Joint Optimization of Task Offloading and Resource Allocation in Tactical Edge Networks

Zhaofeng Zhang, Xuanli Lin, Guoliang Xue, Yanchao Zhang, Kevin S. Chan

***Abstract*—Recent years have witnessed explosive growth in deploying IoT devices over tactical edge networks. Due to the limited computing resources of local edge devices, there is an urgent need to offload the computing tasks to edge servers. A central question here is "How can we design offloading strategies and resource allocation plans so that all the computing resource constraints of servers and communication constraints between devices and servers are satisfied?" In this paper, we formulate the problem of jointly optimizing task offloading and resource allocation (JOA) in tactical edge networks as maximizing the total task completion rewards subject to various resource constraints. We design two algorithms to solve the JOA problem. The first is an exact algorithm using a search tree, where the branch-cutting criterion is well-crafted based on the property of the JOA problem. The second is a meta-heuristic algorithm based on the simulated annealing approach. We conduct numerical evaluations and demonstrate that the proposed exact algorithm can obtain the optimal task offloading strategy and resource allocation plan. The heuristic algorithm can efficiently provide competitive performance.**

1. Introduction

With the rapid advancement of mobile computing and the emergence of Artificial Intelligence of Things (AIoT), millions of IoT devices are now deployed across tactical edge networks, generating vast amounts of data on the battlefield. However, advancing AI capabilities to the edge of tactical mobile computing ecosystem remains a complex challenge due to resource constraints of computing, network, and environment dynamics. Traditionally, the prevailing approach has been to transmit all data from IoT devices to cloud data centers for analysis. However, such data transfer operates with limited resources (e.g., limited power, memory, and network bandwidth) in a dynamic and adversarial environment, which may lead to tremendous transmission delays. Additionally, there are serious concerns about potential privacy breaches during data transfer. An alternative solution involves on-device analytics, where AI applications are executed directly on IoT devices, allowing for localized processing of data generated by these devices [3], [6], [11]. Despite this, the approach encounters significant limitations, particularly regarding inadequate performance and energy efficiency. Many AI applications require substantial computational resources, exceeding the capabilities of resource-constrained and energy-limited IoT devices [12], particularly in the demanding environment of the battlefield. Therefore, an urgent need is to offload these computing tasks to edge servers. On one hand, edge servers always have more powerful computing capacities than resource-constrained local devices. On the other hand, the communication time of offloading to the edge servers will significantly decrease compared to multi-hop wireless transmission to the cloud.

Nevertheless, the co-design of offloading strategies and resource allocation plans has not been well understood, especially for the tactical edge networks. In particular, local devices always only have limited power supplies. That is to say, the transmit power of each device needs to be carefully quantified to satisfy the sophisticated battlefield requirements (e.g., achieving minimal signal-to-interference-plus-noise ratio and not exceeding the maximum transmission time). In addition, each edge server has limited computing resources. Hence, a proper task offloading strategy must ensure that each edge server's computing capacities are not exceeded. To address the above issues, in this work, we seek to answer the following critical question: *"How can we design offloading strategies and resource allocation plans so that all the computing resource constraints of servers and communication constraints between devices and servers are satisfied?"*

One recent work [7] studies the problem of jointly optimizing task offloading and resource allocation (JOA) in tactical edge networks and formulates it as a multi-objective optimization problem. Motivated by this work, we take a step further and formulate this problem as maximizing the total task completion rewards subject to various resource constraints, including servers' computing capacities, local devices' transmit power, and tasks' completion deadlines. Although this problem is challenging to solve directly since it is a mixed nonlinear integer programming problem, we observe two critical properties by exploiting its structure. The first property reveals that, given an infeasible offloading policy, the new offloading policy will still be infeasible if an extra task is added. Then, we propose an exact algorithm to solve the JOA problem using a search tree where the cutting criterion is based on the above observation. The second property is that an arbitrary offloading policy's feasibility can be determined efficiently by solving a linear programming problem. Based on this, we further design a heuristic algorithm using the simulated annealing approach to solve the JOA problem.

Zhaofeng Zhang, Xuanli Lin, Guoliang Xue, and Yanchao Zhang are affiliated with Arizona State University and CoE-FutureG. Email: {xlin54, zzhan199, xue, yczhang}@asu.edu. Kevin S. Chan is affiliated with DEVCOM US Army Research Laboratory. Email: kevin.s.chan.civ@army.mil. Research was sponsored by the DEVCOM Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-23-2-0225. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DEVCOM Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

The main contributions of this paper are the following:

- We formulate the problem of jointly optimizing task offloading and resource allocation in tactical edge networks as maximizing the total task completion rewards subject to various resource constraints.
- We design two algorithms to solve the JOA problem respectively. The first is an exact algorithm using a search tree where the branch-cutting criterion is based on the property of the JOA problem. The second is a heuristic algorithm based on the simulated annealing approach.
- We conduct numerical evaluations and demonstrate that the proposed exact algorithm can obtain the optimal task offloading strategy and resource allocation plan. The heuristic one can provide competitive performance efficiently.

We present the system model in §2 and formulate the JOA problem in §3. We present exact and heuristic algorithms in §4 and §5, respectively. We present evaluation results in §6 and conclude the paper in §7.

2. System Model

We consider a multi-server tactical edge network where each edge server can provide computation offloading services to resource-constrained local mobile users, such as soldiers' wearable devices. Typically, each server is endowed with moderate computing capabilities allocated by the network operator and is capable of wireless communication with mobile users. Local users can offload machine learning (ML) inference tasks to an edge server. We assume that there are a set $\mathcal{U}$ of N users $\mathcal{U} = \{u_1, u_2, ..., u_N\}$ and a set $\mathcal{S}$ of M servers $\mathcal{S} = \{s_1, s_2, ..., s_M\}$. We describe the system model of user computation tasks, task uploading transmissions, and server computing resources in the following subsections.

A. User Computation Tasks

Each user $u_n \in \mathcal{U}$ has the following attributes:

- *Position*: The user u_n's position is represented by a coordinate (x_n, y_n) in two-dimensional Euclidean space.
- *Deadline*: The ML task's deadline for user u_n is δ_n.
- *Data size*: D_n specifies the input data required to transfer (including system settings, program codes, and input parameters) from the local user u_n to the server.
- *ML algorithm and its accuracy requirement*: Each user can choose one ML algorithm k from K candidates. The user u_n's ML algorithm $k\,(k = 1, ..., K)$ is characterized by a 4-tuple of parameters, $\langle \text{CPU}_{n,k}, \text{RAM}_{n,k}, \tau^{\text{exe}}_{n,k}, r_{n,k} \rangle$, in which $\text{CPU}_{n,k}$ is a positive integer indicating the user u_n's required number of CPU core for processing ML algorithm k on the server's virtual machine, $\text{RAM}_{n,k}$ is a positive integer indicating the user u_n's required RAM for running ML algorithm k on the server's virtual machine, and $\tau^{\text{exe}}_{n,k}$ denotes the algorithm k's execution time when user u_n runs it, and $r_{n,k}$ is a positive integer denoting the reward the user u_n earns when the algorithm k is selected. Although we do not consider that the task can be processed on the local device, our proposed approaches can still be applied to this setting.

B. Communication during Task Offloading

When user u_n offloads its task associated with the ML algorithm k to the server s_m, the resulting delay encompasses: (i) The communication time $\tau^{\text{up}}_{n,m}$ required for transmitting the input to the edge server via the uplink, (ii) The execution time $\tau^{\text{exe}}_{n,k}$ entailed in processing the task at the edge server, and (iii) The communication time is used to transmit the output from the edge server back to the user via the downlink. Given that the output size, typically represented by the ML inference result, tends to be considerably smaller than the input size, and further considering the higher data rate of the downlink compared to the uplink, we choose to disregard the delay associated with transmitting the output [2].

Here, we consider a wireless communication system employing orthogonal frequency-division multiple access (OFDMA) as its multiple access protocol in the uplink. Note that when another multiple access protocol (e.g., TDMA) is adopted, our proposed algorithms can still be adapted to that scheme. Within the OFDMA scheme, the operational frequency bandwidth B is partitioned into L equal sub-bands, each of width $W = \frac{B}{L}$. To ensure the orthogonality of offloading transmissions among users affiliated with the same server, each user is allocated to a distinct sub-band. Consequently, each server can serve at most L users at the same time. Let $\mathcal{L} = \{1, ..., L\}$ be the set of sub-bands at each server. We define the task offloading variables, which integrate both the uplink sub-band scheduling and the selection of machine learning algorithms, as $a_{n,m,k,l} \in \{0, 1\}$, where $a_{n,m,k,l} = 1$ indicates that the task from user u_n is offloaded to server s_m via the sub-band l and processed by the ML algorithm k, and $a_{n,m,k,l} = 0$ otherwise. As each task can be offloaded and executed to at most one server, a feasible offloading policy must satisfy the following constraint

$$\sum_{m,k,l} a_{n,m,k,l} \leq 1, \ \forall u_n \in \mathcal{U}. \tag{1}$$

Additionally, we denote $\mathcal{U}_{\text{off}}$ as the set of users that offload their tasks, and let $m(n)$, $k(n)$ and $l(n)$ be such that $a_{n,m(n),k(n),l(n)} = 1$.

We further let P_n to denote the transmit power of user u_n, $\forall u_n \in \mathcal{U}$, and assume

$$P_n = 0, \quad \forall u_n \notin \mathcal{U}_{\text{off}}, \tag{2}$$

$$0 \leq P_n \leq P_{\text{max}}, \quad \forall u_n \in \mathcal{U}_{\text{off}}, \tag{3}$$

where P_n is the transmission power of user u_n subject to a maximum budget P_{max}. Since users employ distinct sub-bands when transmitting to the same server, the uplink interference associated with the same server is effectively alleviated. Nevertheless, these users remain susceptible to interference among different servers. In this case, the Signal-to-Interference-plus-Noise Ratio (SINR) from user $u_n \in \mathcal{U}_{\text{off}}$ to server $s_{m(n)}$ on sub-band $l(n)$ is given by

$$\text{SINR}_{n,m(n),l(n)} = \frac{\frac{P_n}{(d(n,m(n)))^\alpha}}{\nu + \sum\limits_{\substack{n',n'\neq n \\ m',m'\neq m(n) \\ k}} a_{n',m',k,l(n)} \frac{P_{n'}}{(d(n',m(n))^\alpha}}, \ \forall u_n \in \mathcal{U}_{\text{off}},$$

where ν is the background noise, $d(n, m(n))$ is the distance between user u_n and server $s_{m(n)}$, and $\alpha \in [2, 4]$ is the path loss exponent [4], [8]. Hence, the transmission time of user u_n when sending its task to server $s_{m(n)}$ can be calculated as

$$\tau^{\text{up}}_{n,m(n)} = \frac{D_n}{W \log_2(1 + \text{SINR}_{n,m(n),l(n)})}, \ \forall u_n \in \mathcal{U}_{\text{off}}. \quad (4)$$

C. Server Computing Resources

Each server $s_m \in \mathcal{S}$ has the following attributes:

- Position: The server s_m's position is represented by a coordinate (x_m, y_m) in two-dimensional Euclidean space.
- CPU capacity: The server s_m's CPU capacity is a positive integer C_m, indicating the number of CPU cores.
- RAM capacity: The server s_m's RAM capacity is denoted by a positive integer R_m.

3. Problem Formulation

A. Joint Task Offloading and Resource Allocation problem

We let $\mathcal{A} = \{a_{n,m,k,l} | \forall u_n \in \mathcal{U}, s_m \in \mathcal{S}, k, l\}$ denote the task offloading policy, and $\mathcal{P} = \{P_n | \ \forall u_n \in \mathcal{U}_{\text{off}}\}$ denotes the transmit power allocation plan. Based on the above system model, we formulate the joint task offloading and resource allocation (JOA) problem as the maximization problem:

$$\max_{\mathcal{A},\mathcal{P}} R = \sum_{n,m,k,l} a_{n,m,k,l} \times r_{n,k}, \quad (5)$$

$$\text{s.t.} \quad \sum_{n,k,l} a_{n,m,k,l} \times \text{CPU}_{n,k} \leq C_m, \ \forall s_m \in \mathcal{S}, \quad (6)$$

$$\sum_{n,k,l} a_{n,m,k,l} \times \text{RAM}_{n,k} \leq R_m, \ \forall s_m \in \mathcal{S}, \quad (7)$$

$$\sum_{n,k,l} a_{n,m,k,l} \leq 1, \ \forall u_n \in \mathcal{U}, \quad (8)$$

$$a_{n,m,k,l} \in \{0, 1\}, \forall u_n \in \mathcal{U}, s_m \in \mathcal{S}, k, l, \quad (9)$$

$$\tau^{\text{up}}_{n,m(n)} + \tau^{\text{exe}}_{n,k(n)} \leq \delta_n, \ \forall u_n \in \mathcal{U}_{\text{off}}, \quad (10)$$

$$\text{SINR}_{n,m(n),l(n)} \geq \beta, \ \forall u_n \in \mathcal{U}_{\text{off}}, \quad (11)$$

$$0 \leq P_n \leq P_{\max}, \ \forall u_n \in \mathcal{U}_{\text{off}}. \quad (12)$$

Constraints (6) and (7) ensure that the CPU and RAM usage on each edge server does not exceed what is available. Constraint (8) ensures that each task can be offloaded and executed to at most one server. Constraint (10) guarantees that each user's total completion time does not exceed the task deadline. Constraint (11) ensures that the server successfully decodes the information transmitted from the user, i.e., the SINR must be larger than β. Constraint (12) ensures that the transmission power of each user does not exceed its budget. The above JOA optimization problem is a Mixed non-linear integer programming problem, which is generally challenging to solve [1]. We aim to design low-complexity solutions that achieve good performance while being practical to implement.

B. Properties of JOA

We observe the following two critical properties by exploiting the structure of the objective function and constraints of the JOA problem in (5).

Property 1. *Given an infeasible offloading policy, the new offloading policy will still be infeasible if an extra task is added.*

Proof. This follows directly from the fact that, given an infeasible offloading policy, any violated constraints among Constraints (6), (7), (10), and (11) will still be violated if an extra task is added. □

Property 2. *For a given offloading policy* $\mathcal{A} = \{a_{n,m,k,l} | \forall u_n \in \mathcal{U}, s_m \in \mathcal{S}, k, l\}$ *satisfying Constraint* (6)*,* (7)*,* (8)*, and* (9)*, its feasibility can be determined efficiently by solving the following system of linear inequalities*

$$\frac{P_n}{(d(n,m(n)))^\alpha} \geq \max\{\beta, 2^{\frac{\sigma_n}{B}} - 1\} \times \left\{ \nu + \sum_{\substack{n',n'\neq n \\ m',m'\neq m(n) \\ k}} a_{n',m',k,l(n)} \frac{P_{n'}}{(d(n',m(n))^\alpha} \right\}, \ \forall u_n \in \mathcal{U}_{\textit{off}},$$

$$0 \leq P_n \leq P_{\textit{max}}, \ \forall u_n \in \mathcal{U}_{\textit{off}}, \quad (13)$$

where $\sigma_n = \frac{D_n}{\delta_n - \tau^{exe}_{n,k(n)}}$, $\forall u_n \in \mathcal{U}_{\textit{off}}$*. This is a Linear Programming (LP) feasibility problem.*

Proof. Constraint (10) can be rewritten as

$$\text{SINR}_{n,m(n),l(n)} \geq 2^{\frac{\sigma_n}{B}} - 1, \ \forall u_n \in \mathcal{U}_{\text{off}}. \quad (14)$$

Thus, constraints (10) and (11) can be replaced by

$$\text{SINR}_{n,m(n),l(n)} \geq \max\{\beta, 2^{\frac{\sigma_n}{B}} - 1\}, \ \forall u_n \in \mathcal{U}_{\text{off}}. \quad (15)$$

This can be further rewritten as

$$\frac{P_n}{(d(n,m(n)))^\alpha} \geq \max\{\beta, 2^{\frac{\sigma_n}{B}} - 1\} \times \left\{ \nu + \sum_{\substack{n',n'\neq n \\ m',m'\neq m(n) \\ k}} a_{n',m',k,l(n)} \frac{P_{n'}}{(d(n',m(n))^\alpha} \right\}, \ \forall u_n \in \mathcal{U}_{\text{off}},$$

which finishes the proof. □

4. An Exact Algorithm

In this section, we design a branch and bound algorithm to obtain an optimal solution to the JOA problem using a search tree. Although the branch and bound algorithm paradigm has been known for a long time [9], we note that *our proposed bounding technique is novel* and can effectively obtain an optimal solution compared to the exhaustive search.

Specifically, our proposed algorithm traverses a search tree with the height of N, where each non-root level corresponds to user u_n's offloading strategy. Each node is presented by a 4-tuple (n, m, k, l). When $m = 0$, it means that the n-th user's task will not be offloaded; When $m \neq 0$, it means that

the n-th user's task will be offloaded to the m-th server using the l-th sub-band, and the task will be processed by the k-th ML algorithm. Each non-leaf node (n, m, k, l) has $MLK+1$ children, i.e., the $n+1$-th user has MLK offloading choices, in addition to not offloading. Our proposed exact algorithm traverses the search tree using our novel bounding technique, calculates the current total reward R_{tmp}, and updates the current best reward R_{best} if needed. This process follows a pre-order traversal. We use the following example to illustrate the traversal process.

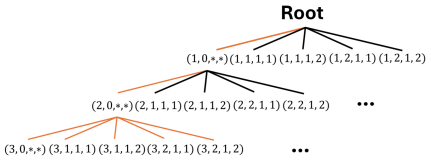


Fig. 1: Traversing process in the proposed algorithm

Example 1. *Let $N = 3$, $M = 2$, $K = 1$, and $L = 2$. As illustrated in Fig. 1, the traversing process starts from the root of the search tree and the node $(1, 0, *, *)$ using the proposed branch-cutting criteria. If we do not cut its branch, we will check its child $(2, 0, *, *)$. If we do not cut $(2, 0, *, *)$'s branch, we will check its children in the order of $(3, 0, *, *)$, $(3, 1, 1, 1)$, $(3, 1, 1, 2)$, $(3, 2, 1, 1)$, and $(3, 2, 1, 2)$. Then, the process backtracks to $(2, 0, *, *)$ and check its sibling $(2, 1, 1, 1)$. The traversal will repeat the above process until all the nodes are checked or cut out.*

The general idea of the proposed bounding techniques is that we cut *the branches not containing a better or feasible solution*. Specifically, we have the following branch-cutting criteria:

- *Infeasible*: If a node causes the server resource usage to exceed capacity or fails the communication time/SINR requirements, i.e, any of the constraints (6), (7), (10) and (11) is violated, its branch will be cut. This branch-cutting criteria directly follows Property 1, which indicates that this node's children are infeasible and do not need to be considered.
- *Inferior*: For any node associated with the n-th user, we define
$$C(n) = \sum_{n'=n}^{N} \sum_{m,k,l} a_{n',m,k,l} \times r_{n',k}, \tag{16}$$
as the maximum possible reward from the remaining unexplored users after the n-th user's parent. If $R_{\text{tmp}}+C(n)$ is less than the current best reward R_{best}, we will cut the branch of this node. This is because it is impossible to improve the R_{best} by exploring the children of this node.

Let $X = (n, m, k, l)$ be an arbitrary node in the search tree. We define $CH_i(X)$ as one of its children, where i is its index. We summarize the traversing process in Algorithm 1. Our exact algorithm calls $\texttt{Preorder}(root, A_{null}, 0)$ where $A_{null} = \{a_{n,m,k,l} = 0 | \forall u_n \in \mathcal{U}, s_m \in \mathcal{S}, k, l\}$ is the offloading strategy in which none of the tasks is offloaded.

Besides the proposed branch-cutting criteria, we implement the following optimizations to reduce the search space further:

Algorithm 1: $\texttt{Preorder}(X, \mathcal{A}_{\text{best}}^{\text{before}}, R_{\text{best}}^{\text{before}})$

Input: X: node to check; $\mathcal{A}_{\text{best}}^{\text{before}}$ and $R_{\text{best}}^{\text{before}}$: the best solution and its corresponding object function value before checking node X
Output: $\mathcal{A}_{\text{best}}^{\text{after}}$ and $R_{\text{best}}^{\text{after}}$: the best solution and its corresponding object function value after checking node X

if X is *infeasible* or *inferior* **then**
  return $(\mathcal{A}_{\text{best}}^{\text{before}}, R_{\text{best}}^{\text{before}})$;
$\mathcal{A}_{\text{best}}^{\text{after}} \leftarrow \mathcal{A}_{\text{best}}^{\text{before}}$; $R_{\text{best}}^{\text{after}} \leftarrow R_{\text{best}}^{\text{before}}$;
Calculate current total reward R_{tmp} according to (5);
if $R_{\text{tmp}} > R_{\text{best}}^{\text{after}}$ **then**
  $R_{\text{best}}^{\text{after}} \leftarrow R_{\text{tmp}}$; Update $\mathcal{A}_{\text{best}}^{\text{after}}$ corresponding to X;
for $i = 1, 2, \ldots, MLK+1$ **do**
  $(\mathcal{A}_{\text{tmp}}, R_{\text{tmp}}) \leftarrow \texttt{Preorder}(CH_i(X), \mathcal{A}_{\text{best}}^{\text{after}}, R_{\text{best}}^{\text{after}})$;
  if $R_{\text{tmp}} > R_{\text{best}}^{\text{after}}$ **then**
    $R_{\text{best}}^{\text{after}} \leftarrow R_{\text{tmp}}$;
    Update $\mathcal{A}_{\text{best}}^{\text{after}}$ corresponding to $CH_i(X)$;
Output $(\mathcal{A}_{\text{best}}^{\text{after}}, R_{\text{best}}^{\text{after}})$

- We define the maximum distance for any user to transmit (without any interference) as
$$d_{\max} = \left(\frac{P_{\max}}{\nu\beta}\right)^{-\alpha}. \tag{17}$$
If the distance between the n-th user and the m-th server is larger than $d_{\max}$, the n-th user will not offload its task to the m-th server, as it is impossible for the user u_n to achieve an SINR higher than β when communicating with m.
- If a node (n, m, k, l) is infeasible due to violating SINR constraint (Constraint (11)), we do no subsequent calculations on its siblings $(n, m, k', l), \forall k' \neq k$. This is because it is impossible to improve SINR by changing the ML algorithm tiers.

5. A Heuristic Algorithm

Besides the exact algorithm mentioned above, we propose a meta-heuristic approach based on the simulated annealing (SA) method to tackle the JOA problem. SA is a rule-based heuristic search process used to improve the current solution to a new one, representing the better objective value for a given optimization problem [10], [5]. In particular, for the proposed JOA problem, the initial feasible task offloading strategy $\mathcal{A}_{\text{old}}$ is randomly selected. After the perturbation process, which generates a new trial point $\mathcal{A}_{\text{new}}$ by making a small perturbation from the current solution $\mathcal{A}_{\text{old}}$, the new strategy that results in improvements in solution quality, i.e., the solution that increases the value of the objective, is automatically accepted; the new strategy decreasing the objective is also accepted with a certain probability $exp(-\frac{|R_{\text{new}}-R_{\text{old}}|}{T})$. Introducing this stochastic criterion allows the worse solution to be accepted to avoid being trapped in a local maximum. Nevertheless, each iteration must reduce this probability to avoid accepting an inferior solution.

Let $R(\mathcal{A}) = \sum_{n,m,k,l} a_{n,m,k,l} \times r_{n,k}$. The heuristic SA algorithm with LP feasibility-based criterion is summarized in

Algorithm 2: $\texttt{SA}(\mathcal{A}_{\text{old}}, T, \mu)$

Input: $\mathcal{A}_{\text{old}}$: initial offloading policy; T: temperature; μ: coefficient for reducing accepting probability
Output: $\mathcal{A}_{\text{best}}$ and $\mathcal{P}_{\text{best}}$: best-known offloading and power allocation plan; R_{best}: earned reward value

1 Set initial temperature T to a positive number.
2 Set $\mathcal{A}_{\text{old}}$ to an initial feasible solution and compute $R_{\text{old}} \leftarrow R(\mathcal{A}_{\text{old}})$.
3 Set $\mathcal{A}_{\text{best}} \leftarrow \mathcal{A}_{\text{old}}$ and $R_{\text{best}} \leftarrow R_{\text{old}}$.
4 **for** $i = 1, ..., \texttt{number\_of\_iterations}$ **do**
5 $\mathcal{A}_{\text{new}} \leftarrow \texttt{Perturbation}(\mathcal{A}_{\text{old}}, H, M, K, L)$;
6 **if** $\mathcal{A}_{\text{new}}$ satisfies Constraints (6), (7), (8) and
7 problem (13) has a feasible solution $\mathcal{P}_{\text{new}}$ **then**
8 $R_{\text{new}} \leftarrow R(\mathcal{A}_{\text{new}})$;
9 Generate a random number $c \in (0, 1)$;
10 **if** $R_{\text{new}} > R_{\text{old}}$ or $c < exp(-\frac{|R_{\text{new}} - R_{\text{old}}|}{T})$ **then**
11 $\mathcal{A}_{\text{old}} \leftarrow \mathcal{A}_{\text{new}}$;
12 $\mathcal{P}_{\text{old}} \leftarrow \mathcal{P}_{\text{new}}$;
13 $R_{\text{old}} \leftarrow R_{\text{new}}$;
14 **if** $R_{\text{new}} > R_{\text{best}}$ **then**
15 $\mathcal{A}_{\text{best}} \leftarrow \mathcal{A}_{\text{new}}$; $\mathcal{P}_{\text{best}} \leftarrow \mathcal{P}_{\text{new}}$;
16 $R_{\text{best}} \leftarrow R_{\text{new}}$;
17 $T \leftarrow \mu T$;
18 **Output** $\mathcal{A}_{\text{best}}$, $\mathcal{P}_{\text{best}}$, and R_{best}

Algorithm 3: $\texttt{Perturbation}(\mathcal{A}, H, M, K, L)$

Input: $\mathcal{A}$: current offloading strategy; H: constant; M: number of servers; K: number of sub-bands; L: number of ML algorithm tiers
Output: perturbed version of $\mathcal{A}$: $\mathcal{A}'$

1 Randomly select H users out of $\mathcal{U}$.
2 **for** $u_n \in \mathcal{U}_{\text{selected}}$ **do**
3 **if** $u_n \in \mathcal{U}_{\text{off}}$ **then**
4 $a_{n,m(n),k(n),l(n)} \leftarrow 0$;
5 **else**
6 $m(n) \leftarrow 0$; $k(n) \leftarrow 0$; $l(n) \leftarrow 0$;
7 **while** True **do**
8 $m'(n) \leftarrow \texttt{Random}(0, M)$;
9 $k'(n) \leftarrow \texttt{Random}(0, K)$;
10 $l'(n) \leftarrow \texttt{Random}(0, L)$;
11 **if** $m'(n) = 0$ **then**
12 $k'(n) \leftarrow 0$;
13 **if** $m'(n) \neq m(n)$ or $k'(n) \neq k(n)$ or $l'(n) \neq l(n)$ **then**
15 **break**
16 **if** $m'(n) \neq 0$ **then**
17 $a_{n,m'(n),k'(n),l'(n)} \leftarrow 1$;
18 **Output** $\mathcal{A}'$

Algorithm 2. Compared to the traditional SA, the proposed SA algorithm implements an extra accepting criterion of offloading strategy (lines 6-8). This accepting criterion is based on Property 2, in which we check the new candidate $\mathcal{A}_{\text{new}}$'s feasibility by solving the LP feasibility problem (13),

Due to the binary nature of the task offloading indicator, the perturbation process $\texttt{Perturbation}()$ needs to be appropriately designed in Algorithm 2. We summarize the perturbation process in Algorithm 3, where the function $\text{Random}(a, b)$ randomly returns an integer number from $[a, b]$. Specifically, given an offloading strategy $\mathcal{A}$, we randomly select H users out of the set $\mathcal{U}$. Let $\mathcal{U}_{\text{selected}}$ be the set of the selected H users. For each of these selected H users, we then change the server to which its task is being offloaded and the tile of its ML algorithm. This process yields a new perturbed version of the original offloading strategy. Note that the perturbed offloading strategy must not be the same as the original one.

6. Performance Evaluation

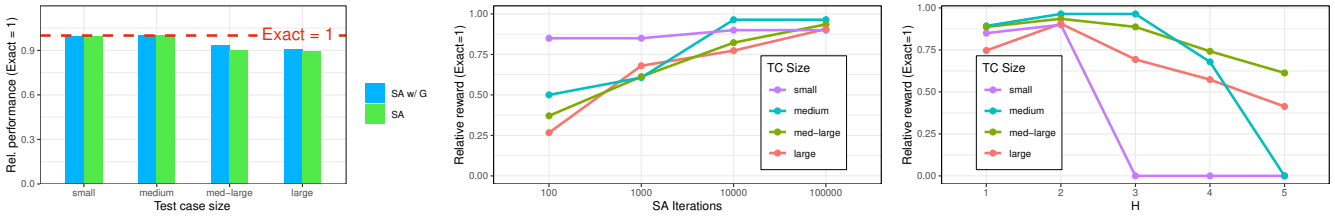
A. Evaluation Setup

We generate four types of test cases with different sizes: `small` with 5 users, 2 servers, 2 ML algorithms, and 2 sub-bands; `medium` with 10 users, 3 servers, 2 ML algorithms, and 2 sub-bands; `med-large` with 15 users, 5 servers, 2 ML algorithms, and 3 sub-bands; and `large` with 20 users, 5 servers, 2 ML algorithms, and 3 sub-bands. We set the users' maximum transmit power to $P_{\max} = 5$ W; the bandwidth of each channel is $B = 20$ MHz; the background noise is assumed to be $\nu = -65$ dBm; the path loss exponent is $\alpha = 3$; and signal quality threshold β is set to 15 dB for all cases. For SA, the initial temperature T is 1000. We normalize the total rewards in all the presented results. The experiment is conducted on a workstation running Ubuntu 22.04 with Intel i9-12900 CPU (24 cores @ 5.0 GHz) and 64 GB memory. All algorithms are implemented and executed in Python 3.11. We use Gurobi 11 to solve the LP feasibility problem (13).

We experiment with two kinds of initial policies for the simulated annealing: a "blank" policy where A_{old} is all 0's, and a greedy policy. The greedy policy is obtained as follows: we first sort the users' tasks by their rewards $r_{n,k}$, then attempt to offload tasks in the descending order of their reward values. For each task, the greedy strategy tries to assign it to a server as near as possible, where Constraints (6), (7), (10), and (11) are satisfied. If no such assignment is feasible, the task will not be offloaded. If a user already has a task with a higher reward offloaded, their remaining tasks (with lower rewards) are not considered. In the subsequent plots, the greedy initial solution is referred to as "`SA w/ G`". We choose iteration numbers that scale with the test case size for both SA initial starting points. Specifically, the iteration number is $100 \times MLK$. This number is chosen to balance between running time and performance.

B. Evaluation Results

In Figures 2a and 2b, the results are averaged over 30 test cases per test case size. Figure 2a reveals that the simulated annealing approach can be effective for different test case sizes, with its relative reward tapering off as the test cases increase. The greedy initial policy offers slightly better rewards than the blank policy. Figure 2b shows that our SA algorithm exhibits increased performance as the number of iterations increases. Smaller test cases plateau at small iteration numbers, while larger cases have increased gains (albeit slower) at high iteration numbers. For `small` to `med-large` cases, $10,000$ iterations yields at least 91% of the rewards the exact algorithm obtains. At $100,000$ iterations, the `large` case yields about



(a) Total rewards across test cases of different sizes ($H = 2$)

(b) Total rewards obtained by heuristic algorithm across different iterations ($H = 2$)

(c) Total rewards obtained by heuristic algorithm across different perturbation sizes ($100,000$ iterations)

Fig. 2: Performance evaluation

83% of the rewards the exact algorithm obtains. We note that larger cases have exponentially bigger search trees; thus, given a fixed number of iterations, the SA algorithm can only use a smaller portion of the search space on larger cases. As such, given a fixed H and iteration number, the effectiveness of SA decreases as the test case increases in size. Due to the randomness in the test case generation and SA execution, certain data points may appear to be outliers (e.g., `medium` and `med-large` at $10,000$ iterations), but overall this trend holds.

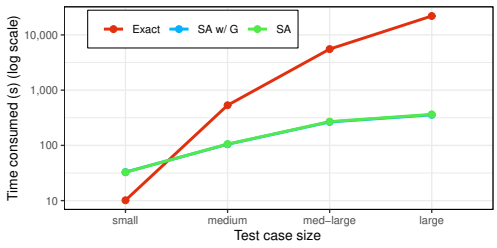


Fig. 3: Running time of proposed algorithms ($k = 2$)

Figure 2c presents the performance of the proposed heuristic algorithm across different perturbation sizes H using $100,000$ iterations. For our test cases, $H = 2$ or $H = 3$ yields the best results for all sizes. However, as H increases past 3, the reward drops sharply. This is due to the increasing number of users making changes simultaneously per iteration, making it more likely to result in an assignment that violates one or more constraints.

Size	`small`	`medium`	`med-large`	`large`
γ	3.63×10^{-2}	2.06×10^{-7}	4.63×10^{-16}	1.78×10^{-22}

TABLE I: γ in different test case sizes

In Fig. 3, the running time of both SA is approximately linear to the input size, while the exact algorithm exhibits exponential growth in time. To investigate the effectiveness of our bounding techniques for the Exact algorithm, we count the total number of nodes in the search tree visited by the algorithm and divide that by the number of nodes visited using an exhaustive search on the tree. This ratio, γ, is averaged over 30 cases for each test case size. As seen in Table I, γ decreases as the test case size increases. This is because when the test case gets bigger, it cuts off more nodes each time the algorithm bounds from a node. Despite the effectiveness of the bounding techniques, the Exact algorithm runs in a low-order exponential time overall due to the size explosion of the search tree.

7. Conclusion

In this work, we study the problem of jointly optimizing task offloading and resource allocation (JOA) in tactical edge networks, and formulate it as maximizing the total task completion rewards subject to various resource constraints. We develop two algorithms to address the JOA problem. The first is an exact algorithm employing a search tree, with branch-cutting criteria derived from the inherent properties of the JOA problem. The second is a meta-heuristic algorithm based on the simulated annealing approach. Through numerical evaluations, we demonstrate that the proposed exact algorithm is capable of obtaining the optimal task offloading strategy and resource allocation plan, while the heuristic algorithm efficiently delivers competitive performance.

References

[1] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

[2] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2014.

[3] Y. Chen, K. Zhao, B. Li, and M. Zhao, "Exploring the use of synthetic gradients for distributed deep learning across cloud and edge resources," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[4] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, 2000.

[5] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[6] S. Lin, Z. Zhou, Z. Zhang, X. Chen, and J. Zhang, *Edge intelligence in the making: Optimization, deep learning, and applications*. Springer, 2021.

[7] J. Perazzone, M. Dwyer, K. Chan, C. Anderson, and S. Brown, "Enabling machine learning on resource-constrained tactical networks," in *2022 IEEE Military Communications Conference (MILCOM)*. IEEE, 2022, pp. 932–937.

[8] J. Tang, G. Xue, and W. Zhang, "Interference-aware topology control and qos routing in multi-channel wireless mesh networks," in *Proceedings of the 6th ACM International Symposium on Mobile ad hoc networking and computing*, 2005, pp. 68–77.

[9] Y. Wan, X. Lin, A. Sabur, A. Chang, K. Xu, and G. Xue, "IoT system vulnerability analysis and network hardening with shortest attack trace in a weighted attack graph," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, 2023, pp. 315–326.

[10] G.-L. Xue, "Parallel two-level simulated annealing," in *Proceedings of the 7th international conference on Supercomputing*, 1993, pp. 357–366.

[11] Z. Zhang, S. Lin, M. Dedeoglu, K. Ding, and J. Zhang, "Data-driven distributionally robust optimization for edge intelligence," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2619–2628.

[12] Z. Zhang, S. Yue, and J. Zhang, "Towards resource-efficient edge AI: From federated learning to semi-supervised model personalization," *IEEE Transactions on Mobile Computing*, 2023.