Probabilistic Task Assignment in Edge Computing

Seth Brozdzik
Department of Electrical and Computer Engineering
Rowan University
Glassboro, NJ
brodzi22@students.rowan.edu

Michael Paulson
Department of Computer Science
Texas State University
San Marcos, TX
mrp160@txstate.edu

Xiao Chen
Department of Computer Science
Texas State University
San Marcos, TX
xc10@txstate.edu

***Abstract*—Recently, edge computing has attracted a lot of attention from academia and industry. One of the important problems in edge computing is the task assignment problem. Many task assignment optimization problems in the literature do not consider uncertain parameters. In this paper, we adopt the chance-constrained method as a powerful paradigm to model uncertainty in our task assignment optimization problem. Chance-constrained programming is one of the most difficult classes of optimization problems. To solve our defined problem, we propose a method called FMS that finds the optimal solution accurately and quickly. We first transform the original probabilistic problem into an equivalent problem using the Gauss error function, and then rely on an auxiliary problem to decrease the search space and find the candidate points that lead to the optimal solution and, therefore, solve the defined problem. Simulation results confirm the correctness and efficiency of our method.**

***Index Terms*—candidate points, chance-constrained, edge computing, optimization, search space**

I. Introduction

Recently, edge computing has attracted a lot of attention from academia and industry [9], [11]. It pushes mobile computing, network control, and storage to distributed devices at the network edge, providing server resources, data analysis, and artificial intelligence closer to data collection sources. It offers faster data processing, generates less network traffic, and is less costly than cloud computing.

One of the important problems in edge computing is the task assignment problem [16], which involves the transfer of resource-intensive computational tasks to external platforms or devices on the edge. Numerous people have worked on various task assignment optimization problems [3], [7], [18], but most of the problems do not involve random parameters. In reality, many parameters are uncertain; for example, the latency between the time the client sends out a task to an edge device and the time the client receives a reply is not predictable. In this paper, we adopt the *chance-constrained* method [1] as a powerful paradigm to model uncertainty in our optimization problem. This method allows us to formulate an optimization problem that ensures that the probability of meeting a certain constraint is above a certain level. Chance-constrained methods have a plethora of applications from telecommunication and medicine to finance [17].

In this paper, we consider a chance-constrained task assignment optimization problem. More specifically, we assume that a client needs to choose some of n edge devices to process tasks. If the client chooses device i, there is a network delay d_i associated with it. The delay d_i is a stochastic variable which we assume follows a Gaussian distribution with mean μ_i and variance σ_i^2, i.e., $d_i \sim N(\mu_i, \sigma_i^2)$, $i = 1, 2, \cdots, n$. We also know that the amount of work each device can finish in its capacity is w_i. The devices process the work sequentially. Our goal is to assign work to devices such that, with a probability (confidence level) p, the total delay is minimized and the total amount of work finished exceeds a certain amount W.

The chance-constrained method is a relatively robust approach to model uncertainty; however, it is often difficult to solve [2]. One way to try is brute force, where all possible assignments are enumerated. However, this method is not scalable with an increase in n. Inspired by the ideas from [13], [14], we propose an algorithm called *Finding the Minimal Solution* (FMS) to quickly reach the minimum solution by substantially reducing the search space. The main idea is as follows: We first transform the original chance-constrained probabilistic problem, which we call $\mathcal{P}1$, using the Gauss error function [4] into an equivalent problem $\mathcal{P}2$. Then, we relate $\mathcal{P}2$ to an auxiliary problem $\mathcal{P}3$. With a series of theorems, we prove that we can significantly decrease the search space and find the candidate points that lead to the solution to $\mathcal{P}2$ through $\mathcal{P}3$. After we obtain the candidate points of $\mathcal{P}2$, we choose the one that minimizes our optimization goal as the solution to $\mathcal{P}2$, and therefore solve the original problem $\mathcal{P}1$. To verify the correctness of our algorithm, we conduct simulations comparing the proposed FMS method and a variation method called *Finding the minimal solution using Binary Search* (FBS) with the ground truth solution provided by the brute forth algorithm (BF). Simulation results show that, using the brute force method as a benchmark, most of the time, both MFS and FBS can produce the exact same minimal solutions as the BF method. And when the solutions are different, the deviation is only about a few percentage points from the optimal. In the process of finding the optimal solution, both MFS and FBS have greatly narrowed down the search space, with FBS using a few more attempts than MFS due to its simplicity while MFS successfully pinpointing the solution in a couple of tries thanks to its consideration of the problem structure.

The differences of our work from others and the key contributions of our work are as follows:

- We define a chance-constrained task assignment opti-

mization problem in an edge network.
- We propose a method to solve the probabilistic problem with high accuracy and a small search space.
- Simulation results confirm the correctness and efficiency of our method.

The rest of the paper is organized as follows: Section II references the related work, Section III presents the problem, Section IV provides the solution, Section V describes the simulations conducted, and the conclusion is in Section VI.

II. Related Work

In the literature, many papers have discussed task assignment optimization problems in edge networks [3], [7], [18]. In [3], the authors proposed a multi-objective optimization solution to assign different application tasks to different edge devices while minimizing the energy consumption of edge devices and the computation time of tasks. In [7], the authors put forward a new joint task assignment and resource allocation approach in a multi-user environment to minimize energy consumption with application latency constraint. And in [18], the authors presented a task offloading and power assignment optimization algorithm for minimizing task completion time under mobile device energy constraint. In these optimization problems, uncertain parameters are not considered, which is what we aim to address in this paper.

To model optimization problems under uncertainty, we adopt the chance-constraint approach, a formulation of an optimization problem to ensure that the probability of meeting a certain constraint is above a certain level. Some early work dates back to the 1950s, with pioneers such as Charnes and Cooper [6], Charnes et al. [5], Miller and Wagner [12], and Prékopa [15], who considered problems with individual or joint chance constraints. Chance-constrained problems are one of the most difficult classes of optimization problems [10] and are generally difficult to solve, except for special cases such as the minimum spanning tree [8] and linear optimization [17].

The closely related papers to our problem are [13] and [14]. In [13], the authors consider the problem of finding the shortest paths in a graph with independent randomly distributed edge lengths. The goal is to maximize the probability that the path length does not exceed a given threshold. They assume each edge has independent normally distributed length and relate the problem to the linear combination of mean and variance. In [14], the author focuses on a general framework for reliable stochastic combinatorial optimization that includes mean-risk minimization and models involving the probability tail of the stochastic cost of a solution. Inspired by these ideas, we develop a solution to our defined problem in this paper.

III. Problem Definition

The problem we want to solve in this paper is defined as follows: A client has some tasks that need to be processed by edge devices and there are n wireless edge devices available. If a client uses device i, there is a network delay d_i associated with the device. Delay d_i is assumed to follow a Gaussian distribution with mean μ_i and variance σ_i^2, i.e., $d_i \sim N(\mu_i, \sigma_i^2)$, $i = 1, 2, \cdots, n$. The amount of work each device can finish in its capacity is w_i. The devices process the work sequentially. The goal is to assign work to devices such that with probability p, the total delay is minimized and the total amount of work finished exceeds W. The mathematical representation of our problem, which we call $\mathcal{P}1$, is as follows:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & D \\ \text{s. t.} \quad & P(\sum_{i=1}^{n} d_i x_i < D) \geq p \\ & \sum_{i=1}^{n} w_i x_i \geq W \\ & x_i \in \{0, 1\}, i = 1, 2, \cdots, n \end{aligned} \tag{$\mathcal{P}1$}$$

In $\mathcal{P}1$, D is the total delay. Parameter x_i is either 0 or 1. If a device i is chosen, $x_i = 1$; Otherwise, $x_i = 0$. The vector $X = \{x_1, x_2, \cdots, x_n\}$ that can minimize D is the optimal solution we want to find to solve $\mathcal{P}1$.

IV. Our Method

A. Transformation

In order to find the optimal solution to $\mathcal{P}1$, we first make the following transformation. Since each d_i follows a Gaussian distribution $d_i \sim N(\mu_i, \sigma_i^2)$, $i = 1, 2, \cdots, n$, the summation of these random variables $\sum_{i=1}^{n} d_i x_i$ also follows a Gaussian distribution with mean $\mu = \sum_{i=1}^{n} \mu_i x_i$ and standard deviation $\sigma = \sqrt{\sum_{i=1}^{n} \sigma_i^2 x_i}$. We rewrite the probabilistic condition in terms of the Gauss error function (erf) [4] as follows.

$$P(\sum_{i=1}^{n} d_i x_i < D) = \frac{1}{2}(1 + erf(\frac{D - \mu}{\sqrt{2}\sigma})) \geq p$$

So,
$$D \geq \mu + [\sqrt{2} erf^{-1}(2p - 1)]\sigma$$

If p is given, then $\sqrt{2} erf^{-1}(2p - 1)$ is a constant. We denote it by A, i.e., $A = \sqrt{2} erf^{-1}(2p - 1)$. Then,

$$D \geq \mu + A\sigma$$

Therefore, to minimize D, we need to find the minimum value of $\mu + A\sigma$. So, we have the following minimization problem, which we call $\mathcal{P}2$, that is equivalent to problem $\mathcal{P}1$. From now on, we will focus on finding the optimal solution to $\mathcal{P}2$ and thereafter solving $\mathcal{P}1$.

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \mu + A\sigma \\ \text{s. t.} \quad & \sum_{i=1}^{n} w_i x_i \geq W \\ & x_i \in \{0, 1\}, i = 1, 2, \cdots, n \end{aligned} \tag{$\mathcal{P}2$}$$

Now let us consider the time complexity of finding the minimum value of $\mu + A\sigma$. As we know, if a device i is chosen, its $x_i = 1$, otherwise $x_i = 0$. Since there are n

devices, there are 2^n choices of these devices. Therefore, the time complexity is $O(2^n)$ if we use the brute force method, which is exponential and not scalable when n is large.

B. Auxiliary Problem

Next, we create an auxiliary problem to problem $\mathcal{P}2$, which we call problem $\mathcal{P}3$, as follows:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & \mu + k\sigma^2 \\ \text{s. t.} \quad & \sum_{i=1}^{n} w_i x_i \geq W \\ & x_i \in \{0,1\}, i = 1, 2, \cdots, n \end{aligned} \tag{$\mathcal{P}3$}$$

The goal of $\mathcal{P}3$ is similar to that of $\mathcal{P}2$, but it is a linear function of (μ, σ^2) with a coefficient k(where $k \geq 0$) instead of (μ, σ) with a constant A. Here, k can take different values and plays a very important role in solution search. We denote the solution X to $\mathcal{P}3$ at k as $X(k)$. The constraints $\sum_{i=1}^{n} w_i x_i \geq W$ in the two problems are the same. That is, if we find a solution vector X that satisfies the constraint in $\mathcal{P}3$, it must satisfy the constraint in $\mathcal{P}2$, even though the minimum solution to $\mathcal{P}3$ may not be the minimum solution to $\mathcal{P}2$. However, we can find all the candidate points of $\mathcal{P}2$ through $\mathcal{P}3$, as we will later prove that the candidate points of $\mathcal{P}3$ are also the candidate points of $\mathcal{P}2$. With the candidate points, we can quickly find the minimal solution to $\mathcal{P}2$.

The following theorems show the properties of $\mathcal{P}3$.

Theorem 1: For a pair (μ, σ^2), $\mu + k\sigma^2$ is a monotonically increasing function with respect to k.

Proof. Suppose (μ_1, σ_1^2) and (μ_2, σ_2^2) are the minimum solutions to problem $\mathcal{P}3$ at k_1 and k_2 ($k_2 > k_1$), respectively. We want to prove that $\mu_1 + k_1\sigma_1^2 < \mu_2 + k_2\sigma_2^2$. Since the pair ($\mu_1$, σ_1^2) is the minimum solutions to $\mathcal{P}3$ at k_1, we have $\mu_1 + k_1\sigma_1^2 \leq \mu_2 + k_1\sigma_2^2$. And since $k_2 > k_1$, we have $\mu_2 + k_1\sigma_2^2 < \mu_2 + k_2\sigma_2^2$. Combining the two, we have $\mu_1 + k_1\sigma_1^2 \leq \mu_2 + k_1\sigma_2^2 < \mu_2 + k_2\sigma_2^2$. Therefore, $\mu + k\sigma^2$ is a monotonically increasing function with respect to k. □

Theorem 2: Suppose (μ_1, σ_1^2) and (μ_2, σ_2^2) are the minimum solutions to problem $\mathcal{P}3$ at k_1 and k_2, respectively. If $k_2 > k_1$, then $\sigma_2^2 \leq \sigma_1^2$.

Proof. Because (μ_1, σ_1^2) is the minimum solution at k_1, we have $\mu_1 + k_1\sigma_1^2 \leq \mu_2 + k_1\sigma_2^2$. Also, because ($\mu_2$, σ_2^2) is the minimum solution at k_2, we have $\mu_1 + k_2\sigma_1^2 \geq \mu_2 + k_2\sigma_2^2$. Subtracting the two sides of these two, we get $(k_2 - k_1)\sigma_1^2 \geq (k_2 - k_1)\sigma_2^2$. Therefore, $\sigma_2^2 \leq \sigma_1^2$. □

C. Our Proposed Algorithm FMS

Now we present the main algorithm *Finding the Minimum Solution to* $\mathcal{P}2$ (FMS) in Fig. 1. It has three steps. In the first two steps, we rely on the auxiliary problem $\mathcal{P}3$ to produce the candidate points of our target $\mathcal{P}2$ problem. In Step (1), we call Algorithm *Reducing Search Space* (RSS) in Fig. 2 to reduce the search space of $\mathcal{P}3$ by narrowing the range of k from [0, ∞] to [0, h]. In Step (2), we call Algorithm *Finding Candidate Points* (FCP) in Fig. 3 to return all the candidate points of $\mathcal{P}3$ in the k range [0, h]. These candidate points are also the candidate points of $\mathcal{P}2$ (proof below). Finally, in Step (3), we find the minimum solution to $\mathcal{P}2$ from these candidate points.

Algorithm FMS: Finding the Minimum Solution to $\mathcal{P}2$

Require: Input: μ_i, σ_i, w_i for $i = \{1, 2, \cdots, n\}$, A, W
Output: the minimum solution X to $\mathcal{P}2$

1: Call Algorithm RSS to get the search range of [0, h] in $\mathcal{P}3$
2: Call Algorithm FCP to obtain all the candidate points in the search range [0, h] in $\mathcal{P}3$
3: Compare all the X values of the candidate points using the objective function $\mu + A\sigma$ to get the minimum solution to $\mathcal{P}2$

Fig. 1. Algorithm to find the minimum solution to $\mathcal{P}2$

In the following, we will explain these algorithms in detail.

Step 1. Algorithm RSS

In Step (1) of Algorithm FMS, we call Algorithm RSS in Fig. 2 to decrease the search range of k in $\mathcal{P}3$ from [0, ∞] to [0, h], where $h = \frac{A}{\sigma}$. In RSS, we first solve $\mathcal{P}3$ at 0 and obtain $X(0)$ (line 1). In line 2, we assign σ_0 to σ. The notation σ_k represents the σ at k. As long as $k\sigma \neq A$, we repeat the following (lines 4-6): we update k to $\frac{A}{\sigma}$, solve $\mathcal{P}3$ at k to get $X(k)$, and assign σ_k to σ. After the loop terminates, we get the upper bound h of the search range. We return the k range [0, h] and the corresponding solutions $X(0)$ and $X(h)$.

Algorithm RSS: Reducing Search Space

Require: Input: μ_i, σ_i, w_i for $i = \{1, 2, \cdots, n\}$, A, W
Output: range [0, h] and the corresponding $X(0)$ and $X(h)$

1: solve $\mathcal{P}3$ at 0 to get $X(0)$
2: $\sigma = \sigma_0$
3: **while** $k\sigma \neq A$ **do**
4: $\quad k = \frac{A}{\sigma}$
5: $\quad$ solve $\mathcal{P}3$ at k to get $X(k)$
6: $\quad \sigma = \sigma_k$
7: **end while**
8: $h = k$
9: output range [0, h] and the corresponding $X(0)$ and $X(h)$

Fig. 2. Finding the search range of k in $\mathcal{P}3$

In Theorem 3, we show that the minimum solution to $\mathcal{P}2$ cannot lie in the range of k from h to ∞ in $\mathcal{P}3$. Therefore, we only need to search the range of k from 0 to h in $\mathcal{P}3$ to find the minimum solution to $\mathcal{P}2$, substantially reducing the search space for $\mathcal{P}2$.

Theorem 3: The minimum solution to $\mathcal{P}2$ cannot lie in the k range of ($\frac{A}{\sigma}$ ∞) in $\mathcal{P}3$.

Proof. Suppose (μ, σ^2) is the minimum solution to $\mathcal{P}3$ at $\frac{A}{\sigma}$. We denote (μ', σ'^2) as the minimum solution to $\mathcal{P}3$ for any $k' > \frac{A}{\sigma}$. We want to show that $\mu + A\sigma \leq \mu' + A\sigma'$. Since ($\mu$, σ^2) is the minimum solution to $\mathcal{P}3$ at $\frac{A}{\sigma}$, we have $\mu + A\sigma = \mu + \frac{A}{\sigma}\sigma^2 \leq \mu' + \frac{A}{\sigma}\sigma'^2$. And from Theorem 2,

we have $\mu' + \frac{A}{\sigma}\sigma'^2 \leq \mu' + \frac{A}{\sigma'}\sigma'^2 = \mu' + A\sigma'$. Putting these together, we have $\mu + A\sigma \leq \mu' + A\sigma'$. Since k' is any number greater than $\frac{A}{\sigma}$, this means we cannot get a lower value in $\mathcal{P}2$ by trying any k' in the range of ($\frac{A}{\sigma}$, ∞) in $\mathcal{P}3$. Therefore, the minimum solution to $\mathcal{P}2$ cannot lie in the k range of ($\frac{A}{\sigma}$, ∞) in $\mathcal{P}3$. □

Step 2. Algorithm FCP

In Step (2) of Algorithm FMS, we call Algorithm FCP in Fig. 3 to find all the candidate points in the k search range [0, h] in $\mathcal{P}3$. We show that the candidate points in $\mathcal{P}3$ are also the candidate points in $\mathcal{P}2$.

The following theorem discloses the design idea of FCP.

Theorem 4: For a k range [a, b], if $X(a) = X(b)$, then for any $c \in [a, b]$, we have $X(c) = X(a) = X(b)$.

Proof. From Theorem 2, we know that $\sigma_b^2 \leq \sigma_c^2 \leq \sigma_a^2$. If $X(a) = X(b)$, then $\sigma_a^2 = \sigma_b^2$. Therefore, the only way for $\sigma_b^2 \leq \sigma_c^2 \leq \sigma_a^2 = \sigma_b^2$ to hold is to let $\sigma_a^2 = \sigma_c^2 = \sigma_b^2$. Since c can be any value in the range [a, b], then all the k values in this range have the same σ^2 value.

Since (μ_a, σ_a^2) is the solution to $\mathcal{P}3$ at a, we have $\mu_a + a\sigma_a^2 \leq \mu_c + a\sigma_c^2$. From the above, $\sigma_a^2 = \sigma_c^2$. Therefore, $\mu_a \leq \mu_c$. Now, we show that μ_a cannot be less than μ_c. We prove this by contradiction. Suppose μ_a is strictly less than μ_c, i.e., $\mu_a < \mu_c$. Since c can be any value in the range [a, b], if $c = b$, then $\mu_c = \mu_b$. Therefore, $\mu_a < \mu_b$. But if $X(a) = X(b)$, then $\mu_a = \mu_b$. We reach a contradiction. So, μ_a can only be equal to μ_c and therefore equal to μ_b as well. This means that the μ values of all the ks in the range of [a, b] are the same. Combining the result of the σ^2 values above, we have proved that $X(c) = X(a) = X(b)$. □

Theorem 4 tells us that, for a range [a, b] of k, if $X(a) = X(b)$, then we need only consider a or b as a candidate point and can ignore all other values in the range. However, if $X(a) \neq X(b)$, then there may be more candidate points in the range. In that case, we pick an intermediate point c. If $X(a) = X(c)$, then, besides a, we need only search the range (c, b] for more candidate points. If $X(c) = X(b)$, then, besides b, we need only search the range [a, c) for more candidate points. In this way, we can further narrow the search space.

How do we determine an intermediate point c in the range of [a, b]? The choice of c will affect the efficiency and accuracy of our method. Ideally, it is a candidate point that causes (μ, σ^2) to change. We need to search for it. To simplify, we can use binary search, picking the middle point in the range each time to reduce the search space. Alternatively, we can use the following method that considers the property of the functions: we look at two functions defined as $f_1(k) = \mu_a + k\sigma_a^2$ and $f_2(k) = \mu_b + k\sigma_b^2$. In these two functions, k is a variable and the others are constants. Therefore, these two functions are lines and they intersect in the range of [a, b] because $f_1(a) < f_1(b)$ and $f_2(a) > f_2(b)$. To get the cross point, we let $f_1(k) = f_2(k)$. Thus, the cross point c is: $-\frac{\mu_a - \mu_b}{\sigma_a^2 - \sigma_b^2}$. We treat the cross point as a candidate point.

With these preparations, we have Algorithm FCP to find all the candidate points in the range of k from 0 to h in $\mathcal{P}3$.

Algorithm FCP: Finding Candidate Points

Require: Input: μ_i, σ_i, w_i for $i = \{1, 2, \cdots, n\}$, [0, h], $X(0)$, $X(h)$, W
Output: candidate point set S_p

1: **if** $X(0) == X(h)$ **then**
2: add element $\{0, X(0)\}$ to the candidate point set S_p
3: return
4: **end if**
5: add elements $\{0, X(0)\}$ and $\{h, X(h)\}$ to the candidate point set S_p
6: add range [0, h] to the range set S_r
7: **while** S_r is not empty **do**
8: pop the first range [a, b] from S_r
9: $c = -\frac{\mu_a - \mu_b}{\sigma_a^2 - \sigma_b^2}$
10: **if** $c \neq a$ and $c \neq b$ **then**
11: solve $\mathcal{P}3$ at c to get $X(c)$
12: **if** $X(c) == X(a)$ and $X(c) \neq X(b)$ **then**
13: add range (c, b] to S_r
14: **else if** $X(c) \neq X(a)$ and $X(c) == X(b)$ **then**
15: add range [a, c) to S_r
16: **else if** $X(c) \neq X(a)$ and $X(c) \neq X(b)$ **then**
17: add element $\{c, X(c)\}$ to S_p
18: add ranges [a, c) and [c, b] to S_r
19: **end if**
20: **end if**
21: **end while**

Fig. 3. Finding all the candidate points in the k range of [0, h] in $\mathcal{P}3$

In Algorithm FCP, we first check if the X values (from Algorithm RSS) at 0 and h are the same (line 1). If so, according to Theorem 4, we only need to add element $\{0, X(0)\}$ to the candidate point set S_p and return (line 2-3). In S_p, each element contains the k value and the corresponding $X(k)$. If the X values at 0 and h are different, we add two candidate points 0 and h, and their corresponding X values into the candidate point set S_p (line 5). We also add the range [0, h] into the range set denoted by S_r (line 6). Then, as long as the range set S_r is not empty, we do the following. We pop the first range [a, b] from S_r (line 8). We calculate the cross point c (line 9). If c is not equal to a nor b (line 10), then we solve $\mathcal{P}3$ at c to get $X(c)$. Then there are three cases (lines 12-19). If $X(c)$ is the same as $X(a)$ but not $X(b)$ (line 12), then there cannot be new candidate points in the range [a, c] from Theorem 4. So, we only add (c, b] to the range set S_r (line 13) to further explore the candidate points. Similarly, we only add range [a, c) to S_r if $X(c)$ is the same as $X(b)$ but not $X(a)$ (lines 14-15). Finally, if $X(c)$ is not equal to $X(a)$ nor $X(b)$ (line 16), we need to add c and its corresponding X value to the candidate point set (line 17) and both [a, c) and [c, b] to the range set (line 18) because there can be more candidate points in both sections. The output of Algorithm FCP will be all the candidate points and their corresponding

X values in $\mathcal{P}3$. The following theorem shows that these candidate points are also the candidate points of $\mathcal{P}2$.

Theorem 5: The candidate points in $\mathcal{P}3$ are also the candidate points in $\mathcal{P}2$.

Proof. In Algorithm FCP, we only add an intermediate point c in the range of $[a, b]$ to the candidate point set S_p when $X(c)$ is different from either $X(a)$ or $X(b)$ in $\mathcal{P}3$. If there is a change in the X value at c from that at a or b, there may be a change in the corresponding μ and σ^2 in $\mathcal{P}3$ and, consequently, a change in the objective function $\mu + A\sigma$ in $\mathcal{P}2$. On the other hand, if there is no change in μ and σ^2 in $\mathcal{P}3$, there should be no change in $\mu + A\sigma$ in $\mathcal{P}2$. Therefore, the candidate points we consider to solve $\mathcal{P}3$ are also the candidate points we consider to solve $\mathcal{P}2$. □

Step 3. Solving Problem $\mathcal{P}2$

The final step of the FMS algorithm is to compare all the X values of the candidates points using the objective function $\mu + A\sigma$ to get the minimum solution to the $\mathcal{P}2$ problem. If $\mathcal{P}2$ is solved, then the original equivalent $\mathcal{P}1$ is solved.

V. Simulations

In this section, we evaluate the performance of our proposed algorithm using a customized simulator written in Matlab.

A. Algorithms Compared

We compared the following algorithms to solve the $\mathcal{P}2$ problem, and thus the $\mathcal{P}1$ problem.

1) FMS: our proposed algorithm.
2) FBS: finding the minimal solution using binary search. In this algorithm, we search for candidate points using binary search in the k range $[0, h]$.
3) BF: brute force algorithm. We enumerate all the possibilities of X to find the minimum solution to $\mathcal{P}2$. This provides the ground truth to evaluate the accuracy of our proposed algorithm.

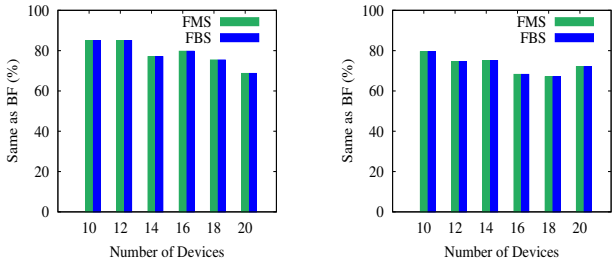
B. Metrics

We used the following metrics.

1) The accuracy of FMS and FBS compared with the brute force algorithm BF. We calculated the percentage of cases in which algorithms FMS and FBS can produce the same results as BF.
2) If the solutions from FMS and FBS differ from that of BF, how much is the difference? We calculated the deviation of FMS and FBS from BF.
3) The number of candidate points evaluated to find the minimum solution to $\mathcal{P}2$.

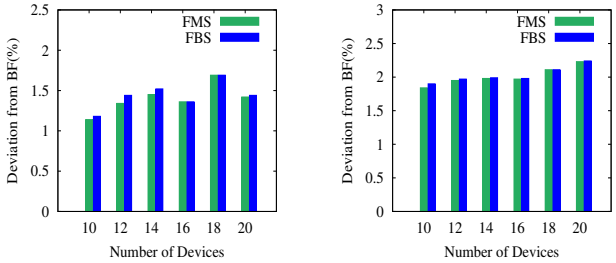
C. Settings

In our simulations, we tried n values from 10 to 20 in increments of 2. For each device i, we randomly generated μ_i from the range [10, 100], σ_i from the range $[-5, 5]$, and w_i from the range [100, 200]. We set the probability p to 0.95 and 0.99, and W to 500. For each simulation, we ran 500 times and averaged the results.



(a) $p = 0.95$ (b) $p = 0.99$

Fig. 4. Percentages FMS and FBS finding the same result as BF



(a) $p = 0.95$ (b) $p = 0.99$

Fig. 5. The deviation of FMS and FBS from BF when results are different

D. Experiments

1) Experiment 1: We compared the optimal solutions found by FMS and FBS with that of the BF algorithm. We calculated the percentage of cases in which algorithms FMS and FBS produced the exact same minimal $\mu + A\sigma$ (rounded to four decimal places) as BF.

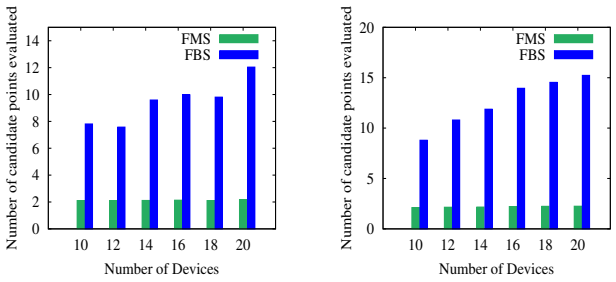
The simulation results are shown in Figs. 4 (a) and (b) with $p = 0.95$ and $p = 0.99$, respectively. In both figures, $70\% - 80\%$ of the time, FMS and FBS found exactly the same optimal solution as BF. Comparing FMS and FBS, there is not much difference between the solutions they found. Despite the simplicity of FBS, its accuracy is satisfactory.

Next, we look at the solutions where FMS and FBS differ from that of the BF. We used the following formula to calculate the deviation of the minimal values produced by FMS or FBS from that of the BF.

$$\frac{opt_A - opt_{BF}}{opt_{BF}} \tag{1}$$

In formula (1), variable opt_A refers to the minimal $\mu + A\sigma$ found by FMS or FBS and opt_{BF} is the minimal $\mu + A\sigma$ generated by BF. The results of this metric are shown in Figs. 5(a) and (b).

We can see from the results that when $p = 0.95$, the deviations of FMS and FBS from BF are less than 2% and when $p = 0.99$, the deviations are less than 2.5%. This means that both algorithms can produce results quite close to, if not the same as, the actual optimal. Comparing FMS and FBS, we observe that FBS has slightly higher deviation than FMS due to the fact that it does not take the structure of the problem into account.



(a) $p = 0.95$ (b) $p = 0.99$

Fig. 6. The number of candidate points compared in FMS and FBS

Number of devices	FMS	FBS	BF
10	2.1067	7.8067	2^{10}
12	2.0833	7.5733	2^{12}
14	2.1167	9.5900	2^{14}
16	2.1467	9.9733	2^{16}
18	2.1067	9.8033	2^{18}
20	2.1600	12.0400	2^{20}

Fig. 7. Number of candidate points compared in FMS, FBS, and BF to find the optimal solution when $p = 0.95$

2) Experiment 2: We compared the number of candidate points evaluated in FMS, FBS, and BF to find the solution to $\mathcal{P}2$. We ran the simulations 500 times and averaged the results, which are shown in Figs. 6(a) and (b) and table 7.

Figs. 6(a) and (b) compare the results of FMS and FBS only. In both $p = 0.95$ and $p = 0.99$, FBS evaluated significant more candidate points than FMS to find the optimal solution. This is because FBS simply adds the middle point of a range without considering the properties of the problem, while FMS looks for the turning point based on the problem structure. The number of candidate points evaluated by FMS is only around 2, which means FMS is very quick in pinpointing the optimal solution.

Since the BF algorithm traverses all the possibilities to find the minimal solution, the number of candidate points it evaluates is 2^n, which rises exponentially as n increases. It is hard to put its numbers with those of the other two into the figures. So we use the table in Fig. 7 to compare the three algorithms. The table illustrates how FMS and FBS, particularly FMS, can save the effort required to find the optimal solution to the problem. Here, we just present the table when $p = 0.95$ due to space limitations; it is similar when $p = 0.99$.

To summarize, we can see that both MFS and FBS produce the exact same optimal solutions as BF most of the time, and when the solutions differ, the deviation is only a few percentage points away from the optimal. In finding the optimal solution, both MFS and FBS have substantially reduced the search space, with FBS getting the optimal solution in a few more attempts than MFS due to its simple idea, and MFS successfully pinpointing the solution in only a couple of attempts thanks to its digestion of the problem structure.

VI. Conclusion

In this paper, we worked on a chance-constrained task assignment optimization problem to deal with uncertainty and proposed a method, FMS, to solve it. We first transformed the original probabilistic problem into an equivalent problem using the Gauss error function, and then relied on an auxiliary problem to decrease the search space and find the candidate points, and then obtain the optimal solution to our defined problem. We conducted simulations to evaluate the performance of our method. Simulation results confirmed the correctness and efficiency of our method. In this paper, we assumed that the delay of each device follows a Gaussian distribution. In the future, we will explore cases where parameters follow other distributions and where devices process tasks concurrently.

Acknowledgment

This material is based upon work supported by the National Science Foundation under REU grant #2149950.

References

[1] Chance-constraint method. https://optimization.mccormick.northwestern.edu/index.php/Chance-constraint_method.

[2] W. Ackooij, R. Zorgati, R. Henrion, and A. Möller. Chance Constrained Programming and Its Applications to Energy Management. *Stochastic Optimization - Seeing the Optimal for the Uncertain*, 2011.

[3] S. Almasri, M. Jarrah, and B. Al-Duwairi. Multi-objective optimization of task assignment in distributed mobile edge computing. *J Reliable Intell Environ*, 8:21–33, 2022.

[4] L. C. Andrews. *Special functions of mathematics for engineers*. SPIE Press, 1998.

[5] A. Charnes, W. Cooper, and G. Y. Symonds. Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Management Science*, 4(3):235–263, 1958.

[6] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6:73–79, 1959.

[7] B. Dab, N. Aitsaadi, and R. Langar. Joint optimization of offloading and resource allocation scheme for mobile edge computing. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7, 2019.

[8] H. Ishii, S. Shiode, T. Nishida, and Y. Namasuya. Stochastic spanning tree problem. *Discrete Applied Mathematics*, 3(4):263–273, 1981.

[9] A. Jebamani and G. Winster. A Survey of Edge Computing in IOT devices, 2022.

[10] S. Küçükyavuz and R. Jiang. Chance-constrained optimization under limited distributional information: A review of reformulations based on sampling and distributional robustness, 2021.

[11] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi. Resource scheduling in edge computing: A survey, 2021.

[12] B. L. Miller and H. M. Wagner. Chance constrained programming with joint constraints. *Operations Research*, 13(6):930–945, 1965.

[13] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher. Stochastic Shortest Paths Via Quasi-convex Maximization. In *Algorithms – ESA 2006*, pages 552–563. Springer Berlin Heidelberg, 2006.

[14] Evdokia Nikolova. Approximation algorithms for reliable stochastic combinatorial optimization. In *APPROX-RANDOM*, 2010.

[15] A. Prékopa. Contributions to the theory of stochastic programming. *Mathematical Programming*, 4(1):202221, 1973.

[16] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, and et al. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 2021.

[17] A. Shapiro, D. Dentcheva, and A. Ruszczynski. Lectures on Stochastic Programming: Modeling and Theory, Second Edition. *Society for Industrial and Applied Mathematics*, 2014.

[18] X. Wang, Z. Zhou, H. Chen, and Y. Zhang. Task offloading and power assignment optimization for energy-constrained mobile edge computing. In *2021 Ninth International Conference on Advanced Cloud and Big Data (CBD)*, pages 302–307, 2022.