wong 1987; Swami and Gupta 1988; Swami 1989*b*,*a*; Ioannidis and Kang 1990] will attempt to aggressively cut down the very large search space, i.e., with $m \geq 10$ involved in a join query, to a small search space. This small search space will contain a small number of plans considered by the query optimiser.

In summary, regardless of any search algorithm (limited or exhaustive) used by a query optimiser, the total number of plans considered by the optimiser will be kept *small* but yet likely to incorporate many low-cost plans in the small search space.

Therefore, the CPU time cost can be ignored to iterate over the small number of plans to find a near-optimal or the "optimal" plan because compared with the *more weighted* I/O cost incurred by `cost(eval)`, the CPU time cost will be negligible. Hence, in what follows we will no longer consider the CPU time cost as part of `cost(opt)`.

However, in each iteration of the limited or exhaustive search, `cost(opt)` is still involved with the cost for selectivity estimation by a selectivity estimation method used. Here are two scenarios for `cost(opt)` when non-sampling-based and sampling-based selectivity estimation methods are used for selectivity estimation.

6.1.1 Non-sampling-based methods

The majority of the cost of non-sampling-based methods occurs off-line, in the collection and maintenance of the statistical information used by the method. Size estimation by these methods would have a very small and *negligible* cost at run-time, assuming that:

- the statistical information is held in memory which is the case for many non-sampling-based methods, e.g., histogram, ASE, IASE, etc.

- and the total number of execution plans considered by the query optimiser is small.

If we wish to compare two non-sampling-based estimation methods, A and B, to prove that A is better than B, using (6.1) for the total cost of a join query, we start by considering whether:

$$\texttt{cost(opt[A])} + \texttt{cost(eval[A])} \; < \; \texttt{cost(opt[B])} + \texttt{cost(eval[B])}$$

Since both estimation methods are employed by the same search algorithm (implying that the same search space is considered) and if the total number of plans considered by the optimiser is small, then it would be reasonable to disregard the following two terms on the basis that they will both be similar,

$$\texttt{cost(opt[A])} \;\approx\; \texttt{cost(opt[B])}$$

and they are fairly small, compared with `cost(eval)`, namely:

$$\texttt{cost(opt[}\mathit{method}\texttt{])} \; << \; \texttt{cost(eval[}\mathit{method}\texttt{])}$$

where *method* is either A or B and << is the symbol for "much less than". That is, the `cost(eval[A])` and `cost(eval[B])` will have much more weight than `cost (opt[A])` and `cost(opt[B])` because the first two are involved with doing I/O operations, while the second two are involved with computation in memory for selectivities. Thus, the overall comparison between A and B reduces to:

$$\texttt{cost(eval[A])} \; < \; \texttt{cost(eval[B])} \tag{6.2}$$

In other words, we are simply interested in which method produces the "best" query execution plan.

6.1.2 Sampling-based methods

The `cost(opt)` for sampling-based methods clearly includes the sampling procedure which entails disk I/O operations. Since this must be performed on-line, this cost component can no longer be considered insignificant.

If HYBRID, a sampling-based method, and UNF, a non-sampling-based method, are compared in the same query optimiser, then one should consider:

$$\texttt{cost(opt[HYBRID])} + \texttt{cost(eval[HYBRID])} < \texttt{cost(opt[UNF])} + \texttt{cost(eval[UNF])}$$

to justify that HYBRID generally outperforms UNF. But since the `cost(opt[UNF])` of UNF is considered a negligible cost as justified by the two reasons above, the inequality would then reduce to:

$$\texttt{cost(opt[HYBRID])} + \texttt{cost(eval[HYBRID])} < \texttt{cost(eval[UNF])} \tag{6.3}$$

If HYBRID and SRSWR (or SRSWOR), a sampling-based method are compared, then one should look at:

$$\texttt{cost(opt[HYBRID])} + \texttt{cost(eval[HYBRID])} < \texttt{cost(opt[SRSWR])} + \texttt{cost(eval[SRSWR])} \tag{6.4}$$

to justify that HYBRID generally outperforms SRSWR. The inequality (6.4) can however, be further reduced to:

$$\texttt{cost(eval[HYBRID])} < \texttt{cost(eval[SRSWR])} \tag{6.5}$$

if the sampling for both selection and join selectivities by both approaches is done with the same sampling fraction. (It makes sense to compare both of them using the same sampling fraction.) Note that this inequality (6.5) is similar to the inequality (6.2) that we derived for non-sampling-based methods. Note also that we are not required to use the same sampling fraction for selections and joins; it is sufficient that the sampling fraction for selections be the same for each approach, and similarly the sampling fraction for joins.

It is one of the main aims in this chapter to confirm (6.3) and (6.5).

6.2 Introduction (structure of presentation)

The following is a logical series for the presentation in this chapter.

We first give the background of how join selectivities are calculated by sampling and UNF methods in Section 6.3.

Olken [Olken 1993] mentioned in his thesis that Antoshenkov [Antoshenkov 1993] uses sampling for the estimation of selectivities of range selection predicates in Rdb/VMS, a commercial database system. This indicates that there exists a practical use of on-line sampling for selection selectivity estimation. This further suggests that sampling for selection selectivities is feasible to be used in real database systems.

Some query optimisation researchers, e.g., [Ioannidis and Poosala 1995; Ioannidis 1993; Chen and Roussopoulos 1994] question about how feasible it is to use sampling particularly for the estimation of join selectivities (as a join is a most time-consuming operation). This is due to that fact that the query optimiser has to consult the selectivity estimator frequently for both join and selection selectivities. We will show by a simplified analysis in Section 6.4 that a sampling fraction can be selected such that the total cost for all necessary sampling would be much less than the total query execution cost. This further implies that the total sampling time will only slightly interfere in the total query execution time.

Such an analysis is important to give a reasonable upper bound on the sampling fraction (i.e., maximum sampling fraction) used. Any sampling-based join selectivity estimator should never perform sampling beyond this upper bound because the overall time spent for optimising join queries can be excessive. To our knowledge, to date there has been no such analysis in the literature of query optimisation and selectivity estimation.

Working from our simplified analysis, we propose in Section 6.5 two possible approaches to the implementation of a query optimiser which uses on-line sampling for the selectivity estimation. The first approach is called *semi-dynamic* and the second is called *fully dynamic*. We also discuss the advantages and disadvantages of each approach. In the current work here, we will only evaluate the performance of the former approach. Therefore two selectivity estimation sampling-based methods

HYBRID and SRSWR considered here will be employed by the query optimiser with the former approach.

In Section 6.6, we consider the exhaustive search as the query optimisation algorithm. The reason we select this search algorithm was described earlier in Section 6.1. As mentioned in Chapter 1, even though the search space is small and the optimiser employs the exhaustive search, the optimiser can produce more costly query plans if the selectivity estimation method used is inaccurate. We compare UNF (a less accurate estimation method) and HYBRID in the query optimiser with the exhaustive search algorithm by using a small size of 5-relation databases so as to see the effectiveness in producing query execution plans of each estimation method.

In Section 6.7, we describe how to calculate the cost of a query execution plan generated by the exhaustive search algorithm, both for the estimated and actual cost of a plan. The estimated cost of a plan is used by the query optimiser to determine the optimal plan in the search space whereas the actual cost, which is observed while executing the query, is used for comparison between two selectivity estimation methods in the same query optimiser.

Since a major main aim in this chapter is to test whether one selectivity estimation method is superior to another, all of the experimental setup must be identical except the two selectivity estimation methods used in the individual query optimisers. The factors that we have to control are: the search strategy (exhaustive search in both cases), the databases, the query sets and the join algorithm. Among many join algorithms, e.g., nested loop, sort merge and hash (see a very extensive survey of join algorithms in [Mishra and Eich 1992]), we choose the hash-join algorithm [DeWitt et al. 1984] as the method to join relations. The reason is that the hash-join algorithm has been shown to work well with many kinds of databases (see the study of three variants of hash-join algorithms in [Shapiro 1986] for example); it does not rely on the sortedness of data; and it has a low algorithm complexity, $O(|R_1| + |R_2|)$ (i.e., only needs a single pass through each relation R_1 and R_2 to compute the join). Hence, calculating the actual/estimated cost for a join, we base the calculation on the complexity of the hash-join algorithm.

In each step of the calculation for the estimated/actual cost of a query plan, a temporary intermediate relation will be produced. Section 6.8 shows how to create schemas for the temporary relations produced. Calculating the estimated/actual cost of a plan requires the knowledge of the schemas of intermediate temporary relations produced in each step of the calculation. Let us define a notation $attrlen($an (intermediate) relation) as the sum of all attributes' lengths in the (intermediate) relation.

As described in Section 6.1.2, the sampling cost for selection and join selectivities incurred by any sampling-based method must be added on to the actual cost of executing the join query if one were to compare between a sampling-based and non-sampling-based method. We describe how to add it in Section 6.9.

In Section 6.10, using the query optimiser with the exhaustive search, we conduct a set of experiments whose aims are as follows.

- A query optimiser with a better selectivity estimation method used can in general produce cheaper query execution plans. We demonstrate this by comparing (1) HYBRID and UNF and (2) HYBRID and SRSWR.
- With the resampling technique, the cost in resampling for more accurate join selectivities is relatively low, compared with the total of query execution costs. Thus it is practical to introduce the technique into actual use of query optimisers by real database systems.
- The resampling technique for join selectivities can assist in improving query execution plans.

We propose that HYBRID be implemented in database systems as the selectivity estimation method of choice. The first reason is that HYBRID can exploit any sortedness of the data to produce better query size estimates. Second the experimental results both for join and selection selectivity estimation in Chapter 3 and in this chapter for the quality of query execution plans, demonstrate the capability of HYBRID.

Section 6.11 contains possible future directions that we plan to work on.

6.3 Notations and background for join selectivities

Given a database D with $|D|$ relations, $R_1, R_2, \ldots, R_{|D|}$ and given a star join with m $(2 \leq m \leq |D|)$ participating relations $(R_{i_1} \bowtie R_{i_2} \bowtie \cdots \bowtie R_{i_m})$, let $R_{i_1,i_2,\ldots,i_m}$

be the output relation of the star join, where $1 \leq i_j \leq |D|,\ j = 1, 2, \ldots, m$. The selectivity $\mu_{i_1,i_2,\ldots,i_m}$ of the star join is defined by:

$$\mu_{i_1,i_2,\ldots,i_m} = \frac{|R_{i_1,i_2,\ldots,i_m}|}{|R_{i_1}||R_{i_2}|\cdots|R_{i_m}|} \tag{6.6}$$

6.3.1 Heuristic procedure: performing selections before joins

In this section, since the proposal for the query optimiser prototype in this chapter relies on a heuristic procedure which is widely used in many current database systems [Ullman 1988*a*; Elmasri and Navathe 1991; Ozsu and Valduriez 1991; Hellerstein and Stonebraker 1993], we will justify why the heuristic procedure is reasonable. The heuristic procedure is that selections are always done first before any joins; that is, relations would be reduced by the selection predicates in the given query prior to proceeding to do any joins in the query.

This heuristic procedure is *generally reasonable* to be based on and used by database systems although it is not always the case; for example, it was reported in [Hellerstein and Stonebraker 1993] that some expensive selection predicates, e.g., functions implemented in a general-purpose programming language such as C or C++, cause the use of the heuristic to produce more expensive plans.

Still the heuristic procedure is reasonable for many database query classes which are not involved with the use of expensive functions because it helps to reduce the complexity of the query optimisation problem – makes the problem tractable. The following is a justification and an example.

Given an example of a join query between R_1 and R_2 with two simple predicates $pred_1$ and $pred_2$ on relations R_1 and R_2, respectively, if we use the heuristic procedure, then the query is only executed by:

- reduce the two relations by $\sigma_{pred_1}(R_1) \rightarrow R_1'$ and $\sigma_{pred_2}(R_2) \rightarrow R_2'$, and join $R_1' \bowtie R_2'$ to produce the final result.

However, if the selections can be done before or after the join, then here are three more alternatives to execute this simple join query:

- $R_1 \bowtie R_2 \rightarrow R_{12}$ and reduce $\sigma_{(pred_1 \text{ and } pred_2)}(R_{12})$.
- Reduce $\sigma_{pred_1}(R_1) \rightarrow R_1'$, join $R_1' \bowtie R_2 \rightarrow R_{12}'$, and reduce $\sigma_{pred_2}(R_{12}')$.

- Similarly, reduce $\sigma_{pred_2}(R_2) \rightarrow R'_2$, join $R_1 \bowtie R'_2 \rightarrow R'_{12}$, and reduce $\sigma_{pred_1}(R'_{12})$.

The number of alternatives above each of which can produce the same final result is still quite a lot, considering that this is just to process a simple join query. Imagine that if the number of relations in a given join query is more than 2 and there are more selections in the query, then the number of alternatives will become extremely large.

6.3.2 Star joins after applying selections (actual results)

Let $\breve{R}_{i_j}$ be a reduced relation as a result of applying a reduction by the selection predicate on relation $R_{i_j}, j = 1, 2, \ldots, m$. Given a star join after the reductions $(\breve{R}_{i_1} \bowtie \breve{R}_{i_2} \bowtie \cdots \bowtie \breve{R}_{i_m})$, let $\breve{R}_{i_1,i_2,\ldots,i_m}$ be the output relation of the join among the reduced relations. The size of $\breve{R}_{i_1,i_2,\ldots,i_m}$ is calculated by:

$$|\breve{R}_{i_1,i_2,\ldots,i_m}| = \breve{\mu}_{i_1,i_2,\ldots,i_m} * (|\breve{R}_{i_1}||\breve{R}_{i_2}| \cdots |\breve{R}_{i_m}|) \tag{6.7}$$

where $\breve{\mu}_{i_1,i_2,\ldots,i_m}$ is the selectivity of this star join. In actuality, while optimising for the optimal execution plan for a join query, the optimiser will never know the *actual star join selectivity* $\breve{\mu}_{i_1,i_2,\ldots,i_m}$, not until the database system has finished executing the star join and observed the output size $|\breve{R}_{i_1,i_2,\ldots,i_m}|$. It is a main function of any query optimiser to use any kind of selectivity estimation methods in order to approximate such a star join selectivity.

In comparing between two estimation methods, say A and B, in a query optimiser to see which one outperforms in producing cheaper execution plans, we need to know the intermediate relation sizes $|\breve{R}_{i_1,i_2,\ldots,i_m}|$'s, generated in each step of a join. The sizes of all the intermediate relations partly form a query execution plan cost for an estimation method used (any of the two). Two query execution plan costs by the two methods can then be compared.

6.3.3 Star joins after applying selections (estimated results)

Let $\widehat{R}_{i_j}$ be a reduced estimated relation as a result of applying a reduction by the selection predicate on relation $R_{i_j}, j = 1, 2, \ldots, m$. Given a star join $(\widehat{R}_{i_1} \bowtie \widehat{R}_{i_2} \bowtie \cdots \bowtie \widehat{R}_{i_m})$ among the estimated relations, let $\widehat{R}_{i_1,i_2,\ldots,i_m}$ be the output estimated relation of the join. The size of $\widehat{R}_{i_1,i_2,\ldots,i_m}$ is calculated by:

$$|\widehat{R}_{i_1,i_2,\ldots,i_m}| = \hat{\mu}_{i_1,i_2,\ldots,i_m} * (|\widehat{R}_{i_1}||\widehat{R}_{i_2}|\cdots|\widehat{R}_{i_m}|) \tag{6.8}$$

where $\hat{\mu}_{i_1,i_2,\ldots,i_m}$ is the estimated selectivity of this star join. Since we typically do not know this selectivity, we need some method for estimating it. The following two sections 6.3.4 and 6.3.5 describe two approaches to estimating the star join selectivity $\hat{\mu}_{i_1,i_2,\ldots,i_m}$.

6.3.4 Estimated join selectivities by a sampling method

Let R'_{i_j} be a sample relation of relation $R_{i_j}, j = 1, 2, \ldots, m$. Given a star join $(R'_{i_1} \bowtie R'_{i_2} \bowtie \cdots \bowtie R'_{i_m})$ among the sample relations, let $R'_{i_1,i_2,\ldots,i_m}$ be the output relation of the join. The selectivity $\mu'_{i_1,i_2,\ldots,i_m}$ of this star join is defined by:

$$\mu'_{i_1,i_2,\ldots,i_m} = \frac{|R'_{i_1,i_2,\ldots,i_m}|}{|R'_{i_1}||R'_{i_2}|\cdots|R'_{i_m}|} \tag{6.9}$$

This is an approach to the estimated selectivity $\hat{\mu}_{i_1,i_2,\ldots,i_m}$.

6.3.5 Estimated join selectivities by UNF

Based on the uniform distribution assumption [Selinger et al. 1979*a*; Swami and Schiefer 1994] and given a star join $(\widehat{R}_{i_1} \bowtie \widehat{R}_{i_2} \bowtie \cdots \bowtie \widehat{R}_{i_m})$, Figure 6.1 shows the calculation towards the estimated selectivity $\hat{\mu}_{i_1,i_2,\ldots,i_m}$ of the join. Using the uniform distribution assumption, suppose that the number of distinct values in relation R_{i_j} is equal to d_j, where $j = 1, 2, \ldots, m$ and thus the number of tuples per distinct value would be $\frac{|R_{i_j}|}{d_j}$. In Figure 6.1, each element under a column freq.(R_{i_j}) represents the number of tuples per distinct value $\frac{|R_{i_j}|}{d_j}$, where $j = 1, 2, \ldots, m$. Also let $d = \min(d_1, d_2, \ldots, d_m)$.

Given an ith distinct value, where $i = 1, 2, \ldots, d$, the total number of tuples in the output relation as a result of the join on this distinct value among the m relations is calculated by:

$$\frac{|R_{i_1}|}{d_1} * \frac{|R_{i_2}|}{d_2} * \cdots * \frac{|R_{i_m}|}{d_m} \text{ which is equal to: } \prod_{j=1}^{m} \frac{|R_{i_j}|}{d_j}$$

dist.	freq.(R_{i_1})	freq.(R_{i_2})	$\cdots$	freq.(R_{i_m})	tuples per dist.
1	$\frac{\|R_{i_1}\|}{d_1}$	$\frac{\|R_{i_2}\|}{d_2}$	$\cdots$	$\frac{\|R_{i_m}\|}{d_m}$	$\prod_{j=1}^{m} \frac{\|R_{i_j}\|}{d_j}$
2	$\frac{\|R_{i_1}\|}{d_1}$	$\frac{\|R_{i_2}\|}{d_2}$	$\cdots$	$\frac{\|R_{i_m}\|}{d_m}$	$\prod_{j=1}^{m} \frac{\|R_{i_j}\|}{d_j}$
.	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
.	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
d	$\frac{\|R_{i_1}\|}{d_1}$	$\frac{\|R_{i_2}\|}{d_2}$	$\cdots$	$\frac{\|R_{i_m}\|}{d_m}$	$\prod_{j=1}^{m} \frac{\|R_{i_j}\|}{d_j}$
	total tuples				$d * \prod_{j=1}^{m} \frac{\|R_{i_j}\|}{d_j}$

Figure 6.1: The estimated total number of tuples based on UNF

as shown under the "tuples per dist." column in Figure 6.1.

Note that the calculation in the figure uses the full relations R_{i_j}'s, not the reduced estimated relations $\widehat{R}_{i_j}$'s. This is due to the *independence assumption* between the join and selection operations [Christodoulakis 1984] used by System-R's query optimiser[1] — join selectivities are estimated independently of selection selectivities. That is, irrespective of whatever has resulted by earlier selections on some relations, the estimated join selectivity $\hat{\mu}_{i_1,i_2,\ldots,i_m}$, which is calculated from the full relations, can be used to approximate the size of the output relation $\widehat{R}_{i_1,i_2,\ldots,i_m}$.

The independence assumption between join and selection operations in a join query can be implied from the *usual attribute independence assumption* that any attribute in a relation is independent of one another. Given a join query which consists of (1) a join operation with a join attribute in the relation and (2) a simple selection predicate on another attribute of the same relation, these two attributes (one join attribute and one selection attribute) will also be treated independently of each other, like other attributes in this relation. The approximate frequency distributions of each attribute in this relation would thus be maintained independently of one another for selection and join selectivity estimation. Due to the attribute independence assumption used, the estimated size of the join operation after the selection in the given join query will not take account of the dependence between the two attributes (even if there is) and will be worked out from the individual approximate frequency distributions of the two attributes.

Using the estimated total number of tuples in Figure 6.1 (the last row in the

[1]UNF was proposed as the query size estimator of System-R's query optimiser.

figure), the selectivity $\hat{\mu}_{i_1,i_2,\ldots,i_m}$ for the star join $(\widehat{R}_{i_1} \bowtie \widehat{R}_{i_2} \bowtie \cdots \bowtie \widehat{R}_{i_m})$ would be:

$$\hat{\mu}_{i_1,i_2,\ldots,i_m} = \frac{d * \prod_{j=1}^{m} \frac{|R_{i_j}|}{d_j}}{\prod_{j=1}^{m} |R_{i_j}|} = \frac{d}{\prod_{j=1}^{m} d_j} \tag{6.10}$$

In fact, the formula (2.7) which we have derived earlier for join selectivity near page 37 of Chapter 2 is subsumed under the formula here.

6.4 Sampling cost vs subplan cost

In this section, we will illustrate, by a simple example, the relative costs of sampling and executing a subplan. We will also show that:

Statement 6.1 ***A sampling fraction can be selected such that the sampling cost for a subplan would be much less than the execution cost for the subplan.***

We define the sampling cost and execution cost for a subplan below. If the statement is correct, then it is reasonable to draw a conclusion that:

- The total cost for all necessary sampling for selectivity estimation would be much less than the total execution cost for a join query.
- As a result of the first point, the total sampling time will only slightly interfere in the total query execution time.

For illustration purposes throughout the chapter, we consider a database with 5 relations each with a star join attribute on it. All possible star joins (namely, combinations of star joins in the database) which could occur in user queries are shown in Table 6.1. Let all the relations in the database have the same cardinality x.

Given a join query $(R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4 \bowtie R_5)$ and given also that the query optimiser uses a sampling method to estimate selectivities, to select what is the optimal order in executing the query, the join selectivities for all combinations of the star joins as shown in Table 6.1 must be approximated by sampling. Let the following be the optimal order in executing the join query specified by the optimiser:

$$((((R_1 \bowtie R_3) \bowtie R_4) \bowtie R_2) \bowtie R_5)$$

rels	combinations	total
2	$R_1R_2, R_1R_3, R_1R_4, R_1R_5$ $R_2R_3, R_2R_4, R_2R_5, R_3R_4$ R_3R_5, R_4R_5	$C_2^5 = 10$
3	$R_1R_2R_3, R_1R_2R_4, R_1R_2R_5$ $R_1R_3R_4, R_1R_3R_5, R_1R_4R_5$ $R_2R_3R_4, R_2R_3R_5, R_2R_4R_5$ $R_3R_4R_5$	$C_3^5 = 10$
4	$R_1R_2R_3R_4, R_1R_2R_3R_5$ $R_1R_2R_4R_5, R_1R_3R_4R_5$ $R_2R_3R_4R_5$	$C_4^5 = 5$
5	$R_1R_2R_3R_4R_5$	$C_5^5 = 1$
	total	26

Table 6.1: All combinations of star joins for a 5-relation database

Consider a subplan of the whole order, $((R_1 \bowtie R_3) \bowtie R_4)$. If it is worth doing a sampling for this subplan, then this:

$$\text{samp. cost for subplan } ((R_1 \bowtie R_3) \bowtie R_4) \; << \; \text{exe. cost for the subplan } ((R_1 \bowtie R_3) \bowtie R_4)$$

should hold and it is also reasonable to say:

$$\overbrace{\mu_{123}(\beta x)^3 + \mu_{124}(\beta x)^3 + \mu_{125}(\beta x)^3 + \cdots + \mu_{345}(\beta x)^3}^{\text{samp. cost for the subplan } ((R_1 \bowtie R_3) \bowtie R_4)} \quad << \quad \overbrace{\mu_{134}x^3}^{\text{exe. cost for the subplan}} \qquad (6.11)$$

That is, the sum of sizes for all the output relations each of which is created by a sampling with the sampling fraction β for one of the star joins as shown in Table 6.1 where rels = 3, should be much less than the size of the output relation created by executing the subplan $((R_1 \bowtie R_3) \bowtie R_4)$.

(6.11) can be arranged to:

$$(\mu_{123} + \mu_{124} + \cdots + \mu_{345})(\beta x)^3 \quad << \quad \mu_{134}x^3 \qquad (6.12)$$

Let:

$$\bar{\mu} = \frac{(\mu_{123}+\mu_{124}+\cdots+\mu_{345})}{C_3^5}$$

The inequality (6.12) can thus reduce to:

$$C_3^5\bar{\mu}(\beta x)^3 << \mu_{134}x^3 \qquad (6.13)$$

Now consider the right hand side of the inequality (6.13). In fact, any subplan with 3 relations other than the subplan $((R_1 \bowtie R_3) \bowtie R_4)$ can also be selected by the query optimiser as part of the entire order. In other words, any star joins with 3 relations as shown in the table can be selected by the query optimiser as the optimal subplan for the join query given. That is, all of the following inequalities are also correct:

$$\begin{aligned}
C_3^5\bar{\mu}(\beta x)^3 &<< \mu_{123}x^3 \\
C_3^5\bar{\mu}(\beta x)^3 &<< \mu_{124}x^3 \\
C_3^5\bar{\mu}(\beta x)^3 &<< \mu_{125}x^3 \\
\cdots\cdots &<< \cdots \\
C_3^5\bar{\mu}(\beta x)^3 &<< \mu_{345}x^3
\end{aligned} \tag{6.14}$$

if the optimal subplans selected by the optimiser are: $((R_1 \bowtie R_2) \bowtie R_3)$, $((R_1 \bowtie R_2) \bowtie R_4), ((R_1 \bowtie R_2) \bowtie R_5), \ldots, ((R_3 \bowtie R_4) \bowtie R_5)$, respectively. The left hand side of (6.14) is all the same no matter what optimal subplan is selected.

To further simplify all the inequalities in (6.14) for more analysis, we can use the average case for all the inequalities. That is, we can use $\bar{\mu}$ as the representative selectivity value for all the selectivities on the right hand side, i.e, $\mu_{123}, \mu_{124}, \mu_{125}, \ldots, \mu_{345}$. We can then reduce all the inequalities (6.14) to:

$$\overbrace{C_3^5\bar{\mu}(\beta x)^3}^{\text{samp. cost for a subplan}} \quad << \quad \overbrace{\bar{\mu}x^3}^{\text{exe. cost for the subplan}}$$

which is reduced to: $C_3^5\beta^3 << 1$. Multiplying both sides by 100 to turn them to percentage, this gives:

$$\overbrace{(100 * C_3^5\beta^3)\%}^{\text{samp. cost for a subplan in \%}} \quad << \quad \overbrace{100\%}^{\text{exe. cost for the subplan in \%}} \tag{6.15}$$

(6.15) says that if it is worthwhile doing a sampling for a subplan, then a sampling fraction β selected should make $100 * C_3^5\beta^3\%$ (the left-hand side) much less than 100% (the right-hand side). In fact, by the average case the inequality (6.15) is also correct no matter what subplan is considered.

We can then generalise that with a database of m relations and with a subplan of y relations involved, if it is worthwhile doing sampling for the subplan, then a sampling fraction β selected should make:

$$\overbrace{(100 * C_y^m \beta^y)\%}^{\text{samp. cost for a subplan in }\%} \quad << \quad \overbrace{100\%}^{\text{exe. cost for the subplan in }\%} \tag{6.16}$$

Using the left-hand side of formula (6.16), we create Table 6.2 where $m = 10$ and the sampling fraction β ranges from 3, 5, 7, 10, 12 and 15 %, respectively. The purpose of the table is to give us an idea of which sampling fraction should be selected as the upper bound for the sampling fraction to be used by the query optimiser. Any sampling fraction beyond the bound should not be attempted because the query optimiser may spend excessive time for optimising join queries.

		sampling cost for a subplan in % = $(100 * C_y^m \beta^y)$ %					
y	C_y^m	$\beta = 0.03$	$\beta = 0.05$*	$\beta = 0.07$	$\beta = 0.10$	$\beta = 0.12$	$\beta = 0.15$
2	45	4.05	11.25	22	45	64.8	101.25
3	120	.3240	1.5	4.116	12	20.736	40.5
4	210	.01701	.13125	.50421	2.1	4.35	10.63
5	252	.00061	.007875	.04235	.252	.62705	1.91
6	210	.15309e-4	.000328	.00247	.021	.06271	.23920
7	120	.26244e-6	.9375e-5	.0000988	.0012	.00430	.02050
8	45	.29525e-8	.17578-6	.25942e-5	.000045	.00019	.00115
9	10	.19683e-10	.1953e-8	.40354e-7	1e-6	.51598e-5	.38443e-4
10	1	.59049e-13	.97656-11	.28248e-9	1e-8	.61917e-7	.57665e-6

Table 6.2: Sampling cost for a subplan in percent

Let us describe Table 6.2. The first column y shows the number of relations involved in a subplan and the second column C_y^m shows the number of all combinations for the star joins with y relations involved. Given an example, when $\beta = 0.03$ (3% sampling fraction) and $y = 2$, the number of all combinations for 2-relation star joins is $C_2^{10} = 45$ and the sampling cost for the subplan with 2 relations is 4.05%.

We can draw the following conclusion from Table 6.2.

- The more relations y involved in a subplan, the less the sampling cost for the subplan. The reason is that the growth of β^y grows considerably faster than the growth of the number of star join combinations C_y^m. The sampling cost in percent dramatically decreases when y increases from 2 to 10. See the decrease under any sampling fraction column.

- The sampling cost for 2 relations involved, namely, $y = 2$ is the most expensive, especially when the sampling fraction is more than 5%, i.e., $\beta > 0.05$.

  Consider the first row when $y = 2$ from left to right. When $\beta = 0.03$, the sampling cost is 4.05%; when $\beta = 0.05$, the sampling cost is 11.25%; when $\beta = 0.07$, the sampling cost is 22% and so on. The clear trend is that with the subplan with two relations involved ($y = 2$), the sampling cost increases from left to right, i.e., when increasing the sampling fraction.

  Note that the sampling cost for a subplan can be so large that this cost itself is even more than the execution cost of the subplan. When $y = 2$ and $\beta = 0.15$ (15%), the sampling cost for this subplan is 101.25% which is more than the execution cost.

  This number of relations $y = 2$ is very significant and very fundamental to any join queries because all join queries must begin from 2 relations involved. We believe that any sampling fraction selected should not incur more than 10% of the execution cost of any subplan in the database. As a result, any sampling fraction from 7% onwards (see the first row of the table where $y = 2$ for an example) should be avoided as the time spent for sampling can be excessive and hence, too much competes with the time to execute the query itself. (The aim in do sampling for selectivity estimation is not to interfere too much in the time spent to execute the query itself.) Although the 5% sampling fraction is a bit expensive – incurring the sampling cost 11.25% –, we believe that it still can be used in any database system and also for the sake of more accurate selectivities than any sampling fraction lower than 5%. Hence, we use the 5% sampling fraction for all experiments in Section 6.10 both for join and selection selectivity estimation.

- From the table, given that only 3 and 5% sampling fractions are considered to be used by a query optimiser (other sampling fractions incur excessive optimisation cost), any number of relations involved in a subplan with $y =$

$3, 4, \ldots, 10$, incurs an insignificant sampling cost. See the two columns under $\beta = 0.03$ and $\beta = 0.05$.

6.5 Semi-dynamic and fully dynamic approaches for query optimisers

In this section, we propose two approaches to the implementation of a query optimiser which uses sampling for selectivity estimation. The first is *semi-dynamic* and the second is *fully dynamic*.

Semi-dynamic approach

A join query we have defined consists of two operations in it: selection and join. The term *incoming query* used below is a join query.

The semi-dynamic approach works as follows:

- For the join operation, do resampling a number of times through any incoming queries to build all possible join selectivities that can occur in the database and store them in the profile catalog. With a small sampling fraction used, for each join selectivity, one may consider to do η-time resampling so that each join selectivity would be more reliable and accurate enough, where η is an integer whose value is more than or equal to 1.

  After all the join selectivities are built into the profile catalog, stop doing any sampling for them to any incoming queries and use the stored join selectivities in the profile catalog for optimising any incoming queries. Do the resampling process again periodically to update the join selectivities stored in the catalog.

- For the selection operation, use *one-time sampling* to estimate the selectivities for selection predicates in any incoming queries. Recall that one-time sampling is the typical sampling procedure proposed by all of previous sampling-based methods. As a result, despite a high bias as a consequence of a single sample used for approximating a selectivity, the selectivities for selections would always be up-to-date for being used by the query optimiser. Recall that resampling in the case of selections would be infeasible due to the exponential storage requirement to store all possible selection selectivities in the profile catalog.

With this approach, the selection and join operations are treated independently of one another. This is the independence assumption between the selection and join operations as used by many current database systems. Many current systems maintain approximate frequency distributions by single-dimensional histograms for the individual attributes of a relation. This clearly implies that two attributes in a join query, namely one join attribute and one selection attribute, from the same relation would have their own single-dimensional histograms from which the join selectivity and the selection selectivity of the two attributes are estimated independently of one another.

Fully dynamic approach

The fully dynamic approach works as follows:

- For both selection and join operations, use one-time sampling to estimate the selectivities for (1) selection predicates and (2) all combinations of star joins in a given join query. For example, if a join query is involved with 5 relations, then the selectivities for all combinations of star joins as shown in Table 6.1 must be approximated by the one-time sampling. Unlike the semi-dynamic approach, the reason one cannot use resampling for join selectivities for the fully dynamic approach would be described shortly below.

  As sampling is newly done every time a new join query arrives at the database system, the selectivities for both joins and selections would always be up-to-date for being used by the query optimiser but of course, with a high bias for each estimated selectivity used.

Using the typical heuristic procedure that selections are always done first before joins, relations would be reduced by selections prior to doing any joins. Given a join query, sampling would then work on the reduced relations in order to estimate the join selectivities for all combinations of star joins in the join query. With this approach, the selection and join operations would depend on one another because the sampling for join selectivities would work on the reduced relations, i.e., takes account of the results of selections in the query. (Unlike the semi-dynamic approach, sampling works on the original full relations for all join selectivities and does not take account of the results of selections in the query.) This is the *dependence assumption* between the selection and join operations.

By the dependence assumption used, it would be infeasible to use resampling for any join selectivity. The reason can be described as follows. Because join selectivities must be computed after the reductions of relations by selection predicates in a given join query and there are an *innumerable* number of possible reduced relations as a result of the reductions, the possible number of join selectivities on the reduced relations will also be *innumerable*. Similar to the exponential storage requirement for selection selectivities to be stored in the profile catalog, the storage requirement for join selectivities after selections would also be too large.

Semi-dynamic approach's advantages and disadvantages

Here are the advantage and disadvantage of the semi-dynamic approach:

Advantage The query optimiser runs, i.e., optimises a join query, much faster, perhaps several orders of magnitude, than the query optimiser that uses the fully dynamic approach, in finding an optimal query execution plan. This is because at optimisation time for a join query, we save the major sampling cost for join selectivities which are already stored in the profile catalog. (In general, the sampling cost for join selectivities would be far more than the sampling cost for selection selectivities.)

Disadvantage The query optimiser should, in principle, produce poorer query execution plans than the optimiser that uses the fully dynamic approach. First, this is due to the independence assumption that treats joins and selections independently of one another. Second, the join selectivities in the profile catalog can be out-of-date after some time. Third, there is a storage cost for a number of join selectivities stored in the catalog for which this approach has to pay.

Fully dynamic approach's advantages and disadvantages

Here are the advantage and disadvantage of the fully dynamic approach:

Advantage The query optimiser should, in principle, produce cheaper query execution plans due to the dependence assumption used that treats selections and joins together. This approach never suffers with out-of-date join selectivities. Unlike the semi-dynamic approach, there is no storage cost for join selectivities by this approach as they are obtained by on-line sampling.

Disadvantage The query optimiser runs perhaps much slower than the semi-dynamic approach, in finding an optimal query execution plan. This is because at optimisation time for a join query, sampling must be done for both join and selection selectivities in the query, especially the sampling for join selectivities which can take much more time than the one for selection selectivities. In addition, each estimated join selectivity obtained can have a high bias as a consequence of one-time sampling used.

6.6 Query optimisation algorithm

Given a join query Q, an optimisation algorithm is shown in Figure 1.5 near page 9 of Chapter 1. The algorithm is general for any kind of selectivity estimation methods used.

Due to the exhaustiveness of the algorithm, the query optimiser considers the entire search space. An entire search space of a join query Q consists of a factorial number of plans $m!$, where m is the number of relations involved in the query.

There are two optimisation objectives commonly used by query optimisers: *minimum total cost* and *minimum response time* [Hevner and Yao 1979]. Here we choose the former as the optimisation objective for the experimental query optimiser. In the algorithm in Figure 1.5, the optimiser will therefore seek to minimise the total cost $estcost(plan)$.

The details of the cost calculation for a plan, $estcost(plan)$, are described in Section 6.7.

6.7 Cost calculation for query execution plans

Given a join query, it is reasonable to consider that the total estimated cost $estcost(plan)$ for a query plan consists of:

$$estcost(plan) = \texttt{estcost(selection)} + \texttt{estcost(exeplan)} \tag{6.17}$$

where `estcost(selection)` is the total estimated cost for all selections in the query and `estcost(exeplan)` is the estimated execution plan cost, which is the estimated cost for a join order[2] in the query. Equation (6.17) is thus the optimisation objective we want to minimise.

6.7.1 Estimated cost for selections

Since all selections are performed independently of one another, the total estimated cost for selections is simply the sum of the estimated cost of the individual selections.

Consider, for example, join query Q shown in Figure 6.3 of Section 6.8. There are three selection predicates $R_1.a_3 > 40932$, $R_3.c_3 \neq 6377$ and $R_4.d_2 < 11264$. If the estimated cost for selection on relation R_i is denoted by $\texttt{est\_select}(R_i)$, then the total estimated cost for all the three selections can be expressed as:

$$\texttt{estcost(selection)} = \sum_{\forall i \in \{1,3,4\}} \texttt{est\_select}(R_i) \tag{6.18}$$

Recall that the notation $\widehat{R}_i$ is the estimated reduced relation as a result of applying a reduction by the selection predicate on relation R_i. If there is no selection predicate on R_i, then the notation $\widehat{R}_i$ is simply equivalent to R_i itself. For any execution plan in the search space, namely, any join order among the 4 relations in Q, e.g.:

$$(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2),\ (((\widehat{R}_2 \bowtie \widehat{R}_1) \bowtie \widehat{R}_4) \bowtie \widehat{R}_3),\ \text{or}\ (((\widehat{R}_2 \bowtie \widehat{R}_1) \bowtie \widehat{R}_4) \bowtie \widehat{R}_3)$$

the total estimated cost `estcost(selection)` for the three selections would be the same in (6.18) for all of the plans in the search space. This will hold for any query optimiser that applies the heuristic of performing selections before joins in the query.

[2]There are a number of "equivalent" join orders (or query execution plans) for the given join query each of which can produce the same final output relation.

Of course, the query optimiser will attempt to optimise how to perform selections on each relation most efficiently. For example, if a relation has an index on an attribute specified in a simple predicate of the query, then the query optimiser will most probably use the index on the attribute to retrieve data in the relation, rather than performing the linear scan on the relation.

In summary, given certain storage structures used by the relations in a given join query (e.g., B^{+}-trees, hash structures and so on) and a set of selections in the query, whatever execution plan selected from the search space would always have an identical total estimated cost for all the selections. This also implies that the *total estimated cost for all the selections* does not depend on the ordering of joins in the query.

Thus, either using

$$estcost(plan) = \overbrace{\texttt{estcost(selection)}}^{\text{fixed for all plans}} + \texttt{estcost(exeplan)}$$

from (6.17) or simply

$$estcost(plan) = \texttt{estcost(exeplan)} \tag{6.19}$$

as the optimisation objective, makes no difference in the optimal plan selection. That is, if a plan, say *optimal_plan* is optimal by the former equation, then the plan *optimal_plan* would also still be optimal by the latter equation.

As a result, in calculating an estimated cost for an execution plan of a join query in Section 6.7.3, we will ignore the total estimated cost for the selections in the query and thus use (6.19) as the optimisation objective.

6.7.2 Actual cost for selections

Like the *total estimated cost* for all selections in a join query, the *total actual cost* for all selections in a join query would also be the same, regardless of whatever execution plan is selected from the search space. This is because of the heuristic procedure used by many database systems.

In comparing between two estimation methods, say A and B, to see which one outperforms in producing cheaper execution plans, given the optimal execution plan for a query by a method *method*, one can look at the *actual* total cost of the optimal plan, i.e.:

$$\texttt{cost(eval[}\mathit{method}\texttt{])} = \overbrace{\texttt{cost(selection[}\mathit{method}\texttt{])}}^{\text{fixed for A or B}} + \texttt{cost(exeplan[}\mathit{method}\texttt{])} \tag{6.20}$$

where *method* is either A or B to see which total `cost(eval[`*method*`])` between the two is lower. `cost(selection[`*method*`])` is the actual cost for all selections in the query and `cost(exeplan[`*method*`])` is the actual execution plan cost. Logically the equation (6.20) is the same as the optimisation objective (6.17), except that the equation (6.20) specifies on the *actual* cost, while the optimisation objective specifies on the *estimated* cost.

Due to the heuristic procedure used, the `cost(selection[`*method*`])` in (6.20) will always be the same no matter what estimation method is used (A, B or whatever). This is also under the condition that the comparison between two estimation methods must be done by using relations with the same storage structures. (In comparing between two methods, it makes sense that one would use the same set of storage structures on the relations.)

As a result, for the comparison purpose between any two methods we can ignore the cost `cost(selection[`*method*`])` and simply use:

$$\texttt{cost(eval[}\mathit{method}\texttt{])} = \texttt{cost(exeplan[}\mathit{method}\texttt{])}$$

for the comparison. In Section 6.7.4, we will consider this actual execution plan cost.

6.7.3 Estimated cost for an execution plan

Given a join query $(\widehat{R}_1 \bowtie \widehat{R}_2 \bowtie \widehat{R}_3 \bowtie \widehat{R}_4)$, a possible join order to execute the query, namely, a query execute plan, is: $(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2)$. This order will proceed to execute as shown in Figure 6.2.

To calculate the intermediate size $|\widehat{R}_i|$ in Figure 6.2 where $i = 1, 2, 3, 4$, if HYBRID is used and the relation R_i is sorted, then apply systematic sampling with a β

$$\begin{array}{lc}
\text{Step 1)} & (((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2) \\
\text{Step 2)} & ((\widehat{R}_{13} \bowtie \widehat{R}_4) \bowtie \widehat{R}_2) \\
\text{Step 3)} & (\widehat{R}_{134} \bowtie \widehat{R}_2) \\
 & \widehat{R}_{1234}
\end{array}$$

Figure 6.2: A join order to execute a join query

sampling fraction to obtain and use the systematic sample relation to approximate the size $|\widehat{R}_i|$. But if R_i has none of its attributes sorted, then apply simple random sampling with replacement with the same sampling fraction to obtain and use the sample relation to approximate the size $|\widehat{R}_i|$. As for UNF, the intermediate size $|\widehat{R}_i|$ is calculated by formula (2.2) multiplied by $|R_i|$ which we have derived near page 29 of Chapter 2.

Recall that through HYBRID an estimated selectivity of a star join is calculated as follows: samples of the relations with the sorted join attributes are created via SYSSMP and samples of the relations with the unsorted join attributes are created via SRSWR. All the samples are then joined together to produce the estimated selectivity of the star join.

To calculate the intermediate size $|\widehat{R}_{i_1,i_2,\ldots,i_m}|$ $(m \geq 2)$ in each step of the figure, use equation (6.8) in Section 6.3.3. Let us reproduce it here for ease in reference:

$$|\widehat{R}_{i_1,i_2,\ldots,i_m}| = \hat{\mu}_{i_1,i_2,\ldots,i_m} * (|\widehat{R}_{i_1}||\widehat{R}_{i_2}| \cdots |\widehat{R}_{i_m}|)$$

where $\hat{\mu}_{i_1,i_2,\ldots,i_m}$ is the estimated selectivity of the star join $(\widehat{R}_{i_1} \bowtie \widehat{R}_{i_2} \bowtie \cdots \bowtie \widehat{R}_{i_m})$. If HYBRID is used for selectivity estimation, then use formula (6.9) in Section 6.3.4 for the estimated join selectivity. If UNF is used, then use formula (6.10) in Section 6.3.5 for the estimated join selectivity.

In what follows, we show how to calculate the *estimated cost* for the query execution plan in Figure 6.2. Recall that the hash-join algorithm is used for any join and thus given that $(\widehat{R}_{i_1} \bowtie \widehat{R}_{i_2})$, the read cost for the two estimated relations would be the cost in reading $|\widehat{R}_{i_1}|$ and $|\widehat{R}_{i_2}|$ tuples into memory.

Cost1 (in Step 1): The cost of the join $(\widehat{R}_1 \bowtie \widehat{R}_3)$ is the cost of reading $|\widehat{R}_1|$ and $|\widehat{R}_3|$ tuples from disk to memory and of writing the output temporary relation $\widehat{R}_{13}$, which is $|\widehat{R}_{13}|$ tuples back to disk.

Convert the number of tuples read/written to a total number of bytes. The read cost for $\widehat{R}_1$ is thus $attrlen(\widehat{R}_1) * |\widehat{R}_1|$. $attrlen$(an (intermediate) relation) is the sum of all attributes' lengths in the (intermediate) relation. Likewise, the read cost for $\widehat{R}_3$ is $attrlen(\widehat{R}_3) * |\widehat{R}_3|$. The write cost for $\widehat{R}_{13}$ is $attrlen(\widehat{R}_{13}) * |\widehat{R}_{13}|$.

Instead of measuring the read and write costs above in terms of "bytes", another way which is most commonly used is to convert the number of tuples read/written to in terms of "pages" by:

$$\frac{attrlen(\widehat{R}_{i_1,i_2,\ldots,i_m}) * |\widehat{R}_{i_1,i_2,\ldots,i_m}|}{PageSize}$$

where $m \geq 1$ and $PageSize$ is the number of bytes per disk page used in the database system. However, since either conversion used makes no change in selection of the optimal plan – it is just a matter of linearly scaling straight through any read/write cost in bytes by the constant $PageSize$ to turn each into a cost in pages –, we will still use the conversion in terms of "bytes".

Cost2 (in Step 2): The cost of the join $(\widehat{R}_{13} \bowtie \widehat{R}_4)$ is the cost of reading $\widehat{R}_{13}$, $attrlen(\widehat{R}_{13}) * |\widehat{R}_{13}|$ bytes and reading $\widehat{R}_4$, $attrlen(\widehat{R}_4) * |\widehat{R}_4|$ bytes into memory and of writing $\widehat{R}_{134}$, $attrlen(\widehat{R}_{134}) * |\widehat{R}_{134}|$ bytes back to disk.

Cost3 (in Step 3): The cost of the join $(\widehat{R}_{134} \bowtie \widehat{R}_2)$ is the cost of reading $\widehat{R}_{134}$, $attrlen(\widehat{R}_{134}) * |\widehat{R}_{134}|$ bytes and reading $\widehat{R}_2$, $attrlen(\widehat{R}_2) * |\widehat{R}_2|$ bytes into memory and of writing $\widehat{R}_{1234}$, $attrlen(\widehat{R}_{1234}) * |\widehat{R}_{1234}|$ bytes back to disk.

Hence, the estimated cost in executing the query execution plan: $(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2)$ is the sum of Cost1+Cost2+Cost3.

6.7.4 Actual cost for the optimal execution plan

Recall that the notation $\breve{R}_i$ is the actual reduced relation as a result of applying a reduction by a selection predicate on relation R_i. If there is no selection predicate on R_i, then the notation $\breve{R}_i$ is simply equal to R_i. Suppose that the order $(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2)$ defined above is the one which provides the cheapest estimated cost to execute the query.

The steps to calculate the actual cost in executing the query execution plan: $(((\breve{R}_1 \bowtie \breve{R}_3) \bowtie \breve{R}_4) \bowtie \breve{R}_2)$ would be the same as the ones in Section 6.7.3 and the only change that needs to be done is to substitute the notation $\widehat{R}$ by $\breve{R}$.

To calculate the intermediate size $|\breve{R}_i|$, one just has to observe the actual size $|\breve{R}_i|$ after reducing relation R_i by the selection predicate on R_i.

To calculate the intermediate size $|\breve{R}_{i_1,i_2,\ldots,i_m}|$ $(m \geq 2)$ in each step of Figure 6.2, use formula (6.7) in Section 6.3.2.

6.8 Schemas of temporary relations

To calculate *attrlen*(an intermediate relation), one needs the knowledge of the schema for the intermediate relation. Here in this section, we describe the creation of schemas for intermediate temporary relations.

Table 6.3 shows the relational schemas of 5 relations in a database. Query Q in Figure 6.3 is a star join query from which all join attributes $R_3.c_2, R_1.a_2, R_2.b_3, R_4.d_3$ and $R_3.c_2$ which appear in the join predicates can join one another.

$R_1(a_1, a_2, \ldots, a_{u_1})$,	$R_2(b_1, b_2, \ldots, b_{u_2})$
$R_3(c_1, c_2, \ldots, c_{u_3})$,	$R_4(d_1, b_2, \ldots, d_{u_4})$
$R_5(e_1, e_2, \ldots, e_{u_5})$	

where $a_{u_1}, b_{u_2}, c_{u_3}, d_{u_4}$ and e_{u_5} each are the last attribute of the relations.

Table 6.3: Schemas of 5 relations in a database

Query Q:

```
    select   R1.a5, R2.b1, R2.b4, R3.c4, R4.d1
    from     R1, R2, R3, R4
    where    R3.c2 = R1.a2 and
             R2.b3 = R1.a2 and
             R4.d3 = R3.c2 and
             R1.a3 > 40932 and
             R3.c3 ≠ 6377 and
             R4.d2 < 11264;
```

Figure 6.3: A star join query

By the heuristic procedure (applying selections as soon as possible), before any join in query Q begins its execution, the respective schemas of R_1, R_3 and R_4 would be: $R_1(a_2, a_5)$, $R_3(c_2, c_4)$, $R_4(d_1, d_3)$ while the schema of R_2 would still remain unchanged in its original form. That is, after a selection on a relation is done, the attributes of the relation which would appear in the schema of a temporary reduced relation are (1) the ones that will be in use in any join predicates of the query and (2) the ones that appear in the select clause of Q, i.e., select $R_1.a_5, R_2.b_1, R_2.b_4, R_3.c_4, R_4.d_1$. Given an example, consider the resulting schema of R_1 after the selection $R_1.a_3 > 40932$ has been done, namely $R_1(a_2, a_5)$. a_2 of R_1 will be in use in the join predicate $R_3.c_2 = R_1.a_2$ and a_5 of R_1 appears in the select clause of Q.

In what follows, we consider the creation of schemas of temporary relations as a result of executing a join. A schema of a temporary relation is created based on two kinds of attributes:

join attributes Any join attributes appearing in subsequent joins must be "carried over" by keeping them in the schema of the temporary relation. Let us illustrate by an example: if the join $(R_3.c_2 \bowtie R_1.a_2)$ is performed before the join $(R_2.b_3 \bowtie R_1.a_4)$, then the schema of R_{13} must "carry" attribute a_4 because the next join $(R_2.b_3 \bowtie R_1.a_4)$ requires the attribute a_4.

select-clause attributes Any attributes appearing in the select clause must also be "carried over" by keeping them in the schema of the temporary relation. Let us illustrate by using query Q: the attributes $R_1.a_5$ and $R_3.c_4$ must appear in the result of the join $(R_3.c_2 \bowtie R_1.a_2)$ (i.e. R_{13}) because the two attributes appear in the select clause of Q.

Using the optimal join order in executing query Q as described in Section 6.7, i.e., $(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2)$, Table 6.4 shows an example of how schemas of temporary relations are created.

			schema of temp. rel.	
Step	A remaining order	temp. rel.	join	select-clause
1	$(((\widehat{R}_1 \bowtie \widehat{R}_3) \bowtie \widehat{R}_4) \bowtie \widehat{R}_2))$	$\widehat{R}_{13}$	$(a_2,$	$a_5, c_4)$
2	$((\widehat{R}_{13} \bowtie \widehat{R}_4) \bowtie \widehat{R}_2)$	$\widehat{R}_{134}$	$(a_2,$	$a_5, c_4, d_1)$
3	$(\widehat{R}_{134} \bowtie \widehat{R}_2)$	$\widehat{R}_{1234}$	$(a_2,$	$a_5, c_4, d_1, b_1, b_4)$

Table 6.4: Schemas of temporary relations

As Q is a star join query from which all join attributes $R_3.c_2, R_1.a_2, R_2.b_3, R_4.d_3$ and $R_3.c_2$ can join one another, we can rename those join attributes to a single name.

Suppose we rename them to a_2. As a result, in Table 6.4 the column named "join" would always carry the attribute a_2 but this is not necessary for general cases of join queries as the schema creation approach described above is applicable to handling all kinds of join queries.

6.9 Overall cost of query optimisation

We have demonstrated that sampling can produce accurate query size estimates, which, in turn, lead the query optimiser to generate efficient query execution plans. However, this accuracy comes at a cost – the amount of time spent performing query optimisation increases considerably. In order to make a definitive statement that sampling methods are superior to other methods, we need to show that the total cost (query optimisation plus query evaluation) is lower for sampling than for other methods.

Let us consider a comparison between query optimisation based on UNF query size estimation and query optimisation using sampling. UNF is a parametric method with very little overhead in generating query size estimates for query optimisation. On the other hand, a query optimiser that uses any sampling method (such as HYBRID) would incur sampling overhead for selectivity estimation.

In order to justify that HYBRID generally outperforms UNF, we must demonstrate the following inequality:

$$\texttt{cost(opt[HYBRID])} + \texttt{cost(eval[HYBRID])} < \texttt{cost(eval[UNF])}$$

which we have defined and justified earlier in (6.3). We have shown the calculation for `cost(eval)` for both HYBRID and UNF in Section 6.7. Here we will show the calculation for `cost(opt[HYBRID])` due to the sampling cost.

For sampling-based methods, there are two types of sampling cost for `cost(opt)` that must be summed up with the cost of an optimal query execution plan. The first is a sampling cost as a consequence of sampling to estimate all join selectivities that must be stored in the profile catalog. This is described in Section 6.9.1. The second is a sampling cost as a consequence of sampling to estimate selection selectivities whose selection predicates appear in a given join query. This is described in Section 6.9.2.

By the end of Section 6.9.2, we will know the total cost for the optimal plan of a join query, say Q, which incorporates the two types of sampling cost.

6.9.1 Sampling cost for join selectivities

The following is the procedure to calculate the sampling cost for all join selectivities that must be stored in the profile catalog.

Using the semi-dynamic approach for query optimisation and using the 5-relation database for illustration here, all combinations of star joins as shown in Table 6.1 imply the total number of join selectivities needed to be stored in the profile catalog.

Given a star join $(R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie R'_4)$ from the table, here are steps to calculate the sampling cost for this star join.

$$\begin{array}{rc} \text{Step 1)} & (((R'_1 \bowtie R'_2) \bowtie R'_3) \bowtie R'_4) \\ \text{Step 2)} & ((R'_{12} \bowtie R'_3) \bowtie R'_4) \\ \text{Step 3)} & (R'_{123} \bowtie R'_4) \\ & R'_{1234} \end{array}$$

In fact, the sampling cost calculation here is the same as the cost calculation for query execution plans as shown in Section 6.7 (see also Figure 6.2 for comparison). From the steps shown above, notice that we do not attempt to optimise the order of the star join $(R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie R'_4)$; we simply use a simple and straightforward order of the star join from left to right. That is, join the relations together from left to right.

In regard to the creation of intermediate temporary schemas in each step, a resulting temporary schema of the output relation only needs to carry a single star join attribute which is to be joined with the sample relation in the next step.

The size $|R'_i|$ is calculated by $\beta * |R_i|$. To calculate the intermediate size $|R'_{i_1,i_2,\ldots,i_m}|$ $(m \geq 2)$ in each step above, arrange equation (6.9) in Section 6.3.4 to:

$$|R'_{i_1,i_2,\ldots,i_m}| = \mu'_{i_1,i_2,\ldots,i_m} * (|R'_{i_1}||R'_{i_2}| \cdots |R'_{i_m}|)$$

Each step incurs a sampling cost as follows:

SampCost1 (in Step 1): The cost of the join $(R'_1 \bowtie R'_2)$ is the cost of reading

R'_1, $attrlen(R'_1) * |R'_1|$ bytes and reading R'_2, $attrlen(R'_2) * |R'_2|$ bytes into memory and of writing the output temporary relation R'_{12}, $attrlen(R'_{12}) * |R'_{12}|$ bytes back to disk.

SampCost2 (in Step 2): The cost of the join $(R'_{12} \bowtie R'_3)$ is the cost of reading R'_{12}, $attrlen(R'_{12}) * |R'_{12}|$ bytes and reading R'_3, $attrlen(R'_3) * |R'_3|$ bytes into memory and of writing R'_{123}, $attrlen(R'_{123}) * |R'_{123}|$ bytes back to disk.

SampCost3 (in Step 3): The cost of the join $(R'_{123} \bowtie R'_4)$ is the cost of reading R'_{123}, $attrlen(R'_{123}) * |R'_{123}|$ bytes and reading R'_4, $attrlen(R'_4) * |R'_4|$ bytes into memory and of writing R'_{1234}, $attrlen(R'_{1234}) * |R'_{1234}|$ bytes back to disk.

Hence, the sampling cost for the star join $(R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie R'_4)$ is the sum of SampCost1+SampCost2+SampCost3. Let SampCost$(R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie R'_4)$ be the sampling cost for the star join, i.e., equal to the sum of SampCost1+SampCost2+SampCost3.

To calculate a sampling cost for other star joins shown in Table 6.1, apply the same process like shown above for the star join $(R'_1 \bowtie R'_2 \bowtie R'_3 \bowtie R'_4)$.

As a result, the total of the sampling costs for all the star joins in the table is equal to:

$$\sum_{\text{star join combinations}} \text{SampCost(a star join combination)} \tag{6.21}$$

In case the number of resamplings done for these star joins is η times, then the total would be:

$$\sum_{i=1}^{\eta} \text{(equation (6.21))} \tag{6.22}$$

Recall that the query optimiser with the semi-dynamic approach must perform resampling η times for each join selectivity stored in the profile catalog. After that, the query optimiser will stop sampling (for the join selectivities) and use the stored join selectivities in the profile catalog to optimise any incoming join queries. (Sampling to update the current join selectivities will be resumed periodically.) The cost of η-time resamplings in (6.22) will therefore be carried over to any of incoming queries, i.e., they all must be responsible for this cost.

In all our experiments in Section 6.10, we use 100 queries and these 100 queries must be responsible for the total join sampling cost in (6.22). That is, each of the queries (also for query Q in Figure 6.3) will need to pay `cost(joinsamp)`:

$$\texttt{cost(joinsamp)} = \frac{\text{equation (6.22)}}{100}$$

on average for its join sampling cost.

6.9.2 Sampling cost for selection selectivities

Using query Q in Figure 6.3, there are 3 selection predicates, namely, $R_1.a_3 > 40932$, $R_3.c_3 \neq 6377$, and $R_4.d_2 < 11264$. The sampling cost `cost(selsamp)` for all the selections would then be:

$$\texttt{cost(selsamp)} = \sum_{\forall i \, \in \, \{1,3,4\}} (\beta * |R_i|) * attrlen(\text{all attributes of } R_i)$$

bytes to read the 3 relations with the sampling fraction β into memory for selectivity estimation. Notice that unlike joins, for selections there is no cost for writing a temporary output relation back to disk.

As a consequence, the total cost for the optimal execution plan for query Q would be:

$$\texttt{cost(opt[HYBRID])} + \texttt{cost(eval[HYBRID])}$$

where `cost(opt[HYBRID])` is:

$$\texttt{cost(opt[HYBRID])} = \texttt{cost(joinsamp[HYBRID])} + \texttt{cost(selsamp[HYBRID])}$$

and `cost(eval[HYBRID])` is:

$$\texttt{cost(exeplan[HYBRID])}$$

where `cost(exeplan[HYBRID])` is the cost in executing Q by the optimal plan with no sampling cost added. Compared with UNF, since `cost(opt[UNF])` is a negligible cost and there is no sampling cost for UNF, the cost in executing Q is simply `cost(exeplan[UNF])`.

6.10 Experimental results

6.10.1 Experimental setup

All of the five-relation databases (six databases) described in Table 3.11(d) near page 3.5.3.1 of Chapter 3 are reused for all experiments in this chapter. The configurations for the join attributes (their frequency distributions) are described in the table. These databases were also earlier used in Chapter 4 (Improving join selectivities by bootstrap method) for all the experiments in the chapter.

All relations in the databases have five attributes and a cardinality of 10,000. 100 join queries are used in all experiments, each of which is of the form like the query in Figure 6.3 and is a star join query among 4–5 relations.

The data in the relations and the queries are generated in such a way as to cover a wide range of various frequency distributions, various numbers of distinct values, various join predicates and various selection predicates that can appear in join queries.

Christodoulakis [1984] commented that due to the nature of database environments, functional dependencies (FD) are something typical rather than unusual to appear on relations. Real data tend to have some relationships among attributes in relations. (We have raised the significance in Section 3.8.1 of Chapter 3 that data generated in a relation should have some forms of FDs on them.) To make data in all relations in Table 6.5 more like realistic data, FDs are added on those relations. Since each relation has only 5 attributes, only one functional dependency defined on a few attributes is added on each of those relations. See Table 6.5 for the functional dependencies on the relations.

Table 6.6 shows the numbers of distinct values for all attributes in the six databases. In Table 3.11(d) near page 3.5.3.1 which shows the configurations for the five-relation databases, we only show the numbers of distinct values for the join attributes, but not for other attributes.

All sampling is done with $\beta = 5\%$ sampling fraction – based on the analysis in Section 6.4 – for both selection and join selectivities.

Since each database has five relations, for the HYBRID sampling scheme (some relations are sorted and the rest unsorted) three relations out of the five are selected

	funcdep	dist mode
R1	$a_4 \rightarrow a_3$	zipf(260,0.266)
R2	$b_5 \rightarrow b_2$	norm(3661.377,257.170)
R3	$c_5 \rightarrow c_1c_3$	unf(3403.931,3590.454)
R4	$d_1 \rightarrow d_2d_5$	semizipf(160,0.5)
R5	$e_1 \rightarrow e_3$	exp(139.064)

(a) SJ1, star join DB 1

	funcdep	dist mode
R1	$a_1 \rightarrow a_3$	exp(120.850)
R2	$b_4b_5 \rightarrow b_2$	zipf(450,0.118)
R3	$c_1 \rightarrow c_4$	norm(2734.131,206.915)
R4	$d_5 \rightarrow d_1$	semizipf(750,0.5)
R5	$e_3 \rightarrow e_1$	unf(3977.661,4177.661)

(b) SJ2, star join DB 2

	funcdep	dist mode
R1	$a_3a_4 \rightarrow a_5$	unf(3169.800,3369.800)
R2	$b_4 \rightarrow b_1b_5$	norm(3223.145,124.649)
R3	$c_5 \rightarrow c_3c_4$	zipf(220,0.922)
R4	$d_1d_4 \rightarrow d_2$	exp(81.317)
R5	$e_5 \rightarrow e_3e_4$	semizipf(320,0.5)

(c) SJ3, star join DB 3

	funcdep	dist mode
R1	$a_3a_5 \rightarrow a_4$	semizipf(520,0.5)
R2	$b_1 \rightarrow b_2$	norm(2459.717,327.025)
R3	$c_1c_3 \rightarrow c_4$	unf(2755.615,2955.615)
R4	$d_1d_2 \rightarrow d_4$	zipf(872,0.069)
R5	$e_1 \rightarrow e_3$	fdist(30,10.789)

(d) SJ4, star join DB 4

	funcdep	dist mode
R1	$a_1 \rightarrow a_5$	exp(120.213)
R2	$b_2 \rightarrow b_1$	semizipf(380,0.5)
R3	$c_1 \rightarrow c_3c_4$	zipf(560,0.289)
R4	$d_2 \rightarrow d_5$	norm(2675.293,331.425)
R5	$e_3 \rightarrow e_1e_4$	fdist(30,11.302)

(e) SJ5, star join DB 5

	funcdep	dist mode
R1	$a_1a_4 \rightarrow a_5$	norm(1817.017,167.010)
R2	$b_1 \rightarrow b_2$	unf(3076.050,3276.050)
R3	$c_1c_5 \rightarrow c_3$	zipf(660,0.998)
R4	$d_4 \rightarrow d_1d_2$	exp(139.485)
R5	$e_4 \rightarrow e_5$	semizipf(390,0.5)

(f) SJ6, star join DB 6

Table 6.5: FDs on databases

	distinct values of 5 attrs
R1	260, 403, 112, 260, 99
R2	491, 747, 313, 456, 1297
R3	5, 113, 176, 308, 201
R4	160, 4, 383, 201, 113
R5	686, 199, 24, 131, 169

(a) SJ1, star join DB 1

	distinct values of 5 attrs
R1	600, 400, 26, 8, 1085
R2	598, 375, 343, 450, 450
R3	1077, 60, 661, 567, 1085
R4	429, 1026, 450, 731, 750
R5	111, 476, 201, 8, 1085

(b) SJ2, star join DB 2

	distinct values of 5 attrs
R1	330, 489, 201, 201, 183
R2	417, 1026, 363, 695, 442
R3	487, 144, 190, 197, 220
R4	442, 374, 600, 442, 697
R5	330, 177, 23, 6, 320

(c) SJ3, star join DB 3

	distinct values of 5 attrs
R1	330, 700, 520, 7, 520
R2	1591, 823, 413, 731, 697
R3	201, 555, 201, 173, 1085
R4	872, 872, 81, 495, 697
R5	13, 700, 11, 8, 1085

(d) SJ4, star join DB 4

	distinct values of 5 attrs
R1	608, 800, 33, 8, 453
R2	286, 380, 800, 731, 697
R3	560, 619, 391, 406, 1085
R4	598, 1609, 143, 731, 629
R5	8, 173, 9, 3, 1085

(e) SJ5, star join DB 5

	distinct values of 5 attrs
R1	907, 465, 33, 907, 600
R2	201, 171, 497, 731, 697
R3	660, 663, 418, 722, 660
R4	406, 511, 900, 679, 697
R5	330, 900, 33, 390, 329

(f) SJ6, star join DB 6

Table 6.6: Numbers of distinct values

as the sorted relations, assuming that the other two relations are unsorted. This poses a question which are the three sorted relations ? The answer is that the three are randomly selected from the five relations.

To sufficiently cover a few other cases, not just one combination of the three relations randomly selected, we randomly generated three combinations altogether for the sorted relations, i.e., R1R2R3, R1R4R5, and R2R3R5. These three combinations are then used throughout all experiments.

With the use of the resampling technique, 15-time resampling is performed for all experiments which compare HYBRID with SRSWR. Given that each of the 100 queries is involved with 4–5 relations, approximately in the first 15 queries the selectivity estimator will spend time resampling to estimate and improve all possible star join selectivities and store them in the profile catalog. Thus, probably the accuracy of join selectivities at the outset may be low due to too high a bias on a single sample used but after more queries are processed by the database system towards the 15th query, the accuracy will be incrementally higher.

Note that we use the term "query execution plan" throughout Section 6.10 to mean the *optimal execution plan.*

We have attempted to summarise the results of the experiments by partitioning the 100 queries into five categories based on the "size" of actual query execution costs. In Figure 6.4, we give the routine used to calculate total query execution costs for the five categories, i.e., one total cost for one category.

size	category
$\leq 10^6$	1
$> 10^6$ **but** $\leq 10^7$	2
$> 10^7$ **but** $\leq 10^8$	3
$> 10^8$ **but** $\leq 10^9$	4
$> 10^9$	5

Table 6.7: 5 categories

The categorisation using the size of query execution costs is shown in Table 6.7. In the comparison (1) between HYBRID and UNF and (2) between HYBRID and SRSWR, the routine is applied to calculate total query execution costs for the five categories.

The following are the meanings of the symbols which appear in many result tables in Section 6.10.2.

samp-cost means the sampling cost per query (i.e., `cost(joinsamp)` + `cost(selsamp)` incurred by HYBRID whose calculation we have introduced in Section 6.9.

no-samp-cost There are times that we consider the query execution plan by HY-

```
input:   ExeCost[i][method], actual execution cost for query i by method, which is either
         method_A, selectivity estimation method A or
         method_B, selectivity estimation method B

output:  total[method], total actual execution cost for 100 queries by method
         total[method][class], total actual execution cost for
             a method in a class of { 1stClass, 2ndClass, 3rdClass, 4thClass, 5thClass }
         total[queries][class], total number of queries in a class

foreach method { method_A, method_B } do
  total[method] = 0
  foreach class { 1stClass, 2ndClass, 3rdClass, 4thClass, 5thClass } do
    total[method][class] = 0
    total[queries][class] = 0
  endforeach
endforeach
for query $i = 1$ to 100 do
  total[method_A] += ExeCost[i][method_A]
  total[method_B] += ExeCost[i][method_B]
  if ExeCost[i][method_A] $\leq 10^6$ then
    total[method_A][1stClass] += ExeCost[i][method_A]
    total[method_B][1stClass] += ExeCost[i][method_B]
    total[queries][1stClass] += 1
  else if ExeCost[i][method_A] $> 10^6$ and ExeCost[i][method_A] $\leq 10^7$ then
    total[method_A][2ndClass] += ExeCost[i][method_A]
    total[method_B][2ndClass] += ExeCost[i][method_B]
    total[queries][2ndClass] += 1
  else if ExeCost[i][method_A] $> 10^7$ and ExeCost[i][method_A] $\leq 10^8$ then
    total[method_A][3rdClass] += ExeCost[i][method_A]
    total[method_B][3rdClass] += ExeCost[i][method_B]
    total[queries][3rdClass] += 1
  else if ExeCost[i][method_A] $> 10^8$ and ExeCost[i][method_A] $\leq 10^9$ then
    total[method_A][4thClass] += ExeCost[i][method_A]
    total[method_B][4thClass] += ExeCost[i][method_B]
    total[queries][4thClass] += 1
  else
    total[method_A][5thClass] += ExeCost[i][method_A]
    total[method_B][5thClass] += ExeCost[i][method_B]
    total[queries][5thClass] += 1
  endif
endfor
```

Figure 6.4: Routine to calculate total query execution costs for 5 categories

BRID without including its sampling cost and hence we use the symbol no-samp-cost for that purpose.

#q means the number of queries in a category.

diff% and **imp%** are the difference and improvement by percentage, respectively, both of which are defined as:

$$\frac{\text{(the cost by the right hand approach - the cost by the left hand approach)}}{\text{(the cost by the left hand approach)}} * 100$$

The left and right hand approaches can be described by examples. Consider, for example, Table 6.8(a). The left hand approach is HYBRID and the right hand approach is UNF (see the table heading). As another example, consider Table 6.9(a). The left hand approach is no-samp-cost and the right hand approach is samp-cost.

The symbol diff% is literally just the difference of two values in percentage but is not meant to carry any meaning of improvement, whereas the symbol imp% is the difference which is meant to carry an improvement when one estimation method is used to compare with the other method. The meanings of diff% and imp% will become clearer when seeing the result tables.

If the value of diff% or imp% is positive (we ignore the + symbol), then the result of the left hand approach has a smaller value than the result of the right hand approach. If the value is negative, then the opposite way is true.

There are total values which we reproduce in two places. There are some rounding-off errors in computation which make the two total values slightly different.

6.10.2 Improvement of query execution plans

A series of experiments has been conducted whose aims are as follows:

1st aim

Compare the UNF and HYBRID approaches. HYBRID uses the one-time sampling (i.e, 1-time resampling which all previous sampling work proposes) to estimate all join selectivities. Apart from its query execution cost, each query execution plan produced by HYBRID must also take account of its sampling cost which we described in Section 6.9. Given a join query, the aim is to see that generally:

$$\texttt{cost(joinsamp[HYBRID])} + \texttt{cost(selsamp[HYBRID])} + \texttt{cost(exeplan[HYBRID])} < \texttt{cost(exeplan[UNF])}$$

The results obtained in Table 6.8 confirm the aim above and overall HYBRID improves the quality of query execution plans over UNF by several orders of magnitude, and the cost of sampling does not outweigh the gains from improved query execution.

2nd aim

Show that the cost for both one-time sampling (1-time resampling) and 15-time resampling for all both join and selection[3] selectivities is comparatively low, namely,

[3]The sampling for selection selectivities is only one-time sampling.

far less than 1%, relative to the total of query execution plan costs for 100 queries. See Tables 6.9 and 6.10 for the cost. Therefore, it seems practical to introduce the resampling technique into real-life query optimisers.

Comparing Tables 6.9 and 6.10 in the column diff% at the same sorted relations, it is natural to see that the cost of 15-time resampling would be more expensive than that of 1-time resampling.

3rd aim

Show that the resampling technique assists in improving query execution plans. (Earlier in Chapter 4 we have claimed that it improves join selectivities by reducing bias.) We compare HYBRID between 1-time and 15-time resampling for all join selectivities. All query execution plans by both 1-time and 15-time resampling take account of sampling cost. Note that for the 15-time resampling, each query execution plan takes account of its 15-time resampling cost for all join selectivities while for the 1-time resampling, each query plan takes account of its 1-time resampling cost for all join selectivities.

The results in Table 6.11 in general confirm the reliability of the resampling technique in obtaining better query plans. There is only one place which has the worse result in Table 6.11(e) for the sorted relations R1R4R5. Probably this is because the 15-time resampling still does not help much improve the overall quality of join selectivities, compared with the higher resampling cost that must be paid for. However this is the only one place out of all and the rest all produce better results.

In fact, Table 6.11 is the reproduction of the samp-cost column with 1-time resampling from Table 6.9 and the samp-cost column with 15-time resampling from Table 6.10. This is to make it easy to compare the results to check against the 3rd aim.

4th aim

Due to the improvement by the resampling technique in the 3rd aim, we then proceed to compare between HYBRID and SRSWR with 15-time resampling for both of them. For a join query, the aim is to see that generally:

$$\texttt{cost(joinsamp[HYBRID])} + \texttt{cost(selsamp[HYBRID])} + \texttt{cost(exeplan[HYBRID])}$$
$$< \texttt{cost(joinsamp[SRSWR])} + \texttt{cost(selsamp[SRSWR])} + \texttt{cost(exeplan[SRSWR])}$$

However, since both of the sampling techniques incur a similar sampling cost, i.e., `cost(joinsamp)` + `cost(selsamp)` because of the fact that they both do sampling with the same sampling fraction for both join and selection selectivities. The simplified aim is thus reduced to:

$$\texttt{cost(exeplan[HYBRID])} < \texttt{cost(exeplan[SRSWR])}$$

The results in Table 6.12 in general confirm the aim above. HYBRID, which uses the sortedness of data whenever possible, proves to provide slightly better query execution plans with the same sampling cost as SRSWR.

Miscellaneous issues

Table 6.9 is the table for HYBRID with 1-time resampling. In this table, the column samp-cost is the reproduction from Table 6.8 at the last row of the column hybrid. Let us see an example. Consider the sorted relations R1R2R3 at the samp-cost column of Table 6.9(a). The total cost is equal to 1.0748e+11 which comes from Table 6.8(a) at the last row in the column hybrid.

Similarly, Table 6.10 is the table for HYBRID with 15-time resampling. In this table, the column no-samp-cost is the reproduction from Table 6.12 at the last row of the column hybrid. Let us see an example. Consider the sorted relations R1R2R3 at the no-samp-cost column of Table 6.10(a). The total cost is equal to 1.0727e+11 which comes from Table 6.12(a) at the last row in the column hybrid.

6.11 Conclusion

We have shown that more accurate selectivity estimation techniques can indeed assist in the selection of better query execution plans.

We believe that for next-generation databases where there may be a large number of relations, i.e., more than 10 relations involved in join queries, more accurate

hybrid	unf	imp%	#q
5.8221e+06	6.2685e+06	7.667	10
3.1266e+07	5.3694e+07	71.733	8
1.0080e+09	1.2537e+09	24.379	21
1.3671e+10	1.8462e+10	35.044	33
9.2759e+10	1.0583e+11	14.095	28
1.0748e+11	1.2561e+11	16.872	100

(a) SJ1, R1R2R3

hybrid	unf	imp%	#q
5.8222e+06	6.2685e+06	7.665	10
3.1286e+07	5.3694e+07	71.622	8
9.1454e+08	1.1528e+09	26.052	20
1.4039e+10	1.8563e+10	32.222	34
9.4017e+10	1.0583e+11	12.569	28
1.0901e+11	1.2561e+11	15.229	100

(b) SJ1, R1R4R5

hybrid	unf	imp%	#q
6.4616e+06	6.7824e+06	4.965	11
3.4248e+07	5.3180e+07	55.281	7
9.5356e+08	1.2537e+09	31.476	21
1.2689e+10	1.7379e+10	36.963	32
9.3970e+10	1.0692e+11	13.777	29
1.0765e+11	1.2561e+11	16.680	100

(c) SJ1, R2R3R5

hybrid	unf	imp%	#q
6.0436e+06	5.4858e+06	-9.229	11
5.6807e+07	1.0251e+08	80.459	13
1.6126e+09	1.8317e+09	13.588	38
9.0717e+09	9.5860e+09	5.669	31
9.9079e+09	1.0132e+10	2.262	7
2.0655e+10	2.1658e+10	4.854	100

(d) SJ2, R1R2R3

hybrid	unf	imp%	#q
7.1687e+06	6.6712e+06	-6.940	13
6.5096e+07	1.1640e+08	78.806	14
1.6207e+09	1.9769e+09	21.979	37
8.7012e+09	1.0353e+10	18.980	30
8.0393e+09	9.2050e+09	14.501	6
1.8433e+10	2.1658e+10	17.491	100

(e) SJ2, R1R4R5

hybrid	unf	imp%	#q
6.4844e+06	5.9961e+06	-7.530	12
6.8621e+07	1.1707e+08	70.603	15
1.5807e+09	1.9769e+09	25.067	37
8.7117e+09	1.0353e+10	18.837	30
7.8631e+09	9.2050e+09	17.066	6
1.8231e+10	2.1658e+10	18.798	100

(f) SJ2, R2R3R5

hybrid	unf	imp%	#q
4.6402e+06	3.9708e+06	-14.425	9
3.5065e+07	4.0720e+07	16.129	9
1.2578e+09	1.5497e+09	23.202	27
1.5180e+10	1.6689e+10	9.941	43
2.6546e+10	2.5482e+10	-4.009	12
4.3024e+10	4.3765e+10	1.724	100

(g) SJ3, R1R2R3

hybrid	unf	imp%	#q
4.0694e+06	3.5280e+06	-13.304	8
5.1469e+07	5.0984e+07	-0.943	11
1.2420e+09	1.5398e+09	23.985	26
1.4841e+10	1.6689e+10	12.450	43
2.5332e+10	2.5482e+10	0.593	12
4.1470e+10	4.3765e+10	5.534	100

(h) SJ3, R1R4R5

hybrid	unf	imp%	#q
3.9377e+06	3.5280e+06	-10.405	8
4.5537e+07	5.0984e+07	11.962	11
1.2380e+09	1.5398e+09	24.382	26
1.4648e+10	1.6689e+10	13.931	43
2.4876e+10	2.5482e+10	2.436	12
4.0812e+10	4.3765e+10	7.237	100

(i) SJ3, R2R3R5

Table 6.8: Results between UNF and HYBRID with 1-time resampling with samp-cost added

1-time resampling = Resampling has been done only once for all star join selectivities.

hybrid	unf	imp%	#q
6.5036e+06	6.5624e+06	0.904	12
2.4585e+07	3.0808e+07	25.311	6
1.4801e+09	2.5073e+09	69.404	29
1.0367e+10	1.2404e+10	19.647	27
1.2763e+11	1.3221e+11	3.587	26
1.3951e+11	1.4715e+11	5.483	100

(j) SJ4, R1R2R3

hybrid	unf	imp%	#q
5.8409e+06	6.0883e+06	4.235	11
2.9927e+07	3.1282e+07	4.528	7
1.4254e+09	2.3170e+09	62.551	28
9.9385e+09	1.2594e+10	26.724	28
1.2781e+11	1.3221e+11	3.442	26
1.3921e+11	1.4715e+11	5.710	100

(k) SJ4, R1R4R5

hybrid	unf	imp%	#q
6.1697e+06	6.3668e+06	3.194	12
2.8864e+07	3.1004e+07	7.414	6
1.3872e+09	2.3170e+09	67.026	28
9.9435e+09	1.2594e+10	26.659	28
1.2714e+11	1.3221e+11	3.988	26
1.3850e+11	1.4715e+11	6.247	100

(l) SJ4, R2R3R5

hybrid	unf	imp%	#q
1.2417e+07	4.1976e+08	3280.6	24
3.3225e+07	1.2565e+08	278.18	9
9.7540e+08	1.8214e+09	86.735	27
1.4057e+10	1.9077e+10	35.710	34
7.7252e+09	9.9362e+09	28.621	6
2.2804e+10	3.1380e+10	37.611	100

(m) SJ5, R1R2R3

hybrid	unf	imp%	#q
1.2239e+07	5.3844e+08	4299.2	24
3.3815e+07	1.2649e+08	274.08	10
1.0560e+09	1.9318e+09	82.933	28
1.3796e+10	1.8847e+10	36.617	32
7.7281e+09	9.9362e+09	28.571	6
2.2626e+10	3.1380e+10	38.691	100

(n) SJ5, R1R4R5

hybrid	unf	imp%	#q
1.2254e+07	4.1976e+08	3325.621	24
3.2811e+07	1.2565e+08	282.955	9
9.7628e+08	1.8214e+09	86.565	27
1.4068e+10	1.9077e+10	35.612	34
7.7252e+09	9.9362e+09	28.621	6
2.2814e+10	3.1380e+10	37.548	100

(o) SJ5, R2R3R5

hybrid	unf	imp%	#q
3.1258e+06	2.3966e+06	-23.326	6
9.9405e+07	2.3585e+08	137.262	19
1.2281e+09	3.5998e+09	193.115	27
6.1427e+09	6.5575e+09	6.754	16
2.3684e+11	3.2153e+11	35.755	32
2.4432e+11	3.3192e+11	35.857	100

(p) SJ6, R1R2R3

hybrid	unf	imp%	#q
2.8548e+06	2.4111e+06	-15.542	6
9.1157e+07	2.2780e+08	149.896	17
1.1565e+09	2.5998e+09	124.801	26
5.6485e+09	6.3672e+09	12.725	18
2.3883e+11	3.2272e+11	35.126	33
2.4573e+11	3.3192e+11	35.075	100

(q) SJ6, R1R4R5

hybrid	unf	imp%	#q
3.7334e+06	2.9063e+06	-22.154	7
9.8305e+07	2.3534e+08	139.397	18
1.1292e+09	2.5998e+09	130.231	26
6.2518e+09	7.5576e+09	20.886	17
2.3631e+11	3.2153e+11	36.059	32
2.4380e+11	3.3192e+11	36.147	100

(r) SJ6, R2R3R5

Table 6.8: Results between UNF and HYBRID with 1-time resampling with samp-cost added

1-time resampling = Resampling has been done only once for all star join selectivities.

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	1.0747e+11	1.0748e+11	0.005
R1R4R5	1.0900e+11	1.0901e+11	0.005
R2R3R5	1.0765e+11	1.0765e+11	0.005

(a) SJ1

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	2.0650e+10	2.0655e+10	0.026
R1R4R5	1.8428e+10	1.8433e+10	0.029
R2R3R5	1.8225e+10	1.8231e+10	0.030

(b) SJ2

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	4.3018e+10	4.3024e+10	0.012
R1R4R5	4.1465e+10	4.1470e+10	0.013
R2R3R5	4.0806e+10	4.0812e+10	0.013

(c) SJ3

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	1.3950e+11	1.3951e+11	0.004
R1R4R5	1.3920e+11	1.3921e+11	0.004
R2R3R5	1.3850e+11	1.3850e+11	0.004

(d) SJ4

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	2.2798e+10	2.2804e+10	0.023
R1R4R5	2.2621e+10	2.2626e+10	0.023
R2R3R5	2.2809e+10	2.2814e+10	0.023

(e) SJ5

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	2.4431e+11	2.4432e+11	0.002
R1R4R5	2.4573e+11	2.4573e+11	0.002
R2R3R5	2.4379e+11	2.4380e+11	0.002

(f) SJ6

Table 6.9: Total costs by HYBRID with 1-time resampling before and after adding sampling costs

Total cost = A total of query execution costs for all 100 queries used.
1-time resampling = Resampling has been done only once for all star join selectivities.
no-samp-cost = Consider only the query execution cost per query without samp-cost.

techniques will play a more important role in selecting the optimal execution plans. This is because one can no longer employ the exhaustive search algorithm to search for the optimal plan (as the factorial number $m!$ of plans grows too large for the thorough search to be done in a practical time) and instead, has to use any of the limited search algorithms which basically searches only a portion of an entire search space for a near-optimal plan, hence leaving out the other portion of the search space unsearched. As a consequence, by the limited search space one should very much rely on the quality obtained of the estimated cost for execution plans calculated by an accurate estimation technique so that such estimated costs will be reliable and thus the plan selected as near-optimal will be trustworthy in most of the times, namely, will be the best among all plans in the limited space.

Either the exhaustive or limited search algorithm can plug in the query optimiser architecture we have proposed in this chapter. The analysis for the query evaluation cost and optimisation cost we have derived is also flexible to handle both search algorithms.

Interesting future work consists of (1) comparing the optimiser with the semi-dynamic and fully dynamic approaches and (2) comparing the semi-dynamic or fully dynamic query optimiser with HIST, the most popular histogram technique

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	1.0727e+11	1.0729e+11	0.016
R1R4R5	1.0703e+11	1.0705e+11	0.016
R2R3R5	1.0737e+11	1.0738e+11	0.016

(a) SJ1

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	1.8075e+10	1.8094e+10	0.104
R1R4R5	1.8063e+10	1.8082e+10	0.103
R2R3R5	1.8051e+10	1.8069e+10	0.104

(b) SJ2

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	4.0543e+10	4.0560e+10	0.042
R1R4R5	4.0538e+10	4.0555e+10	0.042
R2R3R5	4.0598e+10	4.0615e+10	0.042

(c) SJ3

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	1.3867e+11	1.3869e+11	0.013
R1R4R5	1.3854e+11	1.3856e+11	0.013
R2R3R5	1.3831e+11	1.3833e+11	0.013

(d) SJ4

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	2.2538e+10	2.2555e+10	0.075
R1R4R5	2.2614e+10	2.2631e+10	0.075
R2R3R5	2.2470e+10	2.2487e+10	0.075

(e) SJ5

sorted rel	no-samp-cost	samp-cost	diff%
R1R2R3	2.4286e+11	2.4288e+11	0.008
R1R4R5	2.4156e+11	2.4158e+11	0.008
R2R3R5	2.4156e+11	2.4158e+11	0.008

(f) SJ6

Table 6.10: Total costs by HYBRID with 15-time resampling before and after adding sampling costs

Total cost = A total of query execution costs for all 100 queries used.
15-time resampling = Resampling has been done 15 times for all star join selectivities.
no-samp-cost = Consider only the query execution cost per query without samp-cost.

implemented in many commercial database systems.

	resamp.		
sorted rel	15	1	imp%
R1R2R3	1.0729e+11	1.0748e+11	0.175
R1R4R5	1.0705e+11	1.0901e+11	1.828
R2R3R5	1.0738e+11	1.0765e+11	0.251

(a) SJ1

	resamp.		
sorted rel	15	1	imp%
R1R2R3	1.8094e+10	2.0655e+10	14.155
R1R4R5	1.8082e+10	1.8433e+10	1.946
R2R3R5	1.8069e+10	1.8231e+10	0.892

(b) SJ2

	resamp.		
sorted rel	15	1	imp%
R1R2R3	4.0560e+10	4.3024e+10	6.073
R1R4R5	4.0555e+10	4.1470e+10	2.256
R2R3R5	4.0615e+10	4.0812e+10	0.484

(c) SJ3

	resamp.		
sorted rel	15	1	imp%
R1R2R3	1.3869e+11	1.3951e+11	0.587
R1R4R5	1.3856e+11	1.3921e+11	0.467
R2R3R5	1.3833e+11	1.3850e+11	0.124

(d) SJ4

	resamp.		
sorted rel	15	1	imp%
R1R2R3	2.2555e+10	2.2804e+10	1.101
R1R4R5	2.2631e+10	2.2626e+10	-0.023
R2R3R5	2.2487e+10	2.2814e+10	1.455

(e) SJ5

	resamp.		
sorted rel	15	1	imp%
R1R2R3	2.4288e+11	2.4432e+11	0.590
R1R4R5	2.4158e+11	2.4573e+11	1.719
R2R3R5	2.4158e+11	2.4380e+11	0.915

(f) SJ6

Table 6.11: Total costs by HYBRID between 1-time and 15-time resampling with samp-cost added

Total cost = A total of query execution costs for all 100 queries used.
15-time resampling = Resampling has been done 15 times for all star join selectivities.

hybrid	srswr	imp%	#q
5.3174e+06	5.3174e+06	0.000	10
3.0848e+07	3.0848e+07	0.000	8
1.0058e+09	1.0066e+09	0.080	21
1.3661e+10	1.4419e+10	5.545	33
9.2566e+10	9.3915e+10	1.457	28
1.0727e+11	1.0937e+11	1.964	100

(a) SJ1, R1R2R3

hybrid	srswr	imp%	#q
5.3174e+06	5.3174e+06	0.000	10
3.0848e+07	3.0848e+07	0.000	8
1.0014e+09	1.0066e+09	0.526	21
1.3680e+10	1.4419e+10	5.398	33
9.2315e+10	9.3915e+10	1.733	28
1.0703e+11	1.0937e+11	2.189	100

(b) SJ1, R1R4R5

hybrid	srswr	imp%	#q
5.3174e+06	5.3174e+06	0.000	10
3.0848e+07	3.0848e+07	0.000	8
9.5421e+08	1.0066e+09	5.499	21
1.3666e+10	1.4419e+10	5.509	33
9.2709e+10	9.3915e+10	1.300	28
1.0736e+11	1.0937e+11	1.873	100

(c) SJ1, R2R3R5

hybrid	srswr	imp%	#q
5.7544e+06	5.6778e+06	-1.331	12
5.8784e+07	5.4083e+07	-7.997	14
1.5561e+09	1.6716e+09	7.421	38
8.5915e+09	9.1758e+09	6.800	30
7.8627e+09	8.8382e+09	12.406	6
1.8075e+10	1.9745e+10	9.241	100

(d) SJ2, R1R2R3

hybrid	srswr	imp%	#q
6.4295e+06	8.0516e+06	25.229	13
5.6366e+07	5.1709e+07	-8.261	13
1.5588e+09	1.6716e+09	7.239	38
8.5786e+09	9.1758e+09	6.961	30
7.8627e+09	8.8382e+09	12.406	6
1.8063e+10	1.9745e+10	9.314	100

(e) SJ2, R1R4R5

hybrid	srswr	imp%	#q
5.7544e+06	5.6778e+06	-1.331	12
5.6964e+07	5.4083e+07	-5.058	14
1.5589e+09	1.6716e+09	7.229	38
8.5661e+09	9.1758e+09	7.118	30
7.8627e+09	8.8382e+09	12.406	6
1.8050e+10	1.9745e+10	9.390	100

(f) SJ2, R2R3R5

hybrid	srswr	imp%	#q
4.1845e+06	4.6321e+06	10.696	9
4.3895e+07	4.9913e+07	13.710	10
1.3323e+09	1.3339e+09	0.120	27
1.4428e+10	1.4677e+10	1.728	42
2.4734e+10	2.5328e+10	2.401	12
4.0543e+10	4.1394e+10	2.100	100

(g) SJ3, R1R2R3

hybrid	srswr	imp%	#q
4.6430e+06	4.6321e+06	-0.235	9
4.3851e+07	4.9913e+07	13.825	10
1.3316e+09	1.3339e+09	0.169	27
1.4426e+10	1.4677e+10	1.741	42
2.4731e+10	2.5328e+10	2.413	12
4.0538e+10	4.1394e+10	2.112	100

(h) SJ3, R1R4R5

hybrid	srswr	imp%	#q
4.6430e+06	4.6321e+06	-0.235	9
4.9913e+07	4.9913e+07	0.000	10
1.3312e+09	1.3339e+09	0.197	27
1.4478e+10	1.4677e+10	1.378	42
2.4734e+10	2.5328e+10	2.403	12
4.0597e+10	4.1394e+10	1.962	100

(i) SJ3, R2R3R5

Table 6.12: Results with 15-time resampling and with no-samp-cost between HYBRID and SRSWR

For both HYBRID and SRSWR,
15-time resampling = Resampling has been done 15 times for all star join selectivities.
no-samp-cost = Consider only the query execution cost per query without samp-cost.

hybrid	srswr	imp%	#q
5.2529e+06	5.4131e+06	3.050	11
2.9286e+07	2.5063e+07	-14.418	7
1.4842e+09	1.4967e+09	0.838	29
9.8338e+09	1.0306e+10	4.808	27
1.2732e+11	1.2831e+11	0.778	26
1.3867e+11	1.4014e+11	1.062	100

(j) SJ4, R1R2R3

hybrid	srswr	imp%	#q
5.8563e+06	6.1951e+06	5.785	12
2.4281e+07	2.4281e+07	0.000	6
1.5736e+09	1.5733e+09	-0.020	30
1.0023e+10	1.0230e+10	2.061	26
1.2691e+11	1.2831e+11	1.102	26
1.3854e+11	1.4014e+11	1.159	100

(k) SJ4, R1R4R5

hybrid	srswr	imp%	#q
5.2529e+06	5.4131e+06	3.050	11
2.9564e+07	2.5063e+07	-15.224	7
1.4834e+09	1.4967e+09	0.897	29
9.8338e+09	1.0306e+10	4.808	27
1.2696e+11	1.2831e+11	1.065	26
1.3831e+11	1.4014e+11	1.326	100

(l) SJ4, R2R3R5

hybrid	srswr	imp%	#q
1.1683e+07	1.5030e+08	1186	25
3.2747e+07	3.4280e+07	4.683	9
1.0283e+09	1.0898e+09	5.988	28
1.3740e+10	1.3798e+10	0.418	32
7.7248e+09	7.7248e+09	0.000	6
2.2538e+10	2.2797e+10	1.150	100

(m) SJ5, R1R2R3

hybrid	srswr	imp%	#q
1.1683e+07	1.5030e+08	1186	25
3.2747e+07	3.4280e+07	4.683	9
1.0508e+09	1.0898e+09	3.709	28
1.3794e+10	1.3798e+10	0.029	32
7.7248e+09	7.7248e+09	0.000	6
2.2614e+10	2.2797e+10	0.810	100

(n) SJ5, R1R4R5

hybrid	srswr	imp%	#q
1.1683e+07	1.5030e+08	1186	25
3.3204e+07	3.4280e+07	3.242	9
1.0324e+09	1.0898e+09	5.566	28
1.3735e+10	1.3798e+10	0.455	32
7.6571e+09	7.7248e+09	0.884	6
2.2470e+10	2.2797e+10	1.457	100

(o) SJ5, R2R3R5

hybrid	srswr	imp%	#q
3.2927e+06	3.2865e+06	-0.190	7
9.7999e+07	9.3528e+07	-4.563	18
1.1368e+09	1.1832e+09	4.081	26
6.2470e+09	6.1989e+09	-0.770	17
2.3537e+11	2.3738e+11	0.852	32
2.4286e+11	2.4486e+11	0.824	100

(p) SJ6, R1R2R3

hybrid	srswr	imp%	#q
4.6230e+06	4.2685e+06	-7.666	8
9.3402e+07	9.2546e+07	-0.917	17
1.2090e+09	1.2535e+09	3.688	27
6.0310e+09	6.1286e+09	1.618	16
2.3422e+11	2.3738e+11	1.350	32
2.4156e+11	2.4486e+11	1.368	100

(q) SJ6, R1R4R5

hybrid	srswr	imp%	#q
3.7247e+06	3.7589e+06	0.916	7
9.5085e+07	9.3055e+07	-2.135	18
1.1147e+09	1.1510e+09	3.262	26
6.1335e+09	6.2311e+09	1.592	17
2.3421e+11	2.3738e+11	1.352	32
2.4156e+11	2.4486e+11	1.366	100

(r) SJ6, R2R3R5

Table 6.12: Results with 15-time resampling and with no-samp-cost between HYBRID and SRSWR

For both HYBRID and SRSWR,
15-time resampling = Resampling has been done 15 times for all star join selectivities.
no-samp-cost = Consider only the query execution cost per query without samp-cost.

CHAPTER 7

Conclusion

In this thesis, we examined the general problem of efficiently determining accurate estimates of query result sizes to assist in the task of query optimisation. The thesis contributes the following results:

- For both join and selection selectivity estimation, we have proposed and demonstrated the capability of HYBRID, a variant of sampling-based methods. The method is meant for centralised database systems where on-line sampling is appropriate.

- When on-line sampling is inappropriate, such as in distributed database systems, selectivity estimation by local regression, as a non-sampling-based method can be used. Local regression appears to generalise other major non-sampling-based methods proposed in the literature, e.g., UNF, all kinds of histogram methods, IASE and ASE such that they can all be implemented under a single framework. The main strength of the single framework implementation is that different methods have different strengths in dealing with different data distributions. Individual methods handle only some classes of data distribution well; however, their strengths are complementary and it appears that a combined scheme under our proposed framework might well provide a general solution.

- Two possible approaches *semi-dynamic* and *fully dynamic* are proposed to the implementation of a query optimiser that uses sampling for selectivity estimation. To date, there have been a large number of studies about selectivity estimation based on sampling techniques but to our best knowledge, no previous work has studied and analysed the feasibility of on-line sampling as a technique for join and selection selectivity estimation that can be used by query optimisers in database systems.

The remaining future work consists mainly of experimental work to clarify the following points:

- Compare many of the local regression variants proposed in Chapter 5 with serial histograms V-Optimal(V,A) and MaxDiff(V,A) to determine the performance of these methods with respect to different kinds of data distributions.

- Compare the query optimiser that uses the semi-dynamic and fully dynamic approaches.

- Compare the semi-dynamic query optimiser with non-sampling-based methods such as histogram and local regression. Likewise, compare the fully dynamic query optimiser with such non-sampling-based methods.

Bibliography

Aha, D. W. [1990], A Study of Instance-Based Algorithms for Supervised Learning Tasks: Mathematical, Empirical, and Psychological Evaluations, PhD thesis, Department of Information and Computer Science, University of California, Irvine, CA 92717.

Aha, D. W., Kibler, D. and Albert, M. K. [1991], 'Instance-Based Learning Algorithms', *Machine Learning* **6**(1), 37–66.

Aho, A. V., Sethi, R. and Ullman, J. D. [1985], *Compilers: Principles, Techniques and Tools*, Addison-Wesley.

Anderberg, M. R. [1973], *Cluster Analysis for Applications*, Academic Press.

Antoshenkov, G. [1993], Dynamic Query Optimization in rdb/VMS, *in* 'Proc. IEEE Int'l. Conf. on Data Engineering', Vienna, Austria, pp. 538–547.

Bayer, R. and McCreight, E. M. [1972], 'Organization and maintenance of large ordered indexes', *Acta Informatica, Springer Verlag (Heidelberg, FRG and NewYork NY, USA) Verlag* **1**(3), ˙Also published in/as: ACM SIGFIDET 1970, pp.107–141.

Belussi, A. and Faloutsos, C. [1995], Estimating the Selectivity of Spatial Queries using the Correlation's Fractal Dimension, *in* U. Dayal, P. M. D. Gray and S. Nishio, eds, 'VLDB '95: proceedings of the 21st International Conference on Very Large

Data Bases, Zurich, Switzerland, Sept. 11–15, 1995', Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, pp. 299–310.

Bennett, K. P., Ferris, M. C. and Ioannidis, Y. E. [1991], A Genetic Algorithm for Database Query Optimization, Technical Report TR 1004, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.

Bickel, P. J. and Freedman, D. A. [1981], 'Some Asymtotic Theory for the Bootstrap', *Annals of Statistics* **9**, 1196–1217.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. [1984], *Classification and Regression Tress*, Chapman & Hall, Inc.

Chakravarthy, S. [1991], Divide and conquer: A basis for augmenting a conventional query optimizer with multiple query processing capabilities, *in* 'International Conference on Data Engineering', IEEE Computer Society Press, Los Alamitos, Ca., USA, pp. 482–490.

Chao, T. J. and Egyhazy, C. J. [1986], Estimating temporary files sizes in distributed relational database systems, *in* 'Proc. IEEE Int'l. Conf. on Data Engineering', Los Angeles, CA, pp. 4–12.

Chen, C. M. and Roussopoulos, N. [1994], Adaptive Selectivity Estimation using Query Feedback, *in* 'Proceedings of 1994 ACM-SIGMOD International Conference on Management of Data'.

Choenni, R., Kersten, M. L., van den Akker, J. F. P. and Saad, A. [1996], On Multi-Query Optimization, Technical Report CS-R9638, National Research Institute for Mathematics and Computer Science (CWI), P.O. Box 94079, 1090 GB Amsterdam, Netherlands.

Christodoulakis, S. [1983*a*], Estimating Block Transfers and Join Sizes, *in* D. J. DeWitt and G. Gardarin, eds, 'SIGMOD'83, Proceedings of Annual Meeting', San Jose, California, pp. 40–54.

Christodoulakis, S. [1983*b*], 'Estimating Record Selectivities', *Information System* **8**(2), 105–115.

Christodoulakis, S. [1984], 'Implications of certain assumptions in data base performance evaluation', *ACM Transactions on Database Systems* **9**(2), 163–186.

Cleveland, W. S. [1979], 'Robust Locally Weighted Regression and Smoothing Scatterplots', *Journal of the American Statistical Association* **74**, 829–836.

Cleveland, W. S. and Devlin, S. J. [1988], 'Locally Weighted Regression: An approach to regression analysis by local fitting', *Journal of the American Statistical Association* **83**, 596–610.

Cleveland, W. S. and Grosse, E. H. [1991], 'Computational Methods for Local Regression', *Statistics and Computing* **1**, 47–62.

Cleveland, W. S. and Loader, C. R. [1996], Smoothing by local regression: Principles and methods, *in* W. Hardle and M. G. Schimek, eds, 'Statistical Theory and Computational Aspects of Smoothing', Physica Verlag, Heidelberg.

Cochran, W. G. [1963], *Sampling Techniques*, second edition edn, John Wiley & Sons, Inc.

Cosar, A., Lim, E.-P. and Srivastava, J. [1993], Multiple query optimization with depth-first, branch-and-bound and dynamic query ordering, *in* B. Bhargava, T. Finin and Y. Yesha, eds, 'Proceedings of the 2nd International Conference on Information and Knowledge Management', ACM Press, New York, NY, USA, pp. 433–438.

Cox, D. R. [1952], 'Estimation by Double Sampling', *Biometrika* **39**, 217–227.

Dasarathy, B. V., ed. [1991], *Nearest Neighbor (NN) Norms : NN Pattern Classification Techniques*, IEEE Computer Society Press.

Dayal, U. [1984], Query Processing in a Multidatabase System, *in* W. Kim, D. S. Reiner and D. S. Batory, eds, 'Query Processing in Database Systems', Springer Verlag, pp. 81–108.

Delobel, C., Lécluse, C. and Richard, P. [1995], *Databases: From Relational to Object-Oriented Systems*, International Thomson Computer Press.

DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R. and Wood, D. [1984], 'Implementation Techniques for Main Memory Database Systems', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **14**(2), 1–8.

Dogac, A., Halici, U., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Dengi, C., Mancuhan, S., Arpinar, B., Koksal, P. and Evrendilek, C. [1996], Metu Interoperable Database System, Demo Description, *in* 'Proceedings of ACM Sigmod Intl. Conf. on Management of Data'. Montreal.

Draper, N. R. and Smith, H. [1966], *Applied Regression Analysis*, John Wiley & Sons, Inc.

Du, W., Chan, M. C. and Dayal, U. [1995], Reducing Multidatabase Query Response Time by Tree Balancing, *in* 'ACM SIGMOD Intl. Conf. on Management of Data'.

Du, W., Krishnamurthy, R. and Shan, M. [1992], Query Optimization in Heterogeneous DBMS, *in* 'Proceedings of the 18th VLDB Conference', pp. 277–291.

Duda, R. O. and Hart, P. E. [1974], *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York.

Efron, B. [1979], 'Bootstrap Methods: Another Look at the Jackknife', *Annals of Statistics* **7**, 1–26.

Elmasri, R. and Navathe, S. B. [1991], *Fundamentals of Database Systems*, Benjamin/Cummings, chapter 18, pp. 501–534. Query Processing and Optimization.

Epstein, R. and Stonebraker, M. [1980], Analysis of Distributed Data Base Processing Strategies, *in* 'VLDB 1980', pp. 92–101. Proceedings of the Very Large Database Conference.

Evrendilek, C., Dogac, A., Nural, S. and Ozcan, F. [1995], Query Decomposition, Optimization and Processing in Multidatabase Systems, *in* 'Proc. of Workshop on Next Generation Information Technologies and Systems'. Naharia, Israel.

Freytag, J. C. [1987], A rule-based view of query optimization, *in* U. Dayal and I. Traiger, eds, 'Proceedings of the ACM SIGMOD Annual Conference', acm, ACM Press, San Francisco, CA, pp. 173–180.

Gardalin, G., Gruser, J. R. and Tang, Z. H. [1995], A Cost Model for Clustered Object-Oriented Databases, *in* 'Proceedings of the 21st VLDB Conference', Zürich, Switzerland, pp. 323–334.

Gardarin, G., Gruser, J.-R. and Tang, Z.-H. [1996], Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases Database Systems, *in* 'International Conference on Very Large DataBase, VLDB'96'. Bombay, India.

Gardy, D. and Puech, C. [1989], 'On the Effect of Join Operations on Relation Sizes', *ACM Transactions on Database Systems* **14**(4), 574–603.

Grichting, W. L. [1995], 'Bootstrapping and Resampling Stats: what are the differences ?', A news-net article in reply to the question. Author's address: Social Science, University of Tasmania, Launceston, Tasmania, Australia.

Gruser, J. R., Florescu, D., Naacke, H., Tang, Z. H. and Ziane, M. [1996], 'Flora - a Query Optimizer for OODBMSs', *Ingineering of Information Systems (ISI), AFCET* .

Haas, L. M., Freytag, J. C., Lohman, G. M. and Pirahesh, H. [1989], 'Extensive query processing in starburst', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **18**(2), 377–388.

Haas, P. J., Naughton, J. F., Seshadri, S. and Stokes, L. [1995], Sampling-based Estimation of the Number of Distinct Values of an Attribute, *in* 'Proceedings of the 21st VLDB Conference', pp. 311–322. Zurich, Switzerland.

Haas, P. J., Naughton, J. F., Seshadri, S. and Swami, A. N. [1993], Fixed-Precision Estimation of Join Selectivity, *in* 'Proc. 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', pp. 190–201.

Haas, P. J., Naughton, J. F., Seshadri, S. and Swami, A. N. [1996], 'Selectivity and Cost Estimation for Joins Based on Random Sampling', *Journal of Computer and System Sciences* **52**(3), 550–569.

Haas, P. J. and Swami, A. N. [1992], Sequential Sampling Procedures for Query

Size Estimation, *in* 'ACM SIGMOD Conference on the Management of Data', pp. 341–350.

Haas, P. J. and Swami, A. N. [1995], Sampling-Based Selectivity Estimation for Joins Using Augmented Frequency Value Statistics, *in* 'The International Confererence on Data Engineering', pp. 522–531.

Harangsri, B., Shepherd, J. A. and Ngu, A. H. H. [1996*a*], Query Classification in Multidatabase Systems, *in* 'Seventh Australian Database Conference', Australian Computer Society, pp. 147–156. Melbourne, Australia.

Harangsri, B., Shepherd, J. and Ngu, A. [1996*b*], Query Optimisation in Multidatabase Systems using Query Classification, *in* '11th Annual ACM Symposium ON Applied Computing (SAC'96)', ACM, pp. 173–177. Marriott Hotel, Philadelphia, Pennsylvania.

Harangsri, B., Shepherd, J. and Ngu, A. [1996*c*], Query Size Estimation using Systematic Sampling, *in* 'International Symposium on Cooperative Database Systems for Advanced Applications', pp. 400–403. December 5–7, Heian Shrine, Kyoto, Japan.

Harangsri, B., Shepherd, J. and Ngu, A. [1997], Query Size Estimation using Machine Learning, *in* 'Database Systems for Advanced Applications 1997 (DASFAA'97)'. Melbourne, Australia.

Harangsri, B., Shepherd, J. and Ngu, A. [1998], Building More Efficient Histograms by Systematic Sampling, *in* 'International Workshop on Issues and Applications of Database Technology (IADT'98)', Berlin, Germany, July 6–9. Accepted for publication.

Hellerstein, J. M. and Stonebraker, M. [1993], Predicate Migration: Optimizing Queries with Expensive Predicates, *in* 'Proc. ACM SIGMOD Conf.'.

Hevner, A. R. and Yao, S. [1979], 'Query Processing in Distributed Database Systems', *IEEE Transactions on Software Engineering* **5**(3), 177–187.

Holland, J. H. [1975], Adaptation in natural and artificial systems, Technical report, University of Michigan Press.

Hou, W.-C. and Ozsoyoglu, G. [1991], 'Statistical Estimators for Aggregate Relational Algebra Queries', *ACM Transactions on Database Systems* **16**(4), 600–654.

Hou, W.-C., Özsoyoğlu, G. and Dogdu, E. [1991*a*], 'Error-constrained COUNT query evaluation in relational databases', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **20**(2), 278–287.

Hou, W., Ozsoyoglu, G. and Dogdu, E. [1991*b*], Error Constrained COUNT Query Evaluation in Relational Databases, *in* 'ACM-SIGMOD Conference on the Management of Data', pp. 278–287.

Hou, W., Ozsoyoglu, G. and Taneja, B. K. [1988], Statistical Estimators for Relational Algebra Expressions, *in* 'Proceedings of the 7th ACM Symposium on Principles of Database Systems', pp. 276–287.

Hou, W., Ozsoyoglu, G. and Taneja, B. K. [1989], Processing Aggregates Relational Queries with Hard Time Constraints, *in* 'ACM-SIGMOD Conference on the Management of Data', pp. 68–77.

INGRES, U. [1988], 'INGRES version 8.9', Public domain source code from University of California, Berkeley.

InterBase [1998]. InterBase Software Corporation, [http://www.interbase.com].

Ioannidis, Y. [1993], Universality of Serial Histograms, *in* 'Proceedings of the 19th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Dublin'.

Ioannidis, Y. E. and Christodoulakis, S. [1991], On the Propagation of Errors in the Size of Join Results, *in* 'Proceedings of the ACM-SIGMOD Intl. Conf. on Management of Data', pp. 268–277.

Ioannidis, Y. E. and Christodoulakis, S. [1993], 'Optimal Histograms for Limiting Worst-case Error Propagation in the Size of the Join Results', *ACM Transactions on Database Systems* **18**(4), 709–748.

Ioannidis, Y. E. and Kang, Y. C. [1990], Randomized Algorithms for Optimizing Large Join Queries, *in* 'Proceedings of the 1990 ACM-SIGMOD Conference', pp. 312–321.

Ioannidis, Y. E., Ng, R. T., Shim, K. and Sellis, T. K. [1992], Parametric query processing, *in* 'Proceedings of the 18th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Vancouver'.

Ioannidis, Y. E. and Poosala, V. [1995], Balancing Histogram Optimality and Practicality for Query Result Size Estimation, *in* 'ACM SIGMOD International Conference on Management of Data', pp. 233–244.

Ioannidis, Y. E. and Wong, E. [1987], Query Optimization by Simulated Annealing, *in* 'Proceedings of the 1987 ACM-SIGMOD Conference', pp. 9–22. San Francisco, CA, June 1987.

Jarke, M. [1984], Common subexpression isolation in multiple query optimization, *in* W. Kim, D. S. Reiner and D. S. Batory, eds, 'Query Processing in Database Systems', Springer Verlag, pp. 191–205.

Kang, Y. C. [1991], Randomized Algorithms for Query Optimization, PhD thesis, Computer Sciences, University of Wisconsin, Madison, Wisconsin.

Kibler, D., Aha, D. W. and Albert, M. K. [1989], 'Instance-Based Prediction of Real-Valued Attributes', *Computational Intelligence* **5**, 51–57.

Kirkpatrick, S., Jr., C. D. G. and Vecchi, M. P. [1983], 'Optimization by Simulated Annealing', *Science* **220**, 671–680.

Korth, H. F. and Silberschatz, A. [1991], *Database System Concepts*, McGraw-Hill, New York.

Krisnamurthy, R., Boral, H. and Zaniolo, C. [1986], Optimization of Nonrecursive Queries, *in* 'Proceedings of the 12th Int. Conf. on Very Large Data Bases', pp. 128–137.

Laarhoven, P. J. M. V. and Aarts, E. H. L. [1988], *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, Holland.

Lanzelotte, R. S. G. [1990], OPUS: An Extensible OPtimizer for Up-to-date Database Systems, PhD thesis, Computer Science, La Pontificia University.

Ling, Y. and Sun, W. [1995], An Evaluation of Sampling-Based Size Estimation Methods for Selections in Database Systems, *in* 'The International Confererence on Data Engineering', pp. 532–539.

Lipton, R. J. and Naughton, J. F. [1989], Estimating the Size of Generalized Transitive Closures, *in* 'Proceedings of the Fifteenth International Conference on Very Large Data Bases', Amsterdam, pp. 165–171.

Lipton, R. J. and Naughton, J. F. [1990], Query Size Estimation by Adaptive Sampling, *in* ACM, ed., 'Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems: April 2–4, 1990, Nashville, Tennessee', pp. 40–46.

Lipton, R. J., Naughton, J. F. and Schneider, D. A. [1990], Practical Selectivity Estimation through Adaptive Sampling, *in* 'Proceedings of ACM SIGMOD', pp. 1–12.

Loader, C. R. [1997*a*], 'Bandwidth Selection', [http://cm.bell-labs.com/stat/project/locfit/index.html]. Chapter 9 on the web page.

Loader, C. R. [1997*b*], 'Locfit: An Introduction'. To appear in Statistical Computing and Graphics Newsletter.

Loader, C. R. [1997*c*], Software Package for Local Regression. [http://cm.bell-labs.com/stat/project/locfit].

Loader, C. R. [1997*d*], 'Univariate Local Regression', [http://cm.bell-labs.com/stat/project/locfit/index.html]. Chapter 2 on the web page. There are many other useful chapters, e.g., Local regression, Multivariate local regression, Methods and visualization, Local Likelihood Estimation, Density Estimation, Bandwidth Selection and Adaptive Fitting.

Makinouchi, A., Tezuka, M., Kitakami, H. and Adachi, S. [1981], The optimization strategy for query evaluation in RDB/V1, *in* 'Proceedings of the Seventh International Conference on Very Large Data Bases', pp. 518–529.

Manber, U. [1989*a*], *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, chapter 4, pp. 61–90. A brief introduction to data structures.

Manber, U. [1989*b*], *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, chapter 3, pp. 37–60. Analysis of algorithms.

Mandelbrot, B. B. [1983], *The Fractal Geometry of Nature*, updated and augmented [ed.] edn, W.H. Freeman, New York. First published as Fractals. in 1977.

Mannino, M. V., Chu, P. and Sager, T. [1988], 'Statistical profile estimation in database systems', *ACM Computing, Springer Verlag (Heidelberg, FRG and NewYork NY, USA)-Verlag Surveys* **20**(3).

Melli, G. [1997], 'SCDS - a Synthetic Classification Data Set Generator: program to generate data sets', [http://fas.sfu.ca/cs/people/GradStudents/melli/SCDS/]. email melli@cs.sfu.ca.

Merz, C. J. and Murphy, P. M. [1996], 'UCI Repository of Machine Learning Databases', [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Mishra, P. and Eich, M. [1992], 'Join Processing in Relational Databases', *compsurv* **24**(1), 63–113.

Mitchell, G. A. [1993], Extensible Query Processing in an Object-Oriented Database, CS-93-16, Department of Computer Science, Brown University, Providence, Rhode Island, 02912.

Morzy, T., Matysiak, M. and Salza, S. [1994], Tabu Search Optimization of Large Join Queries, *in* 'Advances in Database Technology – EDBT'94', pp. 311–322.

Munoz, A. [1994], An Extensible Query Optimizer Architecture for the TIGUKAT Objectbase Management System, TR94-01, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

Muralikrishna, M. [1988], Optimization of Multiple-Disjunct Queries in a Relational Database System, PhD thesis, Computer Sciences, University of Wisconsin, Madison.

Muralikrishna, M. and DeWitt, D. [1988], Equi-depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, *in* 'Proceedings of the ACM SIGMOD Conf. on Management of Data', pp. 28–36.

Murthy, M. N. and Rao, T. J. [1988], *Systematic Sampling with Illustrative Examples*, Vol. 6, Elsevier Science Publishers, chapter 7, pp. 147–185. Handbook of Statistics.

Ngu, A. H. H., Yan, L.-L. and Wong, L. [1993], Heterogeneous Query Optimization using Maximal Sub-Queries, *in* S. C. Moon and H. I. (Eds.), eds, 'Proceedings of the 3rd International Conference on Database Systems for Advanced Applications (DASFAA)', World Scientific Press, Singapore, pp. 413–420. Daejon, Korea.

Olken, F. [1993], Random Sampling from Databases, PhD thesis, Computer Science, University of California at Berkeley.

Oracle [1996*a*], *Oracle7 Server Concepts Manual.* Release 7, Oracle Corporation.

Oracle [1996*b*], *Oracle7 Server Tuning.* Release 7, Oracle Corporation.

Ozcan, F., Nural, S., Koksal, P., Evrendilek, C. and Dogac, A. [1996], Dynamic Query Optimization on a Distributed Object Management Platform, *in* 'Proc. of Fifth International Conference on Information and Knowledge Management (CIKM '96)'. Maryland, USA.

Ozkan, C., Dogac, A. and Altinel, M. [1996], 'A Cost Model for Path Expressions in Object-Oriented Queries', *Journal of Database Management* **7**(3), unknown page numbers.

Ozsu, M. and Valduriez, P. [1991], *Principles of Distributed Database Systems*, Prentice-Hall, New Jersey.

Park, J. and Segev, A. [1988], Using common subexpressions to optimize multiple queries, *in* 'Proc. IEEE Int'l. Conf. on Data Eng.', Los Angeles, CA, p. 311.

Pearl, J. [1984], *Heuristics – Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Co., Reading, MA.

Piatetsky-Shapiro, G. and Connell, C. [1984], Accurate Estimation of the Number of Tuples Satisfying a Condition, *in* 'Proceedings of the ACM SIGMOD Conference', pp. 256–276. Boston, Mass, June, ACM, New York.

Pongpinigpinyo, S. [1996], Distributed Query Optimisation using Two Stage Simulated Annealing, Master's thesis, Department of Computer Science, University of Tasmania, GPO Box 252C, Hobart, Tasmania, 7001, Australia.

Poosala, V. [1997], Histogram-Based Estimation Techniques in Database Systems, PhD thesis, Computer Science, University of Wisconsin.

Poosala, V. and Ioannidis, Y. E. [1997], Selectivity Estimation without the Attribute Value Independence Assumption, *in* 'VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases', pp. 486–495.

Poosala, V., Ioannidis, Y. E., Haas, P. J. and Shekita, E. J. [1996], Improved Histograms for Selectivity Estimation of Range Predicates, *in* H. V. Jagadish and I. S. Mumick, eds, 'Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data', Montreal, Quebec, Canada, pp. 294–305.

Quinlan, J. R. [1992], Learning with Continuous Classes, *in* 'Proceedings AI'92, Adams and Sterling, Eds', World Scientific, Singapore, pp. 343–348.

Quinlan, J. R. [1993*a*], *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, California.

Quinlan, J. R. [1993*b*], Combining Instance-Based and Model-Based Learning, *in* 'Proceedings of Machine Learning', Morgan Kaufmann.

Quinlan, J. R. [1996], 'Why choosing 3 queries other than other numbers ?', Private communication.

Rumelhart, D. E., McClelland, J. L. and the PDP research group. [1986], *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations*, MIT Press.

Salzberg, B. [1988], *File Structures: An Analytic Approach*, Prentice-Hall International.

Scheaffer, R. L., Mendenhall, W. and Ott, L. [1990], *Elementary Survey Sampling*, fourth edn, PWS-KENT Publishing Company.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. and Price, T. G. [1979*a*], Access Path Selection in a Relational Database Management System, *in* 'Proc. ACM-SIGMOD International Conference on Management of Data', ACM, Boston New York, pp. 23–34.

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. and Price, T. G. [1979*b*], Access Path Selection in a Relational Database Management System, Technical Report RJ2429, IBM Research Laboratory, San Jose.

Sellis, T. K. [1988], 'Multiple query optimization', *ACM Transactions on Database Systems* **13**(1).

Shapiro, L. D. [1986], 'Join Processing in Database Systems with Large Main Memories', *ACM Transactions on Database Systems* **11**(3), 239–264.
URL: *http://www.acm.org/pubs/toc/Abstracts/tods/6315.html*

Simon, J. L. and Bruce, P. [1997], 'Probability and statistics the resampling way', http://www.statistics.com/. An article on the Resampling Stats home page.

Singh, K. [1981], 'On the Asymtotic Accuracy of Efron's Bootstrap', *Annals of Statistics* **9**, 1187–1195.

Smith, E. P. and Belle, G. V. [1984], 'Nonparametric Estimation of Species Richness', *Biometrics* **40**, 119–129.

Software AG Americas, I. [1998], 'Cost based optimizer vs. rule based optimizer'. [http://www.sagus.com].

SOLID [1998]. Solid Information Technology LTD., [http://www.solidtech.com].

Stillger, M. and Spiliopoulou, M. [1996], Genetic Programming in Database Query Optimization, *in* '1st Annual Conference on Genetic Programming', Standard.

Straube, D. D. [1990], Queries and Query Processing in Object-Oriented Database Systems, TR90-33, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

Sun, W., Ling, Y., Rishe, N. and Deng, Y. [1993], An Instant and Accurate Size Estimation Method for Joins and Selection in a Retrieval-Intensive Environment, *in* 'Proceedings of ACM SIGMOD', pp. 79–88.

Swami, A. [1989*a*], Optimization of Large Join Queries, PhD thesis, Standard University.

Swami, A. [1989*b*], Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques, *in* 'Proceedings of the 1989 ACM-SIGMOD Conference', pp. 367–376.

Swami, A. and Gupta, A. [1988], Optimization of Large Join Queries, *in* 'Proceedings of the 1988 ACM-SIGMOD Conference', pp. 8–17. Chicago, IL, June 1988.

Swami, A. and Schiefer, K. B. [1994], On the Estimation of Join Result Sizes, *in* 'Proc. International Confererence on Extending Database Technology (EDBT'94)', Springer-Verlag, Berlin.

Thomson, S. [1992], *Sampling*, John Wiley & Sons, Inc. Basic and Advanced Sampling Methods.

Turney, P. and Jankulak, M. [1993], 'Summary Table of Database Statistics', File can be obtained from ftp://ftp.ics.uci.edu/pub/machine-learning-databases/SUMMARY-TABLE. The analysis of 64 real-world databases donated by the authors.

Ullman, J. D. [1988*a*], *Principles of Database and Knowledge-base Systems*, Vol. 2, Computer Science Press.

Ullman, J. D. [1988*b*], *Principles of Database and Knowledge-base Systems*, Vol. 1, Computer Science Press.

Winston, P. H. [1984], *Artificial Intelligence, Second Edition*, Addison-Wesley. A good introductory book on artificial intelligence. Winston introduces the reader to many of the different facets of AI. Very readable.

Wong, E. and Youssefi, K. [1976], 'Decomposition—a Strategy for Query Processing', *ACM Transactions on Database Systems* **1**(3), 223–241.

Yoo, H. [1990], Intelligent Search in Query Optimization, PhD thesis, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan, 48109-2122 USA.

Zhu, Q. [1992], Query Optimization in Multidatabase Systems, *in* 'Proceedings of the 1992 CAS Conference', Vol. 2, Toronto, Canada, pp. 111–127.

Zhu, Q. [1993], An Integrated Method for Estimating Selectivities in a Multidatabase System, *in* 'Proceedings of Distributed Computing (CASCON' 93)', Vol. 2, Toronto, Ontario, Canada, pp. 832–847.

Zhu, Q. and Larson, P. A. [1994], A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System, *in* 'Data Engineering', pp. 144–153.

Ziarko, W. [1991], *The Discovery, Analysis, and Representation of Data Dependencies in Databases*, AAAI Press / The MIT Press, 445 Burgess Drive, Menlo PArk, California 94025, chapter 11, pp. 107–123. Knowledge Discovery in Databases, edited by Gregory Piatetsky-Shapiro and William J. Frawley.

Zipf, G. K. [1949], *Human behaviour and the principle of least effort*, Addison-Wesley, Reading, MA.

Author Index