ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/257429396

A Market-Based Solution to the Multiple Traveling Salesmen Problem

Article *in* Journal of Intelligent and Robotic Systems · January 2013

CITATIONS
5

READS
136

3 authors:

Elad Kivelevitch
University of Cincinnati
29 PUBLICATIONS **74** CITATIONS
SEE PROFILE

Kelly Cohen
University of Cincinnati
214 PUBLICATIONS **1,035** CITATIONS
SEE PROFILE

Manish Kumar
University of Cincinnati
99 PUBLICATIONS **361** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Unmanned Aerial Systems View project

https://www.researchgate.net/publication/304087998_Genetically_Tuned_LQR_Based_Path_Following_for_UAVs_un
View project

All content following this page was uploaded by Kelly Cohen on 12 January 2017.

The user has requested enhancement of the downloaded file. All in-text references underlined in blue are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.

J Intell Robot Syst (2013) 72:21–40
DOI 10.1007/s10846-012-9805-3

A Market-based Solution to the Multiple Traveling Salesmen Problem

Elad Kivelevitch · Kelly Cohen · Manish Kumar

Received: 14 November 2011 / Accepted: 19 December 2012 / Published online: 30 January 2013
© Springer Science+Business Media Dordrecht 2013

Abstract This paper describes a market-based solution to the problem of assigning mobile agents to tasks. The problem is formulated as the multiple depots, multiple traveling salesmen problem (MTSP), where agents and tasks operate in a market to achieve near-optimal solutions. We consider both the classical MTSP, in which the sum of all tour lengths is minimized, and the Min-Max MTSP, in which the longest tour is minimized. We compare the market-based solution with direct enumeration in small scenarios, and show that the results are nearly optimal. For the classical MTSP, we compare our results to linear programming, and show that the results are within 1 % of the best cost found by linear programming in more than 90 % of the runs, with a significant reduction in runtime. For the Min-Max case, we compare our method with Carlsson's algorithm and show an improvement of 5 % to 40 % in cost, albeit at an increase in runtime. Finally, we demonstrate the ability of the market-based solution to deal with changes in the scenario, e.g., agents leaving and entering the market. We show that the market paradigm is ideal for dealing with these changes during runtime, without the need to restart the algorithm, and that the solution reacts to the new scenarios in a quick and near-optimal way.

Keywords Vehicle routing · Traveling salesman problem · Agent-based group decision making · Market-based approach

Mathematics Subject Classification (2010)
68T42 · 93A14 · 93C85 · 68T37 · 90B06

E. Kivelevitch (✉)
School of Aerospace Systems,
University of Cincinnati, 2851 Woodside Dr,
Cincinnati, OH 45221, USA
e-mail: elad.kivelevitch.uc@gmail.com

K. Cohen
School of Aerospace Systems,
University of Cincinnati, Cincinnati,
OH 45221-0070, USA

M. Kumar
Department of Mechanical, Industrial and
Manufacturing Engineering (MIME),
University of Toledo, Toledo, Ohio,
43606-3390, USA

1 Introduction

1.1 Problem Formulation

This research focuses on the problem of allocating a group of m mobile autonomous agents to perform a set of n tasks defined by their location. The problem of allocating resources to tasks is a fundamental problem in optimization of a variety of autonomous systems. Some examples include

allocating computer resources [14, 15, 20], allocating network resources to consumers [15, 29], allocating parts and processing machines in a factory [3], and allocating targets to unmanned vehicles [34–37]. Therefore, the benefits of a good resource allocation algorithm are far reaching.

In this problem, the goal is to assign mobile agents to visit the location of tasks. Thus, we formulate the problem as the Multiple Traveling Salesman Problem (MTSP). Each agent has an initial location from which it starts its motion, also called a *depot*. Some agents may share the same initial location.

Following the problem definition in [13], let $G = (V, A)$ be a graph, where $V = \{v_1, v_2, ..., v_n\}$ is a set of n vertices (task locations) and A is a set of edges connecting these vertices. Let the cost of traveling from vertex v_i to vertex v_j along the edge $e_{ij} \in A$ connecting them be $d_{ij} \geq 0$. The cost d_{ij} is assumed to be symmetric, i.e., $d_{ij} = d_{ji}$. Furthermore, the costs are assumed to satisfy the triangle inequality, namely $d_{ij} + d_{jl} \geq d_{il}$. These assumptions correspond to costs associated with the distance of traveling between vertices, the travel time, the fuel consumed during the travel, or the cost of this fuel.

In the MTSP there are m agents. The MTSP is a problem of assigning at most m tours, one tour per agent. Denoting the cost of agent k's tour by J_k, where $k \in \{1, 2, ..., m\}$, we can calculate the tour cost using the definitions of costs given above as:

$$J_k = \sum_{e_{ij} \in A} w_{ijk} d_{ij} \tag{1}$$

Where w_{ijk} is a binary variable that indicates that agent k travels from vertex i to vertex j in its tour R_k.

There are two possible goals in such an assignment:

- Case 1, denoted *MinSum*: Minimizing the sum of traveling distances of the entire group, which is the classic MTSP goal, as given in [13].
- Case 2, denoted *MinMax*: Minimizing the longest traveling distance of any agent in the group. By defining the cost this way, we implicitly reduce the time to perform all the tasks, and require that the tasks will be shared between the agents. The multiple depot version of this problem was considered by [8].

By using various weighting functions, other cost functions can be defined as linear combinations of the above cost functions. The mathematical formulation of the above cost functions is given as follows:

$$J^{\star}_{MinSum} = \min \sum_{k=1}^{m} J_k \tag{2}$$

$$J^{\star}_{MinMax} = \min J_{k^+}, \; where \tag{3}$$
$$k^+ = argmax_{k \in \{1,...,m\}} J_k$$

Equations 2 and 3 are essentially the same as the cost functions defined by Shima et al. [38]. The optimization process, for any of the above cases, is subject to the following constraints: each vertex has to be visited exactly once and there is no more than a single tour assigned to a single agent. In addition, the solution must eliminate any subtours, i.e., parts of the tour that are not connected to agent tours [13].

The MTSP is known to be a computationally hard problem. In fact, for m agents and n tasks, assuming that each task is assigned to only one agent, the number of all possible assignments is given by [38]:

$$N_a = m^n \cdot n! \tag{4}$$

1.2 Requirements from a Good Algorithm

The MTSP is known to be $NP-hard$, in the sense that a guaranteed optimal solution requires a computation time that is non-polynomial. As a result, solutions that cannot guarantee optimality, but which meet other requirements, were proposed. We define the criteria for a good solution as follows:

- *Near-optimality*: assuming an optimal cost of a certain scenario is known, the cost computed by our algorithm deviates from the optimal cost by as little as possible. The measurement is given as percentage above the optimal solution, where 0 % above is the best. Note that

in many cases, the optimal cost may not be readily known.
- *Computation speed*: the time required for obtaining a solution. This can be compared with the time required for obtaining an optimal solution. In addition, as the number of tasks and agents increases, the computation time should increase in a polynomial scale.
- *Adaptability to dynamic scenarios*: we define *dynamic scenarios* as scenarios during which tasks or agents are added/removed while the tasks are being executed by the agents. Adaptability is the ability to incorporate information regarding such changes during computation and to find a solution to the new situation.

1.3 Key Contributions of This Solution

The following aspects are the main new contributions of this solution:

We present a solution that can solve both the *MinSum* and the *MinMax* cases. As the work in [17] discusses, applying an algorithm designed for one case usually results in poor results for the other case, but that is not the case here.

With respect to the *near-optimality* criterion, for the *MinSum* case our results are comparable to the results obtained by Binary Programming [13], which is the best known solution in terms of optimality, but obtained significantly faster. For the *MinMax* case, the solution presented here obtains better results than the best known solution to the *MinMax* case provided by Carlsson et al. [8].

Furthermore, algorithms that can solve MTSP with dynamic scenarios in an online manner are relatively new and there is a significantly less amount of previous results dedicated to them. Our solution adapts to scenario changes, and an investigation of this capability was given in a separate work we published earlier this year [27].

Lastly, market-based solutions are, by their nature, based on the interaction of multiple encapsulated agents. In theory, it should be easier to implement such solutions in a distributed manner. Although the current implementation of the market-based solution is centralized, we provide some recommendations for a future fully distributed implementation.

2 Literature Survey

2.1 General

The problem of allocating resources to mobile autonomous agents has been studied in a variety of applications and using a plethora of methods. It is well beyond the scope of this literature survey to cover all existing work, and the focus of this survey is to cover the basic related problem, which is the Traveling Salesman Problem (TSP), including its many variants, and the methodology that is based on agents competing in a market.

2.2 The TSP Variants and Commonly Used Solutions

The Traveling Salesman Problem (TSP) is defined in the following way: a salesman has to visit a certain list of locations (cities), such that each city is visited exactly once, and return to his initial location, while minimizing the total cost of travel. The best guaranteed optimal solution, based on Dynamic Programming, was developed by Held and Karp [21], which has an improved time complexity of the order of magnitude of $O(n^2 2^n)$, where n is the number of locations.

Since the Dynamic Programming solution is limited in the number of cities it can be applied to, efforts were made to develop scalable solutions to the TSP, although none of them can guarantee optimality, either by using Linear Programming [18, 31–33], or heuristic methods that use improvement moves [30], *simulated annealing* [26], *genetic algorithms* [24] and *Ant Colony Systems* (ACS) [11]. The state-of-the-art TSP solver, which is based on a combination of Linear Programming and heuristic solutions, is Concorde [1]. For a comprehensive recount of the TSP and its solutions, we refer the interested reader to Applegate et al. [2].

The TSP has been extended in several ways. The first extension involves having multiple traveling salesmen (MTSP), all located in the same initial point. The Vehicle Routing Problem (VRP) usually adds constraints to the MTSP, e.g., demand/capacity constraints, fuel constraints or time windows. The MTSP (and VRP) was extended to cases where the salesmen originate in different

initial locations, often called *depots*, i.e., Multiple Depots Multiple Traveling Salesmen Problem. This problem has two common variants, one tries to minimize the total cost of travel by all salesmen (*MinSum* in our notation) and the other seeks to minimize the maximum cost by any salesman (*MinMax*), as defined by Carlsson.

Another variation for the MTSP and VRP is *Dynamic Vehicle Routing Problem (DVRP)*, in which tasks stochastically arrive in time according to a Poisson distribution to the scenario [5]. Several solutions to the problem were proposed by Bertsimas [5], and Jaillet and Wagner [23]. The quality of such solutions is usually computed using a *competitive ratio*, defined as the worst-case ratio of the online algorithms cost to the cost of an optimal off-line algorithm, where all data are known a priori. In this work we further extend the definition of a dynamic problem to include addition and removal of mobile agents.

In a recent work on DVRP, Bullo et al. [6] extended the dynamic problem to cases in which the vehicles are constrained in their minimum turn radius, i.e. Dubins vehicles. They also provide measurements regarding the quality of their solution to the dynamic problem.

While the dynamic versions of the MTSP and VRP have become the focus of research interest, they are still relatively less common than the solutions to the "static" problems. Similarly, online algorithms, i.e. algorithms that incorporate new knowledge as it becomes available, for solving these dynamic problems are relatively scarce. The market-based solution described in this work is one such solution.

2.3 Multi-agent Techniques

The solution we propose is based on the complex systems paradigm of an economic market, in which agents interact. An *agent* is defined as "a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives" [39].

Furthermore, it is common to build systems that involve multiple agents. A rigorous taxonomy of multi-agent systems was developed by Dudek et al. [12]. According to [39], multi-agent solutions are desired when: (1) the environment is open, uncertain, dynamic, or complex, and it is virtually impossible to prescribe all possible reactions, hence a solution based on autonomously interacting agents, may be the only possible solution, (2) when the problem is more easily described using the metaphor of an autonomous agent, for example a market, (3) when we want to facilitate the distribution of data, control or expertise and (4) when there is a need to incorporate a legacy system, which is a system that was developed in the past and is planned to be in use in the future, into a new solution.

2.3.1 Market-based Optimization

Dealing with resource allocation problems was traditionally done in a centralized way [3]. For centralized solutions to work, a central node has to have complete and perfect information about the resources, their capabilities and availability, the tasks and their requirements, and any deviation from a plan [14]. This becomes more complicated as the number of tasks and resources increases, and, in particular, when the system has to react to dynamic changes.

To alleviate some of the aforementioned issues, decentralized resource allocation processes have been proposed. One type of a decentralized solution is based on economic markets [29]. In such solutions, agents represent tasks and resources, and they act egotistically in an attempt to maximize their own benefit. Based on economic markets theory, this results in *Pareto-optimal* solutions, which are clearly a desired outcome, as they reflect efficient solutions to the problem where making one agent better off can only be done by making other(s) worse off. In addition, economic markets are known to be robust and react to various changes in the system [10].

2.3.2 Consensus-based Decentralized Auctions

Choi et al. [9] present a decentralized auction-based task allocation, where each agent is allowed to have individual situational awareness (SA), and bid on tasks based on their SA. No centralized auctioneer is required to clear the auction, which is resolved by means of consensus. For multiple

tasks they group tasks in bundles and let agents bid on tasks in the bundle.

While the Consensus Based Bundle Algorithm (CBBA) [9] bears some resemblance to our market-based solution, there are significant differences between the two. First, CBBA is strictly based on auction, which roughly corresponds to the auction mechanism we describe in Algorithm 2, but it does not have the other two mechanisms that we use, namely one-to-one trade and complete takeover or agent switch. In addition, in our solution each agent decides which tasks to give away for further auctions after performing a local optimization on its assigned tasks, while in CBBA this process does not exist.

There are two additional notes that should be made regarding the CBBA. The first one is that the CBBA allows for asynchronous auction in the presence of incomplete networks and ambiguous SA. The second note is that the CBBA uses a generalized goal function. These generalizations make the CBBA useful in the presence of real-life situations where communications may be imperfect and goals change.

2.3.3 Probability Collectives

Probability Collectives (PC) have been proposed for the solution of the multiple depots multiple traveling salesmen problem (MDMTSP) by Kulkarni and Tai [28]. As Kulkarni and Tai show, the PC is a distributed solution that achieves good results in various combinatorial optimization problems, e.g. forming ad-hoc networks and clusters and airline scheduling problems with constraints. The distributed nature of the algorithm allows it to achieve better scalability while the solution is comparable and even provides better results than other heuristics, e.g. Simulated Annealing [28]. Furthermore, the PC approach is robust in two senses: first, it can handle optimization problems with irregular cost functions, and second, it can handle cases in which agents failures need to be considered.

In essence, the PC approach is somewhat similar to the market-based approach in the sense that agents work in an iterative process to better their own situation by making local decisions until they reach Nash equilibrium. The aggregate result is an emergent behavior that yields a global near optimal solution. The main difference is that agents choose their actions from a set of possible strategies with probabilities that evolve over time as the search continues in the optimization space. However, the PC approach requires some a-priori knowledge about the environment, available strategies and how other agents act in the environment, which is not required in the market-based solution proposed in this work.

Kulkarni and Tai solve the MDMTSP using the PC approach, where strategies are predefined routes, and the probabilities of using them are based on their costs. This solution is combined with heuristics for inserting and eliminating cities from intermediate solutions, because the aforementioned strategies chosen by the agents may visit the same city more than once or not visit a city in some cases. The authors show a solution for two cases with three depots, three agents, and 15 cities, with an average runtime of between one and two minutes. Using the market based solution proposed in this work, we were able to reproduce the results of the PC approach (using the *MinMax* cost function) with an average runtime of a few seconds. When we used the *MinSum* cost function, the cost was better than the one found by Kulkarni and Tai. Furthermore, there is no evidence that Kulkarni and Tai tried to solve any problems larger than three agents and 15 cities, and so we have no evidence of the scalability of their solution.

2.3.4 Market-based Approach with Look-ahead Agents

A very interesting implementation of a market-based approach to the MTSP was suggested by Karmani et al. [25]. In their work, they propose an algorithm called Market-based Approach with Look-ahead Agents (MALA). Initially, tasks are randomly assigned to the agents. The agents then begin an iterative process where the agents compute their preferred route using a TSP solver, and then undesired tasks are traded between the agents. They compare their work with centralized and decentralized solutions and show significant improvement in runtime and cost.

MALA shares some properties with the method discussed in this document, but the two methods are not the same. The main similarities between the two methods are the use of an agent-to-agent trade to exchange tasks, an intra-agent TSP solver to compute the best tours for each agent, and that agents act egotistically.

However, there are some notable differences between their approach and the market-based solution proposed in this work. The first difference is that in MALA the problem is to maximize task coverage given strict fuel constraints, which is similar to the *MinMax* case, but we use no constraints and so the space of possible solutions is less restricted in our problem. More importantly, the trading mechanism in MALA is peer-to-peer, and there is no central exchange market. According to Karmani et al., there is a guarantee that an optimal solution can be achieved in a finite amount of peer-to-peer steps, however this may take a large number of such trades and also contradicts what we know about markets and their tendency to reach a *Pareto-optimal* solutions rather than a global optimal solution. Our combination of a central market and peer-to-peer trading can provide mechanisms to handle both problems.

3 Proposed Market-based Solution

The solution to the problem presented in Section 1.1 is based on a market in which there are three entities: the tasks, the agents and the group administrator. The tasks can be thought of as a simplified version of an agent: they encapsulate everything that there is to know about the task: its location, priority, type, and the associated benefit function, and they have a simple behavior, which is to always choose the agent whose bid is the lowest. The next subsections describe the behavior of the agents and the group administrator.

3.1 Agent Rules

In this work, the agent is represented by its properties, e.g., initial location, speed, and list of tasks to which it is assigned, and its behavior. The agent behavior is comprised of several basic operations: buying tasks, relinquishing tasks, exchanging tasks with other agents, and taking over all the assigned tasks of another agent. All the agents participate in an iterative market process where they perform all the aforementioned operations at each iteration. The market as a whole is described in Algorithm 1. Following is a short description of each basic operation.

Algorithm 1 Market Algorithm

```
Initialization
for all Tasks do
    Assign task to nearest agent
end for
while Iteration ≤ NumOfIterations do
    {Step 1: Market Auction}
    {Step 2: Agent-to-Agent Trade}
    {Step 3: Agent Takeover / Agents Switch}
    {Step 4: Agents Relinquish Tasks}
end while
return BestAgents, Assignments, Costs, Runtime
```

3.1.1 Bid and Auction

Algorithm 2 describes the bid and auction step. Available tasks are announced by the agents that release them at every iteration of the market process. This process can be done in a distributed way, including when the connectivity network is constrained, see for example [19]. Then, each agent announces its bid for a task, which is based on the cost the agent will incur for performing this task. To decide this difference, the agent checks the order in which the task will be performed

Algorithm 2 Step 1: Market Auction

```
Require: Tasks for bid from previous iteration
for all Tasks up for bid do
    for all Active Agents do
        Calculate Agent Bids {Depends on cost type}
    end for
    Assign task to lowest bidder
end for
Call CheckImprovement Procedure
```

relative to the existing order of the tasks the agent is already committed to perform. Then, the order which minimizes the cost will be assumed and the agent bids based on the new cost. For the *MinSum* case only the additional cost relative to the cost without the new task is the bid, and for the *MinMax* case the entire cost is the bid. The difference stems from the difference between optimizing the performance of the whole group, as in the *MinSum* case and optimizing the performance of each individual agent, as in the *MinMax* case.

Once the bids for all the available tasks are made by the agents, the tasks decide which agent will be assigned to perform them. Each task, in its turn, assigns itself to the agent that offered the lowest bid. This guarantees that the best agent, in terms of lowest cost, is assigned to each task. The chosen agent is then updated with the task and adds it to its list of assigned tasks. However, because this is a greedy behavior which only seeks to exploit cost reduction, this behavior can lead to a local minimum.

3.1.2 Agent-to-Agent Trade

Agents randomly consider the tasks of other agents. Denote agent A as the agent considering the tasks assigned to another agent, B. When agent A considers the tasks of agent B, it queries the ordered set of tasks that B is assigned to do. This allows A to consider the cost associated with each of B's tasks, and A can consider whether it can perform any of these tasks at a lower cost. If so, agent A takes this task over from agent B. This process, described in Algorithm 3, resembles a sub-contracting or trade mechanism in the free market, and is also a greedy process looking to exploit cost reduction for the whole group. Note that this is similar to the market process described in MALA [25].

In a distributed implementation of the market-based solution, this can be done in the following way: agents will periodically decide to seek for agent-to-agent bids. Agent A will decide to contact one of its *neighbors* (in the CBBA sense [9]) as agent B, and ask for its tour. Since the tasks are commonly known, agent A will be able to compute any improvement and ask agent B to transfer the task to agent A.

Algorithm 3 Step 2: Agent-to-Agent Trade

```
Agent1 ← random('unid', NumOfAgents)
{This agent will try to take tasks from Agent 2}
Agent2 ← random('unid', NumOfAgents)
for all Agent2.AssignedTasks do
  if Agent1 cost for Task ≤ Agent2 cost for Task then
    Agent1.AssignedTasks ←
    Agent1.AssignedTasks ∪ Task
    Agent1.AssignedTasks ←
    Agent2.AssignedTasks − Task
  end if
end for
Call CheckImprovement Procedure
```

3.1.3 Complete Takeover or Agent Switch

The next step in the market-based solution is trying to explore solutions that are not in the local minima. This is done differently in the *MinSum* and *MinMax* cases and is given in Algorithm 4.

In the *MinSum* case, the solution cost is the sum of the tour lengths of all the agents, and as a result it is beneficial in many cases to allow only one agent to make a long tour than to split the tours between agents. The reason for that is simple: an agent has also to return to its initial location, and then the agent incurs an extra cost. Therefore, it is generally good to see if one agent can take over all the tasks of another agent, as a whole. Note that this complete takeover might not be achieved simply by agent-to-agent trade, because sometimes only when all the tasks are taken the sum of all tour lengths will decrease, because only then will the agent losing the tasks have zero tour length.

For a *MinMax* case there is no point in taking over all the tasks of another agent, because if either agent is the one that currently owns the longest tour, the resulting tour would be longer, i.e., worse than the current tour, and if neither agent is the one that currently owns the longest tour than, at best, the resulting tour would not reduce the maximum length. Therefore, a different process is needed to stir the group from its current local minimum. The process used is by taking two randomly chosen agents, and having them

Algorithm 4 Step 3: Agent Takeover / Agents Switch

```
if CostType==MinSum then
  {Agent Takeover}
  Agent1 ← random('unid', NumOfAgents) {This agent will take over all the tasks assigned to Agent 2}
  Agent2 ← random('unid', NumOfAgents)
  Agent1.AssignedTasks ← Agent1.AssignedTasks ∪ Agent2.AssignedTasks
  Agent2.AssignedTasks ← ∅
end if
if CostType==MinMax then
  {Agents Switch}
  Agent1 ← random('unid', NumOfAgents)
  Agent2 ← random('unid', NumOfAgents)
  for all Agent1.Tasks do
    if Task inside the polygon defined by Agent2 and Agent2.AssignedTasks then
      Agent2.AssignedTasks ← Agent2.AssignedTasks ∪ Task
    end if
  end for
  for all Agent2.Tasks do
    if Task inside the polygon defined by Agent1 and Agent1.AssignedTasks (before the switch) then
      Agent1.AssignedTasks ← Agent1.AssignedTasks ∪ Task
    end if
  end for
end if
```

exchange tasks between themselves. This method works on an agent-to-agent basis and not centrally via the market, thus providing better chances of improving the result. The general idea is that each agent considers the other agent's tasks and picks the ones that may possibly fit better with their own tour, for example tasks that are encircled within the tour. This not necessarily will result in a shorter tour, but has a good chance of doing so. At the end of this step, the current performance of the group as a whole is examined by a group administrator (see Section 3.2) before the agents start to give tasks away for the next market iteration.

3.1.4 Reordering the Tasks

Following each step, the agents calculate the cost of their current tours by invoking an algorithm to solve the single traveling salesman problem (TSP). Any TSP solution can be used, and, in fact, in the current implementation there are three algorithms used: Dynamic Programming based on Held and Karp [21] for small tours of up to 5 tasks, a solution based on Convex Hull and Nearest Neighbor [16] for larger problems, chosen for its quick near-optimal solutions, and for larger tours, where the Convex Hull fails to get good results, we use Concorde [1]. The reason not to use Concorde for all the cases is that Concorde is very verbose during run, and actually takes more time to run for small problems.

3.1.5 Relinquishing Tasks

Following this process, the agent recalculates the cost associated with each assigned task, and chooses to relinquish some of the tasks assigned to it, see Algorithm 5. There are two ways to choose the tasks to be given away: a greedy way, which will relinquish the tasks that cause the highest increase in cost, or a blind random selection. Agents combine both ways by randomly choosing tasks to relinquish, with the probability assigned

Algorithm 5 Step 4: Agents Relinquish Tasks

```
TasksForBid ← ∅
for all Agents do
  NumTasksToGive ← min(MaxNumOfTasksToTrade, PartOfTasksToTrade ∗
  |Agent.AssignedTasks|)
  while NumTasksRelinquished ≤ NumTasksToGive do
    for all Agent.AssignedTasks do
      MargnialCost ← Agent Cost with and without Task
      Choose a task to give with probability proportional to MarginalCost
      Agent.AssignedTasks ← Agent.AssignedTasks − Task
      TasksForBid ← TasksForBid ∪ Task
    end for
  end while
end for
```

to each task proportional to the cost incurred by adding that task. The relinquished tasks from all the agents are accumulated and serve as the basis for the next iteration.

3.2 Group Administrator

The group administrator can be thought of as a team leader that keeps track of the performance of the whole team. This function can be assigned to one of the agents in the real world, to an external controller, or even can run in a distributed manner by each of the agents. It is worth noting that the administrator is not interested in the details of each agent. In fact, the group administrator makes only one decision: whether the result of an iteration is better than the result of the best iteration so far, and if so, it prompts the agents to save their new list of assigned tasks for each agent.

In the way the algorithm is implemented now, the agents communicate their current cost (calculated after Task Reordering) to the administrator. The administrator calculates the cost for the group based on Eqs. (2) and (3), compares that to the previous best cost, and, if the current cost is lower, each agent is prompted to save its best known solution so far. If the cost is not improved, the group administrator can decide to return to the best known assignment so far and the next market iteration starts from that assignment.

However, the same logic can also be implemented in a completely decentralized way, where the agents broadcast their current cost to all the other agents. Then, each agent performs the same group cost calculation, based on Eqs. (2) and (3), and, in essence, each agent performs the administrator role for itself, deciding whether to revert to a previous best solution or keep the new solution. Since all the agents share the same data, their behavior will be the same. Note that this way the solution is completely distributed and decentralized, and can be calculated in parallel.

While the administrator role is the only group-wide function in the solution, it only deals with a higher level aspect of the solution, leaving the agents to deal with lower level decisions and calculations. Therefore, this function does not limit the scalability of the solution, because it scales linearly with the number of agents in communication, memory and computation time. This function is similar to the function that keeps the best chromosome of a genetic algorithm solution. The algorithm is given in Algorithm 6.

3.3 Termination

Similar to many other iterative solutions, the market-based solution is terminated when the solution does not show any improvement in the last p iterations, where p is a predefined parameter. We use the value of 30 for this parameter in this work, as it has shown to prevent premature termination of the algorithm. We found no need to vary this parameter with the size of the problem.

Algorithm 6 CheckImprovement Procedure

```
for all Agents do
    Order Agent.AssignedTasks {Invokes a TSP solver}
    Calculate total cost and benefits
end for
if NewCost ≤ BestCost then
    {Or best profit if benefits are used}
    BestAgents ← Agents
    BestCost ← NewCost
    Iteration ← 0
else
    Agents ← BestAgents {Only if ReturnToBest==True}
    Iteration ← Iteration + 1
end if
```

4 Results

The ability of the proposed solution to solve the problem was compared to two types of solutions. The first type of solution is the guaranteed optimal solution, which was computed using direct enumeration. Obviously, due to its computational complexity, this solution was obtained only for a relatively simple case, which includes 3 agents performing 8 tasks, denoted as 3×8. A detailed statistical analysis of this case is provided to demonstrate the near-optimality of the market-based solution.

More complicated cases, which are computationally intractable to directly enumerate, are presented in order to show the scalability of the market-based solution. These cases are compared with other near-optimal solutions: the *MinSum* case is compared with Binary Programming and the *MinMax* case is compared with a solution given by Carlsson et al. [8]. Finally, runs of very large problem sets are shown to demonstrate the scalability of the market-based solution.

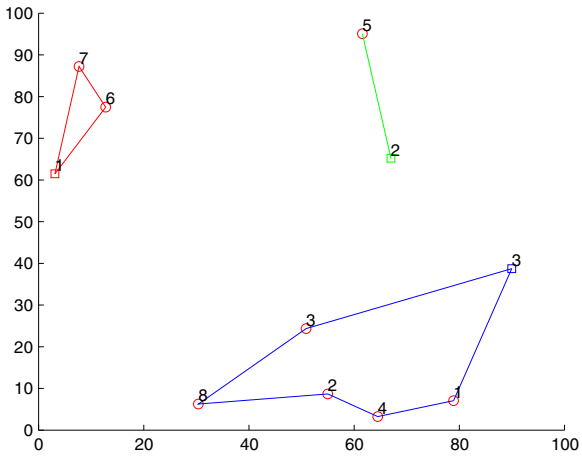
In addition, the adaptability of the market-based solution to changes in the scenario is demonstrated by showing an example of a computation run in which one of the agents is removed from the scenario, leaving two other agents in the market, and then this agent is reintroduced to the market and the three agents solve the problem.

The performance of the market-based solution is measured using three measurements: the quality of the solution vs. the optimal (or approximate) solution, the number of market iterations required to achieve the solution and the time required for the algorithm to run. All the cases were run on a single laptop, and written in MATLAB® running Microsoft Windows XP® operating system on a dual core 2.25 GHz processor machine with 3 GB of memory. However, the inherent encapsulation of the agents in a market-based solution lends itself to parallelization using several machines.

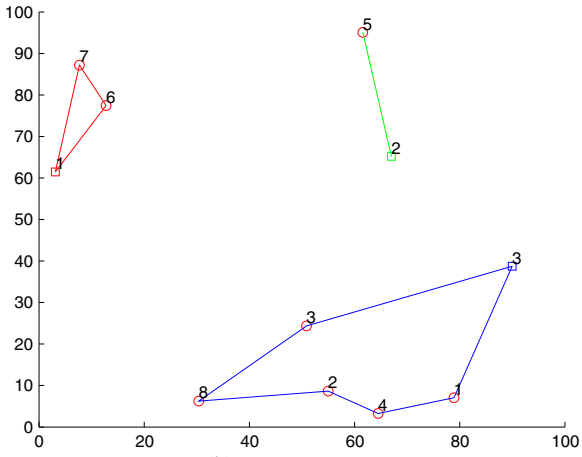
4.1 Optimality

The result for the minimum sum of agent travel distances, as in Eq. 2, denoted *MinSum*, or for the minimum of the longest agent tour, denoted *MinMax*, as in Eq. 3 are quantitatively compared with the market-based solution and the optimal enumeration. A Monte Carlo simulation of 200 scenarios is performed with all four cases: direct enumeration vs. market-based solution, for *MinSum* and *MinMax*. Here, a *scenario* is defined as a given set of agent locations and task locations and a *run* is defined as one solution instance of a scenario using the market-based solution.

Figure 1 depicts an example of a *MinSum* solution for the 3×8 case. This figure shows that the market-based solution achieved the same result of the optimal solution obtained by direct enumeration. As can be seen in Fig. 2, this happens in

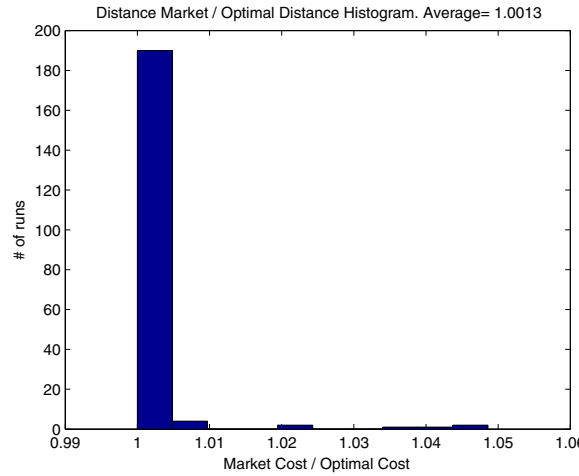


(a) Assignment using the market solution.

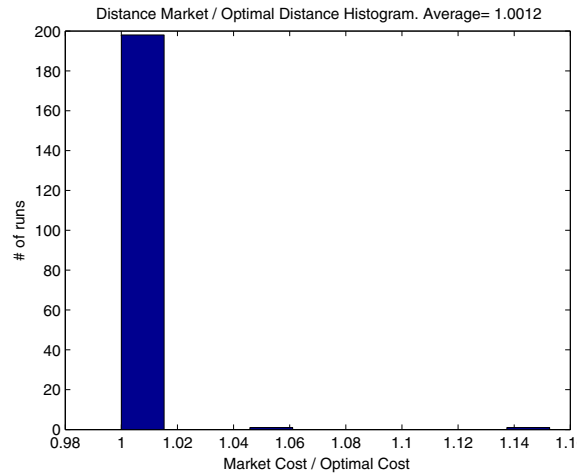


(b) Optimal assignment.

Fig. 1 Results of a 3-by-8 minimum sum case



(a) *MinSum* performance



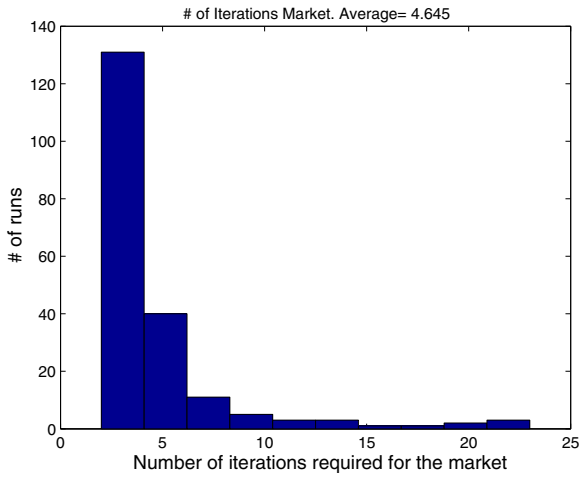
(b) *MinMax* performance

Fig. 2 Histograms of the market based solution quality vs. the optimal solution in the 3×8 *MinSum* and *MinMax* cases

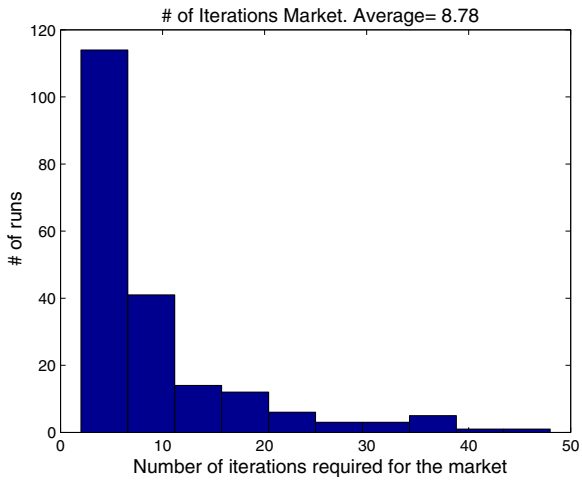
more than 98 % of the scenarios in the *MinMax* case and more than 95 % of the scenarios in the *MinSum* case. On average, the market-based solution was less than 0.2 % higher the optimal solution and the worst solutions, in 200 scenarios, were 10 % to 15 % off in each case. Moreover, these results were obtained very quickly, as can be seen in Fig. 3: it takes, on average, only 8.78 iterations in the *MinMax* case and 4.645 iterations in the *MinSum* case, to find the best solution. Taking into account that each iteration in this 3×8 case executes in about 0.3 s, the solutions are found within 1–2 s. To guarantee that a final result is obtained, the algorithm is allowed to run for 30 more iterations after the best solution is found, so total runtime for each case is higher.

4.2 Statistical Analysis

In the previous subsection the optimality of this solution was shown, based on the results of 200 scenarios. We now continue to statistical analysis based on a Monte-Carlo simulation of 20 scenarios that were run 200 runs each. To measure the performance, the percent of runs that came within $\epsilon = 1$ % of the cost function, as a function of

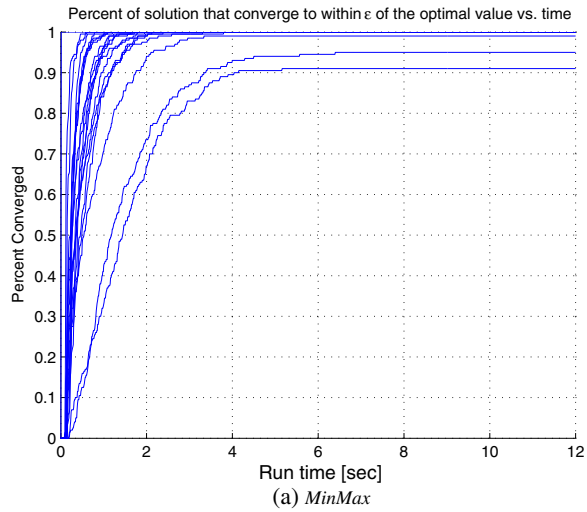


(a) *MinSum* iterations

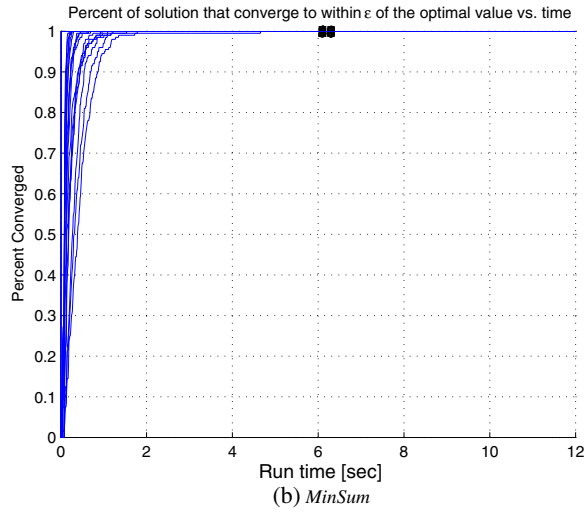


(b) *MinMax* iterations

Fig. 3 Histograms of the number of iterations required for the market algorithm to obtain the best solution in the 3×8 *MinSum* and *MinMax* cases



(a) *MinMax*



(b) *MinSum*

Fig. 4 Results of 20 3×8 scenarios, 200 runs each. Percent of the runs that came within $\epsilon = 1$ % of the optimal cost

time, is computed, as depicted in Fig. 4. Runtime is about 2 s for the *MinSum* case and up to 4 s for the *MinMax* case. For the *MinSum* case, all the runs per each scenario converged to within 1 % of the optimal cost. The *MinMax* case exhibits some variety in the results: most scenarios can be calculated quite accurately, but some have a lower percentage getting to within 1 % of the optimal cost. However, note that of the 20 scenarios, 17 had accurate results, and the other 3 reached the accurate result in more than 90 % of the runs. The reason for that is, presumably, that the *MinMax* case is very susceptible to converging to local *Pareto minima*, and getting away from these local minima is harder, because it requires usually more than one exchange between agents.

4.3 Comparison of the *MinSum* Case to Binary Programming

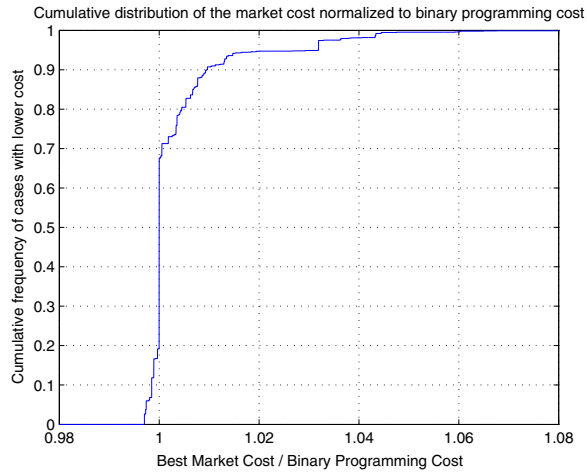
For larger problems, it is impossible to enumerate all the possible solutions, so an approximate solution has to be used. For the *MinSum* case, in which the total cost of travel of all agents is to be minimized, the problem can be formulated as an integer programming problem [4]. We implemented a

binary programming problem based on the formulation proposed in [13] and using the optimization toolbox offered with MATLAB. Unfortunately, integer programming formulations are not easily solved for the *MinMax* case [7] so this subsection will only compare the *MinSum* case to the binary solution. It should be noted that many authors consider integer (and binary) solutions to be accurate, or optimal.

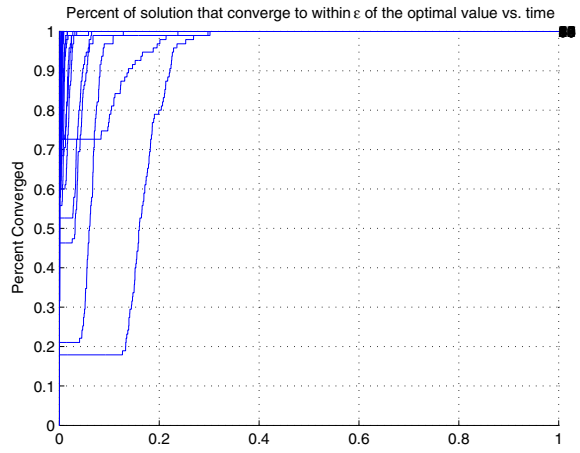
We generate 20 scenarios with 5 agents and 30 tasks. Each scenario was solved by the binary programming algorithm once to get the binary programming cost and runtime for each scenario. These results serve as a baseline for comparison, and the market-based solution was then run 90 times per each of these scenarios, for a total of 1800 runs. For each scenario we calculate the following parameters:

- Normalized cost = the cost calculated by the market-based solution divided by the cost calculated by the binary programming solution.
- Normalized runtime = the runtime required by the market-based solution divided by the runtime required by the binary programming solution.
- Cumulative distribution = the fraction of runs whose normalized cost is less than or equal to a certain normalized cost value, out of the total number of runs.

Figure 5a depicts the results of the binary programming comparison to the market-based solution for the scenarios of 5 agents and 30 tasks. On the left, the cumulative distribution of the market-based solution shows that in about 20 % of the cases, the market-based solution improved the result of the binary programming solution, albeit by a small fraction of less than 0.5 %, probably because the binary programming solution uses rounding in its branch-and-bound techniques that makes several solutions seem identical to the binary programming when in fact they are not. In addition, about 68 % of the runs finished with the market-based solution having the same cost or better, and more than 90 % of the runs finished with the market-based solution getting a cost which is up to 1 % higher the binary programming cost. In the worst run, out of all the runs, the cost found by the market-based solution is only



(a) Cumulative Distribution



(b) Percent Converged vs. Time

Fig. 5 Comparison of the Binary Programming solution to the Market-Based solution for 5 agents, 30 tasks, *MinSum*. Top: cumulative distribution vs. normalized cost. Bottom: the percent of cases that converged to with 9 % of the Binary Programming solution cost vs. time

8.5 % higher than the cost found by the binary programming solution for that case.

In Fig. 5b, the two solutions are compared as a function of the normalized runtime using the percent converged metric for a normalized cost of 9 %, chosen so that all the cases will converge. The graph shows that in all the runs the market-based solution converged to within that boundary in less than 30 % of the time required by the binary programming solution. In fact, as most of the runs are centered in the top left corner, only a small fraction of the runs even required more

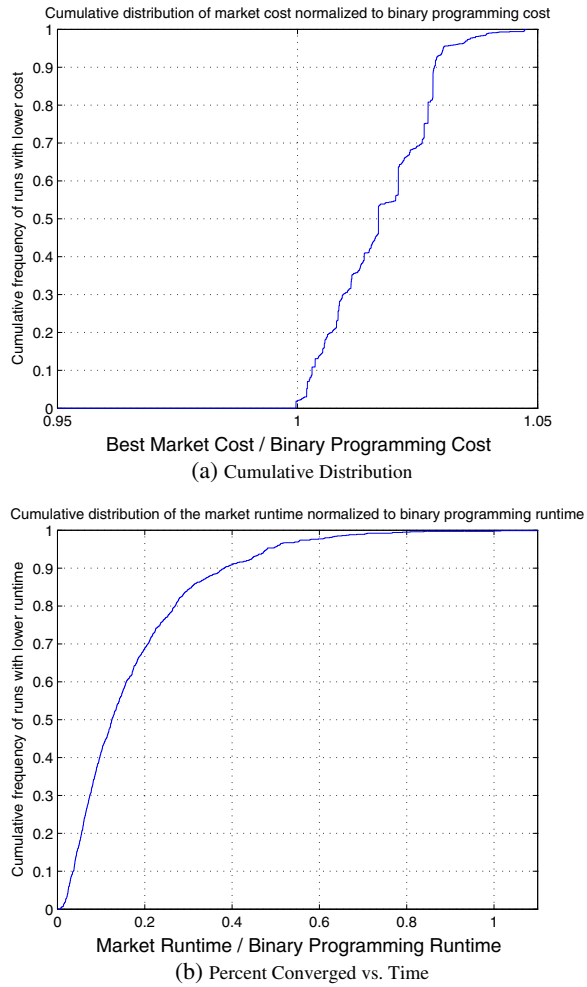
than 10 % of the runtime required by the binary programming solution.

The main reason for the vast improvement in normalized runtime is because the market-based solution scales significantly better than the binary programming solution to larger problems. While the average runtime of the market-based solution increases from about 2 s in the 3 agents and 8 tasks scenarios to about 60 s in the 5 agents and 30 tasks, the runtime of the binary programming solution increased from about 4 s in the smaller scenario to between 38 and 11813 s in the larger scenario, with an average of 1915 s. This shows that not only has the average runtime required for the binary programming solution increased at a much higher rate, but the variation increased significantly as well. Both these phenomena are expected to get only worse as the size of the problem increases, and with it, the number of variables used in the binary programming solution.

To further test the scalability of the market-based solution, we increased the size of the problem to 20 agents and 100 tasks. To obtain solutions for a problem of this size, we use IBM Ilog CPLEX [22], which is a well-known linear programming solver suite. We ran 10 scenarios with 100 runs each for a total of 1000 runs on a faster machine with Intel iCore5, 2.5 GHz, processor and 6 GB RAM. The runtime for the binary solution obtained by CPLEX ranged from 68 to 392 s, which shows the strength of this commercial optimization suite.

Figure 6 depicts the results of the market-based solution normalized to the results of the binary programming solution. From the cost point of view, the median obtained by the market-based solution is about 1.7 % higher than the binary programming cost and the worst case is 4.8 % higher. In 99.7 % of the runs, the runtime for the market-based solution is shorter than the CPLEX runtime. The median normalized runtime is 0.125, i.e., the market-based solution is about 8 times faster than the CPLEX runtime. These results are particularly impressive because the market-based is written in MATLAB and was not optimized and compiled like CPLEX – a commercially used solver.

From these results it is concluded that the market-based solution is comparable to the binary programming in terms of cost, and improves significantly the runtime relative to binary programming.



(a) Cumulative Distribution

(b) Percent Converged vs. Time

Fig. 6 Comparison of the Binary Programming solution to the Market-Based solution for 20 agents, 100 tasks, *MinSum*. Top: cumulative distribution vs. normalized cost. Bottom: cumulative distribution vs. normalized time

4.4 Comparison of the *MinMax* Case to Carlsson's Method

Next, a comparison of the market-based solution to the *MinMax* problem is done with respect to the well-known solution by Carlsson et al. [8], which is considered to be the best known solution to this case. To do that, several scenarios have been generated, all with a uniform distribution

of tasks and agents. The problems range from a single depot, three agents and 100 tasks to 50 depots, 50 agents and 500 tasks. In cases where the number of depots is smaller than the number of agents, we assume that the agents are evenly distributed between the depots. As both the Carlsson algorithm and the market-based algorithm use some randomness in the process, both were run 10 times for each scenario.

Table 1 summarize the results of these runs. Each case is described by the number of different depot locations, the number of agents, and the number of tasks. Then the results of each algorithm are given by three performance measures: the best maximum length found by the algorithm in the 10 runs, the average maximum length by the algorithm in those runs, and the average time required for each run. It is obvious that Carlsson's algorithm has a great runtime in all the cases, and is at least one order of magnitude better than the runtime required for the market-based solution. This is the result of the following differences: the use of a compiled code for most of the recurring calculations; and, most importantly, the iterative nature of the market-based solution, and presumably the excessive number of iterations the market-based solution is allowed before finishing.

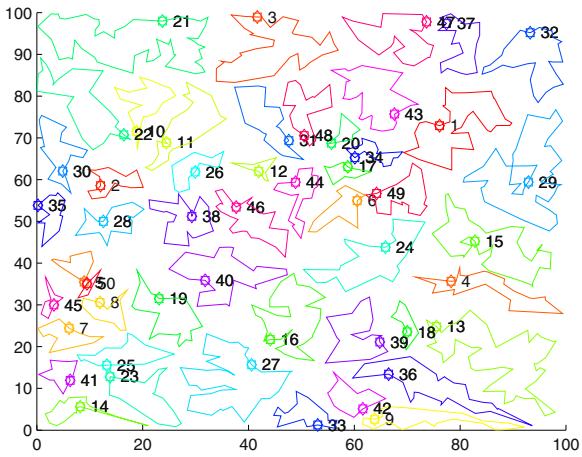
On the other hand, the market-based solution achieves better costs in all the cases, and the differences range from 5 % in the simplest *d1v3c100* case to about 40 % in the *d50v50c500* case. Moreover, the average maximum cost that the market-based solution achieves is better than the best maximum value that Carlsson's algorithm achieves in all the cases except *d1v3c100*, and the market average is usually closer to the market's best value than Carlsson's average is to Carlsson's best value. It should also be noted that in most cases the market-based solution spends more than 90 % of the time reducing the cost by about 5 %, for example see Fig. 8, it takes roughly the same amount of time to get the same cost as the cost given by Carlsson's algorithm. The only difference is that the market-based solution continues to improve over that cost if more time is given to the calculations.

4.5 Handling Very Large Problems

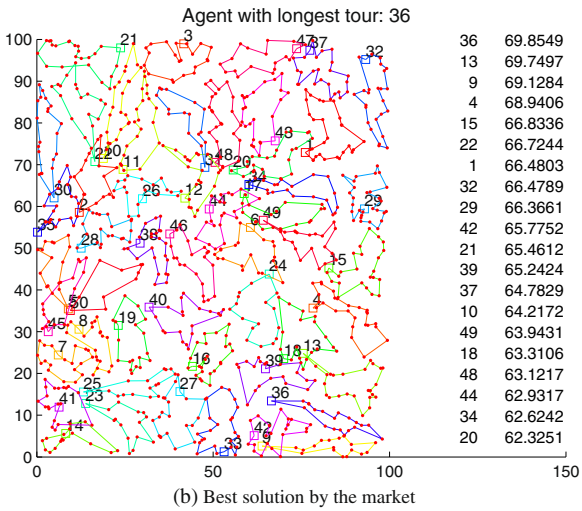
So far, the results shown are discussed vs. other methods, but, in order to demonstrate the ability of the market-based solution to handle very large scenarios, results were obtained for scenarios that include up to 200 agents and 2000 tasks for both the *MinSum* and the *MinMax* cases. Figure 7 depicts the results of one such *MinMax* run for 50 agents and 1000 tasks. The top part of the figure depicts the initial guess, achieved by assigning the nearest agent to each task. This assignment is essentially a Voronoi tessellation of the space, and the market uses it as an initial guess. This initial assignment is very quick, and allows a much better initial solution relative to a randomly

Table 1 Comparison of *MinMax* performance with Carlsson's algorithm

Case	Depots	Agents	Tasks	Carlsson best max	Carlsson avg. max	Carlsson avg. time	Market best max	Market avg. max	Market avg. time
d1v3c100	1	3	100	318.0216	335.8102	1.910	302.9337	322.7905	57.745
d1v10c100	1	10	100	74.4595	80.3051	5.346	68.9637	70.9767	165.6386
d5v5c100	5	5	100	221.2886	229.9663	2.755	177.8745	188.2310	61.9937
d5v5c200	5	5	200	266.6565	282.2576	3.710	240.9694	254.3151	356.5359
d5v10c200	5	10	200	179.1605	201.9222	6.357	144.8087	152.3875	360.3706
d5v20c200	5	20	200	133.6722	148.2029	11.584	116.2462	118.4978	938.154
d10v10c100	10	10	100	140.9398	164.7928	5.466	106.8377	109.9168	173.0095
d10v10c200	10	10	200	166.8934	175.1186	6.263	136.4858	139.3154	564.495
d10v10c500	10	10	500	202.5309	215.5552	7.600	190.6173	201.6343	1227.7721
d20v20c200	20	20	200	110.1314	118.8876	11.577	79.1036	80.4959	412.0668
d20v20c500	20	20	500	132.3455	166.6723	14.006	98.5991	102.0614	1462.8587
d50v50c500	50	50	500	97.8038	122.6068	30.960	56.5013	58.0391	5114.143



(a) Initial guess by nearest agent approach (Voronoi)



(b) Best solution by the market

Fig. 7 Results of a run with 50 agents and 1000 tasks, *MinMax* case. Top: the initial guess. Bottom: the best solution achieved by the market-based solution. To reduce clutter, task locations are shown in red dots. On the right: agent numbers and corresponding tour lengths

chosen assignment from which the market can start the trading. The best solution for the same scenario is shown in Fig. 7b, and it is easily seen that the differences in tour lengths are very small among the top 20 longest tours, given on the right column. This shows that there is very little room for improvement, because in order to reduce the longest tour, one other agent will have to take a task from the agent holding the longest tour, thus increasing the tour length of the agent taking the task. Since the differences are so small, this might cause the tour length of the agent taking the task to be even longer than the optimal tour we already have, and so this will result in a deterioration of the solution and is then ignored by the Administrator (see Section 3.2).

The improvement in the maximum length provided by this solution is shown in Fig. 8. Note how the initial guess, which seems at first glance to be a good guess, has a much higher cost than the best solution achieved by the market. In fact, the market reduced the maximum tour length by more than 50 % relative to the initial guess. It is also obvious that after about 250–300 s, which are about 10 % of the total 3000 s it took to run this scenario, the cost is already within 5 % of the best cost.

To complete the discussion of larger problems, a *MinSum* scenario with 50 agents and 1000 cities is presented in Figs. 9 and 10. It is worth noting that in the *MinSum* case agents often take over all the tasks of other agents, leaving them without any tasks, because then the cost of traveling back and forth to the task is removed. In fact, in smaller problems it is often beneficial to have only one agent perform all the tasks, but in such a large problem like the one here, this is not the case, because having too many tasks often requires the route to become spiral.

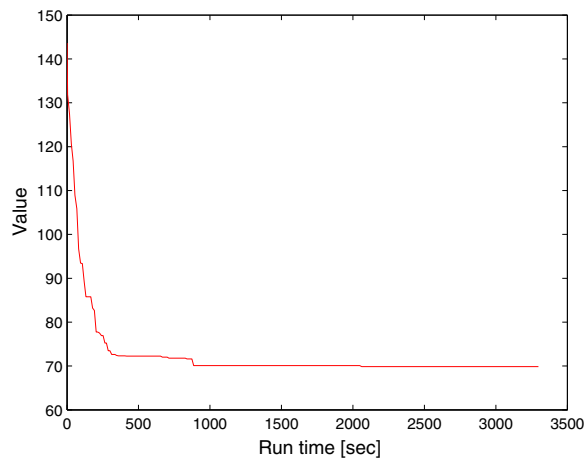
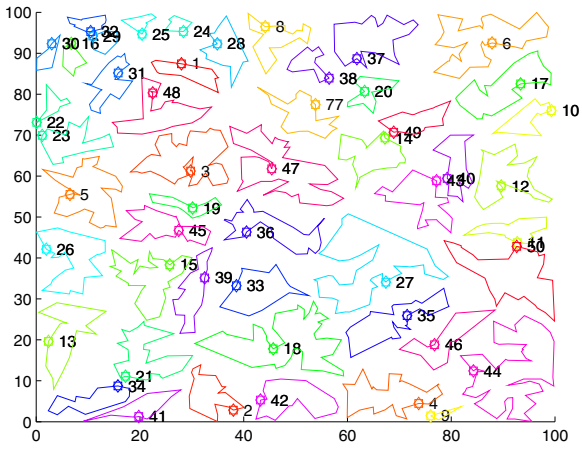
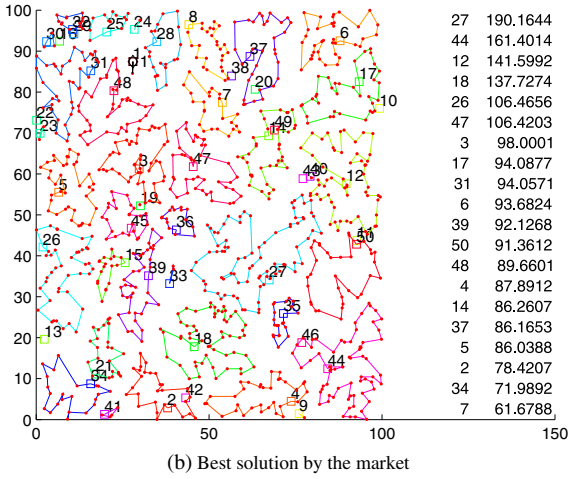


Fig. 8 Time history of the maximum tour length in the 50 agents and 1000 tasks *MinMax* run



(a) Initial guess by nearest agent approach (Voronoi)



(b) Best solution by the market

Fig. 9 Results of a run with 50 agents and 1000 tasks, *MinSum* case. Top: the initial guess. Bottom: the best solution achieved by the market-based solution. To reduce clutter, task locations are shown in red dots. On the right: agent numbers and corresponding tour lengths

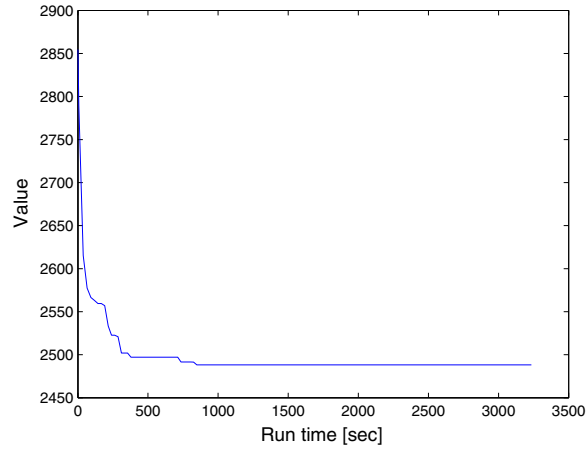


Fig. 10 Time history of the sum of tour lengths in the 50 agents and 1000 tasks *MinSum* run

4.6 Adaptability

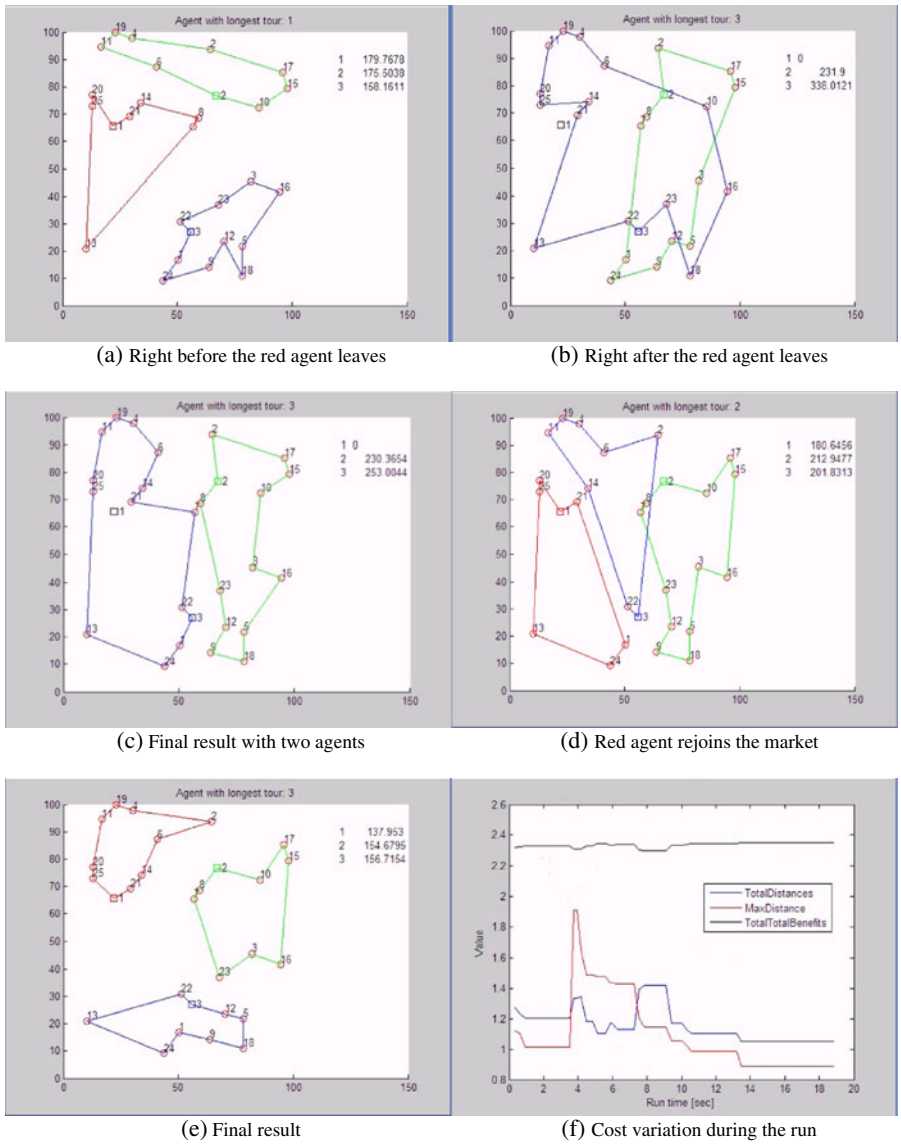
In real life, scenarios change over time. In particular, agents may become available or unavailable during the execution of the scenario, new tasks may become known and old ones may become irrelevant. These changes in the scenario are of particular interest in military operations and transportation operations, when new targets (military) or customers (transportation) become known or picked up, and when vehicles become available or malfunction during the scenario. Most existing solutions to the MTSP, however, do not allow these parameters to be changed during the run of the algorithm, and, if any change is made to the scenario, the algorithm should be restarted, and might never finish execution if the changes happen in shorter time spans than the time required for calculating the solution.

The market-based solution offers a simple and intuitive way to deal with these changes, using a set of simple rules added to the basic rules listed in Section 3:

- Agent arrival rule: if an agent becomes available, let it join the bidding process.
- Agent departure rule: if an agent becomes unavailable, add its tasks to the list of tasks to bid on in the next iteration and exclude it from further bidding.
- Task arrival rule: if a new task becomes available, add it to the list of tasks.
- Task departure rule: if a task is no longer relevant, remove it from an agent's task-list and from the market.

The key here is that the market-based solution has the inherent structure to accommodate these changes, during operation, while taking advantage of solutions that were obtained in previous steps.

Figure 11 shows an example of such a scenario run. The scenario starts with 3 agents and 15 tasks. After 25 iterations, the red agent leaves the market, and the other two agents divide the tasks between them. At iteration 50, the red agent



(a) Right before the red agent leaves

(b) Right after the red agent leaves

(c) Final result with two agents

(d) Red agent rejoins the market

(e) Final result

(f) Cost variation during the run

Fig. 11 Results of a scenario run with an agent leaving and rejoining the market. On the right: agent numbers and corresponding tour lengths

rejoins the market and bids on the tasks. The market then continues to run until a final result is found. Clearly, as seen in Figure 11f, this final result has a better cost than the result obtained by the market before the iteration in which the agent was removed from the scenario and it also has a better result relative to the non-market solution, as shown by the cost ratio being less than 1. Thus, we show the ability of this market-based solution to deal with changes in the scenario.

In a separate work [27], we compared the adaptability of the market-based solution with the adaptability of genetic algorithms. Using a set of 200 scenarios at 4 different rates of events, it was shown that the market-based solution is superior to genetic algorithms in its ability to adapt to changes in the scenario, and therefore improved the overall time required to perform missions. For more details please refer to [27].

5 Conclusions

A task assignment algorithm was developed using a market-based solution, in which agents

operate in bidding on tasks and exchanging them among themselves. The task assignment algorithm shows near optimal results in comparison to a known optimal solution for small scenarios, for which an optimal solution can easily be calculated. Initial results regarding the scalability were obtained by comparing the market-based solution to other existing sub-optimal solutions, in which the market-based solution achieves better results in most cases. The market-based solution provides an excellent paradigm for handling changes in the scenario during runtime, making the solution adaptable to such changes.

There are a few limitations to the proposed market-based solution. Generally, there is no guarantee on runtime and quality of the solution. We have shown that the cost drops rather quickly in the first few iterations, and that the rate of improvement slows down as the solution continues. Thus, for applications that have rigid time constraints, one can decide to stop the solution process after a predefined period and use the best known assignment. In addition, we have not implemented any vehicle constraints in the solution, therefore the current solution is unable to handle instances of the capacitated vehicle routing problem or problems where there are constraints on time windows and/or fuel consumption. We believe that these constraints can be added to the logic of each agent and left it for future work. Lastly, at this point there is full connectivity between the agents. This helps the convergence of the agents, and in future work we would like to add connectivity constraints and compare our results with methods that allow such constraints, e.g. the CBBA solution.

It seems that the market paradigm provides a useful way to handle task priorities and time criticality, using tasks with time-varying benefits, and agents seeking to maximize their profits. A future work will be to extend the market-based solution in this direction. Finally, more work needs to be done to reduce calculation time in large problems and provide better scalability.

References

1. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: Concorde. http://www.tsp.gatech.edu/concorde/index.html (2004)
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study, 1 edn. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, New Jersey (2006)
3. Baker, A.D.: Market-based control: A Paradigm for Distributed Resource Allocation, chap. Metaphor or Reality: A Case Study Where Agents Bid with Actual Costs to Schedule a Factory, pp. 184–223. 9810222548. World Scientific Publishing Co. Pte. Ltd., P O Box 128, Farrer Road, Singapore 912805 (1996)
4. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega **34**(3), 209–219 (2006)
5. Bertsimas, D.J., Ryzin, G.V.: A stochastic and dynamic vehicle routing problem in the euclidean plane. Oper. Res. **39**(4), 601–615 (1991)
6. Bullo, F., Frazzoli, E., Pavone, M., Savla, K., Smith, S.L.: Dynamic vehicle routing for robotic systems. Proc. IEEE **99**(9), 1482–1504 (2011)
7. Campbell, A.M., Vandenbussche, D., Hermann, W.: Routing for relief efforts. Transp. Sci. **42**(2), 127–145 (2008)
8. Carlsson, J., Ge, D., Subramaniam, A.: Lectures on global optimization. In: Chap. Solving Min-Max Multi-Depot Vehicle Routing Problem. Fields Institute Communications, illustrated edn., vol. 55, pp. 31–46. American Mathematical Society (2009)
9. Choi, H.L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE Trans. Robot. **25**(4), 912–926 (2009)
10. Clearwater, S.H.: Market-Based Control: A Paradigm for Distributed Resource Allocation, chap. Preface, pp. v–xi. 9810222548. World Scientific Publishing Co. Pte. Ltd., Singapore (1996)
11. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B **26**(1), 29–41 (1996)
12. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: A taxonomy for multi-agent robotics. Auton. Robots **3**, 375–397 (1996)
13. Feillet, D., Dejax, P., Gendreau, M.: Traveling salesman problem with profits. Transp. Sci. **39**(2), 188–205 (2005). doi:10.1287/trsc.1030.0079
14. Ferguson, D.F., Nickolaou, C., Sairamesh, J., Yemini, Y.: Market-Based Control: A Paradigm for Distributed Resource Allocation, chap. Economic Models for Allocating Resources in Computer Systems, pp. 156–183, 9810222548. World Scientific Publishing Co. Pte. Ltd., Singapore (1996)
15. Gagliano, R.A., Mitchem, P.A.: Market-Based Control: a Paradigm for Distributed Resource Allocation, chap. Valuation of Network Computing Resources, pp. 28–52, 9810222548. World Scientific Publishing Co. Pte. Ltd., Singapore (1996)
16. Giaccari, L.: Tspconvhull. MATLAB Central (2008)
17. Golden, B.L., Laporte, G., Taillard, E.D.: An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. Comput. Oper. Res. **24**(5), 445–452 (1997)

18. Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. Bull. Am. Math. Soc. **64**, 275–278 (1958)
19. Gurfil, P., Kivelevitch, E.: Flock properties effect on task assignment and formation flying of cooperating unmanned aerial vehicles. In: Proceedings of IMechE. Part G: J. Aerospace Engineering, vol. 221, pp. 401–418. Institute of Mechanical Engineers (2007). doi:10.1243/09544100JAERO120
20. Harty, K., Cheriton, D.: Market-Based Control: A Paradigm for Distributed Resource Allocation, chap. A Market Approach to Operating System Memory Allocation, pp. 126–155, 9810222548. World Scientific Publishing Co. Pte. Ltd., Singapore (1996)
21. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. Soc. Ind. Appl. Math. **10**(1), 196–210 (1962)
22. IBM: Cplex. Web. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ (2012). Accessed 10 July 2012
23. Jaillet, P., Wagner, M.R.: Generalized online routing: new competitive ratios, resource augmentation, and asymptotic analyses. Oper. Res. **56**(3), 745–757 (2008)
24. Johnson, D.S., Mcgeoch, L.A.: Local Search in Combinatorial Optimization, chap. The Traveling Salesman Problem: A Case Study in Local Optimization, pp. 215–310. John Wiley and Sons, London (1997)
25. Karmani, R.K., Latvala, T., Agha, G.: On scaling multiagent task reallocation using market-based approach. In: Proceedings of the First IEEE International Conference on Self-adaptive and Self-organizing Systems, pp. 173–182 (2007)
26. Kirkpatrick, S., Gelatt, C.D. Jr., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
27. Kivelevitch, E., Cohen, K., Kumar, M.: Comparing the robustness of market-based task assignment to genetic algorithm. In: Proceedings of the 2012 AIAA Infotech@Aerospace Conference. AIAA, AIAA. AIAA-2012-2451 (2012)
28. Kulkarni, A.J., Tai, K.: Probability collectives: a multi-agent approach for solving combinatorial optimization problems. Appl. Soft Comput. **10**, 759–771 (2010). doi:10.1016/j.asoc.2009.09.006. http://www.sciencedirect.com/science/article/pii/S1568494609001665
29. Kuwabara, K., Ishida, T., Nishibe, Y., Suda, T.: Market-Based Control: A Paradigm for Distributed Resource Allocation, chap. An Equilibratory Market-Based Approach for Distributed Resource Allocation and Its Application to Communication Network Control, pp. 53–73, 9810222548. World Scientific Publishing Co. Pte. Ltd., Singapore (1996)
30. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**(2), 498–516 (1973)
31. Miliotis, P.: Using cutting planes to solve the symmetric travelling salesman problem. Math. Program. **15**, 177–188 (1978). doi:10.1007/BF01609016
32. Mudrov, V.: A method of solution of the traveling salesman problem by means of integer linear programming (the problem of finding the hamiltonian paths of shortest length in a complete graph). Zhurnal Vychislennoi Fiziki (USSR). Abstract in: Int. Abstr. Oper. Res. **5**(3), 1137–1139 (1965)
33. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Rev. **33**, 60–100 (1991). doi:10.1137/1033004. http://dl.acm.org/citation.cfm?id=103864.103868
34. Passino, K., Polycarpou, M., Jacques, D., Pachter, M., Liu, Y., Yang, Y., Flint, M., Baum, M.: Cooperative control for autonomous air vehicles. In: Proceedings of the Cooperative Control Workshop, Florida (2000)
35. Rasmussen, S., Chandler, P., Mitchell, J.W., Schumacher, C., Sparks, A.: Optimal vs. heuristic assignment of cooperative autonomous unmanned air vehicles. In: Proceedings of the AIAA Guidance, Navigation & Control Conference (2003)
36. Schumacher, C., Chandler, P., Pachter, M.: Uav task assignment with timing constraints. AFRL-VA-WP-TP-2003-315. United States Air Force Research Laboratory (2003)
37. Schumacher, C., Chandler, P.R., Rasmussen, S.: Task allocation for wide area search munitions via network flow optimization. In: Proceedings of the 2001 AIAA Guidance, Navigation, and Control Conference (2001)
38. Shima, T., Rasmussen, S.J., Sparks, A.G., Passino, K.M.: Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. Comput. Oper. Res. **33**(11), 3252–3269 (2006). doi:10.1016/j.cor.2005.02.039. Part Special Issue: Operations Research and Data Mining
39. Wooldridge, M.: An Introduction to Multi Agent Systems, 1 edn. John Wiley & Sons, England (2002)

View publication stats