algorithms

MDPI

Article

Hybrid Multiagent Collaboration for Time-Critical Tasks: A Mathematical Model and Heuristic Approach

Yifeng Zhou, Kai Di * and Haokun Xing

School of Computer Science and Engineering, Southeast University, Nanjing 211189, China; yfzhou@seu.edu.cn (Y.Z.); xinghaokun@qingting.fm (H.X.)
* Correspondence: dikai@seu.edu.cn

Abstract: Principal–assistant agent teams are often employed to solve tasks in multiagent collaboration systems. Assistant agents attached to the principal agents are more flexible for task execution and can assist them to complete tasks with complex constraints. However, how to employ principal–assistant agent teams to execute time-critical tasks considering the dependency between agents and the constraints among tasks is still a challenge so far. In this paper, we investigate the principal–assistant collaboration problem with deadlines, which is to allocate tasks to suitable principal–assistant teams and construct routes satisfying the temporal constraints. Two cases are considered in this paper, including single principal–assistant teams and multiple principal–assistant teams. The former is formally formulated in an arc-based integer linear programming model. We develop a hybrid combination algorithm for adapting larger scales, the idea of which is to find an optimal combination of partial routes generated by heuristic methods. The latter is defined in a path-based integer linear programming model, and a branch-and-price-based (BP-based) algorithm is proposed that introduces the number of assistant-accessible tasks surrounding a task to guide the route construction. Experimental results validate that the hybrid combination algorithm and the BP-based algorithm are superior to the benchmarks in terms of the number of served tasks and the running time.

Keywords: multi-agent collaboration; time-critical tasks; heuristic approach

check for updates

Citation: Zhou, Y.; Di, K.; Xing, H. Hybrid Multiagent Collaboration for Time-Critical Tasks: A Mathematical Model and Heuristic Approach. *Algorithms* **2021**, *14*, 327. https://doi.org/10.3390/a14110327

Academic Editor: Pieter Smet

Received: 15 October 2021
Accepted: 3 November 2021
Published: 5 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The hybrid principal–assistant collaboration architecture is often employed to improve the efficiency of task execution in a multiagent system [1,2], where the principal agents in a team are assisted by the assistant agents to complete the arrival tasks. Each assistant agent attaches to one principal agent, and is often more flexible for task execution (e.g., intelligent agents (robots) may be capable of executing tasks with special requirements that can hardly be achieved by humans). The introduction of assistant agents improves the flexibility of collaboration of agents in multiagent systems [1,3]. Actually, the principal–assistant collaboration problem can be considered as a new variety of the traveling salesman problem, concerning the cooperation of the principal agent routing and the assistant agent routing for tasks [4,5]. The principal–assistant systems are also employed in time-sensitive task scenarios [6], where the violation of temporal constraints will result in great losses in these missions.

Unfortunately, recent works [7–9] seldom address this issue. In this paper, we investigate the principal–assistant collaboration problem with deadlines, where each task is associated with an independent deadline and the principal–assistant teams need to find valid routes with temporal constraints. The problem is $\mathcal{NP}$-hard because each deadline-TSP can be reduced to an instance of the principal–assistant collaboration problem with deadlines, where the endurance of assistant agents is zero. If and only if the optimal solution of the deadline-TSP can be found in polynomial time, the optimal solution of the

principal–assistant collaboration problem with deadlines can be found in polynomial time. However, Farbstein and Levin [10] proved that the deadline-TSP is $\mathcal{NP}$-hard in the graph metrics or tree metrics, and thus the principal–assistant collaboration problem with deadlines is $\mathcal{NP}$-hard. The challenges of the principal–assistant collaboration problem with deadlines involve routing and cooperation [4,7]. First, the principal agent and the assistant agent need to cooperate to serve more vertices within the temporal constraints. Second, the combination of several principal–assistant routes is involved in complex allocations of tasks.

To investigate the problem thoroughly, we consider two scenarios involving different principal–assistant agent teams: the 1-principal-1-assistant scenario and the m-principal-u-assistant scenario. In the 1-principal-1-assistant scenario, the principal–assistant team is composed of a principal agent and an assistant agent. We pay attention to the inner coordination between the principal agent and the assistant agent. In the m-principal-u-assistant scenario, there are $m > 1$ principal–assistant teams, each of which is composed of one principal agent and $u > 1$ assistant agents. We focus on studying the source allocation and the cooperation among different team routes. Existing algorithms [4,7–9] cannot be directly employed in these scenarios because their objective is to serve all tasks and minimize the time cost of the entire route, and the arrival time cannot be restricted to each vertex in the routes, so that the arrival time to each vertex may violate certain deadlines. The unmodified application of those algorithms will result in delays, which are not allowed in real-time systems. For the 1-principal-1-assistant scenario, a hybrid combination algorithm is proposed, which finds the optimal combination of partial routes generated by three methods, including a heuristic subroutine, an iterated local search subroutine, and a simulated annealing subroutine. The partial routes prefer to insert the vertex with a smaller deadline and higher profit available per unit time cost in the subsets, which are divided by the time cost of each vertex to the initial vertex. For the m-principal-u-assistant scenario, a BP-based algorithm is proposed. The outline of this algorithm is to use the branch-and-price algorithm [11,12] to search for the optimal combination on the current principal–assistant route set and use the proposed density-based construction to enlarge the principal–assistant route set. The density-based construction evaluates the density of a vertex based on the number of residual assistant agent-accessible vertices surrounding it and the time cost between them and then inserts the vertex with the higher density and smaller deadline as the preference.

In the performance validation, the number of served vertices and the running time of algorithms are regarded as the evaluation criteria. In the 1-principal-1-assistant scenario, the hybrid combination algorithm is compared with an exact integer programming method, two adaptations based on the algorithms in [7,13], and some combinations of inner procedures. The hybrid combination algorithm proposed in this paper is only inferior to the exact method, and its running time is far less than that of the exact method when the graph size is less than 15. When the graph size ranges from 20 to 100, the hybrid combination algorithm serves the most vertices and costs a reasonable running time among the compared algorithms. In the m-principal-u-assistant scenario, the BP-based algorithm embedded with the density-based construction method is compared with two greedy algorithms that employ the line route construction method [14] or a local iterated search method [15] as the route construction and two BP-based algorithms embedded with the line route construction and local iterated search method. The BP-based algorithm embedded with the density-based construction method outperforms the other benchmark algorithms in terms of the number of served vertices and the running time.

The rest of this paper is organized as follows. In Section 2, the related work on the subject is reviewed. In Section 3, the 1-principal-1-assistant scenario is modeled and the hybrid combination algorithm is described. In Section 4, the m-principal-u-assistant scenario is modeled and the BP-based algorithm embedded with the density-based construction method is described. In Section 5, the experimental results of evaluating these algorithms are presented. In Section 6, the conclusions and future work are described.

2. Related Work

2.1. The Fundamental Principal–Assistant Systems

The fundamental hybrid principal–assistant systems proposed by Murray and Chu [7] introduce the Flying Sidekick TSP (FSTSP) model which is closely related to the model proposed in this paper, where the principal agent travels on the graph and repeatedly releases and retrieves the assistant agent. The basic idea of the algorithm in [7] for the FSTSP is to construct an initial principal agent path first by some classic TSP algorithms, and then select a vertex in the route and insert it into another place, or command the assistant agent to serve it so that the reduction on the total time cost is at the maximum. The algorithm then proposed by Agatz et al. [4] constructs a principal agent-only path by the TSP algorithm and then uses a dynamic programming algorithm to split the principal agent path into a principal agent route and an assistant agent route. Moreover, Ha et al. [8] propose a Greedy Randomized Adaptive Search Procedure (GRASP) for the principal–assistant to find a min-cost team route. *Overall, these studies ignore the crucial deadlines factor in the principal–assistant systems, and thus, the task executing time cannot be ensured.*

2.2. The Principal–Assistant Systems with Time Windows

There are only a small number of studies on hybrid principal–assistant systems with time windows. Sawadsitang et al. [9] improve the PDSTSP by introducing uncertainty regarding the takeoff and breakdown of assistant agents and time windows of tasks (GADOP). However, the time windows are coarse-grained, concretely. They provide a decomposition method to solve the problem. Differing from the GADOP problem, each task in this paper has an independent deadline. If the method for GADOP is applied directly, independent deadlines need to be clustered into a constant number of demands, which may give rise to a service delay duration execution, which is impermissible in our hard real-time system. *To sum up, the collaborative relationship between the principal agent and the assistant agent is not well characterized, causing inefficiencies in the system; in their scenarios, the assistant agent starts from the initial vertex and returns to it, so the principal agent cannot make use of the assistant agent to improve the performance of the task execution.*

2.3. The Deadline-TSPs

The deadline-TSP is a different kind of problem from the principal–assistant collaboration problem with deadlines. In deadline-TSPs, a single principal agent attempts to serve the vertices before their deadlines and returns to the initial vertex. Bansal et al. [13] propose an $\mathcal{O}(\log n)$-approximation algorithm for the deadline-TSP. The time complexity of this approximation algorithm is $\mathcal{O}(D_{max}^2 \beta n^{10})$, where D_{max} is the maximum deadline, β is the total price of the graph, and n is the number of vertices. Farbstein et al. [10] solve the deadline-TSP with contiguous deadlines. They propose an $\mathcal{O}\left(1+\epsilon, \frac{\alpha}{1+\epsilon}\right)$-approximation algorithm. The algorithm discretizes contiguous deadlines and uses an $\mathcal{O}(\alpha)$-approximation algorithm to solve the resulting KDTSP, which is a deadline-TSP with k deadlines. The algorithm ensures that the principal agent exceeds the deadlines by a factor of $1+\epsilon$ with $0 < \epsilon < 1$ and serves at least $\alpha/(1+\epsilon)$ of the number of tasks in the optimum. Its time complexity is $\mathcal{O}(n^{2k}\Delta^k)$, where Δ is the maximum distance in the metric space. Although these approximation algorithms run in polynomial time, they cost too much time in practice. *In summary, the existing algorithms cannot be applied in the principal–assistant collaboration problem with deadlines. In this paper, a hybrid combination algorithm and a BP-based algorithm are proposed to address two scenarios, including the 1-principal-1-assistant scenario and the m-principal-u-assistant scenario.*

3. 1-Principal-1-Assistant Scenario

In this section, the principal–assistant collaboration problem with deadlines, where one principal agent is equipped with one assistant agent, is defined formally, and a heuristic algorithm is proposed.

3.1. Problem Formalization

Let $V = \{v_i \mid 0 \leq i \leq n\}$ be the set of vertices, where v_0 is the location of the initial vertex, and the others denote the locations of tasks, each of which contains only one task. Each vertex can be visited several times, but the task at the vertex can be served at most once. The task at vertex $v_i \in V \backslash \{v_0\}$ has an independent deadline d_i, which means that the task at v_i needs to be served before d_i. Additionally, the initial vertex at v_0 is associated with a deadline d_0, which means that the principal–assistant team must return to the initial vertex before d_0. Let $D = \{d_i \mid v_i \in V, d_i \leq d_0\}$ be the set of deadlines.

In such a principal–assistant collaboration system, the principal agent equipped with an assistant agent starts from the initial vertex and returns to it before d_0, releasing and retrieving the assistant agent to serve other tasks repeatedly. The principal agent works in a connected undirected graph $G_1 = (V, M_1)$, where M_1 denotes the set of the edges between the vertices in V. Let $m_1(v_i, v_j)$ represent the minimum time cost between $v_i \in V$ and $v_j \in V$ in G_1. Without loss of generality, it is assumed that $d_i \geq m_1(v_0, v_i)$, and the assistant agent is often assumed to move freely from a vertex to another and spend less time than the principal agent [4]. Meanwhile, it works in a fully connected undirected graph $G_2 = (V, M_2)$, where M_2 denotes the set of edges between the vertices in V. Let $m_2(v_i, v_j) \in M_2$ denote the minimum time cost between $v_i \in V$ and $v_j \in V$ in graph G_2 and $m_2(v_i, v_j) \leq m_1(v_i, v_j)$. The other assumptions about the assistant agent are as follows: (1) the assistant agent has limited endurance η; (2) the assistant agent needs to be released and retrieved at a different vertex, except when the assistant agent is released and retrieved at the initial vertex v_0; and the assistant agent cannot reach other places except the location of the tasks being served; (3) the assistant agent needs to reach the vertex where it is retrieved before the principal agent arrives.

The principal–assistant route is denoted by a route $r = \{principal_r, assistant_r\}$, where $principal_r$ is the principal agent route and $assistant_r$ is the assistant agent route. Let $N(r)$ denote the number of served vertices in the route r. The principal agent route is denoted by $principal_r = \{p_i\}$, where the principal agent departs from $p_0 = v_0$, moves from p_i to p_{i+1}, and serves p_i. The number of vertices in $principal_r$ is denoted by $|principal_r| = g$, and the number of the served vertices in $principal_r$ is $g - 1$. The assistant agent route is denoted by $assistant_r = \{(p_i, p_k, p_j)\}$, where a tuple (p_i, p_j, p_k) indicates that the assistant agent is released at $p_i \in principal_r$ (called the *released vertex*) and retrieved at $p_j \in principal_r$ (called the *retrieved vertex*), serving $p_k \notin principal_r$.

The problem can be then modeled as follows, and Table 1 provides the summaries of the parameters and variable notations.

Table 1. Parameters and variables for the 1-principal-1-assistant scenario.

Notation	Description
$x_{i,t}$	Binary decision variable equals to one if the principal agent arrives v_i at time t; zero otherwise.
$y_{i,k,j}$	Binary decision variable equals to one if the assistant agent is released at v_i, retrieved at v_j, and serves v_k; zero otherwise.
$z_{i,j}$	Binary decision variable equals to one if the principal agent moves from v_i to v_j; zero otherwise.
$b_{i,t}$	Binary decision variable equals to one if t is equal or greater than the released time and less than the retrieved time in the sortie serving v_i; zero otherwise.
s_i	Integer intermediate variables representing the principal agent leaving time from v_i.
e_i	Integer intermediate variables representing the principal agent arrival time to v_i.
t	The time step.
M	A very large number.

$$\max \sum_{v_i \in V} \sum_{0 \le t \le d_0} x_{i,t} + \sum_{v_i \in V} \sum_{v_k \in V} \sum_{v_j \in V} y_{i,k,j} \tag{1}$$

$$\text{s.t.} \quad s_0 = 0 \tag{2}$$

$$s_i = \sum_{0 \le t \le d_0} x_{i,t} t, v_i \neq v_0 \tag{3}$$

$$e_i = \sum_{0 \le t \le d_0} x_{i,t} t, \forall v_i \in V \tag{4}$$

$$\sum_{1 \le t \le d_0} x_{0,t} = 1 \tag{5}$$

$$x_{0,0} = 1 \tag{6}$$

$$\sum_{0 \le t \le d_0} x_{0,t} t \ge \sum_{0 \le t \le d_0} x_{i,t} t, \quad v_i \neq v_0 \tag{7}$$

$$\sum_{0 \le t \le d_0} x_{i,t} \le 1, \quad v_i \neq v_0 \tag{8}$$

$$\sum_{v_i \in V} x_{i,t} \le 1, \quad 0 \le t \le d_0 \tag{9}$$

$$x_{j,t'} t' - x_{i,t} t \ge (x_{j,t'} + x_{i,t} - 1) m_1(v_i, v_j) + (x_{j,t'} + x_{i,t} - 2) M, \forall v_i, v_j \in V, 0 \le t < t' \le d_0 \tag{10}$$

$$z_{i,j} \le 1 + (d_j - \sum_{0 \le t \le d_0} x_{j,t} t)/M, \forall v_i, v_j \in V \tag{11}$$

$$\sum_{v_j \in V} z_{i,j} = \sum_{1 \le t \le d_0} x_{i,t}, \forall v_i \in V \tag{12}$$

$$\sum_{v_i \in V} z_{i,j} = \sum_{1 \le t \le d_0} x_{j,t}, \forall v_j \in V \tag{13}$$

$$e_j - s_i \ge m_1(v_i, v_j) + (z_{i,j} - 1) M, \forall v_i, v_j \in V \tag{14}$$

$$y_{i,k,j} \le 1 - \sum_{v_{i'} \in V} z_{i',k}, \forall v_i, v_k, v_j \in V \tag{15}$$

$$y_{i,k,j} \le z_{i,j}, \forall v_i, v_k, v_j \in V \tag{16}$$

$$y_{i,k,j}(m_2(v_i, v_k) + m_2(v_k, v_j)) \le \eta, \forall v_i, v_k, v_j \in V \tag{17}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} y_{i,k,j} \le 1, \forall v_k \in V \tag{18}$$

$$\sum_{v_k \in V} \sum_{v_j \in V} y_{i,k,j} \le 1, \forall v_i \in V \tag{19}$$

$$\sum_{v_i \in V} \sum_{v_k \in V} y_{i,k,j} \le 1, \forall v_j \in V \tag{20}$$

$$(1 - \sum_{v_{i'} \in V} z_{i',i}) M/3 + (1 - \sum_{v_{j'} \in V} z_{j',j}) M/3 + s_i + m_2(v_i, v_k) + m_2(v_k, v_j) \le e_j + (1 - y_{i,k,j}) M, \forall v_i, v_k, v_j \in V \tag{21}$$

$$(1 - \sum_{v_{i'} \in V} z_{i',i}) M/3 + s_i + m_2(v_i, v_k) \le d_k + (1 - y_{i,k,j}) M, \forall v_i, v_k, v_j \in V \tag{22}$$

$$\sum_{v_i \in V} b_{i,t} \le 1, 0 \le t \le d_0 \tag{23}$$

$$b_{k,t} \le \sum_{v_i \in V} \sum_{v_j \in V} y_{i,k,j}, \forall v_k \in V \tag{24}$$

$$b_{k,t} \le 1 + (t - s_i)/M + (1 - y_{i,k,j}) M,$$

$$\forall v_i, v_k, v_j \in V, 0 \le t \le d_0 \tag{25}$$

$$b_{k,t} \le 1 + (e_j - 1 - t)/M + (1 - y_{i,k,j})M, \forall v_i, v_k, v_j \in V, 0 \le t \le d_0 \tag{26}$$

$$\sum_{0 \le t \le d_0} b_{k,t} \ge e_j - s_i + (y_{i,k,j} - 1)M, \forall v_k \in V \tag{27}$$

$$\sum_{v_i \in V} \sum_{v_j \in V} y_{i,k,j} + \sum_{1 \le t \le d_0} x_{k,t} \le 1, \forall v_k \in V \tag{28}$$

The objective function (1) seeks to maximize the number of served tasks. Constraints (2)–(4) establish the relationship between s_i, e_i, and $x_{i,t}$. Explicitly, except initial vertex v_0, for other v_i, $s_i = e_i$.

Constraints on the principal agent routes. The principal agent travels in the graph, holding constraints (5)–(14). Constraints (5)–(7) ensure that the principal agent departs from the initial vertex v_0 at time 0 and finally returns to it before d_0. Constraint (8) indicates that the principal agent serves at most one vertex at any time, and constraint (9) indicates that the principal agent serves a task at a vertex except initial vertex v_0 at most one time. Constraint (10) states that the moving distance of the principal agent from v_i to v_j is not less than the minimum distance between them. Constraints (11)–(14) link $z_{i,j}$ and $x_{i,t}$, $x_{j,t}$. Constraint (11) ensures $z_{i,j}$ cannot be 1 if the principal agent visits v_j after its deadline. Constraints (12) and (13) ensure that the principal agent serves any tasks at most once. Constraint (14) ensures that if $z_{i,j} = 1$, the moving distance between any two vertices is not less than the minimum distance between them.

Constraints on the assistant agent routes. Constraints (15)–(27) are the constraints on the assistant agent routes. Constraint (15) ensures the assistant agent serves tasks that are not served by the principal agent. Constraint (16) guarantees that the assistant agent will be retrieved by the principal agent and Constraint (17) guarantees that the assistant agent moves in the range. Constraints (18)–(20) ensure the assistant agent serves any task at a vertex at most once. Constraints (21) and (22) ensure that the assistant agent serves the task vertex v_k before its deadline and arrives at retrieved vertex v_j before the principal agent arrives. Constraints (23)–(27) link $b_{k,t}$ and $y_{i,k,j}$. Constraint (23) requires the assistant agent executes at most one task at any time. Constraint (24) ensures $b_{k,t} = 0, 0 \le t \le d_0$ for unserved vertex v_k. Constraints (25)–(27) ensure that if the assistant agent serves v_k, departing from the principal agent at v_i and returning to the principal agent at v_j, $b_{k,t} = 1, s_i \le t < e_j$; otherwise, $b_{k,t} = 0$. Constraint (28) indicates that v_i can only be served by the principal agent and the assistant agent once.

3.2. Hybrid Combination Algorithm

The principal–assistant collaboration problem with deadlines in the 1-principal-1-assistant scenario is $\mathcal{NP}$-hard, and the proof is described as follows.

Theorem 1. *The principal–assistant collaboration problem with deadlines in the 1-principal-1-assistant scenario is $\mathcal{NP}$-hard.*

Proof of Theorem 1. Let *SP* denote the principal–assistant collaboration problem with deadlines in the 1-principal-1-assistant scenario. If a principal–assistant route is *r* for *SP*, the number of served vertices in route *r* is in polynomial time; hence, *SP* is clearly in $\mathcal{NP}$.

Let *CP* denote the deadline-TSP [10]. Then, a reduction from the *CP* to an instance of the *SP* is provided. Each *CP* can be reduced to a corresponding instance of the *SP* with $\eta = 0$. If and only if the optimum of the *CP* can be found in polynomial time can the optimum of the corresponding *SP* with $\eta = 0$ be found in polynomial time. However, the *CP* is proved to be $\mathcal{NP}$-hard [10]. Therefore, the principal–assistant collaboration problem with deadlines in the 1-principal-1-assistant scenario is $\mathcal{NP}$-hard. □

According to the computational complexity analyzed in Theorem 1, this is an $\mathcal{NP}$-hard problem, so we cannot obtain an optimal solution in polynomial time [16]. To obtain a suitable result in a reasonable time, we propose a heuristic algorithm. The basic idea of the algorithm is to divide the vertices into different subsets and find the optimal combination of the partial routes based on the subsets. The division will classify the vertices into different subsets based on their minimum time cost to the initial vertex v_0. The valid partial routes are constructed by the process CADROUTES, which includes three procedures, namely a heuristic method CONSTRUCT, an iterated local search ITERLOCALSERACH and a simulated annealing procedure SIMANNEAL.

The division and combination are described in Section 3.3 and the construction of the partial route is described in Section 3.4 and Section 3.5. The time complexity of the hybrid combination algorithm is analyzed in Section 3.6.

3.3. Division and Combination

The process employs a fact that, in a principal agent route $principal_r$, there exists at least one vertex called the turn vertex $p_{turn} = \arg\max_{p_i \in principal_r} m_1(p_i, v_0)$, and the principal agent route $principal_r$ can be split into two sequences at the turn vertex, including the $\{p_0, p_1, ..., p_{turn}\}$ and the $\{p_{turn}, ..., p_{g-2}, p_{g-1}\}$. In Figure 1, p_5 is the turn vertex and the principal agent route is split into $\{p_0, p_1, p_2, p_3, p_4, p_5\}$ and $\{p_5, p_6, p_7, p_8, p_9\}$. The partial routes contain monotonic parts and wavy parts [17]. The vertices of the monotonic parts are sorted in some order and the vertices of the wavy parts are disordered. Most of the vertices of the former are in monotonic parts, sorted in the non-decreasing order of the time cost to v_0, e.g., $\{p_2, p_3, p_4, p_5\}$. However, there exist some vertices in wavy parts, where a vertex may be closer to v_0 than its previous vertex, e.g., $\{p_1, p_2\}$. In general, the former sequence of the principal agent route $\{p_0, p_1, p_2, p_3, p_4, p_5\}$ is the process of leaving from the initial vertex. Similarly, the latter sequence of the principal agent route $\{p_5, p_6, p_7, p_8, p_9\}$ also has monotonic parts and wavy parts, but it is a return process, where the principal–assistant team returns to the initial vertex. The divisions and combinations are designed based on the feature, which is detailed in Algorithm 1.

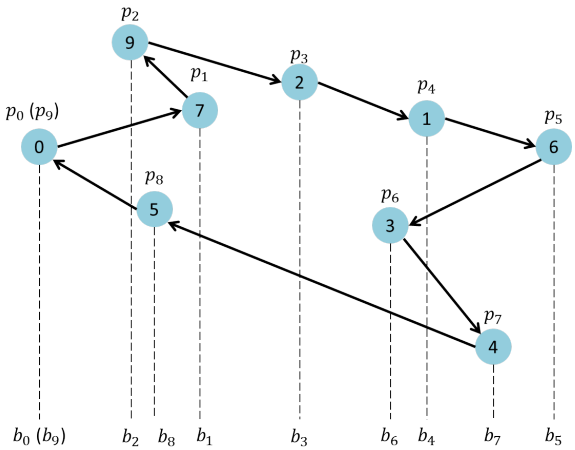

Figure 1. A demonstration of a principal agent route $principal_r = \{p_i \mid 0 \leq i \leq 9\}$. Let $b_i = m_1(p_0, p_i)$, p_i be sorted in the non-decreasing order based on b_i, $b_0(b_9) \leq b_2 \leq b_8 \leq b_1 \leq b_3 \leq b_6 \leq b_4 \leq b_7 \leq b_5$.

In the division step, the turn vertex and the arrival time to it are selected first. In line 2, each vertex except the initial vertex can be regarded as the turn vertex v_{turn}. $U_t = \{v_i \mid m_1(v_i, v_0) \leq m_1(v_{turn}, v_0), v_i \in U\}$ denotes the set of candidate vertices for the principal agent. In line 6, each time t between t_{min} and t_{max} will be regarded as the arrival time to v_{turn}, the time budget of the leave process is t and the time budget of the return process is $d_0 - t$. Given the different subsets and time budgets, the leave processes and return

processes are generated by CADROUTES. The LR and RR represent the set of candidate leave processes and the set of candidate return processes, respectively. In line 11, the optimal principal–assistant route is found. In line 13, the optimal combination of a leaving process from LR and a return process from RR is selected as the output.

Algorithm 1: Hybrid Combination Algorithm

Input: Graphs $G_1 = (V, M_1), G_2 = (V, M_2)$, deadline set D
Output: A principal–assistant route $r = \{principal_r, assistant_r\}$
1 $U \leftarrow V - \{v_0\}, r \leftarrow \{\}$;
2 **for** $v_{turn} \in U$ **do**
3 $\quad U_t = \{v_i \mid m_1(v_i, v_0) \leq m_1(v_{turn}, v_0), v_i \in U\}$;
4 $\quad t_{min} \leftarrow m_1(v_0, v_{turn})$;
5 $\quad t_{max} \leftarrow \min\{d_{turn}, d_0 - m_1(v_0, v_{turn})\}$;
6 $\quad$ **for** $t \in [t_{min}, t_{max}]$ **do**
7 $\quad\quad$ LR $\leftarrow$ CADROUTES$(U, U_t, v_0, v_{turn}, 0, t)$;
8 $\quad\quad U' \leftarrow \{v_i \mid v_i \in U, v_i \notin lr, lr \in \text{LR}\}$;
9 $\quad\quad U'_t \leftarrow \{v_i \mid v_i \in U_t, v_i \notin lr, lr \in \text{LR}\}$;
10 $\quad\quad$ RR $\leftarrow$ CADROUTES$(U', U'_t, v_{turn}, v_0, t, d_0 - t)$;
11 $\quad r_{max} \leftarrow \arg\max_{r' \in \text{LR} \cup \text{RR}} N(r')$;
12 $\quad$ **if** $N(r_{max}) > N(r)$ **then**
13 $\quad\quad r \leftarrow r_{max}$;
14 **return** r;
15 **Function** CADROUTES$(U, U_t, v_s, v_e, st, bdg)$
16 HP $\leftarrow$ CONSTRUCT$(U, U_t, v_s, v_e, st, bdg)$;
17 ILS $\leftarrow$ ITERLOCALSEARCH$(\text{HP}, U - \text{HP})$;
18 SA $\leftarrow$ SIMANNEAL$(U, U_t, v_s, v_e, st, bdg)$;
19 **return** HP, ILS, SA;
20 **End Function**

3.4. Heuristic Construction

The procedure CADROUTES is the core of the algorithm. It provides three partial routes, i.e., HP, ILS, and SA, which are generated by three methods, including the heuristic CONSTRUCT, an iterated local search ITERLOCALSERACH, and a simulated annealing procedure, respectively.

Heuristics are effective methods to address routing problems [18], so that the procedure CONSTRUCT is designed to find a valid solution. Before describing the method, the features of the routes for the principal–assistant team are provided first. The route $r = \{principal_r, assistant_r\}$ can be merged into one structure, as in Figure 2, and the route can be rewritten as $r = \{r_i\}$, where r_i is a partial route. The partial routes can be classified into three types of components, including short line segments, simple triangles, and complex triangles. As Figure 2 shows, in a short line segment, the principal agent moves from one vertex to the other directly, serving one vertex, such as the principal agent route $\{v_0, v_6\}$ and $\{v_3, v_9\}$ without the assistant agent route. In a simple triangle such as the principal agent route $\{v_6, v_3\}$ with the assistant agent route $\{v_6, v_5, v_3\}$, the principal agent departs from the released vertex, releasing the assistant agent, and moves to retrieved vertex directly, retrieving the assistant agent. The principal–assistant serves two vertices. In a complex triangle, the principal agent moves from the released vertex to the retrieved vertex through other vertices, such as the principal agent route $\{v_8, v_1, v_2, v_4\}$ with the assistant agent route $\{v_8, v_7, v_4\}$. It is intuitive to select the partial route that serves the most tasks locally, but a smaller time cost may improve the opportunity to serve more tasks in the future. To balance these two factors, the profit per unit time cost $\phi(r)$ is defined in Definition 1 to help select the partial routes. At every iteration, the heuristic selects a valid partial route with the highest profit available per unit cost.

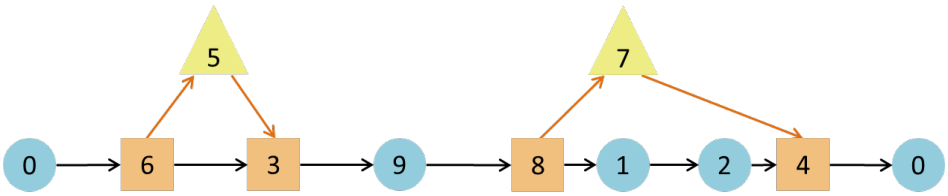

Figure 2. The structure of a principal–assistant route r: The principal agent serves circle and square vertices. The assistant agent serves triangle vertices. The principal agent releases or retrieves the assistant agent at square vertices.

Definition 1. *The profit available per unit cost can be then defined as follows:*

$$\phi(r) = \frac{N(r)}{cost(principal_r)} \tag{29}$$

where $cost(principal_r)$ *is the traveling time of the principal agent in* $principal_r$ *, and* $N(r)$ *is the number of tasks served by route* r*.*

It can be derived that at each iteration, there are $\mathcal{O}(n)$ short line segments, $\mathcal{O}(n^2)$ simple triangles, and $\mathcal{O}(n^n)$ complex triangles to be selected. The solution space of complex triangles is too large to search quickly. However, a complex triangle $\{principal_r, assistant_r\}$ can be approximated by a corresponding long line segment $\{principal_r, \{\}\}$. It can be derived that

$$\begin{aligned} \phi(\{principal_r, \{\}\}) + \epsilon &= \phi(\{principal_r, assistant_r\}) \\ \frac{g-1}{cost(principal_r)} + \epsilon &= \frac{g}{cost(principal_r)} \end{aligned} \tag{30}$$

where $\epsilon > 0$ is a real number. A larger $cost(principal_r)$ indicates a smaller gap between $\{principal_r, assistant_r\}$ and $\{principal_r, \{\}\}$. Because a long line segment can be split into several short line segments, to accelerate the search process, the heuristic only selects partial routes from short line segments and simple triangles.

The procedure CONSTRUCT is outlined in Algorithm 2. Based on the above observation, it searches for the partial routes r' with the highest $\phi(r')$ from the short lines and simple triangles and inserts them into the original routes iteratively, until there are no valid partial routes. In line 6, all the short line segments and simple triangles are found and inserted into the set R. From line 8 to line 11, the algorithm selects the partial route r' with the maximum $\phi(r')$. If more than one partial route has the same $\phi(\cdot)$, the partial route with the minimum d_i, which is the deadline of the end vertex of the partial route, is selected.

Theorem 2. *The Construct Algorithm proposed in Algorithm 2 is an* $\mathcal{O}(\frac{l_{min}}{2 \cdot l_{max}})$*-approximation algorithm, where* l_{min} *is the minimum distance between any two vertices in* G_1 *and* l_{max} *is the maximum distance between any two vertices in* G_1*.*

Proof of Theorem 2. To prove the approximation ratio, the upper bound of the optimum and the lower bound of the Construct Algorithm need to be provided firstly. Assume that the optimum $r^\star$ serves $q^\star$ tasks and the route r' pair created by Construct Algorithm serves q' tasks. In the optimal case, $r^\star$ is totally constituted by simple triangles, each of which serves two tasks and costs l_{min}. Hence, for the optimum,

$$\frac{q^\star}{cost(q^\star)} \leq \frac{2}{l_{min}}. \tag{31}$$

On the other hand, there exists a lower bound of its profit per unit cost. Assume $r' = (r_0, r_1, \cdots)$ and each component r_i serves q_i vertices and costs $cost(r_i)$ time steps. It can be derived that

$$\frac{q'}{cost(r')} = \frac{\sum_{r_i \in r'} q_i}{\sum_{r_i \in r'} cost(r_i)} \geq \min_{r_i \in r'} \frac{q_i}{cost(r_i)} \tag{32}$$

$\frac{q'}{cost(r')}$ is greater than or equal to the minimum local profit per unit cost. In the worst case, the component departs from the start vertex and directly moves to serve the farthest vertex and returns, costing l_{max}. Thus,

$$\frac{q'}{cost(r')} \geq \min_{r_i \in r'} \frac{q_i}{cost(r_i)} = \frac{1}{l_{max}}. \tag{33}$$

The approximate ratio can be calculated by

$$\frac{q'/cost(r')}{q^\star/cost(r^\star)} \geq \frac{1/l_{max}}{2/(l_{min})} = \frac{l_{min}}{2 \cdot l_{max}}. \tag{34}$$

Thus, the Construct Algorithm proposed in Algorithm 2 is an $\mathcal{O}(\frac{l_{min}}{2 \cdot l_{max}})$-approximation algorithm. □

Algorithm 2: Construct Algorithm

Function CONSTRUCT($U, U_t, v_s, v_e, st, bdg$)
$principal_r \leftarrow \{v_s, v_e\}$;
$assistant_r \leftarrow \{\}$;
$p_{cur} \leftarrow v_s$;
while $cost(principal_r) \leq budget$ **do**
  $R \leftarrow \{r' \mid principal_{r'} \leftarrow \{p_{cur}, v_i\}, assistantagent_{r'} \leftarrow \{(p_{cur}, v_j, v_i)\}, v_j \in U_t, v_i \in U, v_i \notin v_j\}$;
  $maxval \leftarrow 0, d \leftarrow d_0, r_{max} \leftarrow \emptyset$;
  for $r' \in R$ **do**
    if r' *is valid* **then**
      if $\phi(r') > maxval$ **then**
        $maxval, \phi(r'), d \leftarrow d_i, r_{max} \leftarrow r'$;
  if $r_{max} \neq \emptyset$ **then**
    $principal_r \leftarrow \{v_s, ..., p_{cur}, v_i, v_e\}$;
    $assistant_r \leftarrow assistant_r \cup assistantagent_{r_{max}}$;
    $U \leftarrow U - \{v_i, v_j\}, U_t \leftarrow U_t - \{v_i\}$;
    $p_{cur} \leftarrow v_i$;
  else
    break;
return $r = \{principal_r, assistant_r\}$;
End Function

3.5. Iterated Local Search and Simulated Annealing

It is possible for the procedure CONSTRUCT to be trapped in local optimum; thus, the iterated local search [15] and the simulated annealing [19,20] are incorporated to improve the opportunity of escaping from local optimum. The two methods contain some basic operations on the principal–assistant routes. There are six basic operations that refer to the operations mentioned by Guanwan et al. [21], as shown in Table 2. The operation Swap and Replace ensures that the new principal–assistant route serves at least as many tasks as the old one does. The operation Insert inserts an unserved task into the principal agent route and deletes the resulting infeasible partial routes of the assistant agent. The operation

Subjoin inserts the partial assistant agent route with the minimum time cost. If more than one assistant agent route has the same time cost, it selects the assistant agent route with the minimum d_i, which is the deadline of the served vertex v_i. The operation Removet deletes one vertex in the principal agent route and the resulting invalid assistant agent routes. The operation Removeu is a safe operation and decreases the quality of the current solution but may lead to a better solution in the following process.

Table 2. Basic operations.

Operations	Description
Swap	Exchange two vertices in the principal agent route.
Replace	Replace a served vertex in the principal agent route with an unserved one.
Insert	Insert one unserved vertex into the principal agent route.
Subjoin	Plan partial assistant agent routes on the current principal agent route, and adopt the partial routes with the shortest time.
Removet	Remove a vertex from the principal agent route.
Removeu	Remove a triad from the assistant agent route.

The procedure ITERLOCALSERACH proposed in Algorithm 3 is a metaheuristic algorithm, which starts from the initial solution HP created by the CONSTRUCT and adopts four operations, i.e., Swap, Replace, Insert and Subjoin, to orderly search the neighbor solutions. The operations will be used iteratively. At Step 1, Swap is used to operate two vertices in the principal agent route, except the start vertex and end vertex. If the Swap operation reduces the time cost, it is adopted to update the route. At Step 2, Replace is used to operate any vertex except the start vertex and end vertex. If it is valid, the operation will be accepted, and the replaced position will not be operated again. Replace is similar to Swap, as it also updates the route and terminates when there is no valid operation. At Step 3, Insert is used to fully utilize the budget. One valid vertex is inserted into the first available position, and the principal agent route is updated. Then, Insert continues to be used in the new route. At Step 4, Subjoin is used to plan new partial routes for the assistant agent. The valid simple or complex triangles that serve other tasks are sorted in the increasing order of their time cost. The triangle with the smaller time cost is subjoined preferentially. The ITERLOCALSERACH will execute until the principal–assistant route cannot be improved or the number of iterations reaches the upper limit cntlimit.

Algorithm 3: ITERLOCALSEARCH

```
1 Function ITERLOCALSEARCH(route, U, U_t, budget)
2 Swap;
3 Replace;
4 Insert;
5 Subjoin;
6 return ILS;
7 End Function
```

The route ILS generated by the ITERLOCALSERACH is not necessarily better than the HP but provides another possible partial principal–assistant route that can be combined with others to generate a better entire partial route.

Simulated annealing can converge to the global optimum in some probabilistic sense [19] and has been applied in many optimization problems [18]. Simulated annealing is different from the iterated local search. It has an environment temperature which will decrease as the procedure runs; when the temperature is smaller than a minimum temperature, the procedure terminates. At the temperature decreasing moment, it generates a new partial route; the new one will be accepted and will replace the current principal–assistant route if the new one is better; otherwise, the new one will be accepted as the current one

with a certain probability, which is related to the current temperature. Simulated annealing includes the following three components: neighbor generation, acceptance probability, and cooling schedule. The neighbor generation is used to generate a neighbor principal–assistant route based on the current partial route. In every generation, a valid operation is selected randomly from all six basic operations, and a new route is created by applying the operation on the current solution. After the generation of the new route, the method needs to decide to accept one route between the original one and the new one. Assume that the gap λ is the number tasks served by the new route minus the number of tasks served by the original route. If λ is greater than 0, the new route will be accepted and is regarded as the current solution. Otherwise, the new route will be accepted with an acceptance probability. The acceptance probability adopts the Metropolis principle [22] and is set as follows:

$$AP = \exp(-\lambda / Temp) \tag{35}$$

where $Temp$ denotes the current temperature. The procedure will terminate once the temperature is less than the minimum temperature $Temp_{min}$. The $Temp$ is decreased by multiplying a positive factor $\alpha < 1$ at each iteration. In some previous papers, α retains a certain value for different instances, but it is not suitable in its own scenario. For example, to create a route r_1 with a larger budget and another route r_2 with a smaller budget, if simulated annealing uses the same α, they will have similar time costs. However, a smaller budget indicates less valid combinations of routes, and many iterations in the construction of r_2 are in vain. To accelerate the route construction with different budgets, let

$$\alpha = const \cdot (bdg / d_0) \tag{36}$$

where $0 < const < 1$ is a constant real value, and bdg is the rest of the budget. $Temp$ decreases faster if the budget is small.

3.6. Analysis of Time Complexity

In the hybrid combination algorithm, the division process selects the turn vertices and their arrival time; the combination assembles a constant number of partial routes. The CADROUTES subroutine generates new partial principal–assistant routes.

Theorem 3. *The time complexity of the hybrid combination algorithm proposed here is* $\mathcal{O}(Kd_0n^4)$, *where* $K = \max\left\{cntlimit, \log_\alpha\left(\frac{Temp}{Temp_{min}}\right)\right\}$.

Proof of Theorem 3. According to Algorithm 1, the number of combinations of the turn vertices and their arrival time is $\mathcal{O}(d_0 n)$. The CADROUTES executes three procedures in order. The procedure CONSTRUCT scans the $\mathcal{O}(n^2)$ components every iteration and it at most executes d_0 iterations. The complexity of the CONSTRUCT is $\mathcal{O}(d_0 n^2)$. These operations at most run in $\mathcal{O}(n^3)$. The time complexity of the procedure ITERLOCALSERACH is $\mathcal{O}(cntlimit \cdot n^3)$. The number of simulated annealing iterations is no more than $k = \log_\alpha(\frac{Temp}{Temp_{min}})$. The time complexity of the SIMANNEAL is $\mathcal{O}(kn^3)$. The total time complexity is denoted by $\mathcal{O}(Kd_0n^4)$, where $K = \max\left\{cntlimit, \log_\alpha\left(\frac{Temp}{Temp_{min}}\right)\right\}$. □

4. *m*-Principal-*u*-Assistant Scenario

4.1. Assumption and Model

The m-principal-u-assistant scenario, where each of m principal agents is equipped with u assistant agents, is defined in this section. Most of the assumptions held in this problem are the same as the descriptions in Section 3, except that there are m principal–assistant teams to complete the tasks. Each principal–assistant team has one principal agent and u assistant agents.

Due to the introduction of multiple assistant agents, the arc-based model becomes overly complicated. For conciseness, the problem is modeled as a path-based integer linear

programming based on [14]. The notations of the parameters and variables are displayed in Table 3. The problem can be modeled as follows.

Table 3. Parameters and variables for the m-principal-u-assistant scenario.

Notation	Description
r	A principal–assistant route.
Ω	The set of all available routes.
c_r	Parameter that indicates the number of tasks served in the route r.
$a_{i,r}$	Parameter that equals one when one task v_i is served in route r, and zero otherwise.
w_r	Binary decision variable equal to one if route r is used, and zero otherwise.

$$\max \sum_{r\in\Omega} c_r \cdot w_r \tag{37}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} a_{i,r} \cdot w_r \leq 1 \quad \forall v_i \in V\backslash\{v_0\} \tag{38}$$

$$\sum_{r\in\Omega} w_r \leq m \tag{39}$$

$$w_r \in \{0,1\}, \forall r \in \Omega \tag{40}$$

The objective (37) seeks an optimal combination of routes that serves the most tasks. Constraint (38) requires that each task is served at most once. Constraint (39) requires the number of principal–assistant teams to be less than the maximum capacity m. Constraint (40) is the definition of the decision variables w_r.

4.2. Branch-and-Price Algorithm

The master problem (37) is hard to solve, because the set Ω is an exceptionally large set. The branch-and-price algorithm [11,12] is a practical method to decompose such a problem, which embeds column generation into the branch-and-bound algorithm.

4.2.1. Linear Relaxation and Decomposition

In the branch-and-price algorithm, the master problem is converted from a 0–1 integer linear problem into a general linear problem, as follows:

$$\max \sum_{r\in\Omega} c_r \cdot w_r \tag{41}$$

$$\text{s.t.} \quad (38) - (39)$$

$$0 \leq w_r \leq 1, \forall r \in \Omega \tag{42}$$

After the linear relaxation, the difficulty of solving the objective (37) has been decreased slightly, but the set Ω is still too large to enumerate completely. However, a fact should be exploited, i.e., that the ratio of the number of actually used routes to the number of all valid routes is exceedingly small, so the final solution can be constructed from a smaller route set $\Omega' \subseteq \Omega$. The column generation formulates a restricted master problem by replacing the set Ω with the set Ω' as follows:

$$\max \sum_{r\in\Omega} c_r \cdot w_r \tag{43}$$

$$\text{s.t.} \quad \sum_{r\in\Omega} a_{i,r} \cdot w_r \leq 1 \quad \forall v_i \in V\backslash\{v_0\} \tag{44}$$

$$\sum_{r\in\Omega} w_r \leq m \tag{45}$$

$$0 \leq w_r \leq 1, \forall r \in \Omega \tag{46}$$

The initial set Ω' can be enlarged by adding new routes. According to the duality theory, a better route r satisfies the following condition:

$$\widehat{c_r} = c_r - \sum_{v_i \in V \setminus \{v_0\}} a_{i,r} \cdot \delta_i - \gamma > 0 \tag{47}$$

where δ_i is the dual variable of the current optimal solution for each constraint in Constraint (44), and γ is the dual variable for Constraint (45). Furthermore, if no route with $\widehat{c_r} > 0$ exists, the global optimum is found. The following process is to find and price an undiscovered route $r^\star$ with $\widehat{c_{r^\star}} = \max_{r \in \Omega - \Omega'} \widehat{c_r}$, which is called the pricing subproblem and is detailed in the next section.

4.2.2. Pricing Subproblem

The pricing subproblem seeks better valid routes. The previous studies proposed many exact or heuristic algorithms to solve the routing problems [12,14,23].

Based on the definition of c_r, it can be rewritten as follows:

$$c_r = \sum_{v_i \in V \setminus \{v_0\}} a_{i,r} \tag{48}$$

Associated with Equation (47), the following can be deduced:

$$\widehat{c_r} = \sum_{v_i \in V \setminus \{v_0\}} a_{i,r} \cdot (1 - \delta_i) - \gamma > 0 \tag{49}$$

Assume that V_c denotes the set of vertices involved in Ω' and $V_u = V - V_c$. From the analysis of the restricted master problem, it is deduced that $\delta_i = 0$, $\forall v_i \in V_u$ and $\delta_i > 0$,$\forall v_i \in V_c$. The route that serves the most $v_i \in V_u$ vertices is the optimal choice in terms of the mathematical formulation. However, temporal constraints restrict the number of vertices in V_u. Additionally, the original allocation of the vertices may not be optimal; the new routes can include some involved vertices so that to replace the old one creates a better entire result. The routing process concerns vertices in the set V_c and V_u.

4.3. Principal–Assistant Route Construction

The outline of the routing algorithm is selecting different subsets of the vertices and constructing partial routes on them, which includes the randomized selection procedure described in Section 4.3.1, the principal agent route construction described in Section 4.3.2, and the assistant agent route construction described in Section 4.3.3. The entire algorithm is described in Section 4.3.4.

4.3.1. Randomized Selection

The randomized selection of vertices is described in Algorithm 4. The vertex v_i is selected with a probability $prob_i$, assuming that $\Omega_{v_i} \in \Omega'$ denotes the set of routes that contain $v_i \in V$. For vertex $v_i \in V_c$, a higher ratio $|\Omega_{v_i}|/|\Omega'|$ indicates that we know more about vertex v_i than the other vertices in V_c. To obtain more information about the whole problem, $v_i \in V_c$ with a lower $|\Omega_{v_i}|/|\Omega'|$ should be selected with a higher probability. The vertex $v_i \in V_c$ is selected with the following probability:

$$prob_i = 1 - \frac{|\Omega_{v_i}|}{|\Omega'|}, \quad v_i \in V_c \tag{50}$$

For the vertices in V_u, each of them is selected with a probability as follows:

$$prob_i = \frac{1}{|V_u|}, \quad v_i \in V_u \tag{51}$$

which indicates that before being explored, they all have equal values. Additionally, the expectation of the number of vertices to be selected is equal to 1, which ensures that all the uninvolved vertices will be included after several selections.

Algorithm 4: Random Selection Algorithm

Input: V_c, V_u
Output: V_t
$V_t \leftarrow \emptyset$;
for $v_i \in V_c$ **do**
  Execute $V_t \leftarrow V_t \cup \{v_i\}$ with probability $prob_i$;
for $v_i \in V_u$ **do**
  Execute $V_t \leftarrow V_t \cup \{v_i\}$ with probability $prob_i$;
return V_t;

4.3.2. Principal Agent Route Construction

The construction of the principal agent routes with temporal constraints is a classic problem, called the orienteering problem with time windows (OPTW) [21,24], where a traveler needs to serve vertices during their time windows and then return to the initial vertex. Two efficient algorithms can be used in the principal agent route construction, including the line construction [21] and an iterated local search algorithm [15].

The outline of the line construction is to insert an unserved vertex between two vertices in a valid route iteratively and ensure that the resulting route is also valid until no vertex can be inserted. If there is more than one option, the operation adding the minimum time cost will be selected. The iterated local search algorithm (ILS) is a two-step process. First, it creates a valid route by line construction, and then the perturbation and local search techniques are used to improve the route within a certain iteration. Two algorithms will be embedded into our integrated algorithm individually, and their performances will be evaluated.

The above two methods outperform the other routing algorithms [23] in the principal agent route construction, but in the construction of the principal–assistant routes, they ignore the influence of the assistant agent routes and result in poor performance in some situations. An example is displayed in Figure 3. In Figure 3, the line construction creates $principal_r = \{v_0, v_3, v_0\}$, and the assistant agent routes $assistant_r^1$, $assistant_r^2$ are empty. However, there is an obvious better principal–assistant route, $principal_r = \{v_0, v_2, v_0\}$, $assistant_r^1 = \{(v_2, v_1, v_0)\}$, and $assistant_r^2 = \emptyset$. It demonstrates a phenomenon that if a principal agent travels to a vertex surrounded by some assistant agent-accessible tasks, it is possible to serve tasks then directly move to a task costing less time. Inspired by the influence of the assistant agents, the density of the vertex is defined in Definition 2. The $density_i$ of v_i is influenced by the number of assistant agent-accessible tasks and their distances to v_i in graph G_2. The distances are squared to improve their weight; the vertex with the most assistant agent-accessible tasks located in an adjacent area is preferred.

Definition 2. *The density of the task v_i is as follows:*

$$density_i^S = \frac{1}{u} \cdot \sum_{dur=1}^{\eta} \frac{\left|V_{dur}^i\right|}{(0.5 \cdot dur)^2} \tag{52}$$

where $V_{dur}^i = \left\{v_j \in S \mid 2 * m_2(v_i, v_j) = dur\right\}$.

The density-based construction algorithm is designed based on the line construction and the concept of the density of a task, which is shown in Algorithm 5. Before using line construction, all the provided tasks are sorted in ascending order based on the density of each task, and a factor $0.0 < \zeta < 1.0$ decides how many tasks have high priority to be inserted in the principal agent route. Assume that V' is the sorted set of tasks; first, $\lceil |V'| \cdot \zeta \rceil$ tasks in V' have high priority. The tasks with high priority will be stored in order in list H, and the rest of the tasks with low priority will be stored in order in L. The

tasks in H will be inserted into principal agent route $principal_r$ by the line construction $lineInsert(principal, S)$, until there is no vertex that can be inserted.

Algorithm 5: Density-Based Algorithm

Input: The set of vertices $V' \subseteq V$, the set of deadlines D, a parameter $0.0 < \zeta < 1.0$, the start vertex v_s, the end vertex v_e
Output: A principal agent route $principal_r$

1 $principal_r \leftarrow \{v_s, v_e\}$;
2 Sort vertices in V' in the descending order based on the density$_i^{V'}$, vertices with the same density are sorted in the ascending order based on the deadline d_i, obtain $V' = \{p_i\}$;
3 $H \leftarrow \{p_i \mid p_i \in V', i \leq \lceil |V'| \cdot \zeta \rceil\}$;
4 $L \leftarrow V' - H$;
5 $lineInsert(principal_r, H)$;
6 $lineInsert(principal_r, L)$;
7 **return** $principal_r$;

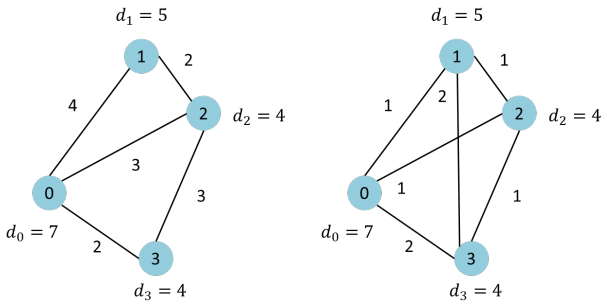

Figure 3. The influence of the density: Subfigure (**a**) shows the tasks V in graph G_1 and Subfigure (**b**) shows the tasks V in graph G_2. The principal–assistant team has 1 principal agent and 2 assistant agents, the duration of which is 2.

Then, the tasks in L will be considered. In the experiments, a suitable ζ can improve the performance, and 0.3 is the best value in our experiments. For example, in Figure 3, density$_1 = \frac{2}{1^2 \cdot 2} + \frac{0}{2^2 \cdot 2} = 1.0$, density$_2 = \frac{3}{1^2 \cdot 2} + \frac{0}{2^2 \cdot 2} = 1.5$, density$_3 = \frac{1}{1^2 \cdot 2} + \frac{0}{2^2 \cdot 2} = 0.5$, and if $\zeta = 0.3$, then $H = \{v_2\}$, $L = \{v_1, v_3\}$, and the principal–assistant route is $principal_r = \{v_0, v_2, v_0\}$, $assistant_r^1 = \{(v_2, v_1, v_0)\}$, $assistant_r^2 = \emptyset$, which is the optimum.

4.3.3. Assistant Agent Route Construction

The assistant agent route construction algorithm is displayed in Algorithm 6. In the beginning, all the valid partial assistant agent routes are collected from Line 2 to Line 5. In Line 6, the partial assistant agent route (v_i, v_k, v_j) in set $Cand$ is sorted in descending order based on the time cost $cost(v_i, v_j)$, and the routes with the same time cost are sorted in ascending order based on the deadline d_k. The partial assistant agent routes are inserted into the suitable places of the initial assistant agent routes in order.

4.3.4. Integrated BP-Based Algorithm

The integrated BP-based algorithm is detailed in Algorithm 7. The main parts of it are described in the previous sections. The framework of the column generation is described in Section 4.2.1, and the principal agent route construction and assistant agent route construction are described in Section 4.3.2 and Section 4.3.3, respectively.

Algorithm 6: Assistant Agent Ranking Algorithm

Input: A principal agent route $principal_r$, V_u
Output: u assistant agent routes

```
1  Cand ← ∅;
2  for v_i, v_j ∈ principal_r do
3      for v_k ∈ V_u do
4          if (v_i, v_k, v_j) is valid then
5              Cand ← Cand ∪ {(v_i, v_k, v_j)};
6  Sort (v_i, v_k, v_j) in the set Cand in the descending order based on the time cost
     cost(v_i, v_j), the routes with the same time cost are sorted in ascending order
     based on the deadline d_k;
7  cnt ← 0;
8  while cnt < u do
9      assistant_r^cnt ← ∅;
10     for e ∈ Cand do
11         if assistant_r^cnt ∪ e is valid then
12             assistant_r^cnt ← assistant_r^cnt ∪ e;
13             Cand ← Cand − {e};
14     cnt ← cnt + 1;
15 return u assistant agent routes;
```

Additionally, the actual moving distance of the assistant agent in the sortie is an important element. As prescribed, the duration of the assistant agent is η. In previous studies, the assistant agent is assumed to take full use of its power and serve all accessible tasks. However, it is not a suitable policy in practice. An example is displayed in Figure 4. In Figure 4, the line construction creates principal agent route $principal_1 = \{v_0, v_3, v_0\}$, and if the full duration is used, the assistant agent routes will be $assistantagent_1^1 = \{(v_3, v_2, v_0)\}$ and $assistantagent_1^2 = \emptyset$. The other principal–assistant team cannot serve any other vertices. However, if the actual duration is reduced to 2, two principal–assistant routes can be created, including a principal agent route $principal_1 = \{v_0, v_3, v_0\}$ and two empty assistant agent routes $assistantagent_1^1 = \emptyset$, $assistantagent_1^2 = \emptyset$, and another principal agent route $principal_2 = \{v_0, v_2, v_0\}$ and two assistant agent routes $assistantagent_2^1 = \{(v_2, v_1, v_0)\}$, $assistantagent_2^2 = \emptyset$. The latter solution is better than the former one. To search for better principal–assistant routes, in Line 2 to Line 9, different actual distances are used to construct principal–assistant routes, and in Line 16, a random actual duration is generated for each principal–assistant route construction.

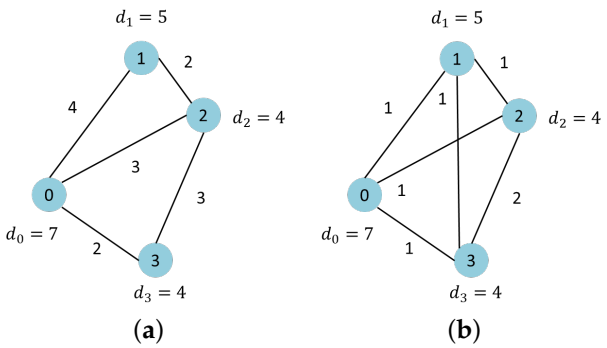

Figure 4. The influence of the duration: subfigure (**a**) shows the tasks V in graph G_1 and subfigure (**b**) shows the tasks V in graph G_2. The principal–assistant team has 2 principal agents and each is equipped with 2 assistant agents, the duration of which is 4.

All the principal–assistant routes will be examined in the ideal environment, but an overly long execution time is impractical. Hence, an iteration upper bound cntlimit is provided to control the execution time.

Algorithm 7: BP-Based Algorithm

Input: Vertex set V, deadline set D, parameters $0.0 < \zeta < 1.0$, $cntlimit$
Output: the set of principal–assistant routes $\widehat{\Omega}$

$\Omega' \leftarrow \emptyset, \widehat{\Omega}' \leftarrow \emptyset, best \leftarrow 0$;
for $dur \in [1, \eta]$ **do**
  $V' \leftarrow V, c \leftarrow 0$;
  while $c < m$ **do**
    $v_s \leftarrow v_0, v_e \leftarrow v_0$;
    $principal_r \leftarrow density\_based(V', D, \zeta, v_s, v_e)$;
    $assistant_r \leftarrow randkedassistantagent(principal_r, V' - principal_r)$;
    $\Omega' \leftarrow \Omega' \cup \{r\}$;
    Update $V', c \leftarrow c + 1$;
Solve integer linear programming based on the set Ω', obtain dual variables $\delta_{i,\gamma}$, current optimum set $\Omega_l \subseteq \Omega'$ and the number of served vertices $score$;
$\widehat{\Omega} \leftarrow \Omega_l, best \leftarrow score$;
$cnt \leftarrow 1$;
while $cnt < cntlimit$ **do**
  $V_c, V_u \leftarrow obtainVers(V, \Omega')$;
  $V' \leftarrow Random\_selection(V_c, V_u)$;
  $dur \leftarrow randomInt(1, \eta)$;
  $principal_r \leftarrow density\_based(V', D, \eta)$;
  $assistant_r \leftarrow randkedassistantagent(principal_r, V' - principal_r)$;
  if $\widehat{c_r} > 0$ **then**
    $\Omega' \leftarrow \Omega' \cup \{r\}$;
    Solve integer linear programming based on the set Ω', obtain dual variables $\delta_{i,\gamma}$, current optimum set $\Omega_l \subseteq \Omega'$ and the number of served vertices $score$;
    if $score > best$ **then**
      $\widehat{\Omega} \leftarrow \Omega_l, best \leftarrow score$;
  $cnt \leftarrow cnt + 1$;
return $\widehat{\Omega}$;

5. Experiments

5.1. Experimental Setup and Evaluation Criteria

The computations are performed on a PC with a Windows 10 OS, 16 GB RAM, and i7-7700 4.2 GHz CPU in Python 3.6. CPLEX (version 12.6) is used to solve all the integer linear programming.

The experiments are conducted on random graphs generated based on the Erdos–Rényi model [25], where all the distance and time costs are discretized into integer numbers. The random graphs ensure that the distance between two adjacent vertices varies from 1 to 10 randomly. In G_1, each vertex is connected to the other with the probability $6/|V|$. The d_0 is twice the diameter of the graph and $d_i, v_i \neq v_0$ is a random variable greater than $m_1(v_i, v_0)$ and less than d_0. In G_2, let $m_2(v_i, v_j)$ be a random variable less than or equal to $m_1(v_i, v_j)$ and greater than 1. The endurance of the assistant agent η is twice the median of the distances between the vertices in G_2.

The evaluation criteria include the number of served vertices and the running time of algorithms. For each graph size n, 100 instances are generated; the average values of the results will be regarded as the performance of algorithms.

5.2. Results in the 1-Principal-1-Assistant Scenario

5.2.1. Compared Algorithms

The hybrid combination algorithm is denoted by HC, and in the simulated annealing, $const = 0.96$, $Temp = 10{,}000$, and $Temp_{min} = 100$ according to the preliminary finding.

The HC is compared to seven algorithms. The ILP denotes the method that solves the integer linear programming method by CPLEX directly. C1 denotes an adaptation of the algorithm proposed by Murray and Chu [7]. The outline of C1 is to construct an initial principal agent route serving all tasks by the saving heuristic proposed by Clarke [26] and then take out a vertex from the principal agent route, which saves the most time, and insert it into other principal agent route or let the assistant agent serve it iteratively until there is no valid vertex to select. C2 is the approximation algorithm for the deadline-TSP without assistant agent proposed by Bansal et al. [13]. The C2+Subjoin uses the operation Subjoin to append the assistant agent route to the principal agent route generated by C2. The CST employs the procedure CONSTURCT on the whole problem directly. CST+CB combines the procedure CONSTURCT and the process of division and combination. The CSC combines the procedure CONSTURCT and the operation Subjoin.

5.2.2. Computational Performance

The number of served vertices and the running time of algorithms are displayed in Figure 5. In Figure 5a,b, until the graph size $n = 15$, the ILP is the optimal method, but its running time exceeds 790 seconds. Because the running time of ILP will grow exponentially as the graph size n grows larger, the ILP is not a suitable method when $n > 15$. When the graph size n ranges from 16 to 20, except for C1 and C2, the other five algorithms are close to each other. However, the running times of C2 and C2+Subjoin exceed 500 s when $n = 19$. As abovementioned, the time complexities of C2 and C2+Subjoin are greater than $\mathcal{O}(n^{10})$, and their actual running times are also extremely large as the graph size $n > 19$. Hence, ILP and C2+Subjoin can be applied in small graphs but are not suitable in the larger graphs due to their running time. Additionally, HC, CST, CST+CB and CSC need to be employed in larger graphs to evaluate their performances. As shown in Figure 5a, the HC serves the most vertices, the CSC is only inferior to the HC, and the CST and the CST+CB serve far fewer vertices than they do. In Figure 5c, the HC costs the most time in each graph size, and when $n = 100$, its running time approaches 900 s. The CSC, CST and CST+CB cost less time, even if $n = 100$, their running time is less than 10 seconds. As a whole, when the graph size ranges from 30 to 100, the HC is the optimal choice, which although costs the most time, serves most vertices.

5.3. Results in the m-Principal-u-Assistant Scenario

5.3.1. Compared Algorithms

The proposed algorithm, called IP-D, combines the BP-based algorithm and the density-based construction. The IP-D is compared with four algorithms. The GL denotes a greedy adaption of line construction [27], which iteratively constructs the principal–assistant routes by line construction and the operation Subjoin based on the residual vertices. The GI denotes a greedy adaption of the line construction [15], which iteratively constructs the principal–assistant routes by the iterated local search construction and the operation Subjoin based on residual vertices. The IP-L combines the BP-based algorithm and the line construction, and the IP-I combines the BP-based algorithm and the iterated local search construction.

5.3.2. Computational Performance

There are three concrete scenarios, including the 3-principal-2-assistant scenario, the 3-principal-3-assistant scenario, and the 4-principal-2-assistant scenario. The results are shown in Figure 6.

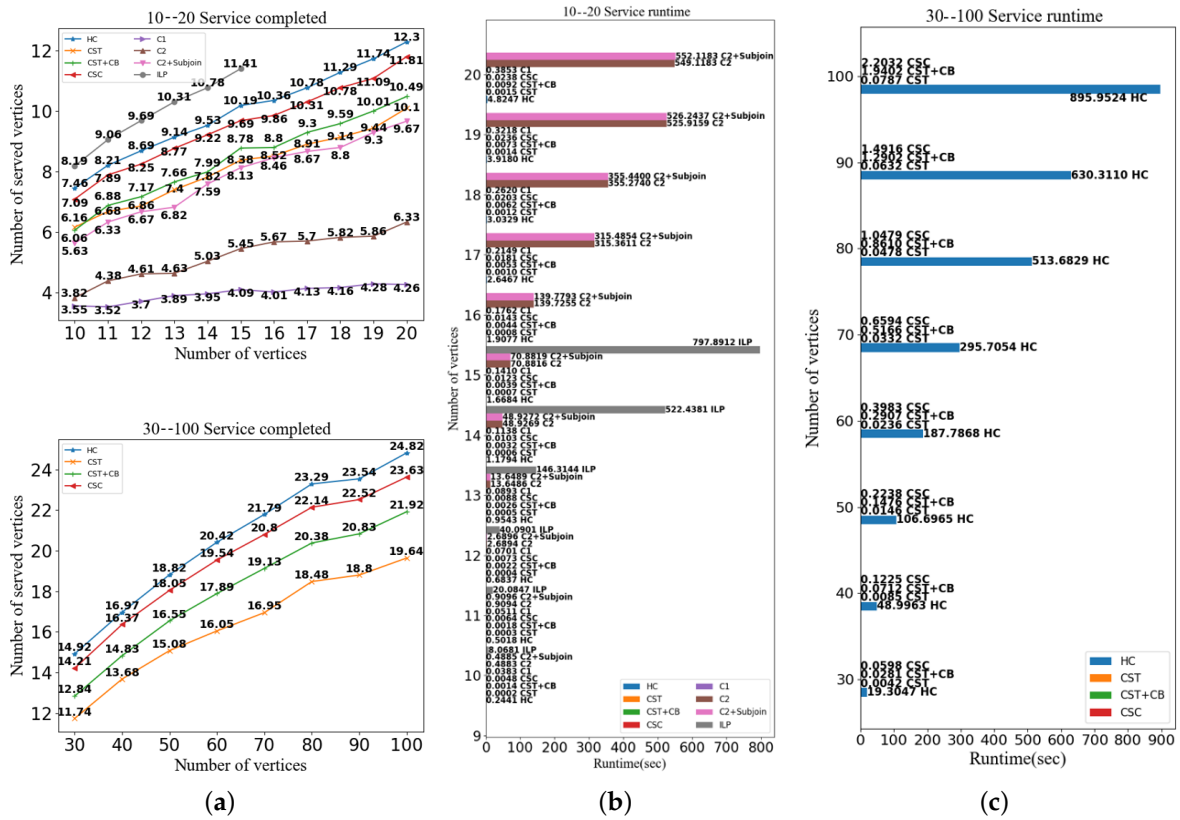

Figure 5. The performance of our approach and baseline algorithms in the 1-principal-1-assistant scenario. Test of the Number of Served Vertices: (**a**); Test of the Running Time: (**b**,**c**).

In the 3-principal-2-assistant scenario, as shown in Figure 6a,c, GL, IP-L, IP-I, and IP-D share similar numbers of served vertices, but GI serves far less vertices than they do. The GL, GI, IP-L, and IP-D cost time less than 40 seconds, even if $n = 20$, but the IP-I costs more time, and exceeds 800 seconds when $n = 20$. The running time of IP-I is too large to be employed in larger graphs. Figure 6b,d shows the algorithm performances in the larger graphs, the size of which ranges from 30 to 100. Except when $n = 30$, the IP-D serves the most vertices. The gap between the number of vertices served by the IP-D and the others grows larger as the sizes of graphs grow larger. With respect to the running time, the IP-D costs the most time among all the algorithms, but its maximum running time is less than 600 s. In the 3-principal-3-assistant scenario shown in Figure 6e–h, and the 4-principal-2-assistant scenario shown in Figure 6i–l, the IP-D has a similar performance. As a whole, the number of vertices served by the IP-D is the largest among all the algorithms, and the running time of the IP-D is the largest among the others but is reasonable in practice.

There is a strange phenomenon where IP-L and IP-D expend more time when $n = 30$ than $n = 100$. The main time cost comes from the integer programming, which is $\mathcal{NP}$-hard. More columns in the integer programming indicate a longer CPU time. Based on the experimental data, when the number of columns that denote the principal–assistant routes in the paper reaches 400, the IP-L and IP-D will expend more time to solve their integer programming problems. Because the d_0 is twice the diameter of the graph, when $n = 30$, one principal–assistant route cannot serve many tasks surrounding the initial vertex, so the new route r, which serves more unserved tasks far from the initial vertex, can be added to Ω'. However, when $n = 100$, the d_0 is very large and the principal–assistant routes seek to serve the vertices surrounding the initial vertex; many generated routes are the same or cannot coexist, and the actual size of the route set Ω' when $n = 100$ is less than the size when $n = 30$.

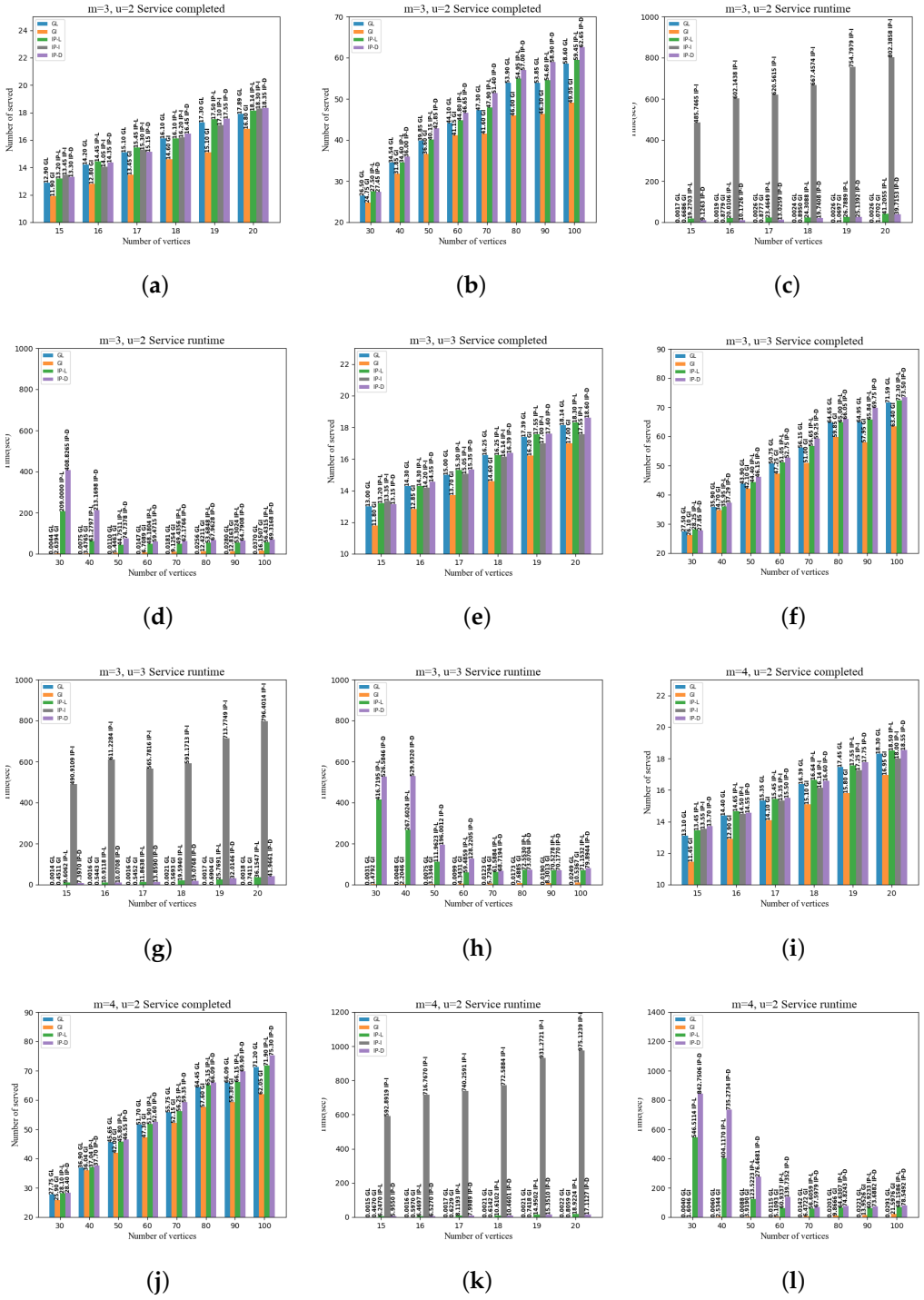

Figure 6. The performance of our approach and baseline algorithms in the *m*-principal-*u*-assistant scenario. Test of the Number of Served Vertices in the 3-principal-2-assistant scenario: (**a**,**b**); Test of the Running Time in the 3-principal-2-assistant scenario: (**c**,**d**); Test of the Number of Served Vertices in the 3-principal-3-assistant scenario: (**e**,**f**); Test of the Running Time in the 3-principal-3-assistant scenario: (**g**,**h**); Test of the Number of Served Vertices in the 4-principal-2-assistant scenario: (**i**,**j**); Test of the Running Time in the 4-principal-2-assistant scenario: (**k**,**l**).

In general, the number of vertices served by the IP-D is largest among all the algorithms, although the running time of the IP-D is also largest among them; however, this is reasonable in practice.

6. Conclusions and Future Research

The principal–assistant collaboration problem with deadlines is investigated in this paper. Two scenarios are systemically discussed here, including the 1-principal-1-assistant scenario and the m-principal-u-assistant scenario. The hybrid combination algorithm is proposed to solve the 1-principal-1-assistant scenario, which combines different partial routes generated by three methods. Through comprehensive experiments, we validate the performance of the hybrid combination algorithm by comparing to several benchmark approaches. It can find a better route than the other algorithms compared except for the exact integer linear programming, but costs far less time than the integer linear programming. Therefore, this method can be feasible in 1-principal-1-assistant collaboration applications. A BP-based algorithm embedded with a density-based construction is proposed for the m-principal-u-assistant scenario. It uses the branch-and-price algorithm to find the optimal combination of the current route set and uses the density-based construction to enlarge the route set. Through comprehensive experiments, we compare the performance of the proposed algorithm with several benchmark approaches. In most cases, the proposed algorithm can serve the most vertices, although it takes more running time but is still reasonable. Therefore, the algorithm is practical in m-principal-u-assistant collaboration scenarios.

In the future, we will mainly explore the adaptation of hybrid principal–assistant collaboration systems from two directions: (1) There are often multiple initial locations of agents in real applications, and multiple tasks may be in the same location; hence, one direction is to consider multiple initial vertices and more than one task in a vertex in the principal–assistant collaboration problem. (2) There are various collaborative relationships between principal agents and assistant agents in different environments; hence, the other direction is to improve the locomotion of the principal–assistant teams by considering appropriate collaboration pattern selection.

Author Contributions: Conceptualization, Y.Z. and K.D.; methodology, Y.Z., K.D. and H.X.; validation, K.D. and H.X.; formal analysis, Y.Z. and K.D.; writing—original draft preparation, Y.Z., K.D. and H.X.; writing—review and editing, Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (No.61807008, 61806053, 61932007, 62076060, and 61703097); and the Natural Science Foundation of Jiangsu Province of China (BK20180369, BK20180356, BK20201394, and BK20171363).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jiang, Y.; Zhou, Y.; Li, Y. Reliable task allocation with load balancing in multiplex networks. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **2015**, *10*, 1–32. [CrossRef]
2. Rahimzadeh, F.; Khanli, L.M.; Mahan, F. High reliable and efficient task allocation in networked multi-agent systems. *Auton. Agents -Multi-Agent Syst.* **2015**, *29*, 1023–1040. [CrossRef]
3. Jiang, Y.; Zhou, Y.; Wang, W. Task allocation for undependable multiagent systems in social networks. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *24*, 1671–1681. [CrossRef]
4. Agatz, N.; Bouman, P.; Schmidt, M. Optimization approaches for the traveling salesman problem with drone. *Transp. Sci.* **2018**, *52*, 965–981.
5. Sloat, S.; Kopplin, I. Daimler to work with Matternet to develop delivery van drones. *Wall Str. J.* **2016**, *45*, 63–86.
6. Scott, J.; Scott, C. Drone delivery models for healthcare. In Proceedings of the Hawaii International Conference on System Sciences, Hilton Waikoloa Village, HI, USA, 4–7 January 2017.
7. Murray, C.C.; Chu, A.G. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transp. Res. Part Emerg. Technol.* **2015**, *54*, 86–109.
8. Ha, Q.M.; Deville, Y.; Pham, Q.D.; Hà, M.H. On the min-cost traveling salesman problem with drone. *Transp. Res. Part Emerg. Technol.* **2018**, *86*, 597–621. [CrossRef]
9. Sawadsitang, S.; Niyato, D.; Tan, P.S.; Wang, P. Joint ground and aerial package delivery services: A stochastic optimization approach. *IEEE Trans. Intell. Transp. Syst.* **2018**, *20*, 2241–2254. [CrossRef]
10. Farbstein, B.; Levin, A. Deadline TSP. *Theor. Comput. Sci.* **2019**, *771*, 83–92. [CrossRef]

11. Dell'Amico, M.; Righini, G.; Salani, M. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transp. Sci.* **2006**, *40*, 235–247.
12. Parragh, S.N.; Cordeau, J.F. Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. *Comput. Oper. Res.* **2017**, *83*, 28–44.
13. Bansal, N.; Blum, A.; Chawla, S.; Meyerson, A. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In Proceedings of the Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 13–15 June 2004; pp. 166–174.
14. Wang, Z.; Sheu, J.B. Vehicle routing problem with drones. *Transp. Res. Part Methodol.* **2019**, *122*, 350–364.
15. Gunawan, A.; Lau, H.C.; Lu, K. An iterated local search algorithm for solving the orienteering problem with time windows. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization, Copenhagen, Denmark, 8–10 April 2015; pp. 61–73.
16. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2009.
17. Peng, K.; Du, J.; Lu, F.; Sun, Q.; Dong, Y.; Zhou, P.; Hu, M. A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery. *IEEE Access* **2019**, *7*, 49191–49200. [CrossRef]
18. Černỳ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl.* **1985**, *45*, 41–51. [CrossRef]
19. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680.
20. Granville, V.; Krivánek, M.; Rasson, J.P. Simulated annealing: A proof of convergence. *IEEE Trans. Pattern Anal. Mach. Intell.* **1994**, *16*, 652–656. [CrossRef]
21. Gunawan, A.; Lau, H.C.; Vansteenwegen, P. Orienteering problem: A survey of recent variants, solution approaches and applications. *Eur. J. Oper. Res.* **2016**, *255*, 315–332. [CrossRef]
22. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of state calculations by fast computing machines. *J. Chem. Phys.* **1953**, *21*, 1087–1092. [CrossRef]
23. Lozano, L.; Medaglia, A.L. On an exact method for the constrained shortest path problem. *Comput. Oper. Res.* **2013**, *40*, 378–384. [CrossRef]
24. Applegate, D.; Bixby, R.; Chvátal, V.; Cook, W. TSP Cuts Which Do Not Conform to the Template Paradigm. In *Computational Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 2001.
25. Erdos, P.; Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* **1960**, *5*, 17–60.
26. Clarke, G.; Wright, J.W. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* **1964**, *12*, 568–581.
27. Kantor, M.G.; Rosenwein, M.B. The orienteering problem with time windows. *J. Oper. Res. Soc.* **1992**, *43*, 629–635. [CrossRef]