Balanced Task Allocation by Partitioning the Multiple Traveling Salesperson Problem

Isaac Vandermeulen
The University of Sheffield
Sheffield, UK
iavandermeulen1@sheffield.ac.uk

Roderich Groß
The University of Sheffield
Sheffield, UK
r.gross@sheffield.ac.uk

Andreas Kolling
iRobot Corporation
Pasadena, California
akolling@irobot.com

ABSTRACT

Task assignment and routing are tightly coupled problems for teams of mobile agents. To fairly balance the workload, each agent should be assigned a set of tasks which take a similar amount of time to complete. The completion time depends on the time needed to travel between tasks which depends on the order of tasks. We formulate the task assignment problem as the minimum Hamiltonian partition problem (MHPP) for m agents, which is equivalent to the minmax multiple traveling salesperson problem (m-TSP). While the MHPP's cost function depends on the order of tasks, its solutions are similar to solutions of the average Hamiltonian partition problem (AHPP) whose cost function is order-invariant. We prove that the AHPP is NP-hard and present an effective heuristic, AHP, for solving it. AHP improves a partitions of a graph using a series of transfer and swap operations which are guaranteed to improve the solution's quality. The solution generated by AHP is used as an initial partition for an algorithm, AHP-mTSP, which solves the combined task assignment and routing problems to near optimality. For n tasks and m agents, each iteration of AHP is $O(n^2)$ and AHP-mTSP has an average run-time that scales with $n^{2.11}m^{0.33}$. Compared to state-of-the-art approaches, our approach found approximately 10% better solutions for large problems in a similar run-time.

KEYWORDS

Task allocation; vehicle routing; combinatorial optimization; multi-agent systems

ACM Reference Format:
Isaac Vandermeulen, Roderich Groß, and Andreas Kolling. 2019. Balanced Task Allocation by Partitioning the Multiple Traveling Salesperson Problem. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Work is shared in multi-agent teams by assigning different tasks to each agent. For tasks associated with physical locations, a mobile agent must travel to the task before completing it. The total time taken for this agent to complete its tasks includes the time needed to travel between consecutive tasks, which depends on the order of tasks. This transit time must be considered when fairly assigning tasks among agents.

Task allocation problems are based on the central objective of maximizing the utility of the tasks allocated to each agent [13, 21]. For heterogeneous teams, different agents have different utilities for each task [12]; for homogeneous teams, the utility depends only on the task [27]. Utility can be defined as the sum over subsets of tasks and is not necessarily the sum of the utility of individual tasks [31, 37]. The objective of task allocation is to maximize either the sum of utilities, or the utility of the agent with the smallest utility which results in a fair allocation [47].

Tasks are commonly allocated using economic approaches such as auctions [5, 12, 30], markets [8, 47], or token exchange [9]. Another approach is using bounties where agents are only rewarded after completing a task [45]. Task assignment and path finding (TAPF) are often combined into a single problem to find optimal assignments while planning collision free paths [26] which can be extended to include additional tasks by using auctions [27] or by not requiring every task to be completed [17].

For mobile agents, the combined task allocation and routing problem is closely related to the multiple traveling salesperson problem (m-TSP) [2, 19, 36, 40, 46]. The m-TSP asks: "What is the quickest way for m salespeople to visit a set of n cities?" and is a generalization of the 1-TSP, which is well-known NP-hard routing problem [33]. While task assignment is primarily a *partition* problem, the m-TSP is primarily a *routing* problem.

There are two common objectives for the m-TSP. In the minsum m-TSP, the objective is to minimize the sum of the path lengths; in the minmax m-TSP, the objective is to minimize the length of the longest path. In practical applications, these objectives correspond to minimizing total distance traveled by the team and minimizing mission time, respectively. These objectives are usually conflicting even for small problems [35]. Minmax objectives are more suited to balanced task allocation; however, a minsum objective with a constraint on agents' path lengths can have a similar result [41, 43].

The 1-TSP (and hence the m-TSP) is NP-hard [33]. Christofides [6] proposed an $O(n^3)$ algorithm which returns a tour with length no more than $3/2$ times the optimal length. Other 1-TSP heuristics consist of an initial tour construction phase followed by an improvement phase [22]. The variable k-opt method proposed by Lin and Kernighan [23] and similar variants [16, 18] are some of the most effective improvement-based heuristics. It is a major component of the freely available Concorde [7] solver.

Solutions to the minmax m-TSP are often found by converting the problem into the 1-TSP by copying one vertex m times [14] or dividing a 1-TSP solution into m pieces [3]. Alternatively, vertices can be assigned to agents, for example by k-means clustering [29], before finding a route for each agent by solving the 1-TSP. A multi-agent version of Christofides' algorithm can guarantee a solution with an optimality bound of $5/2 - 1/m$ [10].

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

Other m-TSP heuristics use a common approach of tour generation, improvement, and recombination. An initial set of paths are generated using a modified version of Christofides' algorithm [4], k-means clustering [20], k-centers clustering [28], nearest neighbor, greedy, or random heuristics. These tours are improved by tabu search [32], simulated annealing [38], compressed annealing [25], or general variable neighborhood search [39]. If many solutions are generated, they can be recombined using evolutionary methods [1], ant colony optimization [24], invasive weed optimization [42], or a memetic algorithm [44].

In this paper, we design a new task allocation algorithm for mobile agents which allocates tasks to minimize the completion time for a team of agents. This algorithm exploits a relationship between average and minimum Hamiltonian cycle lengths which is explained in Section 3. The algorithm consists of partitioning (Section 4) and routing (Section 5) phases, can be decentralized (Section 6), and can incorporate constraints on the agents' start and end locations (Section 7). Some comparisons with existing minmax m-TSP algorithms are presented in Section 8.

2 TASK ALLOCATION AND THE m-TSP

The combined task allocation and routing problem can be defined on a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V}$ representing tasks and edge set $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ representing transit between two tasks. Task completion times are represented by a function $w_\mathrm{v} : \mathcal{V} \to \mathbb{R}_{\geq 0}$ and transit times are represented by $w_\mathrm{e} : \mathcal{E} \to \mathbb{R}_{\geq 0}$. We denote the number of tasks by n and the number of agents by m.

Suppose agent i is assigned a set of tasks, $\mathcal{V}_i \subset \mathcal{V}$ with $n_i = |\mathcal{V}_i|$. The time needed to complete these tasks depends on the order they are completed. This order can be represented as a cycle $c = (e_0, \ldots, e_{n_i})$ which visits each vertex of $\mathcal{V}_i$ once. For an assignment $\mathcal{V}_i$ and route c, the completion time is

$$t_\mathrm{total}(\mathcal{V}_i, c) = \sum_{v \in \mathcal{V}_i} w_\mathrm{v}(v) + \sum_{e \in c} w_\mathrm{e}(e). \tag{1}$$

Each vertex is incident to exactly two edges of the cycle, so we can define $w : \mathcal{E} \to \mathbb{R}_{\geq 0}$ by $w(e) = \frac{1}{2}\left(w_\mathrm{v}(v_0) + w_\mathrm{v}(v_1)\right) + w_\mathrm{e}(e)$ where $e = (v_0, v_1)$ and then rewrite (1) simply as

$$t_\mathrm{total}(\mathcal{V}_i, c) = \sum_{e \in c} w(e). \tag{2}$$

Through out the remainder of the paper, we simplify notation by using the edge weight w instead of w_v and w_e.

A subset of vertices $\mathcal{V}_i \subset \mathcal{V}$ induces a subgraph $\mathcal{G}_i$ of $\mathcal{G}$. A *Hamiltonian* cycle on $\mathcal{G}_i$ is a cycle which visits each vertex exactly once. Let $c^*(\mathcal{G}_i)$ be the shortest Hamiltonian cycle on $\mathcal{G}_i$. Agent i can complete its assigned tasks as quickly as possible by following c^* and so we define the *size* of a subgraph $\mathcal{G}_i$ by

$$S_\mathrm{m}(\mathcal{G}_i) = \sum_{e \in c^*(\mathcal{G}_i)} w(e). \tag{3}$$

Computing S_m is difficult because finding the shortest Hamiltonian cycle is equivalent to solving the 1-TSP which is NP-hard [33]. In Section 3 we define an alternate size of subgraphs which can be computed in polynomial time and provide relationship between it and S_m.

A partition, $\mathcal{P} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$, of $\mathcal{G}$ is a set of subgraphs with each vertex of $\mathcal{G}$ contained in exactly one subgraph $\mathcal{G}_i$. The task assignment problem is to partition $\mathcal{G}$ into m subgraphs while minimizing the size required for the slowest agent to complete all of its tasks. As the time required for agent i to complete the tasks of $\mathcal{V}_i$ is equal to the size $S_\mathrm{m}(\mathcal{G}_i)$, we define the cost of a partition as

$$C_\mathrm{m}(\mathcal{P}) = \max_{\mathcal{G}_i \in \mathcal{P}} \{S_\mathrm{m}(\mathcal{G}_i)\}. \tag{4}$$

We define the task allocation problem as the Minimimum Hamiltonian Partition Problem (MHPP) whose objective is to find the optimal partition $\mathcal{P}^*$ which minimizes C_m.

Problem 1 (MHPP). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ be a complete weighted graph. For a given $m \geq 2$, find a partition $\mathcal{P} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ of $\mathcal{G}$ which minimizes C_m as defined in Equation 4.*

The MHPP is closely related to the minmax multiple traveling salesperson problem (m-TSP). A strategy for the m-TSP is a set of m disjoint cycles, $\mathcal{S} = \{c_1, \ldots, c_m\}$ such that each $v \in \mathcal{V}$ is in exactly one cycle of $\mathcal{S}$. The cost of a strategy is the length of its longest cycle:

$$C(\mathcal{S}) = \max_{c \in \mathcal{S}} \left\{ \sum_{e \in c} w(e) \right\}. \tag{5}$$

The objective of the m-TSP is to find $\mathcal{S}^*$ which minimizes C.

Problem 2 (m-TSP). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ be a complete weighted graph. For a given $m \geq 2$, find a strategy $\mathcal{S} = \{c_1, \ldots, c_m\}$ on $\mathcal{G}$ which minimizes C as defined in Equation 5.*

The MHPP and m-TSP are equivalent problems. A solution to the MHPP can be converted to a solution to the m-TSP by solving the 1-TSP on each subgraph of the partition. A solution to the m-TSP can be converted to a solution to the MHPP by defining each subgraph by the vertices of a single cycle of the m-TSP solution. For the remainder of the paper, all cycles are Hamiltonian.

The m-TSP can also be defined for non-cyclic paths. For the majority of this paper, we consider the cyclic version of the m-TSP and extend these results to non-cyclic paths in Section 7.

3 A PROXY FOR MINIMUM CYCLE LENGTH

The MHPP is difficult to solve because $S_\mathrm{m}(\mathcal{G}_i)$ can only be computed by solving the NP-hard 1-TSP. Instead, we consider a similar partition problem with an easy-to-compute cost function.

Problem 3 (Average Hamiltonian Partition Problem (AHPP)). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a complete weighted graph. For a given $m \geq 2$, find a partition $\mathcal{P} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ of $\mathcal{G}$ which minimizes*

$$C_\mathrm{a}(\mathcal{P}) = \max_{\mathcal{G}_i \in \mathcal{P}} \{S_\mathrm{a}(\mathcal{G}_i)\} \tag{6}$$

where $S_\mathrm{a}(\mathcal{G}_i)$ is the average length of a cycle on the subgraph $\mathcal{G}_i$.

The AHPP uses the average cycle length in its cost function instead of the minimum cycle length. The average cycle length can be computed in quadratic time. Since $\mathcal{G}_i$ is a complete subgraph, each of its edges is equally likely to appear in a cycle. There are $|\mathcal{E}_i| = \frac{n_i(n_i-1)}{2}$ edges in $\mathcal{G}_i$ and n_i edges in a cycle so

$$S_\mathrm{a}(\mathcal{G}_i) = n_i \sum_{e \in \mathcal{E}_i} \frac{2}{n_i(n_i-1)} w(e) = \frac{2}{n_i - 1} \sum_{e \in \mathcal{E}_i} w(e). \tag{7}$$

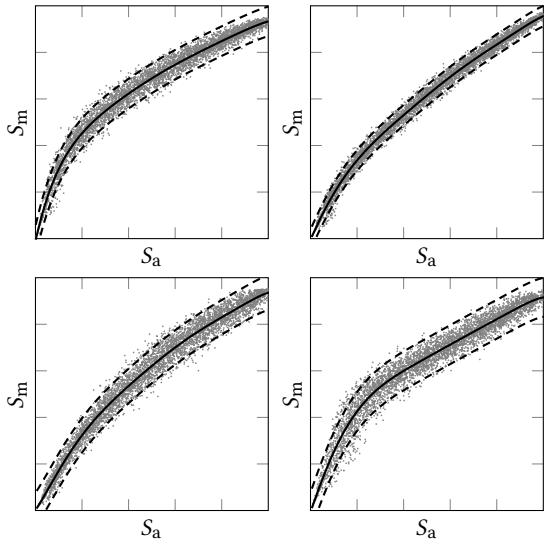

Figure 1: Minimum vs average cycle lengths of randomly sampled subgraphs of a 2D Euclidean graph (top left), a 3D Euclidean graph (top right), an Erdős-Rényi random graph with weights from an exponential distribution with parameter 1 (bottom left), and the graph of the lower 48 US capitals (bottom right). Dots represents the average and estimated minimum cycle lengths (computed using Concorde [7]) of a randomly sampled subgraph, the solid line is the mean relationship between S_a and S_m, and the dashed lines are the 2.5% and 97.5% quantiles.

Using this formula, $S_a(\mathcal{G}_i)$ can be computed in $O(n_i^2)$. Computing $C_a(\mathcal{P})$ requires computation of $S_a(\mathcal{G}_i)$ for all m subgraphs of $\mathcal{P}$. As $\sum_{i=1}^{m} n_i^2 < n^2$, it can be computed in $O(n^2)$.

3.1 Is C_a a good proxy for C_m?

Our intention is to use the solution to the AHPP as a solution to the MHPP. Although $S_a(\mathcal{G}_i)$ is in general much larger than $S_m(\mathcal{G}_i)$ and not proportional to it, there is a relationship between S_a and S_m for subgraphs of the same graph (Figure 1). For many graphs

$$S_m(\mathcal{G}_i) = f(S_a(\mathcal{G}_i)) + \nu \tag{8}$$

where f is a monotonically increasing function and ν is 0-mean noise. For any $\alpha \in (0, 0.5)$, we define $b_\alpha^-, b_\alpha^+ \in \mathbb{R}_{>0}$ such that

$$\mathbb{P}\left[\nu \geq -b_\alpha^-\right] = \mathbb{P}\left[\nu \leq +b_\alpha^+\right] = 1 - \alpha.$$

For $\alpha = 0.25$, $[-b_\alpha^-, b_\alpha^+]$ is the 95% confidence interval for ν. We use these quantiles to establish that solutions to the AHPP are good solutions for the MHPP.

Lemma 1. *Suppose that $\mathcal{G}$ is such that (8) holds, then $C_m(\mathcal{P}) = f(C_a(\mathcal{P})) + \nu$ for any partition $\mathcal{P}$ of $\mathcal{G}$.*

Sketch of proof. Let $\mathcal{G}_m^*, \mathcal{G}_a^* \in \mathcal{P}$ be the subgraphs which maximize S_m and S_a, respectively. This lemma is a result of the facts that $S_a(\mathcal{G}_a^*) \geq S_a(\mathcal{G}_m^*)$, $S_m(\mathcal{G}_m^*) \geq S_m(\mathcal{G}_a^*)$, and that (8) holds for both $\mathcal{G}_m^*$ and $\mathcal{G}_a^*$. □

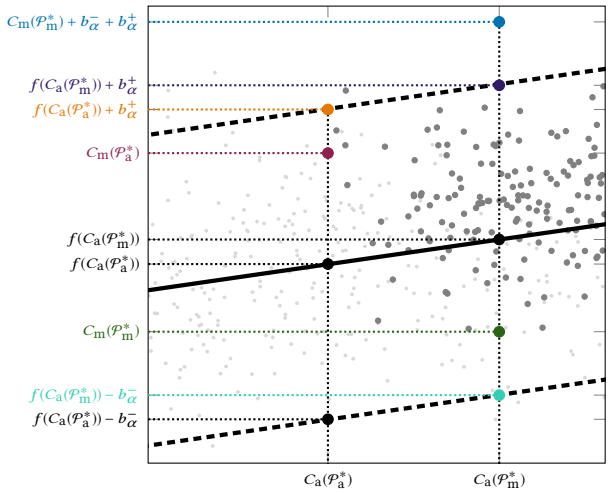

Figure 2: Magnified section of the bottom right graph of Figure 1 showing quantities involved in the proof of Theorem 1. Light gray dots represent $(S_a(\mathcal{G}_i), S_m(\mathcal{G}_i))$ for subgraphs $\mathcal{G}$. Dark gray dots represent $(C_a(\mathcal{P}), C_m(\mathcal{P}))$ for partitions of $\mathcal{G}$. The red and green dots represent the partitions which minimize C_a and C_m.

Theorem 1. *Suppose that $\mathcal{G}$ is such that (8) holds, then*

$$\mathbb{P}\left[C_m(\mathcal{P}_a^*) \leq C_m(\mathcal{P}_m^*) + b_\alpha^- + b_\alpha^+\right] \geq (1-\alpha)^2$$

where $\mathcal{P}_a^*$ and $\mathcal{P}_m^*$ are the partitions that minimize C_a and C_m as defined in (6) and (4).

Proof. Some quantities used in this proof are colored according to Figure 2 to aid understanding. First, we define the events:

$$\mathrm{A} : C_m(\mathcal{P}_a^*) \leq C_m(\mathcal{P}_m^*) + b_\alpha^- + b_\alpha^+$$
$$\mathrm{B} : C_m(\mathcal{P}_a^*) \leq f(C_a(\mathcal{P}_a^*)) + b_\alpha^+$$
$$\mathrm{C} : C_m(\mathcal{P}_m^*) \geq f(C_a(\mathcal{P}_m^*)) - b_\alpha^-.$$

Since $\mathcal{P}_a^*$ minimizes C_a, it is always true that $C_a(\mathcal{P}_a^*) \leq C_a(\mathcal{P}_m^*)$. The monotonicity of f preserves the inequality so $f(C_a(\mathcal{P}_a^*)) \leq f(C_a(\mathcal{P}_m^*))$. If B holds then

$$C_m(\mathcal{P}_a^*) \leq f(C_a(\mathcal{P}_a^*)) + b_\alpha^+ \leq f(C_a(\mathcal{P}_m^*)) + b_\alpha^+.$$

If C also holds, then

$$\begin{aligned} C_m(\mathcal{P}_a^*) &\leq f(C_a(\mathcal{P}_m^*)) - b_\alpha^- + b_\alpha^- + b_\alpha^+ \\ &\leq C_m(\mathcal{P}_m^*) + b_\alpha^- + b_\alpha^+ = C_m(\mathcal{P}_m^*) + b_\alpha^- + b_\alpha^+. \end{aligned}$$

Therefore $\mathrm{B} \cap \mathrm{C} \Rightarrow \mathrm{A}$ so $\mathbb{P}\left[\mathrm{A}\right] \geq \mathbb{P}\left[\mathrm{B} \cap \mathrm{C}\right]$. As B and C depend on different variables ($\mathcal{P}_a^*$ and $\mathcal{P}_m^*$), they are independent so $\mathbb{P}\left[\mathrm{A}\right] \geq \mathbb{P}\left[\mathrm{B}\right] \times \mathbb{P}\left[\mathrm{C}\right]$. By Lemma 1, the probabilities of B and C are both at least $(1-\alpha)$ and therefore $\mathbb{P}\left[\mathrm{A}\right] \geq (1-\alpha)^2$. □

Corollary 1. *Suppose there exists a graph $\mathcal{G}$ and $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that $S_m(\mathcal{G}_i) = f(S_a(\mathcal{G}_i))$ for every subgraph $\mathcal{G}_i$ of $\mathcal{G}$. Then $\mathcal{P}_a^* = \mathcal{P}_m^*$ and so the solution to the AHPP also solves the MHPP.*

Theorem 1 establishes that C_a is a good proxy for C_m when (8) holds. Experimental evidence suggests (8) holds for many common graphs (Figure 1). This result motivates us to use a solution to the AHPP as a proxy for the solutions to the MHPP when developing a task allocation heuristic. As the AHPP can itself be viewed as a heuristic approximation of the MHPP, this approach enables us to find good solutions to the MHPP by solving the AHPP, avoiding the problem of evaluating the MHPP's cost function which is NP-hard.

3.2 Hardness of the AHPP

Our task allocation heuristic relies on solutions to the AHPP. Although the AHPP's cost function can be computed in polynomial time, the overall problem is still NP-hard so we will develop a heuristic for the AHPP instead of solving it exactly.

Theorem 2. *The AHPP is NP-hard.*

Proof. We prove that the AHPP is NP-hard by reducing the known NP-hard number partition problem (NPP) [11] to it. The NPP asks "Given a multiset of positive integers $\mathcal{Z}$ with even sum K, does there exist a partition $\{\mathcal{Z}_1, \mathcal{Z}_2\}$ where $\mathcal{Z}_1$ and $\mathcal{Z}_2$ both sum to at most $\frac{K}{2}$?" We will reduce this problem to a decision version of the AHPP which asks "Given a complete graph $\mathcal{G}$ with positive weights, does there exist a partition $\{\mathcal{G}_1, \mathcal{G}_2\}$ such that the average cycle length on each subgraph is at most L?"

For any instance $(\mathcal{Z}, K)$ of the NPP, we can construct an instance $(\mathcal{G}, L)$ of the AHPP. Let $L = \frac{K}{2}$, $n = |\mathcal{Z}|$, and $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ be a complete weighted graph with $|\mathcal{V}| = n$. Each $v_i \in \mathcal{V}$ corresponds to a $z_i \in \mathcal{Z}$. The weight of an edge $e = (v_i, v_j)$ is defined as $w(e) = \frac{1}{2}(z_i + z_j)$. This reduction can be performed in quadratic time and can be applied to any instance of the NPP.

Let $\mathcal{J}_1$ be a subset of $\{1, \ldots, n\}$. It corresponds to a multiset of integers, $\mathcal{Z}_1$, and a set of vertices, $\mathcal{V}_1$, defined by

$$\mathcal{Z}_1 = \{z_j \in \mathcal{Z} \mid j \in \mathcal{J}_1\}, \quad \mathcal{V}_1 = \{v_j \in \mathcal{V} \mid j \in \mathcal{J}_1\}.$$

It also corresponds to $\mathcal{G}_1$, a subgraph of $\mathcal{G}$ induced by $\mathcal{V}_1$. Using this definition, every subgraph of $\mathcal{G}$ corresponds to a unique subset of $\mathcal{Z}$ and vice versa.

By the definition of w, it is easy to verify that every cycle on $\mathcal{G}_1$ has the same length and so the average cycle length is

$$S_a(\mathcal{G}_1) = \sum_{z \in \mathcal{Z}_1} z.$$

From $\mathcal{J}_1$, we can define $\mathcal{J}_2 = \{1, \ldots, n\} \setminus \mathcal{J}_1$ and corresponding $\mathcal{Z}_2$ and $\mathcal{G}_2$. $\{\mathcal{Z}_1, \mathcal{Z}_2\}$ is a partition of $\mathcal{Z}$ and $\{\mathcal{G}_1, \mathcal{G}_2\}$ is a partition of $\mathcal{G}$. As $\mathcal{J}_2$ is also a subset of $\{1, \ldots, n\}$, it is also true that $S_a(\mathcal{G}_2)$ equals the sum of $\mathcal{Z}_2$.

Suppose the NPP is true. Then there exists a partition $\{\mathcal{Z}_1, \mathcal{Z}_2\}$ which both sum to at most $\frac{K}{2}$. This partition corresponds to a partition $\{\mathcal{G}_1, \mathcal{G}_2\}$ of $\mathcal{G}$. Both $S_a(\mathcal{G}_1)$ and $S_a(\mathcal{G}_2)$ are equal to the sums of $\mathcal{Z}_1$ and $\mathcal{Z}_2$ which are both at most $L = \frac{K}{2}$ so the AHPP is true. Similarly, if the AHPP is true, then there exists a partition $\{\mathcal{G}_1, \mathcal{G}_2\}$ with $S_a(\mathcal{G}_1)$ and $S_a(\mathcal{G}_2)$ at most L so in the corresponding $\{\mathcal{Z}_1, \mathcal{Z}_2\}$, both subsets sum to at most $\frac{K}{2} = L$ and the NPP is true. Therefore the NPP can be reduced to the AHPP and since the NPP is NP-hard, the AHPP must also be NP-hard. □

Corollary 2. *The MHPP is NP-hard.*

4 A TASK ALLOCATION HEURISTIC BASED ON THE AHPP

As the AHPP is NP-hard, we designed a novel heuristic for solving it (Algorithm 3). It consists of two alternating phases—improvement (Algorithm 1) and transferring outliers (Algorithm 2)—which modify an initial partition and create a near optimal solution to the AHPP. Our novel partitioning approach is explicitly based on a minmax criterion and only depends on the value of the edge weight function. Unlike other approaches, such as k-means clustering [29], it can be used on non-Euclidean graphs which are important in real-world problems where travel times are not proportional to as-the-crow-flies distances.

4.1 Improvement through transfers and swaps

The improvement phase (Algorithm 1) transfers and swaps vertices between pairs of subgraphs to decrease their maximum size. This algorithm improves a partition until it is a local minimizer of C_a with respect to the transfer and swap operations. As Algorithm 1 requires computation of $S_a(\mathcal{G}_i)$ for many graphs, we define the total edge weight, $W(\mathcal{G}_i)$, and the marginal edge weight, $\Delta W(\mathcal{G}_i, v)$ by

$$W(\mathcal{G}_i) = \sum_{e \in \mathcal{E}_i} w(e) \tag{9}$$

$$\Delta W(\mathcal{G}_i, v) = \sum_{v' \in \mathcal{V}_i} w((v, v')). \tag{10}$$

Combining (7) and (9), a subgraph's size can be written as

$$S_a(\mathcal{G}_i) = \frac{2}{n_i - 1} W(\mathcal{G}_i). \tag{11}$$

The marginal edge weights will be used to efficiently update $W(\mathcal{G}_i)$ and $S_a(\mathcal{G}_i)$ when a vertex is added or removed from the graph.

Transfers are the simplest modification of a partition. A transfer consists of a single vertex $v \in \mathcal{V}_i$ moving from $\mathcal{G}_i$ to $\mathcal{G}_j$. After a transfer, the new subgraphs $\mathcal{G}_i'$ and $\mathcal{G}_j'$ have sizes

$$S_a(\mathcal{G}_i') = \frac{2}{n_i - 2}\Big(W(\mathcal{G}_i) - \Delta W(\mathcal{G}_i, v)\Big) \tag{12}$$

$$S_a(\mathcal{G}_j') = \frac{2}{n_j}\Big(W(\mathcal{G}_j) + \Delta W(\mathcal{G}_j, v)\Big). \tag{13}$$

Transfers usually result in a large decrease in $S_a(\mathcal{G}_i)$ and an increase in $S_a(\mathcal{G}_j)$ which makes them useful when $S_a(\mathcal{G}_i) \gg S_a(\mathcal{G}_j)$.

When $S_a(\mathcal{G}_i)$ and $S_a(\mathcal{G}_j)$ are nearly equal, we can offset the increase in $S_a(\mathcal{G}_j)$ by simultaneously moving $v' \in \mathcal{G}_j$ from $\mathcal{G}_j$ to $\mathcal{G}_i$. After swapping v and v', the subgraphs' total edge weights are

$$W(\mathcal{G}_i') = W(\mathcal{G}_i) - \Delta W(\mathcal{G}_i, v) + \Delta W(\mathcal{G}_i, v') - w((v, v')) \tag{14}$$

$$W(\mathcal{G}_j') = W(\mathcal{G}_j) + \Delta W(\mathcal{G}_j, v) - \Delta W(\mathcal{G}_j, v') - w((v, v')) \tag{15}$$

and their new sizes can be computed by (11). There are n_i potential swaps and $n_i n_j$ potential swaps so each exchange is $O(n^2)$.

Our improvement heuristic (Algorithm 1) repeatedly searches for transfers or swaps of vertices between pairs of subgraphs. In each iteration of the main loop (lines 1–13), it consider either transfers (lines 2–6) or swaps (lines 8–12) between a pair of subgraphs. There are fewer potential transfers than swaps, so the heuristic prioritizes searching for transfers. We keep track of which pairs of subgraphs have been checked (line 13) and only recheck a pair if one of the subgraphs has changed.

Algorithm 1: Improve partition

Input: Partition $\mathcal{P} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$
Output: Partition $\mathcal{P}'$ with $C_a(\mathcal{P}') \leq C_a(\mathcal{P})$

```
1  while There are unchecked pairs do
2      (G1, G2) ← Pair with unchecked transfers
3      if (G1, G2) exists then
4          v* ← Best transfer from G1 to G2
5          if v* exists then
6              Transfer v* from G1 to G2
7      else
8          (G1, G2) ← Pair with unchecked swaps
9          if (G1, G2) exists then
10             (v1*, v2*) ← Best swap between G1, G2
11             if (v1*, v2*) exists then
12                 Swap v1* and v2*
13     Update subgraph sizes and checked pairs
14 return P' = {G1, ..., Gm}
```

The main loop of Algorithm 1 starts by selecting an unchecked pair of subgraphs to check for transfers (line 2) or swaps (line 8) between. Next, it searches for a transfer (line 4) or swap (line 10) which reduces $\max\{S_a(\mathcal{G}_i), S_a(\mathcal{G}_j)\}$ for this pair. We can compute the effect of a potential move on $S_a(\mathcal{G}_i)$ and $S_a(\mathcal{G}_j)$ in constant time using (11)–(14). If a move which reduces $\max\{S_a(\mathcal{G}_i), S_a(\mathcal{G}_j)\}$ is found, the vertex is transferred (line 6) or the pair of vertices is swapped (line 12) and $S_a(\mathcal{G}_i)$ and $S_a(\mathcal{G}_j)$ are updated (line 13). Once transfers swaps have been checked for all pairs, the main loop terminates and the current partition, which is a local minimum, is returned (line 14).

We are able to check the size of subgraphs after a potential transfer or swap using (11)–(14) if the marginal weights $\Delta W(\mathcal{G}_i, v)$ are known for all $\mathcal{G}_i \in \mathcal{P}$ and $v \in \mathcal{V}$. These variables can be initially computed in $O(n^2)$ using (10). After a swap or transfer is performed, $\mathcal{G}_i$ and $\mathcal{G}_j$ change so $\Delta W(\mathcal{G}_i, v)$ and $\Delta W(\mathcal{G}_j, v)$ must be updated. If v' is transfered from $\mathcal{G}_i$ to $\mathcal{G}_j$, then

$$\Delta W(\mathcal{G}_i', v) = \Delta W(\mathcal{G}_i, v) - w((v, v'))$$
$$\Delta W(\mathcal{G}_j', v) = \Delta W(\mathcal{G}_j, v) + w((v, v'))$$

for all $v \in \mathcal{V}$. Using these equations, each marginal weight is updated in $O(1)$ and all the marginal weights are updated in $O(n)$ in line 13 of Algorithm 1. By storing the nm marginal sizes, we can compute the subgraph sizes after a potential move in $O(1)$ instead of $O(n)$ and find the best move in $O(n^2)$ instead of $O(n^3)$.

4.2 Transfer of outliers

The definitions (9) and (10) can be rearranged as

$$W(\mathcal{G}_i) = \frac{1}{2} \sum_{v \in \mathcal{V}_i} \Delta W(\mathcal{G}_i, v). \tag{16}$$

This identity tells us that $\Delta W(\mathcal{G}_i, v)$ is proportional to vertex v's *contribution* to $S_a(\mathcal{G}_i)$. An effective way to improve the partition, therefore, is to search for vertices $v \in \mathcal{V}_i$ with a large $\Delta W(\mathcal{G}_i, v)$ but a small $\Delta W(\mathcal{G}_j, v)$ for some other graph $\mathcal{G}_j$. Such a vertex, which we call *out-of-place*, would have a smaller contribution to the size of $\mathcal{G}_j$ than it is currently having to the size of $\mathcal{G}_i$.

Although Algorithm 1 tends to move out-of-place vertices to more appropriate subgraphs, it only performs moves which decrease $\max\{S_a(\mathcal{G}_i), S_a(\mathcal{G}_j)\}$. When two subgraphs already have a similar size, transferring an out-of-place vertex would often violate this constraint. In the second phase of the heuristic (Algorithm 2) we allow some violation of this constraint when moving *outlier* vertices to better subgraphs. A vertex $v \in \mathcal{V}_i$ is an outlier if $\Delta W(\mathcal{G}_i, v) > \Delta W(\mathcal{G}_j, v)$ for some $\mathcal{G}_j$ and

$$\Delta W(\mathcal{G}_i, v) > \alpha \sum_{v' \in \mathcal{V}_i} \frac{1}{n_i} \Delta W(\mathcal{G}_i, v') = \alpha \frac{2}{n_i} W(\mathcal{G}_i).$$

This second criterion is that v contributes more to $S_a(\mathcal{G}_i)$ than an average vertex of $\mathcal{G}_i$. The outlier detection threshold, $\alpha \geq 1$, is used to control the number outliers detected which decreases as α increases. We found that $\alpha = 1.5$ gave good results.

Algorithm 2 begins by identifying all outliers (lines 2–9). For each subgraph, it checks if each vertex's contribution is above the detection threshold (line 4) and if it is, checks if the vertex is out-of-place (line 5). Every outlier, along with its current subgraph and the subgraph it fits best in, is added to a list (lines 7–9). After identifying all outliers, they are transferred to the subgraphs they fit best in (line 11) and $W(\mathcal{G}_i)$ and $\Delta W(\mathcal{G}_i, v)$ are updated to reflect this transfer (line 12).

Algorithm 2: Transfer outliers

Input: Partition $\mathcal{P}$, threshold $\alpha \geq 1$
Output: Partition $\mathcal{P}'$ with outliers transferred

```
1  U ← {}                                   /* Set of outliers */
2  for Gi ∈ P do
3      for v ∈ Gi do
4          if ΔW(Gi, v) > α (2/ni) W(Gi) then
5              Gj ← minimizer of ΔW(Gi', v)
6              if Gj ≠ Gi then
7                  U ← U ∪ {v}
8                  Gold(v) ← Gi
9                  Gnew(v) ← Gj
10 for v ∈ U do
11     Transfer v from Gold(v) to Gnew(v)
12     Update W and ΔW for Gold, Gnew, and all v' ∈ V
13 P' ← (G1, ..., Gm)
14 return P'
```

4.3 Overall partition algorithm

Transferring outliers using Algorithm 2 and the transfers and swaps of Algorithm 1 are two effective ways to improve a partition. Alternating between these two algorithms is the basis of our main heuristic for the AHPP (Algorithm 3).

Algorithm 3 starts with a randomly generated partition (line 1) and improves it by alternating between Algorithms 1 and 2. It computes $W(\mathcal{G}_i)$ and $\Delta W(\mathcal{G}_i, v)$ for this partition (line 2) using their

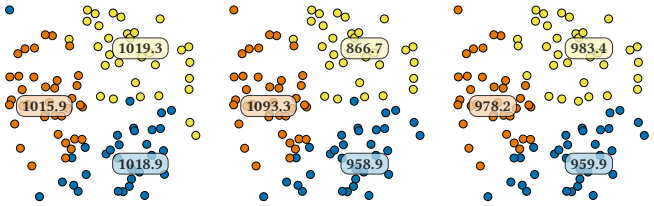

Figure 3: Partition of a graph after one round improvements (left), after transferring outliers with $\alpha = 1.5$ (center), and the final partition produced by AHP-mTSP (right). The numbers overlayed on each subgraph are the average cycle length of that subgraph using Euclidean edge weights.

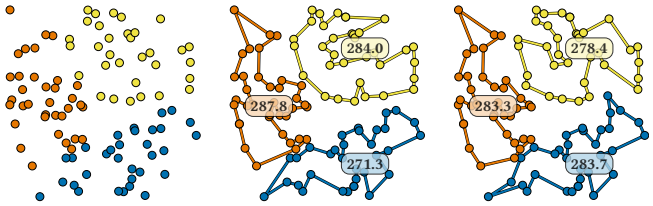

Figure 4: Completed graph partition (left), 1-TSP solutions on each subgraph (center), and m-TSP solution resulting from Algorithm 4 after three transfers (right).

definitions (9) and (10) and then improves the partition as much as possible using transfers and swaps (line 3). In each round of the main loop (lines 4–9), outliers are transferred (line 6) and the resulting partition is improved (line 7). When outliers are transferred, $\Delta W(\mathcal{G}_i, v)$ changes if v has been transferred or $\mathcal{G}_i$ has had at least one vertex transferred to/from it. This phase (line 6) is therefore not guaranteed to improve $C_a(\mathcal{P})$ so it is always followed immediately by a partition improvement phase (line 7). If these two phases improve the partition, we keep the improved partition (line 9); otherwise, the algorithm returns the partition from before the outliers were transferred (line 10). In this way, Algorithm 3 never returns a worse partition as a result of transferring outliers.

Algorithm 3: Average Hamiltonian Partition (AHP)

Input: Complete graph $\mathcal{G}$, number of agents m
Output: Near-optimal solution, $\mathcal{P}$, to the AHPP

```
1  P ← Random partition of G
2  Compute W(G_i), ΔW(G_i, v), for all G_i ∈ P and v ∈ V
3  Improve P by transfers/swaps          /* Algorithm 1 */
4  while Improvements decreased C_a(P) do
5      P' ← P
6      Transfer outliers of P'           /* Algorithm 2 */
7      Improve P' by transfers/swaps     /* Algorithm 1 */
8      if C_a(P') < C_a(P) then
9          P ← P'
10 return P
```

Figure 3 shows how transferring outliers affects a partition. The top lefts blue vertex is an outlier and is detected by Algorithm 2 and transferred to the orange subgraph. Two yellow outliers in the bottom right are also transferred to the blue subgraph. After transferring these outliers, the $C_a(\mathcal{P})$ from 1019.3 to 1093.3 but after another improvement phase $C_a(\mathcal{P}) = 983.4$ which is a 3.6% improvement on the original cost of 1019.3.

5 FROM A PARTITION TO CYCLES

By Corollary 1, if $S_m(\mathcal{G}_i) = f(S_a(\mathcal{G}_i))$ for all $\mathcal{G}_i$, then the solutions to the AHPP and MHPP are identical. This solution solves the task allocation problem and then the routing problem can be solved by solving the 1-TSP on each subgraph of the partition. In reality, the relationship between S_a and S_m contains some noise (Figure 1), so the optimal partitions for the AHPP and MHPP differ slightly. The solution to the AHPP is still useful as an initial partition for an improved m-TSP algorithm (Algorithm 4). The initial m-TSP strategy is obtained by solving the 1-TSP on each subgraph of the partition. The strategy is improved by transferring vertices between cycles to reduce its minmax cost. The best transfer can be found in $O(n^2)$ by checking all pairs of vertices in the longer cycles and locations for insertion in the shorter cycle. The algorithm alternates between transferring vertices between cycles and solving the 1-TSP for each cycle until no more improvements can be made.

Algorithm 4: AHP-mTSP

Input: Complete graph $\mathcal{G}$, number of agents m
Output: Set of m cycles solving the minmax m-TSP

```
1  P ← Minmax partition of G obtained by AHP
2  while Improvements possible do
3      for G_i in P do
4          p_i ← 1-TSP optimal cycle on G_i
5      S ← {p_1, ..., p_m}
6      Improve S by transferring vertices between cycle
7      P ← partition induced by S
8  return S
```

Algorithm 4 involves solving m instances of the 1-TSP. Although the 1-TSP is NP-hard, several open-source solvers [7, 15] have very good performance and runtimes. Furthermore, we are solving m instances of the 1-TSP with $n_1, \ldots n_m$ vertices each such that $n_1 + \ldots + n_m = n$ instead of a single instance with n vertices. Solving these m smaller instances is faster than solving the single large instance because the runtime of TSP solvers is slower than linear in the number of vertices.

In Figure 4, we find cycles using the partition from Figure 3. The initial cycles have similar lengths so Algorithm 4 only made three transfers to reach its final solution. These transfers reduced the strategy's cost from 287.8 to 283.7, a decrease of 0.15%.

6 DECENTRALIZATION

Although we have presented a centralized versions of AHP-mTSP, it is straightforward to decentralize it by decentralizing each of its component algorithms. Algorithm 1 consists of exchanges (transfers in lines 4–6 or swaps in lines 10–12) of vertices between pairs

of subgraphs. As these exchanges only involve 2 agents' graphs, multiple pairs of agents can compute exchanges simultaneously resulting in a decentralized version of Algorithm 1 which is faster than the centralized version if there are 4 or more agents. Algorithm 2 consists of a search for outliers in each graph which doesn't modify any of the graphs (lines 2–9) and then a transfer of these outliers after they have all been identified (lines 10–12). It can be decentralized by having each agent identify outliers in its own graph followed by pairwise communication with other agents to transfer the outliers that were identified. The decentralized versions of Algorithms 1 and 2 result in a decentralized version of Algorithm 3. Algorithm 4 consists of an initial partition by Algorithm 3, solution of the 1-TSP on each subgraph (lines 3–4), and transfers of vertices between pairs of paths (line 6) which can be decentralized like the transfers in Algorithm 1. The solutions to the m instances of the 1-TSP are independent and can be decentralized by each agent solving the 1-TSP on its own subgraph resulting in a decentralized Algorithm 4. Some synchronization is also required to ensure the agents progress through the algorithm at the same rate.

7 PATHS WITH DEPOTS

An agent's path may be constrained to start or end at specific depot vertex. We classify an agent's path into one of five types depending on the constraints:

(1) Cycle with 1 depot: $v_{\text{start}} = v_{\text{end}} = v_{\text{depot}}$
(2) Cycle with 0 depots: $v_{\text{start}} = v_{\text{end}}$
(3) Path with 2 depots: $v_{\text{start}} = v_{\text{depot}} \neq v'_{\text{depot}} = v_{\text{end}}$
(4) Path with 1 depot: $v_{\text{start}} = v_{\text{depot}}$
(5) Path with 0 depots: No constraints

In all of these cases v_{depot} is a specified vertex and v_{start} and v_{end} are the first and last vertices of the agent's path. Another important distinction is whether a depot is unique to one agent or shared by multiple agents.

Problems with different depot constraints can be solved using Algorithm 3 with slight modifications to Algorithms 1–4. In the initial partition, each subgraph must contain its agent's depots and these depots cannot be transferred or swapped in Algorithms 1 or 2. For open paths, $S_{\text{a}}(\mathcal{G}_i) = {}^{2}/_{n_i} \sum_{e \in \mathcal{E}_i} w(e)$ as open paths only contain $n_i - 1$ edges. Once a partition is found, the 1-TSP is solved with the relevant depot constraint. When the paths are being improved by Algorithm 4, the depots again cannot be transferred. This approach can be used to generate solutions to the various types of minmax m-TSPs (Figure 5).

8 RESULTS

We compared our algorithm against two state-of-the-art algorithms for problems with $50 \leq n \leq 5000$ and $3 \leq m \leq 100$ and different depot configurations. Our algorithm was implemented in python and solutions were computed with $\alpha = 1.5$ using a Linux desktop computer with a 3.40 GHz processor and 8 GB of memory.

8.1 Problems with multiple depots

We compared our algorithm with the hierarchical market-based solution (HMS) from Kivelevitch *et al.* [20] for cycles with unique depots for each agent. They considered $n = 5000$ vertices on the square $[0, 100] \times [0, 100]$ with $m \in \{10, 100\}$ agents. As they did not publish the exact locations of the vertices in their instances, we randomly generated a new set of 5000 vertices from the same distribution for each of our 20 tests. Our results show an average improvement of approximately 10% and had a worst case with lower cost than their result for both 10 and 100 agents (Table 1). Furthermore, our solutions required less computation time. The best solutions we found for 10 agents is shown in Figure 6.

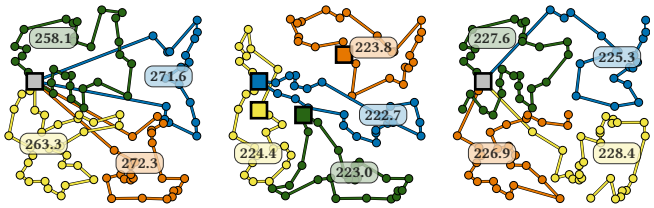

Figure 5: Minmax m-TSP solutions for cycles with one shared depot (left), cycles with unique depots (center), and paths with one depot (right). Colored/grey squares are depots for one/multiple agent(s).

Table 1: Comparison of minmax cost, C, and run-times, t, achieved by HMS [20] with AHP-mTSP. Results for HMS are from the single solutions computed in [20] for 5000 vertices uniformly distributed on $[0, 100] \times [0, 100]$ with 10 or 100 agents. Results for AHP-mTSP are based on 20 solutions with different random seeds.

		HMS		AHP-mTSP		
n	m	C	t	C_{min}	C_{avg}	t_{avg}
5000	10	577	12000	**513.66**	**516.56**	**6199.36**
5000	100	64.738	76477	**55.65**	**56.73**	**4786.58**

8.2 Problems with one depot

For the minmax m-TSP with one shared depot, we compared our approach with the best results found by any of the 6 heuristics that Wang *et al.* compared (Table 2). They computed cyclic solutions for several problems from TSPLIB [34] using the first vertex as a shared depot for all agents. As the number of solutions for the minmax m-TSP increases exponentially with both n and m, a heuristics performance can be best evaluated by its performance on large problems. For the largest problem, with $n = 1173$, our heuristic produced better solutions with lower minmax costs for both the best solution found and average solution cost for 3, 5, 10, and 20 agents. The best solutions we found for the largest problem, pcb1173, are shown in Figure 7.

Our problem had average run-times ranging from less than 1 s to 426 s. For the largest problem ($n = 1173$), our algorithm took between 146 s and 426 s which is the same order of magnitude as the 236 s used by Wang *et al.* [44]. However, as the problems were run on different computers, we cannot make more detailed comparisons.

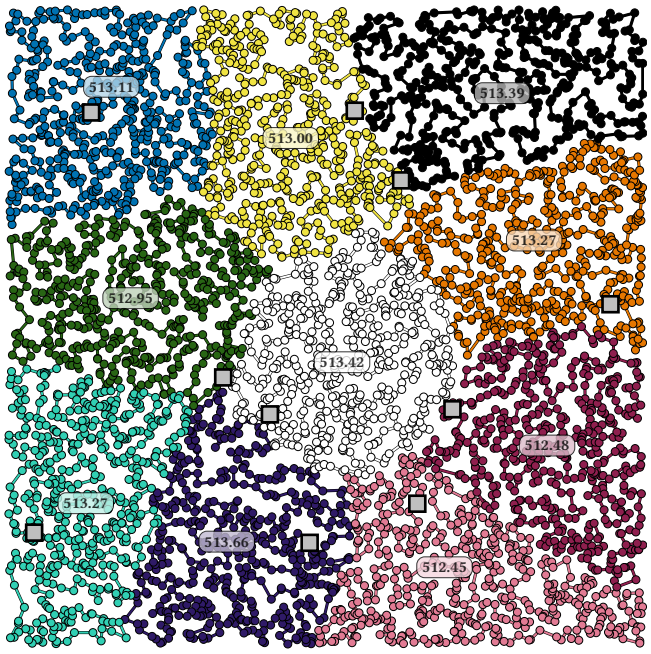

Figure 6: The best solution found for 5000 uniformly distributed vertices with 10 agents and 10 depots.

Table 2: Comparison of Minmax m-TSP costs for pcb1173 [34] obtained using invasive weed optimization (IWO) and a memetic algorithm (MA) [44] with AHP-mTSP. Results are based on 20 solutions with different random seeds. $n = 1173$

	IWO		MA		AHP-mTSP	
m	$C_{\min}$	C_{avg}	$C_{\min}$	C_{avg}	$C_{\min}$	C_{avg}
3	24008.5	24300.3	22443.2	22781.6	**20733.3**	**20999.2**
5	16057.2	16274.6	14557.3	14861.4	**13876.3**	**14179.2**
10	10517.9	10668.0	9222.9	9352.6	**8698.4**	**8871.3**
20	8063.2	8207.9	7063.2	7276.7	**6595.9**	**6670.2**

8.3 Run-time analysis

We analyzed average runtimes for 31 test problems from TSPLIB [34]. These problems have $n \in \{51, 100, 150, 200, 318, 532, 783, 1173\}$ and $m \in \{2, 3, 10, 20\}$. The problems with $n = 1173$ are the same problems as in Table 2. The run-times are averaged over 40 trials of each problem. We assumed the run-time follows a monomial model $t_{\text{avg}} = k_0 n^{k_1} m^{k_2} e^{\epsilon}$ where k_0, k_1, and k_2 are parameters to be estimated and e^{ϵ} captures the uncertainty of the problem. By taking the logarithm of both sides of this equation, we obtain an equation which is linear in the parameters

$$\log(t_{\text{avg}}) = \log(k_0) + k_1 \log(n) + k_2 \log(m) + \epsilon.$$

We estimated k_1 and k_2 by linear regression and obtained the model

$$\widehat{t}_{\text{avg}} = (3.1944 \times 10^{-5}\ \text{s}) n^{2.111} m^{0.325}.$$

The estimates produced using this overall model (Figure 8) are close to the actual average run-times.

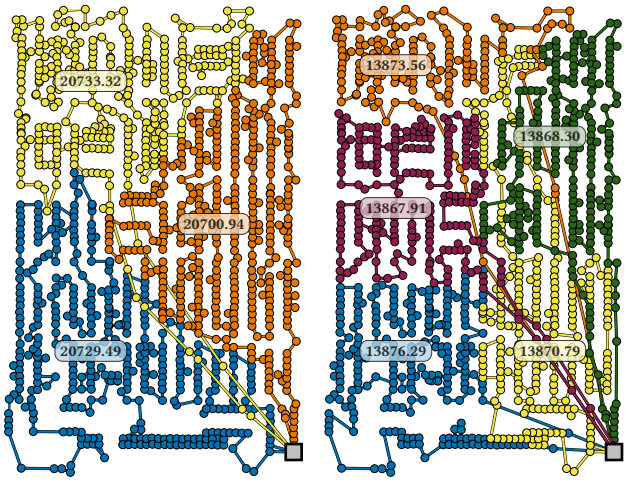

Figure 7: Best solutions found for pcb1173 with 3 (left), 5 (right) agents. The gray square represents the depot. The numbers on top of the cycles are their lengths.

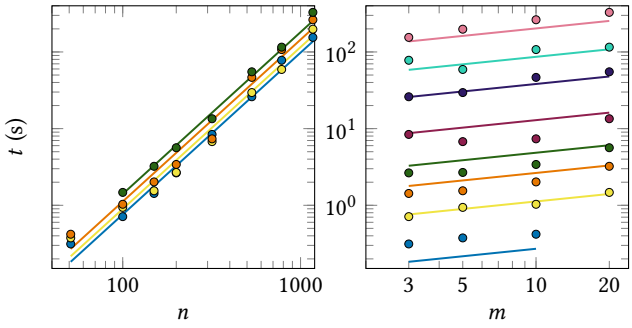

Figure 8: Actual average run-times (dots) and predicted average run-times (lines) for varying n with fixed m (left) and varying m with fixed n (right).

9 CONCLUSIONS

Task assignment and routing are coupled problems for teams of mobile agents. We formulated these combined problems as a partition problem, the MHPP, which is equivalent to the minmax m-TSP. As these problems are NP-hard, we developed a heurstic algorithm, AHP-mTSP, for the combined task assignment and routing problem. It is based on a graph partition heuristic for the AHPP which is a similar problem to the MHPP. Despite the MHPP's simpler cost function, there is a close relationship between the solutions of the two problems. AHP-mTSP uses a solution to the AHPP and computes routes by solving the 1-TSP. The routes are improved slightly by transferring vertices between them resulting in a set of cycles which minimizes the length of the longest cycle. These cycles solve the combined task allocation and routing problems. Using this approach, we solved large task allocation problems and obtained better solutions than have previously been reported using a variety of algorithms. For n tasks and m agents, the algorithm's runtime was proportional to $n^{2.111} m^{0.325}$.

REFERENCES

[1] Raulcézar Alves and Carlos Lopes. 2015. Using genetic algorithms to minimize the distance and balance the routes for the multiple traveling salesman problem. In *Congress on Evolutionary Computation (CEC)*. IEEE, 3171–3178.

[2] Muhammad Arif and Sajjad Haider. 2017. An Evolutionary Traveling Salesman Approach for Multi-Robot Task Allocation.. In *International Conference on Agents and Artificial Intelligence (ICAART)*. IEEE, 567–574.

[3] John Beasley. 1983. Route first cluster second methods for vehicle routing. *Omega* 11, 4 (1983), 403–408.

[4] Rubén Bolaños, Mauricio Echeverry, and John Escobar. 2015. A multiobjective non-dominated sorting genetic algorithm (NSGA-II) for the Multiple Traveling Salesman Problem. *Decision Science Letters* 4, 4 (2015), 559–568.

[5] Han Choi, Luc Brunet, and Jonathan How. 2009. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics* 25, 4 (2009), 912–926.

[6] Nicos Christofides. 1976. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report. DTIC Document.

[7] William Cook. 2016. Concorde TSP Solver. (2016). http://www.math.uwaterloo.ca/tsp/concorde/

[8] Bernardine Dias. 2004. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments.* Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

[9] Alessandro Farinelli, Luca Iocchi, Daniele Nardi, and Vittorio Ziparo. 2006. Assignment of dynamically perceived tasks by token passing in multirobot systems. *Proc. IEEE* 94, 7 (2006), 1271–1288.

[10] Greg Frederickson, Matthew Hecht, and Chul Kim. 1976. Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 216–227.

[11] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA.

[12] Brian Gerkey and Maja Matarić. 2002. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation* 18, 5 (2002), 758–768.

[13] Brian Gerkey and Maja Matarić. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23, 9 (2004), 939–954.

[14] Samuel Gorenstein. 1970. Printing press scheduling for multi-edition periodicals. *Management Science* 16, 6 (1970), B–373.

[15] Keld Helsgaun. [n. d.]. LKH Version 2.0.7. ([n. d.]). http://www.akira.ruc.dk/~keld/research/LKH/

[16] Keld Helsgaun. 2000. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126, 1 (2000), 106–130.

[17] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph Durham, and Nora Ayanian. 2018. Conflict-Based Search with Optimal Task Assignment. In *17th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 757–765.

[18] Daniel Karapetyan and Gregory Gutin. 2011. Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research* 208, 3 (2011), 221–232.

[19] Elad Kivelevitch, Kelly Cohen, and Manish Kumar. 2011. Market-based solution to the allocation of tasks to agents. *Procedia Computer Science* 6 (2011), 28–33.

[20] Elad Kivelevitch, Balaji Sharma, Nicholas Ernest, Manish Kumar, and Kelly Cohen. 2014. A hierarchical market solution to the min-max multiple depots vehicle routing problem. *Unmanned Systems* 2 (2014), 87–100.

[21] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. 2013. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research* 32, 12 (2013), 1495–1512.

[22] Gilbert Laporte. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59, 2 (1992), 231–247.

[23] Shen Lin and Brian Kernighan. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21, 2 (1973), 498–516.

[24] Weimin Liu, Sujian Li, Fanggeng Zhao, and Aiyun Zheng. 2009. An ant colony optimization algorithm for the multiple traveling salesmen problem. In *4th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, 1533–1537.

[25] Manuel López-Ibáñez, Christian Blum, Jeffrey Ohlmann, and Barrett Thomas. 2013. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing* 13, 9 (2013), 3806–3815.

[26] Hang Ma and Sven Koenig. 2016. Optimal target assignment and path finding for teams of agents. In *15th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 1144–1152.

[27] Mitchell McIntire, Ernesto Nunes, and Maria Gini. 2016. Iterated multi-robot auctions for precedence-constrained task scheduling. In *15th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 1078–1086.

[28] Byungsoo Na. 2006. *Heuristic approaches for no-depot k-traveling salesmen problem with a minmax objective.* Ph.D. Dissertation. Texas A&M University, College Station, Texas.

[29] R. Nallusamy, K. Duraiswamy, R. Dhanalaksmi, and P. Parthiban. 2010. Optimization of non-linear multiple traveling salesman problem using k-means clustering, shrink wrap algorithm and meta-heuristics. *International Journal of Nonlinear Science* 9, 2 (2010), 171–177.

[30] Maitreyi Nanjanath and Maria Gini. 2010. Repeated auctions for robust task execution by a robot team. *Robotics and Autonomous Systems* 58, 7 (2010), 900–909.

[31] Nhan Nguyen, Trung Nguyen, and Jörg Rothe. 2017. Approximate solutions to max-min fair and proportionally fair allocations of indivisible goods. In *16th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 262–271.

[32] Joaquín Pacheco and Rafael Martí. 2006. Tabu search for a multi-objective routing problem. *Journal of the Operational Research Society* 57, 1 (2006), 29–37.

[33] Christos Papadimitriou. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4, 3 (1977), 237–244.

[34] Gerhard Reinelt. 2015. TSPLIB. (2015). http://elib.zib.de/pub/mp-testdata/tsp/tsplib/

[35] Alessandro Riva, Jacopo Banfi, Carlo Fanton, Nicola Basilico, and Francesco Amigoni. 2018. A Journey Among Pairs of Vertices: Computing Robots' Paths for Performing Joint Measurements. In *17th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 229–237.

[36] Sanem Sariel and Tucker Balch. 2005. Real time auction based allocation of tasks for multi-robot exploration problem in dynamic environments. In *Workshop on Integrating Planning into Scheduling*. AAAI, 27–33.

[37] Sebastian Schneckenburger, Britta Dorn, and Ulle Endriss. 2017. The Atkinson inequality index in multiagent resource allocation. In *16th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 272–280.

[38] Vui Ann Shim, Kay Chen Tan, and Chun Yew Cheong. 2012. A hybrid estimation of distribution algorithm with decomposition for solving the multiobjective multiple traveling salesman problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 5 (2012), 682–691.

[39] Banu Soylu. 2015. A general variable neighborhood search heuristic for multiple traveling salesmen problem. *Computers & Industrial Engineering* 90 (2015), 390–401.

[40] Johan Thunberg, David Anisi, and Petter Ögren. 2009. A comparative study of task assignment and path planning methods for multi-UGV missions. In *Optimization and Cooperative Control Strategies*. Springer, 167–180.

[41] Pradeep Varakantham, Hala Mostafa, Na Fu, and Hoong Lau. 2015. DIRECT: A scalable approach for route guidance in selfish orienteering problems. In *14th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 483–491.

[42] Pandiri Venkatesh and Alok Singh. 2015. Two metaheuristic approaches for the multiple traveling salesperson problem. *Applied Soft Computing* 26 (2015), 74–89.

[43] Jiří Vokřínek, Antonín Komenda, and Michal Pěchouček. 2010. Agents towards vehicle routing problems. In *9th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 773–780.

[44] Yongzhen Wang, Yan Chen, and Yan Lin. 2017. Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem. *Computers & Industrial Engineering* 106 (2017), 105–122.

[45] Drew Wicke, David Freelan, and Sean Luke. 2015. Bounty hunters and multiagent task allocation. In *14th International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 387–394.

[46] Sukmin Yoon and Jinwhan Kim. 2017. Efficient multi-agent task allocation for collaborative route planning with multiple unmanned vehicles. *IFAC-PapersOnLine* 50, 1 (2017), 3580–3585.

[47] Robert Zlot and Anthony Stentz. 2006. Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research* 25, 1 (2006), 73–101.