A Hierarchical Market Solution to the Min–Max Multiple Depots Vehicle Routing Problem

Article · January 2014

DOI: 10.1142/S230138501450006X

5 authors, including:

Elad Kivelevitch
University of Cincinnati
28 PUBLICATIONS **68** CITATIONS
SEE PROFILE

Nicholas Ernest
Psibernetix Inc.
14 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Manish Kumar
University of Cincinnati
88 PUBLICATIONS **310** CITATIONS
SEE PROFILE

Kelly Cohen
University of Cincinnati
207 PUBLICATIONS **994** CITATIONS
SEE PROFILE

Unmanned Systems, Vol. 2, No. 1 (2014) 87–100

DOI: 10.1142/S230138501450006X

UNMANNED SYSTEMS
http://www.worldscientific.com/worldscinet/us

A Hierarchical Market Solution to the Min–Max Multiple Depots Vehicle Routing Problem

Elad Kivelevitch*,§, Balaji Sharma†, Nicholas Ernest*, Manish Kumar‡, Kelly Cohen*

*School of Aerospace Systems, University of Cincinnati, 2600 Clifton Ave., Cincinnati, Ohio 45208, USA

†The MathWorks, Inc., 3 Apple Hill Dr., Natick, Massachusetts 01760, USA

‡Department of Mechanical, Industrial and Manufacturing Engineering, University of Toledo, 2801 W. Bancroft, Toledo, Ohio 43606, USA

The problem of assigning a group of Unmanned Aerial Vehicles (UAVs) to perform spatially distributed tasks often requires that the tasks will be performed as quickly as possible. This problem can be defined as the Min–Max Multiple Depots Vehicle Routing Problem (MMMDVRP), which is a benchmark combinatorial optimization problem. In this problem, UAVs are assigned to service tasks so that each task is serviced once and the goal is to minimize the longest tour performed by any UAV in its motion from its initial location (*depot*) to the tasks and back to the depot. This problem arises in many time-critical applications, e.g. mobile targets assigned to UAVs in a military context, wildfire fighting, and disaster relief efforts in civilian applications. In this work, we formulate the problem using Mixed Integer Linear Programming (MILP) and Binary Programming and show the scalability limitation of these formulations. To improve scalability, we propose a hierarchical market-based solution (MBS). Simulation results demonstrate the ability of the MBS to solve large scale problems and obtain better costs compared with other known heuristic solution.

Keywords: Task assignment; multiple traveling salesmen problem; vehicle routing problem; multiagent methods; min–max problems; binary programming; mixed integer linear programming.

US

1. Introduction

With the increasing use of Unmanned Aerial Vehicles (UAVs) for military and civilian applications, there is a need to assign groups of UAVs to perform spatially distributed tasks and returning to some default state. Generally speaking, there are several possible goals for such an assignment. Some examples include: minimizing the total cost of fuel consumed by the group, minimizing the total distance traveled by the UAVs on their mission, or performing all the tasks as quickly as possible. In this paper, we consider the latter problem and we formulate it as the Minimum–Maximum (Min–Max) Multiple Depots Vehicle Routing Problem (MMMDVRP).

The Multiple Depots Vehicle Routing Problem (MDVRP) is an extension of the famous Traveling Salesman Problem (TSP). In the MDVRP, there are multiple UAVs (traveling salesmen in the TSP notation or vehicles in the MDVRP) that originate at several initial locations, called *depots*, and the goal is to visit a set of tasks and return to the depots. Traditionally, the goal of the MDVRP was to minimize the total distance traveled by the UAVs; however, formulating the cost function this way can lead to assignments that require long travel distances by a few UAVs, which may result in a long time to complete all the tasks.

In some cases, this is an unacceptable assignment, e.g. engaging mobile targets in military applications [1], fighting rapidly moving forest fires [2], or providing assistance in relief efforts [3]. Consequently, the MMMDVRP, in which the goal is to minimize the longest tour by any UAV in its mission, was proposed.

In this work, we consider the MMMDVRP as defined by Carlsson *et al.* [4], and we put no constraints on the capacities

Received 7 March 2013; Revised 19 December 2013; Accepted 19 December 2013; Published 20 January 2014. This paper was recommended for publication in its revised form by editorial board member, Han-Lim Choi.
Email Address: §*elad.kivelevitch@uc.edu*

of the vehicles or requirements of the individual tasks. Therefore, the search space for the optimization process is less constrained.

Solutions of the standard and min–max MTSP and VRP based on mathematical optimization methods exist and are commonly solved by commercial Linear Programming solvers. By formulating and solving the problem as Mixed Integer Linear Programming (MILP) problem, it is possible to obtain solutions that are considered to be optimal. In this work, we also introduce a formulation of the problem as a pure Binary Programming problem with Binary Search (BPBS), and show that it is possible to solve this formulation using a commercial LP solver for relatively small instances. Unfortunately, both the MILP and BPBS formulations do not scale very well.

Larger scenarios of the MMMDVRP are often solved using heuristic algorithms that perform two stages: first, the set of tasks is partitioned between the traveling salesmen, then each vehicle performs an optimization to find the best TSP solution for the subset of tasks assigned to it. Furthermore, for multi-robot systems, decentralized cooperative decision making is often preferred because of its potential to reduce the computing resources required by any single processor, reduced communication bandwidth, and ability to overcome the loss of a single robot [5].

In this work, we will describe a solution based on a hierarchical economic market, in which multiple agents interact in a simulated economic market. This work is an improvement of our market-based solution (MBS) [6]. We compare the results of our hierarchical market with the results of existing scalable solutions, namely, Carlsson *et al.* [4], Narasimha *et al.* [2] and Ernest and Cohen [7, 8].

This paper extends our previous work in two ways. First, we present a Binary Programming formulation with Binary Search for the MMMDVRP, which, to the best of our knowledge, is the first such formulation. We believe that this formulation can, in some cases, be very effective in finding optimal solutions using commercial mathematical optimization solvers that can utilize a Binary Programming formulation to run faster than a MILP formulation. The second new aspect of this work pertains to the MBS we developed [9]. In this work, we add hierarchy to the market, which improves the ability of our overall solution to handle problems with large numbers of vehicles and tasks.

The paper is organized as follows: in the next section we review existing work specific to min–max problems. Then, in Sec. 3 we introduce the mathematical formulation of the problem using both MILP and Binary Programming and discuss the limitations of such formulation. In Sec. 4, we describe our basic market and the hierarchical market. A comparison of the results of the hierarchical market to the other methods is given in Sec. 5, followed by conclusions and suggestions for future work.

2. Literature

The TSP, MTSP and VRP have been researched extensively in the past decades, and excellent reviews of these works can be found in Applegate *et al.* [10], Bektas [11] and Feillet *et al.* [12]. For the sake of brevity, this section focuses on works specifically tailored to the min–max variant of these problems.

One of the first attempts at the general class of Min–Max problems was made by Golden *et al.* for the single depot cases [13]. They proposed a method based on a tabu search and adaptive memory procedure. Using several classic test cases, which originally had been produced for testing the single depot classical VRP [14, 15], Golden *et al.* showed that their algorithm produced results that were near a theoretical lower bound. A comparison of our work for these cases was given in [16].

Ren introduced a series of heuristic solutions for the min–max problem based on genetic algorithms, improved insertion schemes and tabu search [17, 18]. While this work may be used for multiple depots, solutions were tested for a problem with a single depot, six vehicles and 20 tasks, and all showed the ability to obtain the optimal solution, but with different average cost for the other vehicles.

Carlsson *et al.* [4] formulated the MMMDVRP and proposed a solution based on partitioning the problem into equal areas and solving the resulting single TSP for each vehicle and subset of tasks. Carlsson *et al.* assumed that the tasks are distributed randomly with uniform probability in the region, which allows them to partition the region into equal area sub-regions, which, given enough tasks, will yield tours of equal lengths, and thus minimize the longest tour by any vehicle. They also prove that the optimal solution is within lower and upper bounds, but these bounds are not necessarily tight. Nonetheless, their algorithm is particulary fast [6] and serves as a benchmark for research in the area.

Campbell *et al.* addressed the problem for emergency relief efforts application [3], where time is of the essence and therefore the min–max solution is required. They used a mixed integer formulation given by [19] to formulate the min–max problem but found that commercial solvers struggle to solve such a formulation, because the optimization problem is not formulated in a standard way and the cost function is introduced as a constraint on the time of the longest tour. Instead, Campbell *et al.* [3] introduced a heuristic solution for the insertion scheme and improvement scheme, while ascertaining that it was nontrivial to modify existing heuristics in a way that would be both computationally efficient and would result in a significant improvement to the cost. Based on the above literature survey, it is apparent that there is a need for a computationally efficient heuristic that will yield good results for this problem.

Narasimha *et al.* [2, 20] extended their single depot ant-colony solution of the min–max VRP to the multiple depots case. Similar to Carlsson's solution [4], partitioning was done based on the assumption that the tasks were uniformly distributed in the region, however in this case the partitioning was done between the depots, not individual vehicles. Then, their previously developed single depot ant-colony solution was applied to the tasks and vehicles in each subset of tasks assigned to a depot. Their results improved over Carlsson's algorithm in terms of cost, but required a significant computational effort.

Ernest and Cohen presented a fuzzy-heuristic approximating method for the MDVRP [7, 8]. Their solution seeks to minimize computational cost even for extremely large scale cases while maintaining solution integrity by solving the clustering portion of the problem without analyzing any of the vehicle's actual paths. Their method first performs fuzzy clustering, based on variables such as the area of the convex hull of a cluster, the number of targets within the cluster, the number of targets per unit area (target density), and the distance between polygon mass center and its cluster's home depot. Given an initial nearest neighbor assignment, the solution runs iteratively by exchanging targets between clusters in an attempt to reduce the average polygonal area and distance between mass centers and assigned depots. Once targets are assigned to depots, they are divided between the depot vehicles based on the above parameters and polar coordinates. Then, a TSP solver is used to solve the routing problem of each vehicle. Currently the solution uses either Genetic Algorithm or Lin–Kernighan [21]. However, this solution is still sensitive to depot layouts where multiple depots are being blocked in a corner of the map by another depot can cause significant losses in solution accuracy. Methods for alleviating this are currently being investigated.

Finally, in a previous effort [6, 9] we proposed a solution based on multiple intelligent agents competing for tasks in an economic market. We showed that our solution vastly improved the cost obtained by Carlsson *et al.* [4], by up to 40%, but that our solution required a significantly longer computational time. We attributed the improvement in cost to the fact that the market did not partition the region rigidly between the vehicles, and therefore allowed for improvements of the cost beyond any solution that first partitions the set of tasks as done by Carlsson and Narasimha.

It should be noted that the MBS we described in previous works did not make use of a hierarchical market, which is described in this work, and, as a result, its performance in terms of runtime and quality of the solutions deteriorated as we increased the problem size and the ratio of tasks to vehicles. We also observed the undesired trait of high variability in the quality of the solutions and the time required for its computation. It was clear that we needed an additional way to improve the basic MBS in a way that will address these problems while still avoiding a strict partitioning of the area as done by Carlsson and Narasimha.

3. Mathematical Formulation and Solution

3.1. *Mixed integer programming formulation*

We define $N = \{1, 2, \ldots, n\}$ to be the set of indices for task locations, and $D = \{n+1, n+2, \ldots, n+m\}$ to be the set of indices for depot locations. Note that we let each vehicle have a separate location, so in cases where there are multiple vehicles in a depot, that depot location will be repeated as many times as the number of vehicles in that depot. We let $N_0 = N \bigcup D = \{1, 2, \ldots, n, n+1, n+2, \ldots, n+m\}$ be the set of all indices of both tasks and depots.

Let d_{ij} be the distance between any two tasks $i, j \in N$ and let d^0_{ki} and d^0_{ik} be the distance between depot $k \in D$ and task $i \in N$. We assume that the problem is symmetric so that:

$$d_{ij} = d_{ji} \quad \forall\, i, j \in N, \tag{1}$$

$$d^0_{ki} = d^0_{ik} \quad \forall\, i \in N\ \forall\, k \in D. \tag{2}$$

Furthermore, we assume that the problem satisfies the triangle inequality, so that:

$$d_{ij} + d_{jl} \geq d_{il} \quad \forall\, i, j, l \in N_0. \tag{3}$$

Let x_{ijk} be a binary variable equal to one if vehicle k moves from location i to location j along its tour and zero otherwise, where $i, j \in N_0$ and $k \in D$. It should also be noted that $x_{iik} = 0$ to prevent a vehicle from moving from a certain location to the same location on its tour.

Based on the definitions above, one can formulate the tour length of a single vehicle as:

$$L_k = \sum_{j=1}^{n} x_{kjk} d^0_{kj} + \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ijk} d_{ij} + \sum_{i=1}^{n} x_{ikk} d^0_{ik}. \tag{4}$$

The first term in Eq. (4) represents the cost of travel from the vehicle's depot to the location of the first task along its route. The second term sums the distances along the route from one task to the next, and the last term adds the cost of traveling back to the vehicle's depot.

Our goal is to minimize the longest tour by any vehicle, thus we add a continuous decision variable, $L_{\max}$ to the set of decision variables x_{ijk}, and define the cost function as:

$$\text{Minimize } L_{\max}. \tag{5}$$

The optimization is subject to the following constraints:

$$\sum_{k=1}^{m} x_{kik} + \sum_{j=1}^{n} \sum_{k=1}^{m} x_{jik} = 1 \quad \forall\, i \in N, \tag{6}$$

$$x_{kik} + \sum_{j=1}^{n} x_{jik} - \left(x_{ikk} + \sum_{j=1}^{n} x_{ijk} \right) = 0 \quad \forall i,j \in N, \forall k \in D, \tag{7}$$

$$\sum_{i=1}^{n} x_{kik} \leq 1 \quad \forall k \in D \tag{8}$$

$$\sum_{i=1}^{n} x_{kik} - \sum_{i=1}^{n} x_{ikk} = 0 \quad \forall k \in D, \tag{9}$$

$$L_k \leq L_{\max} \quad \forall k \in D, \tag{10}$$

$$\text{Sub-tour elimination constraints.} \tag{11}$$

The constraint in Eq. (6) guarantees that each task is performed exactly once, and Eq. (7) makes sure that if a vehicle performs a task, the same vehicle continues its tour to other locations, hence no task is the end of the tour. Constraints (8) and (9) make sure that each vehicle has at most one tour and that, if the vehicle does have a tour, the vehicle returns to its initial depot location, respectively. Constraint (10) is used to limit the length of a tour by any vehicle. This constraint is the way to bound from below the length of the decision variable $L_{\max}$.

Finally, sub-tours have to be eliminated. In this context, a sub-tour is any tour that contains at least one task and that does not originate in a depot (see also [22]). In such problems, the number of sub-tours is a determining factor in the computational effort required for solving the MILP formulation. Based on our definition of sub-tours, the number of sub-tours depends on the number of tasks that are closely clustered together such that a tour between them is shorter than a tour that also includes a depot. Thus, as the ratio of tasks to depots increases, the likelihood of such sub-tours increases and the complexity increases as a result.

3.2. *Binary programming formulation with binary search*

Here, we present an alternative way to mathematically formulate and solve the MMMDVRP using a BPBS. This second formulation will be most useful when using mathematical optimization solvers that utilize the binary formulation and when there is a way to define strict bounds on the value of $L_{\max}$, for example by running the MBS first.

We do this by making the only continuous decision variable $L_{\max}$ to be a constraint value, thus leaving the formulation only with binary decision variables, x_{ijk}. By gradually searching for the smallest value of $L_{\max}$, we guarantee that the following cost function will satisfy the min–max cost function as well. We replace the min–max cost function in Eq. (5) with the following min-sum cost function:

$$\text{Minimize} \sum_{k=1}^{m} L_k. \tag{12}$$

While there is a difference between finding a solution that minimizes the sum of all vehicle tour lengths and finding the solution that minimizes the longest tour [13], there are three main reasons to use this goal function. First, this allows formulating the problem in a purely Binary Programming framework that commercial solvers can utilize to solve faster. Second, the requirements to visit each task once and that each vehicle has at most one tour are the same in both problems. Third, and most importantly, in many applications, while the main goal is to minimize the longest tour (find the shortest time to complete all missions), it is still desirable that all the other tours will be optimized as well. When using the cost function as in Eq. (5) any tour connecting a subset of tasks, and whose length is less than the maximum tour length, can be arbitrarily found by the solver, even if it is not an optimal tour through that subset of tasks. By defining the cost function the way we do in Eq. (12), we guarantee that all tours are optimized as well as the longest tour.

3.2.1. *Lower and upper bounds*

Note that the constraint in Eq. (10) limits the maximum tour length by any vehicle to $L_{\max}$, but no definition of $L_{\max}$ was given thus far. To define this constraint we define an upper bound, $L_{\max}^{\text{UB}}$, and a lower bound, $L_{\max}^{\text{LB}}$, on the length of the longest tour.

There are several ways to define the upper bound, $L_{\max}^{\text{UB}}$. Generally speaking, any valid solution of the minimum sum of tour lengths or the min–max can be considered as an upper bound for the min–max length, since it satisfies constraints (6)–(11) and thus all the tasks are visited, there is at most one tour per vehicle, and there are no sub-tours. However, this requires a candidate solution, which may take time to compute, depending on the type of solution one uses. Another option is to use the upper bound given by Carlsson *et al.* in [4]:

$$L_{\max}^{\text{UB}} = \frac{\text{TSP}(N)}{m} + 2 * d(D,N). \tag{13}$$

In Eq. (13), $\text{TSP}(N)$ means the length of the optimal TSP tour through the locations in set N and $d(D,N)$ is the longest distance between any location $p \in D$ and any location $q \in N$. While this computation requires a TSP solution, we can use any TSP solver for this purpose, and solvers that are less accurate will only cause the upper bound to increase (be less tight). As this upper bound is not tight, the accuracy of the TSP solver does not affect the algorithm significantly.

There are mainly two ways to define the lower bound, $L^{\text{LB}}_{\max}$. The first one is also based on Carlsson *et al.* [4]:

$$L^{\text{LB}}_{\max_1} = \frac{\text{TSP}(D \cup N) - \text{TSP}(D)}{m}. \tag{14}$$

Another way is to extend the minimum distance defined by Golden *et al.* [13] to the multiple depots case. In Golden *et al.*, there is only a single depot, and the shortest possible min–max tour length is exactly twice the distance between this depot and the farthest task from it. To extend this bound to the multiple depots case, let us consider the problem from the perspective of the tasks instead of the depots. A task will be best served, in terms of travel cost, by the closest depot to it, ignoring all other tasks in the scenario. This means that, for a single task, the lower bound is:

$$\text{LB}_j = \min_i \{d^0_{ij} + d^0_{ji}\}, \tag{15}$$

where the notation d^0_{ij} is used for the travel distance from depot i to task j and d^0_{ji} is the travel cost back. However, there are several tasks, and each one of them has to be served by a vehicle, so the task that requires the longest tour will present the shortest possible tour by any vehicle to any task. This yields the following lower bound extended to the multiple depot min–max case:

$$L^{\text{LB}}_{\max_2} = \max_j \{\min_i \{d^0_{ij} + d^0_{ji}\}\}. \tag{16}$$

For the single depot case, this bound is exactly the same as the lower bound given by Golden *et al.*

Both lower bounds can be computed independently, and, obviously, the larger one is a tighter bound on the problem as a whole. So, we conclude that:

$$L^{\text{LB}}_{\max} = \max(L^{\text{LB}}_{\max_1}, L^{\text{LB}}_{\max_2}). \tag{17}$$

3.2.2. *The algorithm*

We are now ready to define the algorithm. The algorithm is a binary search for the smallest value of $L_{\max}$ that still satisfies the constraint (10) while a solution to the problem exists. The algorithm begins with the values of the upper and lower bounds on $L_{\max}$ as defined in Eqs. (13) and (17), respectively. Since this is a binary search for a solution, whose cost is known to satisfy: $L^{\text{LB}}_{\max} \leq L_{\max} \leq L^{\text{UB}}_{\max}$, the algorithm is guaranteed to converge by continuously reducing the gap between the lower and upper bounds. This is done in the way explained below.

The value of $L_{\max}$ is set to a value in the middle of that range and a linear programming (LP) solver is used. If a solution exists, then there is an assignment of vehicles that minimizes the total sum of tour lengths that satisfies the constraints. So, we use the longest tour in this assignment as the new upper bound. If there is no solution, the problem is infeasible. In other words, the maximum tour length constraint prevents us from finding a solution, and therefore the lower bound can be set to the value of $L_{\max}$.

As long as the difference between $L^{\text{UB}}_{\max}$ and $L^{\text{LB}}_{\max}$ is greater than a certain tolerance threshold, t, the algorithm continues to narrow it down until a solution is found. This allows finding both the value of $L_{\max}$ that is the solution of the problem and a bound on how far this solution is from the lower bound. The algorithm is given in Algorithm 1.

This algorithm is guaranteed to converge to a solution, and this solution provides a known cost $L_{\max}$ as well as a bound on how low that cost can be from a theoretical value, $L^{\text{UB}}_{\max}$. As long as a small tolerance value (t) is used, the difference can be kept as small as desired. For example, we use a value of 10^{-5} for the tolerance, which guarantees that our cost will be at most 10^{-5} from the lower bound. One may wish to improve this accuracy, if that is indeed required.

Algorithm 1 Min-Max Binary Programming Algorithm

Require: m vehicle depot locations, n tasks, tolerance t

1: $L^{\text{UB}}_{\max} \leftarrow$ Eq. 13
2: $L^{\text{LB}}_{\max} \leftarrow$ Eq. 17
3: **while** $(L^{\text{UB}}_{\max}/L^{\text{LB}}_{\max} - 1) \geq t$ **do** ▷ Binary Search for $L_{\max}$ value
4: $\quad L_{\max} = (L^{\text{UB}}_{\max} + L^{\text{LB}}_{\max})/2$
5: $\quad$ Run LP solver on problem
6: $\quad$ **if** $\exists$*solution* (i.e., the solver found a solution) **then**
7: $\quad\quad L^{\text{UB}}_{\max} = \max_{i \in D} L_i$ ▷ Use max tour length as $L^{\text{UB}}_{\max}$
8: $\quad$ **else**
9: $\quad\quad L^{\text{LB}}_{\max} = L_{\max}$ ▷ Infeasible $L_{\max}$, increase $L^{\text{LB}}_{\max}$
10: $\quad$ **end if**
11: **end while**
Return: $L_{\max}$ ▷ is the solution and the tours are the optimal assignment of vehicles to tasks.

The main issue with this algorithm is its implementation and execution. We implemented the algorithm in *MATLAB* and used the well-known CPlex [23] commercial Linear Programming solver by IBM to solve the LP problems. While this is a state-of-the-art solver, we found that the size of problems that can be solved using this combination is limited, mainly due to excessive memory usage by *MATLAB*. More details are given in Sec. 5.

Furthermore, due to the rapid increase in the number of variables required for this formulation and the increasing number of sub-tours, which increases the time required for finding a solution using this LP formulation, this formulation does not lend itself to large-scale problems. Thus, we present in the next section a heuristic solution based on economic markets that will allow better scalability.

4. Market-Based Solution

4.1. *The basic market*

The basic MBS is described in [6, 9], but for the sake of completeness we provide a description of it here. In this solution, the vehicles are represented by software agents [24] that bid and trade for tasks in a simulated economic market. The MBS begins with a fast nearest-neighbor assignment of tasks to agents. The purpose of this assignment is to initialize the market to a reasonable solution, instead of a randomized one, but we have shown that this initial assignment is still far from the best market solution. Following the initial assignment, the MBS is an iterative process, and in each iteration agents perform a set of actions, see Algorithm 2.

Step 1: Market Auction: At the beginning of each iteration, the agents are informed of available tasks, and each agent computes the additional cost it will incur if that task is assigned to it and bids that cost. The cost is calculated by inserting the task to an existing route in the way that will increase the existing tour through tasks, which have already been assigned to the agent, by the smallest distance, and if no other tasks are assigned, the cost is the travel from the depot to the task and back. The agent with the lowest bid is selected to take the task. This is a classic bid-auction mechanism, which was used in many previous works, e.g. [25].

Step 2: Agent-to-Agent Trade: One agent selects randomly a second agent. The first agent queries the list of tasks from the second agent, and calculates the cost of travel to these tasks. If the first agent can service a task on the list of tasks from the second agent with a lesser cost than the second agent, that task is assigned to the first agent. This action is similar to sub-contracting an agent. This peer-to-peer mechanism was suggested by Karmani *et al.* [26], but they did not use an auction process.

Step 3: Agents Switch: So far the first two steps were greedy actions that may result in the solution reaching local minima. The third step is another peer-to-peer process, but

Algorithm 2 Market Algorithm

Require: m agents, n tasks, r clustering ratio.
  Optional: Previous Assignments, Clusters
1: **if** No Previous Assignment **then** ▷ Previous assignments may come from higher hierarchies run
2: **for all** Tasks **do** ▷ Initialize using nearest neighbor partitioning of the area
3: Assign task to nearest agent
4: **end for**
5: **else** ▷ Previous run assigned clusters to agents
6: **for all** Clusters **do**
7: **for all** Tasks in Cluster **do**
8: task.assignment ← cluster.assignment
9: **end for**
10: **end for**
11: **end if**
12: **while** $Iteration \leq NumOfIterations$ **do**
13: Step 1: Market Auction
14: Step 2: Agent-to-Agent Trade
15: Step 3: Agents Switch
16: Step 4: Agents Relinquish Tasks
17: **end while**
  Return: BestAgents, Assignments, Clusters, Costs, Runtime

it is different than the second action, as it allows the solution to escape local minima. Two agents are randomly chosen and their routes are examined. If the two routes form polygons that intersect each other, the agents exchange tasks in an attempt to resolve that intersection. It should be noted, though, that in some cases this may not result in a better solution overall, but it is a way to improve solutions that reach a local minimum.

Following each action, each agent invokes a single TSP solver. We can use any TSP solver in the solution, and we use Concorde [27] for large TSP instances, Lin–Kernighan [21] for medium sized, and convex hull with nearest neighbor insertion (CHNNI) [28] for small problems. These solvers were chosen to get the best runtime performance while still maintaining acceptable quality of the TSP solutions. The fastest solver is the CHNNI, but the quality of its solutions deteriorates when the number of tasks in a tour increases beyond 50 tasks, so it is used only for TSP tours of up to that number. Lin–Kernighan is faster than Concorde, but more limited in problem sizes, so we use Concorde just for the largest TSP routes, with more than 500 tasks per tour.

Step 4: Agents Relinquish Tasks: Finally, agents release randomly chosen tasks in preparation of the next market iteration. The probability to release a task is proportional to the additional cost incurred by that task. By releasing tasks this way, we allow for the possibility of escaping local minima as well as exploitation of best gradient.

The market stops when the number of iterations since the last improvement is greater than a parameter. An improvement is defined as either: (a) a decrease in the longest tour or (b) same longest tour but smaller sum of all tours. The latter is necessary for the improvement of the tours of other agents, which may open the door to further improvement of the longest tour in following market iterations.

4.2. *The hierarchical market*

4.2.1. *Motivation and rationale*

The MBS described above works well for problems that have a relatively low ratio of tasks to vehicles. When this ratio become higher, and especially when the total number of tasks increases, it becomes more difficult for the MBS to find a good assignment of tasks to the vehicles.

One way to improve this is by obtaining a quick and good partitioning of the tasks, as done by Carlsson *et al.* [4] and Narasimha *et al.* [2, 20]. Then, the MMMDVRP would be simplified to solving a set of TSP problems (preferably, in parallel), by invoking TSP solvers for each sub-region. Yet, it is important to understand that partitioning the problem like this inherently adds artificial constraints on the global optimization — the boundaries between the sub-regions. At best, if the partitioning is perfect, these constraints will not increase the best cost, but if it is imperfect it may increase the global cost. To the best of our knowledge, there is no quick way to perform this partitioning perfectly.

The rationale behind the hierarchical market presented in this work is to recursively refine the partitioning of the problem in a way that does not add rigid artificial constraints to the problem. The way we do this is by gradually clustering tasks into larger and larger sets of tasks, while purposefully ignoring the fine resolution of the problem by representing each area by a single task at the center of the area. It should be noted that the depots are not part of this clustering and remain the same depots at each clustering level. A possible future development could be to cluster depots as well.

To illustrate, suppose the tasks truly represent real towns scattered uniformly across the United States of America. A rigid partitioning will divide the country along state lines, or some major topographical features like riversheds, but some states (or riversheds) are bigger than others and this will result in inequitable partitioning. The result will be that some routes are much longer than others and will result in an increased min–max cost.

Our solution works differently. At first, we group towns into small areas, e.g. we group Cincinnati, Mason, and Dayton in Ohio to one small area called SW Ohio, represented as a task near Monroe, Ohio. The second step is to group small areas into larger areas, for example taking the SW Ohio area from before and grouping it with the Central Ohio area of Columbus, Central Kentucky area of Louisville and Lexington, and SE Indiana and we call this area Ohio–Kentucky–Indiana (OKI) and represent it as a task in the Cincinnati area. We continue by grouping the larger areas into regions, say the whole Ohio Valley. We do this until we get to a point in which the number of such regions is small enough to allow us rapid assignment of large regions to depots.

In essence, we got a quick way to partition the area between the depots. It is possibly a good baseline, but this is not necessarily a perfect partitioning. From this point on, the idea is to refine the partitioning by gradually reintroducing the finer resolution of the assignment.

Getting back to the aforementioned example, suppose we initially assign the Ohio Valley region to a depot in Columbus, Ohio, and the Mid-South region to a depot in Memphis, Tennessee, which means that all the areas within these regions are initially assigned to either Columbus or Memphis. However, when considering the level of smaller areas, we discover that if we reassign the southernmost area of the Ohio Valley region, which is represented by a single task at this level, to the depot in Memphis, the resulting min–max

cost will improve, so we reassign that area to Memphis. This reassignment is done using the market algorithm (Algorithm 2) until the solution settles. Then, we refine it again by running the market on the sub-areas, and this process continues until an assignment at the level of single tasks is done. And, while each time the market runs on a finer resolution there is the potential of many new reassignments, the underlying baseline solution achieved at the coarser resolution level prevents many of these permutations and reassignments are done mainly along the partitioning lines.

Thus, we achieved our goal of reducing the number of permutations by partitioning the problem, but at the same time allowing reassignments from one partition to another and having no constraints on the optimization.

4.2.2. *Algorithm description*

The mechanism is based on a recursive clustering of tasks into larger bundles of tasks, where each bundle is represented by a new task at the centroid of this bundle, then trading on these bundles. We define $r > 1$ as the clustering ratio, which, on average, represents the number of tasks per cluster. Reasonable values for clustering ratio were found to be between 2 and 5, where a clustering ratio of 2 usually improves the quality of the solution, but requires more time, and higher clustering ratios generally reduce computation time but increase the variability in the solution quality. More details can be found in [29].

The algorithm is given in Algorithm 3. With every recursion of the algorithm, the number of tasks the market has to solve for will decrease by a factor equal to the clustering ratio, ending with a small number of tasks to handle in the most aggregated case. The market can very quickly find a solution to these tasks, that represent a subregion containing possibly many tasks, and then each of these aggregate tasks is broken down to allow the agents to deal with smaller bundles of tasks. This continues until the market works on single tasks instead of bundles.

Obviously, the parameter r is a design parameter that can be used to tweak the number of tasks bundled together at each recursion, and, consequently, the number of recursions during the run.

A large clustering ratio will require a smaller number of recursions, which may result in a shorter runtime, but with the caveat that from one recursion to the next, the market has a higher degree of freedom in solving for a larger number of bundles. This may cause the next recursion to run longer, as the market explores a larger solution space.

On the other hand, a large clustering ratio also means that the partitioning is less rigid, and therefore, at least in theory, there is a better chance of finding a better solution. However, since the market is limited in the number of iterations it is allowed to run, finding a better solution is also affected by searching in the appropriate region of the search space. Thus, there is no right answer for how much r should be.

Empirically, it was found that for more complicated problems, which involve a higher ratio of tasks per agent, and a higher number of tasks as a whole, it is best to keep r smaller, while for simple cases, r can be allowed to be larger.

Algorithm 3 Hierarchical Market Algorithm

Require: m agents, $n^{(0)}$ tasks, r

$i \leftarrow 0$
while $n^{(i)} > m * r$ **do** ▷ Recursive process
  $i \leftarrow i + 1$
  $n^{(i)} \leftarrow n^{(i-1)}/r$ ▷ Rounded value using Matlab round
  Cluster tasks to $n^{(i)}$ clusters using k-means clustering
  tasks ← clusters ▷ Each cluster becomes a single task, located at its centroid, in the next hierarchy
end while
▷ At this point, the regular market can run. In the highest hierarchy, there is no previous assignment. The market will initialize with a default nearest neighbor assignment. Once there is a previous assignment in one hierarchy, the next assignment will begin by taking the tasks of each cluster and assigning them to the agent that was assigned to the cluster in the higher hierarchy.
while $i > 0$ **do** ▷ Until the solution runs on the level of single tasks.
  Invoke Market (Algorithm 2) to assign tasks to agents.
  $i \leftarrow i - 1$
end while

5. Results

5.1. *Small-sized problems*

In this section, we compare the solutions provided by the MBS with the results obtained by the mathematical programming formulations presented in Sec. 3. We use a laptop with a second generation Intel iCore 5, 2.5 Ghz, with 6 GB RAM, using MATLAB to formulate the problem and the IBM developed CPLEX [23] to solve it. Due to the high memory demands of the MILP and BPBS solutions, the largest problem sizes we could run was limited to 12 tasks. While this problem is very small, this section demonstrates the ability of the MBS to obtain comparable results.

As discussed in Sec. 3, one of the limiting factors of the MILP and BPBS formulations is the number of sub-tour elimination constraints. Recalling that we define a sub-tour as a tour that consists only of tasks, without a depot, it is clear that sub-tours are more likely when the ratio of tasks to vehicles gets larger. Therefore, we define three classes of problems: *vehicles rich* when the ratio of tasks to vehicles is less than 2, *tasks rich* when the ratio is 4 or larger, and a *medium range* in between. It was observed that the mathematical optimization methods took much less time to run, on average, when the scenarios were vehicles rich than when the scenarios were tasks rich.

For each class, we used 20 scenarios with 12 tasks. The vehicles rich scenarios had eight vehicles, the tasks rich scenarios had three vehicles, and the medium range had six vehicles. These scenarios were randomly generated with uniformly distributed tasks and depots in a 100×100 units region. The scenarios were first solved by the MILP solution to obtain its cost and runtime, and then each scenario was run 100 times using the MBS, for a total of 2000 runs per class, 6000 runs overall.

We consider two metrics: the normalized runtime, defined as the ratio of MBS runtime to MILP runtime, and the *optimality gap*. For the optimality gap, a value of zero means that the two solutions reached the same value. The optimality gap is defined as:

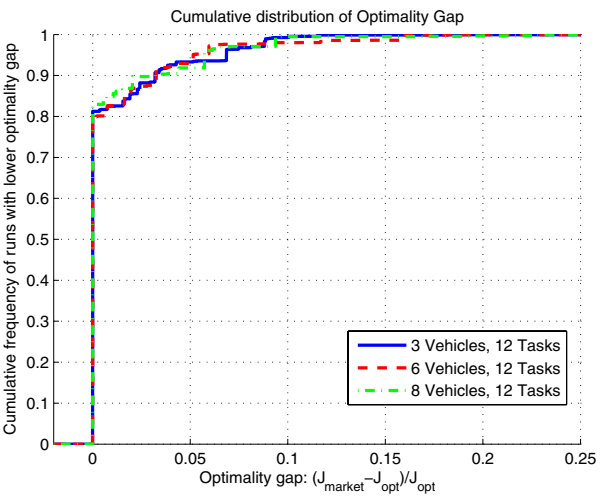
$$\epsilon = \frac{J_{\text{MBS}} - J^*}{J^*}, \tag{18}$$

where J^* is the cost found by the MILP solution.

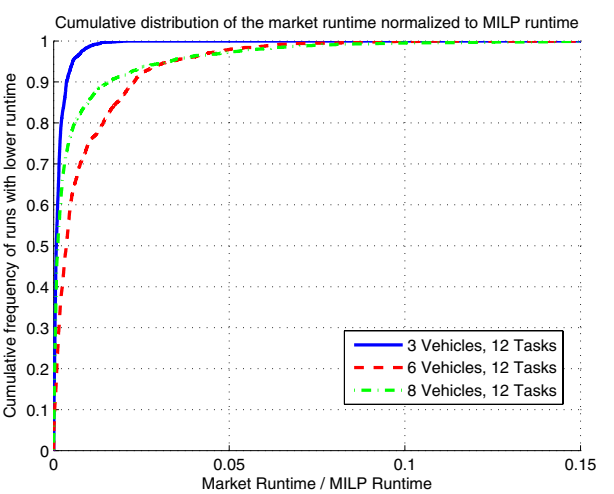
Due to the stochastic nature of the MBS, we present the results in a graphic form using cumulative distributions: the abscissa is the metric used, and the ordinate is the fraction of runs (out of 2000 runs in each class) that obtained a result less than or equal to the value in the abscissa.

In Fig. 1(a), the cumulative distribution of the optimality gap is presented. From the figure, the MBS obtains the perfect optimality gap of 0 in around 80% of the runs: 81.25% of the three vehicles scenarios, 78.63% of the runs in the six vehicles scenarios, 79.75% of the eight vehicles scenarios. 90% of the runs ended with a optimality gap of about 0.03 or less. The worst run in each class was 0.18 for three and six vehicles, and 0.22 for eight vehicles.

Figure 1(b) depicts the normalized runtime, and it is clear that all values are significantly less than 1, in other words the MBS is much faster than mathematical optimization. The worst case in each class is 0.19 for eight vehicles, 0.15 for six vehicles, and 0.02 for three vehicles. These represent an improvement in runtime by a factor of about 5, 6, and 50, respectively. The median runtime ratio for each class is 0.00367 for eight vehicles, 0.00119 for six vehicles, and 0.00077 for three vehicles; again, these represent an improvement by a factor of 272, 840, and 1299, respectively. So, the MBS runs at least one order of magnitude



(a) Cumulative distribution of optimality gap for three different problem classes



(b) Cumulative distribution of market runtime/MILP runtime for three different problem classes

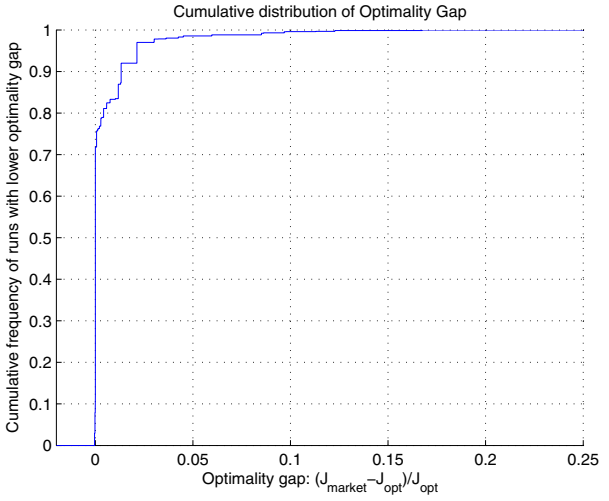
Fig. 1. Comparison of the MBS with MILP.

faster, and in half the cases, two or three orders of magnitude faster, even for such small problems, and even while considering that the mathematical optimization problem is run by a commercial mathematical optimization tool.

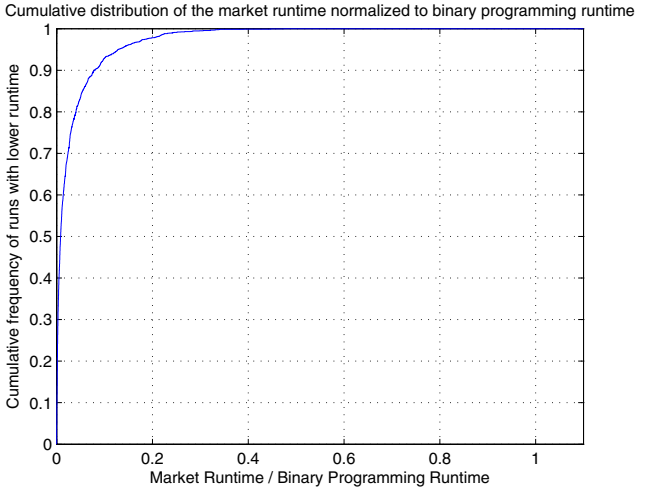
Next, we compare the results of the MBS to results obtained by the BPBS solution for a set of 20 scenarios, with five vehicles and 10 tasks in each scenario, randomly generated in the same way that the previous cases were generated. The results are depicted in Fig. 2 and are similar to the ones obtained for the MILP solution.

From this comparison we conclude that the MBS is much faster than mathematical optimization, but obtains comparable results in the vast majority of the cases. In fact, if we want to improve the probability of obtaining very good results, we can take the best result the MBS obtains in four or five runs and get a marginality gap of zero in a probability of about 99% at a much shorter time.

It is also interesting to see that while the MBS runtime is almost indifferent to the ratio of tasks to vehicles, which determines the number of sub-tour elimination constraints, the mathematical optimization is very sensitive to this ratio. This is evident by the fact that the mathematical optimization takes the longest time to run (worst ratio) in the three vehicles scenarios and in descending order the six vehicles cases and eight vehicles cases.



(a) Cumulative distribution of optimality gap



(b) Cumulative distribution of market runtime/BPBS runtime

Fig. 2. Comparison of the MBS with BPBS solution.

5.2. *Medium-sized problems*

For larger problems, we compare the performance of the MBS with the performance of the other heuristic solutions using two figures of merit: the cost achieved by the solution and the time it took the algorithm to run. All the algorithms use stochastic processes in their computation, and therefore each scenario is run 25 times to obtain some statistics for both the cost and runtime.

Five cases ranging from three depots, six vehicles, and 80 tasks to five depots, 12 vehicles, and 140 tasks were randomly generated for the comparison. Depots and tasks are generated randomly with uniform distribution in a region of 100×100 units. In cases 1–4 the vehicles are distributed evenly between the depots, and in case 5 there are three depots with two vehicles each and two depots with three vehicles. All the cases were run on a PC with Intel second generation multi-core processors. These cases were originally used in the paper by Narasimha *et al.* describing the Ant Colony Optimization (ACO) [20].

Table 1 gives the best results obtained by each algorithm (in 25 runs) for each case. The MBS, with a clustering ratio of 2, yields best costs that are better than the other two algorithms.

In Table 2, we compare the gap of each algorithm relative to the best cost obtained by any algorithm. Note that these cases are far too big for the mathematical optimization methods, so we take the best of the results in each case as the incumbent best. Thus, the gap in this case is defined as: $\frac{J_m - Ji}{J_i}$, where J_m is the cost obtained by method m and J_i is the best cost so far, which in all the cases is

Table 1. Comparison of best costs: Carlsson, ACO and MBS.

Case	d	m	n	Carlsson	ACO	MBS
1	3	6	80	193.7000	186.5168	160.8532
2	3	9	140	172.6441	184.5539	153.2285
3	4	8	80	207.6464	205.1906	148.7433
4	4	12	140	149.4203	146.8195	125.0229
5	5	12	140	129.0139	125.0439	116.6997
6	5	15	140	129.7286	110.2564	96.6106

Table 2. Gap for best results of the Carlsson, ACO and MBS.

Case	*d*	*m*	*n*	Carlsson	ACO	MBS
1	3	6	80	0.2042	0.1595	0
2	3	9	140	0.1267	0.2044	0
3	4	8	80	0.3960	0.3795	0
4	4	12	140	0.1951	0.1743	0
5	5	12	140	0.1055	0.0715	0
6	5	15	140	0.3428	0.1412	0

obtained by the MBS. Carlsson's gaps are between 0.1055 and 0.396, and the Ant Colony algorithm has gaps of 0.0715 to 0.3795.

Next, we check the average costs for the aforementioned cases to check the consistency of each method in getting good results. Table 3 gives the average gap computed by each algorithm for each case. As the table shows, the MBS consistently outperforms the other two solutions with average gap that is between 0.017 and 0.052, while the ACO gets an average gap of between 0.074 and 0.379 and Carlsson between 0.244 and 0.731. This also shows that the variability of the average performance is much larger in the other two algorithms, in particular of Carlsson's algorithm. It should also be noted that the average cost obtained by the MBS is better than the best cost obtained by each of the other two solutions for the same case.

Table 4 gives the time required for each solution to complete computation, on average, for each case. From the table it is clear that Carlsson takes the least amount of time to run, which also seems to be unrelated to the size of the problem. The Ant Colony solution requires a runtime which is generally two orders of magnitude higher than the time required for Carlsson's algorithm. The MBS requires runtime that is generally one order of magnitude higher than Carlsson's runtime. While this is an improvement relative to the solution we published in [6], the same reasons cause our solution to be slower than Carlsson's: the MBS is an iterative process while Carlsson's is not, and in addition, Carlsson's solution mostly relies on compiled code whereas the MBS is written in MATLAB.

Table 3. Comparison of average gap relative to the best cost: Carlsson, ACO and MBS.

Case	*d*	*m*	*n*	Carlsson	ACO	MBS
1	3	6	80	0.398	0.187	0.017
2	3	9	140	0.244	0.24	0.027
3	4	8	80	0.438	0.379	0.032
4	4	12	140	0.453	0.205	0.019
5	5	12	140	0.291	0.074	0.017
6	5	15	140	0.731	0.156	0.052

Table 4. Comparison of runtime (in seconds): Carlsson, ACO and MBS.

Case	*d*	*m*	*n*	Carlsson	ACO	MBS
1	3	6	80	1.78	245.95	40.07
2	3	9	140	4.42	602.49	47.35
3	4	8	80	2.14	225.75	20.31
4	4	12	140	1.76	484.53	60.28
5	5	12	140	4.10	432.60	51.00
6	5	15	140	5.58	443.86	55.98

5.3. *Large-scale problems*

For large-scale problems that include tens of vehicles and a thousand tasks or more, we compare the MBS with Carlsson's algorithm and a new method still in development, Fuzzy Clustering (FCL) solution, described by Ernest and Cohen [8]. The ACO is not used because of its limited scalability to larger problems.

To accommodate the current requirements of the FCL solution, a problem with four depots, 16 vehicles, and 1000 tasks was generated. The depots are located at [250, 250], [250, 750], [750, 250], and [750, 750]. There are four vehicles per depot, and 1000 tasks randomly located in a 10 00 $\times$ 1000 region. Table 5 lists the results of the three algorithms for this case. For this case the best known solution was obtained by the MBS to be 1603.2 units.

Once again, the cost of the MBS is the best out of the three algorithms both for best run and average. For the best runs, the gaps of the other methods are 0.1672 and 0.1015 for Carlsson and FCL, respectively. The average costs of each algorithm have a gap of 0.255, 0.1022, and 0.0212 for Carlsson, FCL, and MBS, respectively. This shows that the results of the MBS are consistent even for larger problems.

The new FCL solution for this problem gets the best runtime out of the three, a mere 16.2 s, with a cost just 10.1% higher than the best MBS cost. However, in other experiments we have done, not shown here, the FCL solution exhibited high sensitivity to the locations of the depots, which increased the gap of its best cost relative to the MBS best cost. The caveat is the runtime required for this problem, which jumps from 35.2 s required for Carlsson's

Table 5. Comparison of Carlsson, FCL and MBS for the case with 4 depots, 16 vehicles and 1000 tasks.

Metric	Carlsson	FCL	MBS
Best gap	0.1672	0.1015	0
Avg. gap	0.2550	0.1022	0.0212
Avg. time (s)	35.2	16.225	1419.4

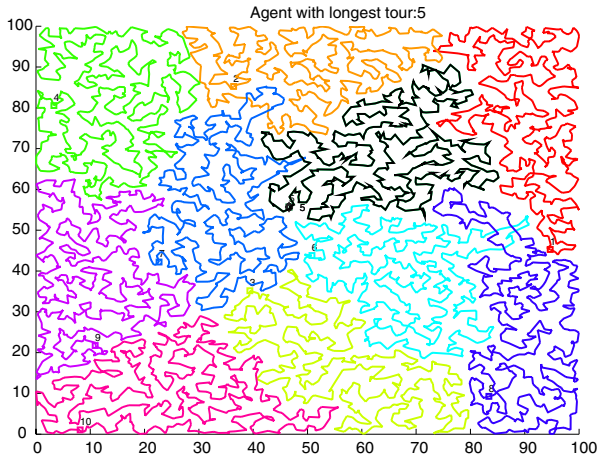


Fig. 3. A solution to the Min–Max case with 10 vehicles and 5000 tasks. Runtime was about 12,000 s on a computer with i7, 3.4 GHz CPU with 16GB RAM.

solution to 1419 s, about 23 min, required by the MBS. It should be noted, however, that this is indeed a large problem. In addition, due to distributed nature of the MBS, it can be implemented in a way that will take advantage of parallel computing, something that is not achievable in Carlsson. This has the potential of reducing the runtime significantly, approximately by a factor of the number of agents.

We end this section with two examples of very large scale problems. In the first there are 10 depots, 10 vehicles and 5000 tasks, which yields a very large ratio of 500 tasks per vehicle. Figure 3 depicts the assignment of tasks to vehicles, with a cost of 577 units. The solution based on Carlsson's algorithm is slightly higher at 590 units. While computing the optimal cost of such problem is beyond our current capabilities, we believe that this assignment is close to optimal, based on the small differences between vehicle costs and the very few intersections between vehicle tours. It seems, therefore, that there is little room for improving this assignment.

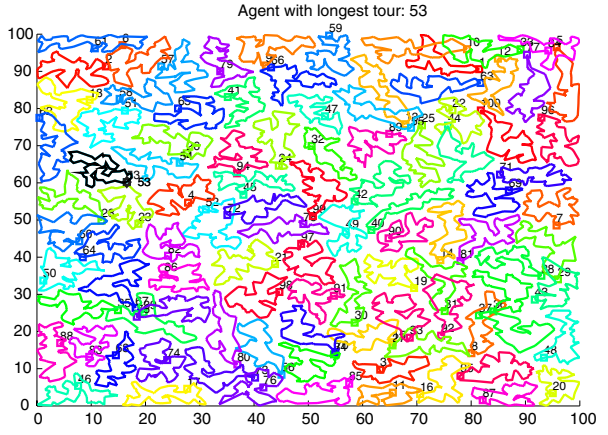


Fig. 4. A solution to the Min–Max case with 100 agents and 5000 tasks. Runtime was about 76,477 s on a computer with i7, 3.4 GHz CPU with 16GB RAM.

In the second problem, depicted by Fig. 4 there are 100 depots, 100 vehicles and 5000 tasks. The cost in this case is 64.7375 units and the difference between the top 20 vehicles is less than 2 units, which may indicate little room for improvement in this problem as well. Attempts to run this case using Carlsson's code resulted in a failure, probably due to the size of the problem.

6. Conclusion

We address the problem of the MMMDVRP. This problem is of extreme importance to the assignment of multiple autonomous UAVs to perform tasks in time-critical applications, both in military and civilian contexts. We formulate the problem using mathematical optimization methods and present two formulations: as a MILP problem and as a purely BPBS for the longest tour constraint.

We extended a solution based on agents interacting in an economic market settings by aggregating tasks in hierarchical manner. This solution allows us to perform a "soft" partitioning of the tasks, and then solve for the optimal route between the tasks assigned to a vehicle. By doing so, we show that the resulting cost we achieve is vastly better than existing state-of-the-art solutions, in particular Carlsson's algorithm.

While the computational effort required by the hierarchical MBS, measured in terms of execution time, is improved relative to the MBS without hierarchies, it is still significantly longer than the time required for Carlsson's algorithm, but much better than the time required for an ACO solution that also improves over the cost obtained by Carlsson's algorithm.

In the future, we intend to focus our research on three directions. The first one is to improve the BPBS so that it will be possible to solve problems of larger scales than what is currently possible. The second avenue is improving the runtime of the hierarchical MBS. We believe that there are two directions that can be combined to improve the results. First, this algorithm can be implemented in a compiled language, e.g. C++ or Java, which will improve overall runtime compared to the scripting language used now. The second direction is to use the inherently distributed nature of the multi-agent market in order to run the algorithm on a parallel processing machine. We believe that the combination of these two efforts can bring the runtime of the hierarchical market to be competitive with Carlsson's runtime.

References

[1] T. Shima, S. J. Rasmussen, A. G. Sparks and K. M. Passino, Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms, *Comput. Oper. Res.* **33**(11) (2006) 3252–3269.

[2] K. V. Narasimha, M. Kumar and E. Kivelevitch, Ant colony optimization technique to solve the min-max multi depot vehicle routing problem, *2012 IEEE American Control Conf.*, June 2012.

[3] A. M. Campbell, D. Vandenbussche and W. Hermann, Routing for relief efforts, *Transport. Sci.* **42** (2008) 127–145.

[4] J. Carlsson, D. Ge and A. Subramaniam, Solving min-max multi-depot vehicle routing problem, in *Lectures on Global Optimization* (American Mathematical Society, 2009), pp. 31–46.

[5] C. A. C. Parker and H. Zhang, Cooperative decision-making in decentralized multiple-robot systems: The best-of-n problem, *IEEE/ASME Trans. Mechatronics* **14** (2009) 240–251.

[6] E. Kivelevitch, K. Cohen and M. Kumar, Market-based solution to the allocation of tasks to agents, *Proc. Comput. Sci.* **6** (2011) 14–19.

[7] N. Ernest and K. Cohen, Fuzzy logic clustering of multiple traveling salesman problem for self-crossover based genetic algorithm, in *Proc. IAAA 50th ASM*, January 2012 (AIAA, Nashville, TN).

[8] N. Ernest and K. Cohen, Fuzzy clustering based genetic algorithm for the multi-depot polygon visiting dubins multiple traveling salesman problem, in *Proc. 2012 IAAA Infotech@Aerospace*, June 2012 (AIAA, Garden Grove, CA).

[9] E. Kivelevitch, K. Cohen and M. Kumar, A market-based solution to the multiple traveling salesmen problem, *J. Intell. Robot. Syst.* (2013).

[10] D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, 1st edn. Princeton Series in Applied Mathematics, (Princeton University Press, Princeton, New Jersey, 2006).

[11] T. Bektas, The multiple traveling salesman problem: An overview of formulations and solution procedures, *Omega* **34** (2006) 209–219.

[12] D. Feillet, P. Dejax and M. Gendreau, Traveling salesman problem with profits, *Transport. Sci.* **39** (2005) 188–205.

[13] B. L. Golden, G. Laporte and E. D. Taillard, An adaptive memory heuristic for a class of vehicle routing problems with minmax objective, *Comput. Oper. Res.* **24**(5) (1997) 445–452.

[14] N. Christofides, A. Mingozzi and P. Toth, The Vehicle Routing Problem, in *Combinatorial Optimization* (Wiley, Chichester, 1979), pp. 313–338.

[15] M. L. Fisher, Optimal solution of vehicle routing problems using minimum k-trees, *Oper. Res.* **42** (1994) 626–642.

[16] E. Kivelevitch, K. Cohen and M. Kumar, On the scalability of the market-based solution to the multiple traveling salesmen problem, in *Proc. 2012 AIAA Infotech@Aerospace Conf.* June 2012 (AIAA).

[17] C. Ren, Solving min–max vehicle routing problem, *J. Software* **6** (2011) 1851–1856.

[18] C. Ren, Heuristic algorithm for min–max vehicle routing problems, *J. Comput.* **7** (2012) 923–928.

[19] J. F. Bard, G. Kontoravdis and G. Yu, A branch-and-cut procedure for the vehicle routing problem with time windows, *Transport. Sci.* **36** (2002) 250–269.

[20] K. V. Narasimha, E. Kivelevitch, B. Sharma and M. Kumar, An ant colony optimization technique for solving minmax multi-depot vehicle routing problem, *Swarm Evol. Comput.* (2013).

[21] S. Lin and B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, *Oper. Res.* (1973).

[22] E. Kivelevitch, K. Cohen and M. Kumar, A binary programming solution to the multiple-depot, multiple traveling salesman problem with constant profits, in *Proc. 2012 AIAA Infotech@Aerospace Conf.*, June 2012 (AIAA).

[23] IBM, Cplex Web (2012), Available at http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[24] M. Wooldridge, *An Introduction to Multi Agent Systems*, 1st edn. (John Wiley & Sons, Chichester, England, 2002).

[25] H.-L. Choi, L. Brunet and J. P. How, Consensus-based decentralized auctions for robust task allocation, *IEEE Trans. Robot.* **25** (2009) 912–926.

[26] R. K. Karmani, T. Latvala and G. Agha, On scaling multi-agent task reallocation using market-based approach, in *Proc. First IEEE Int. Conf. Self-Adaptive and Self-Organizing Systems*, July 2007 pp. 173–182.

[27] D. L. Applegate, R. E. Bixby, V. Chvatal and W. J. Cook, Concorde (2004), Available at http://www.tsp.gatech.edu/concorde/index.html.

[28] L. Giaccari, Tspconvhull, MATLAB Central (2008).

[29] E. Kivelevitch, K. Cohen and M. Kumar, On the scalability of the market-based solution to the multiple traveling salesmen problem, in *Proc. 2012 AIAA Infotech@Aerospace Conf.*, June 2012 (AIAA).

Elad Kivelevitch received his Bachelor of Science and Master of Science in Aerospace Engineering from the Technion–Israel Institute of Technologies in 1997 and 2005, respectively, and his Ph.D. in Aerospace Engineering from the University of Cincinnati, in 2012. He has been an Assistant Professor Educator at the Department of Aerospace Engineering and Engineering Mechanics at University of Cincinnati (UC) since August 2012. Prior to pursuing his Ph.D., he worked in research and development of unmanned systems between 1999 and 2008. At UC, he teaches the following undergraduate courses: Integrated Aircraft Engineering, Fundamentals of Control Theory, Modeling and Simulation of Dynamic Systems, Dynamics; as well as the following graduate courses: Analytical Dynamics, Modern Control. His research interests are Combinatorial Optimization, Unmanned Systems, Complex Adaptive Systems, and Intelligent Systems.

Balaji R Sharma is an engineer with MathWorks since 2013, and engages with academic institutions on curriculum development. He completed his Baccalaureate degree in Mechanical Engineering at Visweswaraya Technological University in India in 2006. He received his Master's degree in Mechanical Engineering in 2010, with a thesis in the field of automotive modal analysis. He received his doctoral degree, also in Mechanical Engineering, at the University of Cincinnati in 2013, with his dissertation work exploring cooperative control algorithms and spatio-temporal estimation methods for improved situational awareness in large-scale events such as wildland fires. His research areas include cooperative control, estimation methods, sensing and robotics. He is the recipient of the University Research Council scholarship at the University of Cincinnati.

Nicholas Ernest is a Ph.D. student of Aerospace Engineering at the University of Cincinnati, where he has studied since beginning his undergraduate degree in the same major in 2006. Starting in 2008, he has worked part-time at Wright-Patterson Air Force Base through a variety of programs in both the Air Force Research Laboratory and Landing Gear Test Facility. His research interests include; genetic fuzzy systems, unmanned combat aerial vehicles, and cooperative multi-agent control.

Manish Kumar received his Bachelor of Technology degree in Mechanical Engineering from Indian Institute of Technology, Kharagpur, India in 1998, and his M.S. and Ph.D. degrees in Mechanical Engineering from Duke University, NC, USA in 2002 and 2004, respectively. After finishing his Ph. D., he worked as a postdoctoral research associate in the Department of Mechanical Engineering and Materials Science at Duke University from 2004 to 2005. In 2005, he received the Research Associateship Award from National Research Council (NRC). This award allowed him to work as a postdoctoral Research Associate with the Army Research Office, NC, USA from 2005 to 2007. As a part of his NRC Associateship program, he was a visiting scholar at General Robotics, Automation, Sensing, and Perception (GRASP) laboratory at the University of Pennsylvania, PA, USA. Subsequently, he worked as an Assistant Professor in the School of Dynamic Systems at the University of Cincinnati, OH, USA where he directed the Cooperative Distributed Systems (CDS) Laboratory and co-directed the Center for Robotics Research. He is currently an Associate Professor in the Department of Mechanical, Industrial, and Manufacturing Engineering in the University of Toledo, OH, USA. His current research interests include complex systems, decision making and control in large-scale systems, development of novel techniques to fuse data from multiple sources, robotics, swarm systems, and multiple robot coordination and control. He is a member of American Society of Mechanical Engineers.

Kelly Cohen an Associate Professor of Aerospace Engineering, has been at the University of Cincinnati (UC) since 2007. Prior to his academic position at UC, he has 13 years of experience working as a program manager in the area of UAV research and development for a government agency. He obtained his Ph.D. in aerospace engineering at the Technion–Israel Institute of Technology in 1999. At UC, he teaches the following undergraduate courses: introduction to systems engineering, integrated aircraft engineering, fundamental control theory, and modeling and simulation of dynamic systems; as well as the following graduate courses: analytical dynamics, fuzzy logic control, systems engineering analysis, and optimal control. His research interests include: intelligent systems, cooperative multi-agent control, unmanned aerial vehicles, dynamic optimization, reduced order modeling, and system identification.