Distributed Task Rescheduling With Time Constraints for the Optimization of Total Task Allocations in a Multirobot System

Joanna Turner, Qinggang Meng, *Member, IEEE*, Gerald Schaefer, *Member, IEEE*, Amanda Whitbrook, and Andrea Soltoggio

***Abstract*—This paper considers the problem of maximizing the number of task allocations in a distributed multirobot system under strict time constraints, where other optimization objectives need also be considered. It builds upon existing distributed task allocation algorithms, extending them with a novel method for maximizing the number of task assignments. The fundamental idea is that a task assignment to a robot has a high cost if its reassignment to another robot creates a feasible time slot for unallocated tasks. Multiple reassignments among networked robots may be required to create a feasible time slot and an upper limit to this number of reassignments can be adjusted according to performance requirements. A simulated rescue scenario with task deadlines and fuel limits is used to demonstrate the performance of the proposed method compared with existing methods, the consensus-based bundle algorithm and the performance impact (PI) algorithm. Starting from existing (PI-generated) solutions, results show up to a 20% increase in task allocations using the proposed method.**

***Index Terms*—Distributed task-allocation, multiagent systems, vehicle routing.**

I. Introduction

Multirobot systems are increasingly employed to complete jobs and missions in various fields including search and rescue [1]–[4], space and underwater exploration [5], support in healthcare facilities [6], surveillance and target tracking [7], [8], product manufacturing [9], [10], pick-up and delivery, and logistics. A team of homogeneous or heterogeneous specialized robots can cover more ground and be more resilient to failures than a single all-purpose robot [11], [12].

Manuscript received June 28, 2016; revised April 6, 2017 and July 28, 2017; accepted August 7, 2017. Date of publication September 28, 2017; date of current version August 16, 2018. The work of Q. Meng was supported by EPSRC with BAE Systems as the leading industrial partner under Grant EP/J011525/1. This paper was recommended by Associate Editor H. M. Schwartz. *(Corresponding author: Joanna Turner.)*

J. Turner, Q. Meng, G. Schaefer, and A. Soltoggio are with the Department of Computer Science, Loughborough University, Loughborough LE11 3TU, U.K. (e-mail: joanna.z.turner@gmail.com; q.meng@lboro.ac.uk; gerald.schaefer@ieee.org; a.soltoggio@lboro.ac.uk).

A. Whitbrook is with the Department of Electronics, Computing and Mathematics, University of Derby, Derby DE22 1GB, U.K. (e-mail: a.whitbrook@derby.ac.uk).

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. The file contains a supplementary table that compares the performance of different task allocation algorithms. The material is 0.0346 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2017.2743164

One challenge in using teams of robots is to co-ordinate them to perform tasks while optimizing one [13], [14] or more objectives [15]–[17]. Considering a search and rescue scenario, in which survivors need to be assisted before specified deadlines, the two main objectives are: 1) to maximize the number of rescued survivors and 2) to minimise the average waiting time before their rescue [18]. Similarly, tasks in an assembly line or manufacturing process require completion with different constraints and optimization objectives often including completion of all tasks in the shortest possible time [9]. As opposed to a factory environment, search and rescue missions often deal with very dynamic conditions, unstructured environments, and limited resources. Increasing the number of survivors and reducing waiting time are top priorities. The novel algorithm presented in this paper applies particularly to search and rescue scenarios, but applications extend to other scenarios in which similar conditions and constraints are present.

The specific problem investigated in this paper is that of maximizing task assignments in time constrained scenarios. Robots, or autonomous vehicles, can only perform one task at a time, each task requires only one vehicle to perform it and each vehicle may be assigned multiple tasks that they execute based on a schedule. Using the Gerkey and Matarić taxonomy [13], [19], this is known as the single-task (ST), single robot (SR), and time-extended assignment (TA) problem. Due to the complexity of the problem, existing heuristic task allocation methods are likely to generate a local optima solution and lack the flexibility to escape from it. Following the principle that tasks are assigned to minimize costs, we introduce a method of measuring the cost of a task assignment, called performance impact (PI)-MaxAss, that effectively shifts task assignments among vehicles to create feasible time slots for unassigned tasks where none would exist otherwise. The maximum number of reassignments can be adjusted to match performance requirements. With this method, existing task assignment solutions are iteratively improved without the need to repeat the whole task allocation procedure. The procedure follows a two-phase task assignment strategy that starts from a solution generated by an existing distributed task allocation algorithm, PI [20], that minimizes average waiting time. The proposed method PI-MaxAss is used in the second stage for maximizing task allocations.

This paper introduces PI-MaxAss, an extension of [21] with further configuration settings and new findings using previously untested simulation scenarios. A new convergence guarantee method is proposed, as well as a complexity analysis. In line with the updating of terminology from [22] to [20] that more precisely reflects the contribution of the PI algorithm, the taxonomy used in [21] has also been updated for this paper.

The remainder of this paper is organized as follows. The task assignment problem and current approaches are presented in Section II. In Section III the concept of PI-MaxAss is introduced. Simulation results are presented in Section IV followed by a discussion in Section V and concluding remarks in Section VI.

II. Problem and Current Approaches

A. Related Work

The search and rescue scenarios considered in this paper have similarities with the traveling salesman problem (TSP), a well-known NP-hard combinatorial optimization problem in graph theory [23]. The objectives considered in this paper are comparable to the constraints of two variants of the TSP: 1) the team orienteering problem with time windows (TOPTW) [24], also known as the multiple tour maximum collection problem and 2) the K-traveling repairmen problem (K-TRP) [25], also known as the minimum latency problem. The TOPTW considers multiple time-limited paths with the objective to maximize the total collected score over a set of vertices. Each vertex is assigned a time window and is to be visited once at most. The K-TRP tries to determine a set of tours for multiple repairmen to visit a set of customers with the objective to minimize the average time a customer must wait before a repairman arrives. These objectives and constraints are applicable to a variety of scenarios such as those found in healthcare, target tracking, pick-up and delivery, logistics, dynamic ride sharing [26], cleaning chemical spills, patrolling, checking for structural integrity of buildings [4], and any scenario that requires many urgent jobs to be completed in a minimum time by multiple agents.

Various algorithms have explored strategies to solve multiobjective TSPs or vehicle routing problems, see [16] for a survey. Paquete and Stützle [15] tackled a bi-objective TSP with a two-phase local search procedure. The first phase generates a solution that optimizes only one objective. The second phase begins the search from the solution generated in the first phase to optimize the second objective. The advantages to using this approach highlighted by [15] are to exploit the strong performance of single objective local search algorithms by chaining them together, and to maintain a flexible modularity and ease of understanding to the procedure that allows for modifications and enhancements. Heuristic methods to solve combinational optimization problems are prone to finding a local optimum [27]; however, a second search can perturb the first phase solution out of local optima to reach an enhanced solution closer to a nondominated global optimum. Algorithms previously developed to solve variants of the TSP problem, such as [15], [16], and [28]–[30], rely on computing a solution with a centralized approach.

Centralized task allocation systems, where a central server gathers information from each vehicle in the team and then computes an allocation for each vehicle, can optimize a chosen global objective based on a complete set of information from all vehicles. The drawbacks are the resulting single point of failure, and the requirement that each vehicle must have a communication link with the central server. Thus, the possible mission range is limited, and a heavy communication and computation burden is put on the central server. Distributed methods for task allocation overcome these limitations. In such cases, the task allocation algorithm runs on each vehicle simultaneously and the solution is reached through the interaction and exchange of information among them [11], [12], [31]. One of the drawbacks of distributed systems is that each vehicle has a different situational awareness, and therefore, consensus procedures are required for the team of vehicles to reach agreement.

ST-SR-TA is a combinatorial optimization problem known to be strongly NP-hard [13], [19]. A subcategory of this problem, for which the cost of a task assignment depends on the other tasks that agent is performing, is called in-schedule dependencies (ID [ST-SR-TA]). Variants of the K-TSP can be modeled under this class of problem [19]. Due to the high complexity of the problem, as the number of tasks and vehicles increases, it is usually too computationally expensive to consider each combination of tasks for each vehicle in order to find the optimal solution. The computational limitations are particularly relevant in search and rescue scenarios in which time and resources could be limited. Therefore, heuristic methods are employed to speed up the process of task allocation while maintaining an efficient and scalable algorithm [11], [28]–[30], [32].

Market-based multirobot (MR) co-ordination approaches [11] have been applied successfully to the ST-SR-TA problem to find suboptimal solutions efficiently and in a distributed fashion. With this approach, teams of self-interested agents iteratively trade tasks to maximize their own profit or minimize their costs. A cost is associated with an agent visiting a task within its path and is often measured as the total estimated use of individual resources to reach that task, such as fuel consumption, distance traveled, or time to reach the target. The local cost of an agent's path is equal to the sum of costs of each task the agent is assigned to [33], and the global cost of an agent team is the sum of costs of all task assignments in the team. An auction is a commonly used market-based approach to assign tasks [34]. The process consists of several rounds of bidding in which agents place bids on each task where the value of a bid for a task is equal to the agent's estimated cost of visiting that task. The agent wins and is allocated those tasks for which it has placed a bid lower than any other agent. The effect of using this market-based approach is that local costs and subsequently global costs are minimized [11].

Zheng and Koenig [35] developed a multirobot, distributed reallocation mechanism called K-swaps that describes multiple task exchanges among multiple agents at a time, and showed empirically that the method can optimize an existing task allocation solution by reducing team costs. Extending the idea of K-swaps, [36]–[38] introduced a decentralized task assignment algorithm considering instantaneous assignment, such that each robot is assigned exactly one task, the SR-ST-IA problem. The algorithm requires the differentiation of two roles, organizer and member robots, and can be used to optimize existing suboptimal task assignments.

B. Formal Problem Description

Consider a search and rescue scenario with n heterogeneous autonomous vehicles and m survivors. In this scenario, attending to a survivor is synonymous with executing a task. The goal is to provide targeted emergency support to the survivors as quickly as possible, e.g., some survivors may require food supplies, while others may require medical provisions. Thus in some scenarios different types of vehicles are necessary to complete different tasks. The distributed vehicles in a network rely on local communication to co-ordinate a rescue plan over multiple iterations.

In the particular scenario considered, each survivor must be visited by one vehicle in order to be deemed rescued. Each vehicle can be assigned multiple targets and will sequentially visit those targets, while not required to return to its initial location. The main challenge is to reach an optimal allocation where allocation numbers are maximized and waiting time minimized, while respecting time constraints.

To formulate the problem mathematically, a set of n heterogeneous autonomous vehicles is defined by $\mathbf{V} = [v_1, \ldots, v_n]$, and a set of m tasks waiting to be completed is defined by $\mathbf{T} = [t_1, \ldots, t_m]$. A list of key symbols used hereafter is provided in Table I. The ordered task allocation of the ith vehicle v_i is stored in $\mathbf{a}_i$, which can contain a variable number of tasks depending on how many tasks are assigned to v_i. Each task is to be assigned to one vehicle only, or left unassigned when time constraints cannot be satisfied.

Different task types can be executed by heterogeneous vehicles with the right capabilities. Thus, each task will be assigned only to vehicles functionally capable of performing them.

A latest start time s_k is defined for each task t_k after which it is too late for the task to be executed successfully; it is therefore necessary to determine whether a vehicle can arrive at the location of a task t_k before the latest start time s_k. The objective of minimizing average waiting time measures the cost of a task assignment as the time it takes to start servicing the task from the start of the vehicle's schedule, i.e., the total time the survivor must wait before being attended to. The time cost of a task t_k in $\mathbf{a}_i$, defined as $c_{i,k}(\mathbf{a}_i)$ in [22], is the predicted time taken by the vehicle v_i to arrive at the location of the task t_k. This time includes the duration of earlier tasks in $\mathbf{a}_i$ and travel time to and from those earlier tasks, but does not include the duration of the execution of t_k. For this particular scenario, the duration of a task is dependent on the task type [22]. Vehicles are additionally assumed to have limited fuel capacity that restricts the time that they can be active for. All tasks must be started before the vehicle reaches its fuel capacity. The latest time at which v_i can arrive at a task before reaching its fuel capacity is defined as f_i. The start time of the kth task must therefore also be no later than f_i such that

$$c_{i,k}(\mathbf{a}_i) \leq \min(s_k, f_i). \tag{1}$$

TABLE I
SYMBOL DEFINITIONS

Symbol	Definition
$\mathbf{V} = [v_1, \ldots, v_n]$	Set of n vehicles
$\mathbf{T} = [t_1, \ldots, t_m]$	Set of m tasks
$\mathbf{a}_i$	Ordered task allocation of the i^{th} vehicle v_i
s_k	Latest start time for task t_k
$c_{i,k}(\mathbf{a}_i)$	The time cost of a task t_k in $\mathbf{a}_i$: the predicted time taken by v_i to arrive at the location of the task t_k in its schedule $\mathbf{a}_i$
f_i	The latest time at which v_i can start a task before running out of fuel
$\|\mathbf{a}_i\|$	The number of tasks assigned to v_i
$w_k^{\ominus}(\mathbf{a}_i, t_k)$	The Removal Performance Impact (RPI) of a task t_k in $\mathbf{a}_i$
$\mathbf{a}_i \ominus t_k$	$\mathbf{a}_i$ with t_k removed
$\boldsymbol{\gamma}_i = [w_1^{\ominus}, \ldots, w_m^{\ominus}]$	Vector on each vehicle to store RPIs
$\mathbf{a}_i \oplus_l t_k$	The inclusion of task t_k at position l in $\mathbf{a}_i$
$w_q^{\oplus}(\mathbf{a}_i, t_q)$	The Inclusion Performance Impact (IPI) of including t_q into $\mathbf{a}_i$
$\boldsymbol{\gamma}_i^{\oplus} = [w_1^{\oplus}, \ldots, w_m^{\oplus}]$	A list to store the IPIs of each task on each vehicle
$\boldsymbol{\beta}_i = [\beta_1, \ldots, \beta_m]$	A vehicle ID list corresponding to the RPI list that keeps track of which task is assigned to which vehicle.
$\boldsymbol{\psi}_i = [t_1, \ldots, t_\varsigma]$	Candidate tasks for inclusion into v_i's task list
$\mathbf{a}_i^{\ominus k}$	Temporary task list with t_k removed
$\mho_{i,k}$	List of tasks that can replace t_k in $\mathbf{a}_i$ while respecting time constraints
SD	Swap Distance: maximum number of permissible reassignments to create a time slot for an unassigned task
r	Reduction rate of RPI-MaxAss for each additional reassignment
$\boldsymbol{\varpi}_i$	A vector that stores the number of times each task has been removed from a vehicle v_i's task list

In [20] and [22], the global objective J is to minimize the average start time of all tasks, such that

$$J = \min\left\{\frac{1}{m}\sum_{i=1}^{n}\sum_{k=1}^{|\mathbf{a}_i|} c_{i,k}(\mathbf{a}_i)\right\} \tag{2}$$

where $|\mathbf{a}_i|$ is the number of tasks assigned to v_i. In [20] and [22], unassigned tasks are given the highest cost and are therefore prioritized for inclusion following the inclusion criteria described later in Section II-E.

The main contribution of this paper is a novel way to measure the PI of a task assignment such that an assignment's cost is correlated to the PI of tasks that it can be replaced with were it to be reassigned to another vehicle. The objective is to maximize the number of allocated tasks. Maximizing task allocations is defined as

$$J^\star = \max\left\{\sum_{i=1}^{n}|\mathbf{a}_i|\right\}. \quad (3)$$

The new version of PI, which maximizes the number of assignments, is referred to as PI-MaxAss, and is the main contribution of this paper. The PI presented in [22] that minimizes average time is referred to as PI-MinAvg in order to distinguish the two.

The motivation for PI-MaxAss is to prioritize assigning the maximum number of tasks in scenarios in which time constraints severely restrict the number of tasks that can be assigned. For scenarios in which all tasks can be assigned, it is recommended to use PI-MinAvg to optimize average waiting time.

C. CBBA, Extensions, and Variations

The consensus-based bundle algorithm (CBBA) [39] is a robust and fully distributed multiassignment task allocation algorithm that employs a greedy auction strategy to enable agents to build a bundle of tasks sequentially. This task building phase is followed by a consensus procedure phase that resolves conflicting assignments. These two stages alternate until consensus has been reached by the team on all task assignments. For an analysis of CBBA's scalability, see [39]. Of the various extensions and modifications, [40] and [41] address MR task assignments and heterogeneous networks for the ST-MR-TA problem in which multiple robots may be required to service one task [13]. Choi *et al.* [40] addressed the case in which a task requires only one single agent, one or two agents, and exactly two agents of different type. Hunt *et al.* [41] proposed the consensus-based grouping algorithm that addresses the problem of multiagent multitask assignment with group and equipment-based dependencies, and which can accommodate any number of robots.

Ponda *et al.* [42] increased the overall efficiency of a task assignment by incorporating time windows of validity and fuel costs as part of the scoring scheme. The scoring scheme rewards agents for arriving at the optimal time for each task and for minimizing fuel consumption. Ponda *et al.* [42] also addressed real-time replanning for broken communication links, solving the problem of conflicting assignments when unconnected sub networks each have an agent assigned to the same task.

The consensus phase of CBBA requires synchronized communication between all agents. In a real-time dynamic environment, co-ordinating a large number of agents to communicate in sync may overburden the network and require artificially delaying the broadcast of new messages until all earlier messages have been received by the network of agents. Johnson *et al.* [43] extended CBBA with an asynchronous communication protocol to permit the agents to run the consensus phase of the algorithm on their own schedule. The asynchronous communication protocol also uses less bandwidth than CBBA. Ponda *et al.* [44] introduced CBBA with Relays algorithm that improves the team of agents' range and ensures network connectivity in a dynamic environment by utilizing agents as communication relays.

Di Paola *et al.* [45], [46] proposed the heterogeneous robots consensus-based allocation (HRCA) algorithm that deals with multiassignments in heterogeneous networked-teams. The algorithm consists of two outer stages. Stage 1 iterates two inner phases that closely resemble the two phases of CBBA. As opposed to CBBA, in Stage 1 of HRCA the maximum task bundle size is ignored. Stage 2 is performed only if there exist bundles exceeding the maximum limit. In this case, iterative task elimination based on least penalty is performed to resize the bundle. Binetti *et al.* [7], [47] developed the decentralized assignment algorithm based on CBBA and HRCA to solve the task allocation problem for assigning critical tasks for heterogeneous agents with limited capacity.

Cui *et al.* [48] introduced a game theory approach for task allocation. As with CBBA, the process of task allocation is split into two phases. A contract net protocol is used for the initial task allocation and a game theory approach is then used to reallocate the tasks to satisfy Pareto optimality. Smith *et al.* [49] extended CBBA to develop the cluster-formed CBBA to reduce the communication necessary for reaching consensus on task allocation. The communication reduction has a tradeoff of a drop in optimality of task allocation as complexity increases.

D. Performance Impact Algorithm

Whitbrook *et al.* [20] and Zhao *et al.* [22] proposed a concept called PI as an extension of CBBA. This method introduces PI, a value used by vehicles to prioritize task assignments. With PI, unlike CBBA, tasks included into a vehicle's task list can push back the execution times of later tasks in that same list, provided that all time constraints are satisfied. Likewise after a task is removed from a task list, the execution times of later tasks in the list may be shifted forward. With the PI algorithm, a vehicle does not release a task until it is reassigned elsewhere at a lower cost, i.e., once a task is assigned it does not become unassigned. PI considers not only the cost of a task assignment but also the impact of that task assignment on the cost of other assignments in the vehicle's task list. The authors demonstrate the effectiveness of PI through a simulated search and rescue scenario with a global objective to minimize the average start times of tasks with deadlines. The PI algorithm was shown empirically to solve time-critical task allocation problems that CBBA could not, and was shown to find a lower average start time compared with CBBA. Despite the improved performance, the PI algorithm still fails to solve some problems that are solvable due to converging to locally optimal but globally suboptimal solutions [20].

The PI algorithm is a distributed task allocation algorithm that runs simultaneously on each vehicle. Using the same two-phase architecture as CBBA, the PI algorithm iterates over a task inclusion phase and a consensus and conflict resolution phase. During the first phase vehicles locally and iteratively build themselves a task bundle; during the second phase vehicles share their assignment lists with neighboring vehicles

Algorithm 1 Task Allocation Outer-Loop Iterative Procedure for CBBA and PI Running on Each Vehicle

```
Initialise Timer $T \leftarrow 1$
$converged \leftarrow false$
while $converged$ is $false$ do
    Task Inclusion Phase
    Communication and Conflict Resolution Phase
    $converged \leftarrow$ Check Convergence.
    $T \leftarrow T + 1$
end while
```

and resolve conflicting assignments. Both phases repeatedly alternate until a global conflict-free task allocation is agreed upon by all vehicles. These main steps in an iteration of the algorithm are expressed with pseudocode in Algorithm 1.

The PI algorithm measures the local impact of a task assignment to the total cost of a vehicle's task list with the removal performance impact (RPI) and the inclusion performance impact (IPI) of a task assignment. The IPIs are computed during the task inclusion phase and determine which task to include next into a task list. The RPIs are computed at the end of the task inclusion phase and are communicated to networked vehicles during the communication and conflict resolution phase. RPIs determine which vehicle keeps a task in case of conflict.

E. PI Task Inclusion Phase

The IPI of a task t_q in $\mathbf{a}_i$, as defined for PI-MinAvg, is measured as the time cost of t_q in $\mathbf{a}_i$ plus the sum of increase in time costs of other tasks in $\mathbf{a}_i$ that have been assigned previously. The increase in time costs occurs if later tasks need to be shifted to create enough time to service t_q. If no tasks have been assigned previously, the IPI of t_q in $\mathbf{a}_i$ is equal to its time cost, i.e., the time for v_i to reach t_q. This is because the sum of increase in time costs of other tasks in $\mathbf{a}_i$ is necessarily equal to 0. Let $\mathbf{a}_i \oplus_l t_q$ be the insertion of task t_q at position l in $\mathbf{a}_i$. The IPI of t_q in $\mathbf{a}_i$ is computed as

$$w_q^{\oplus}(\mathbf{a}_i, t_q) = \min_{l=1}^{|\mathbf{a}_i|+1} \left\{ w_{q,l}^{\triangle}(\mathbf{a}_i, t_q) \right\} \tag{4}$$

where

$$w_{q,l}^{\triangle}(\mathbf{a}_i, t_q) = \sum_{z=l}^{|\mathbf{a}_i|+1} c_{i,z}(\mathbf{a}_i \oplus_l t_q) - \sum_{z=l}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i). \tag{5}$$

Equation (5) computes the IPI of t_q at each position l in $\mathbf{a}_i$, where $c_{i,z}(\mathbf{a}_i)$ denotes the time cost of the task at position z in v_i's task list. Equation (4) finds the smallest IPI and records it as t_q's IPI in $\mathbf{a}_i$. A list to store the IPIs of each task is kept on each vehicle and is defined as $\boldsymbol{\gamma}_i^{\oplus} = [w_1^{\oplus}, \ldots, w_m^{\oplus}]$ for vehicle v_i.

During this task inclusion phase, vehicles select tasks to include into their task lists until no more tasks can be added. This repeating process is depicted on lines 1–21 in Algorithm 2. Before including a task, the algorithm computes the IPIs of all candidate tasks t_q according to (4) and (5),

Algorithm 2 PI Task Inclusion Phase

```
1:  while task list not full do
2:      $w_q^{\oplus} \leftarrow$ highest permissible cost, $w_q^{\oplus} \in \gamma_i^{\oplus}$
3:      for each task $q$ do
4:          if task $q$ is a candidate then
5:              for each insertion position $l$ in task list do
6:                  if $\mathbf{a}_i \oplus_l t_q$ is feasible then
7:                      Compute $w_{q,l}^{\triangle}$ according to (5)
8:                  end if
9:              end for
10:             Compute $w_q^{\oplus}$ and position $l$ according to (4)
11:         end if
12:     end for
13:     Compute $g$ from (6)
14:     if $g > 0$ then
15:         Insert task $q$ yielding $g$ in position $l$ of task list
16:         Update vehicle list $\boldsymbol{\beta}_q = i$
17:         Update time costs of task list
18:     else
19:         break
20:     end if
21: end while
22: Compute $\boldsymbol{\gamma}_i$ (only RPIs in task list will be affected)
```

where candidate tasks are those compatible with v_i's capabilities and not already in $\mathbf{a}_i$. The computation of IPIs is depicted on lines 3–12 in Algorithm 2. When there are already tasks in $\mathbf{a}_i$ that have been assigned previously it is necessary to determine which position in the task list yields the most optimal IPI, i.e., whether it is most optimal to include t_q at the start of $\mathbf{a}_i$, at the end, or in a position between tasks. Thus the IPI of t_q is computed in each position l (lines 5–9) and the position l in which the IPI is lowest is the optimal position (line 10).

After the IPIs of all candidate tasks have been computed, v_i selects for inclusion the task whose IPI can improve upon that task's current RPI the most. At this stage candidate tasks' RPIs will either have their initial value if unassigned, or an updated value received during the communication and conflict resolution phase. RPIs for all tasks are initialized to their highest permissible cost such that RPIs of tasks must be lower than this value once they are assigned. An IPI of t_q in $\mathbf{a}_i$ lower than t_q's RPI in another vehicle's task list $\mathbf{a}_j$ indicates that the global cost can be reduced if t_q is reallocated to v_i. The RPI of a task t_q is referred to formally as $w_q^{\ominus}$ and each vehicle stores the vector $\boldsymbol{\gamma}_i = [w_1^{\ominus}, \ldots, w_m^{\ominus}]$. A task t_q assigned to v_j with an RPI greater than the IPI of t_q in $\mathbf{a}_i$ is written formally as $w_q^{\ominus}(\mathbf{a}_j, t_q) > w_q^{\oplus}(\mathbf{a}_i, t_q)$. Multiple IPIs may improve on the current RPIs, as such, v_i selects for inclusion the task that reduces the global cost most. The maximum difference between the RPIs of all tasks and the IPIs of all tasks is computed as

$$g = \max_{q=1}^{m} \left\{ \boldsymbol{\gamma}_{i,q} - \boldsymbol{\gamma}_{i,q}^{\oplus} \right\}. \tag{6}$$

Line 13 in Algorithm 2 computes g according to (6). If $g > 0$ (line 14), the task corresponding to g is included into the vehicle's ordered task list, leading to the maximum reduction to the global cost. If $g \leqslant 0$, IPIs of all tasks are greater or equal to the current RPIs, meaning that the current assignments cannot be improved upon, or that time constraints of candidate tasks cannot be met. In this case the task inclusion process ends (line 19).

RPIs are updated at the end of the task inclusion phase (line 22). While RPIs are constant for unassigned tasks, once assigned, the RPI is measured as t_k's time cost in $\mathbf{a}_i$ plus the sum of the changes in time cost of remaining tasks in $\mathbf{a}_i$ before and after the removal of t_k. By removing t_k from $\mathbf{a}_i$, v_i may be able to execute its remaining task assignments earlier. The time costs of tasks earlier in the task list than t_k are not affected by the removal of t_k. The RPI of a task t_k in $\mathbf{a}_i$ is formally written as:

$$w_k^{\ominus}(\mathbf{a}_i, t_k) = \sum_{z=b}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i) - \sum_{z=b+1}^{|\mathbf{a}_i|} c_{i,z}(\mathbf{a}_i \ominus t_k) \tag{7}$$

where b is the position of task t_k in v_i's task list, $c_{i,z}(\mathbf{a}_i)$ denotes the time cost of the task at position z in v_i's task list, and $\mathbf{a}_i \ominus t_k$ denotes $\mathbf{a}_i$ with t_k removed. When a global consensus is reached, all vehicles have an identical copy of $\boldsymbol{\gamma}$.

F. PI Communication and Conflict Resolution Phase

Once the task inclusion phase is complete, the RPI list and an m-sized vehicle ID list that keeps track of which vehicle is assigned to which task, are broadcast to neighboring vehicles. The vehicle ID list is necessary for consensus and is defined as $\boldsymbol{\beta}_i = [\beta_1, \ldots, \beta_m]$. Neighboring vehicles are those where a communication link exists between them based on a network topology. This topology may be dynamic and depend on, e.g., communication range and physical distance between two local vehicles. The vehicles communicate once per algorithmic iteration and this paper does not consider a communication cost. As two or more vehicles may be assigned the same task, the consensus procedure introduced in [39] is used to resolve these conflicting assignments. A lower RPI indicates a more optimal assignment, therefore vehicles with a higher RPI for a conflicting assignment release the task. RPIs and associated vehicle IDs are updated during consensus.

The task inclusion and conflict resolution phases repeat until no inclusions or removals can be made. At this point, the system is deemed to have converged and the task allocation procedure ends.

III. Performance Impact for Maximizing Task Assignments

Simulated experiments have shown that the PI algorithm both allocates more tasks and optimizes average waiting time better than CBBA in time critical scenarios with a low task-to-vehicle ratio [20], [22]. However, preliminary experiments showed that when there is a higher ratio of tasks to vehicles, PI can fail to allocate all tasks even though it is possible to do so. Due in part to their scoring strategies, the baseline CBBA and PI do not reassign tasks when this is necessary in order to assign additional tasks. In the search and rescue scenario the safety and rescue of survivors is a high priority; a poorer

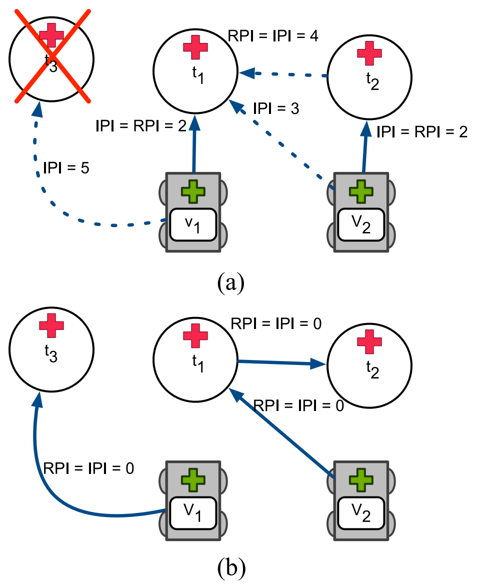


Fig. 1. In this scenario PI-MinAvg is unable to assign all tasks. PI-MaxAss assigns all tasks. Dotted lines connecting vehicles to tasks indicate examples of IPIs computed during the task inclusion phase. Solid lines indicate tasks assigned after reaching consensus. (a) Each task assignment is labeled with its PI-MinAvg IPI or RPI. With PI-MinAvg t_1 is assigned to v_1, t_2 is assigned to v_2, and t_3 is left unassigned. v_2 may also include t_1 if v_2 has not yet received v_1's RPI list. In this case v_2 releases t_1 during the conflict resolution phase due to a higher RPI than v_1. (b) PI-MaxAss reassigns tasks starting from the PI-MinAvg solution and creates a time slot for t_3. Each task assignment is labeled with its IPI and RPI for maximizing the number of task assignments.

quality of solution results in fewer survivors being rescued than is possible with the available resources.

The new version of PI, that maximizes the number of assignments is referred to as PI-MaxAss, and is the main contribution of this paper. An early version of PI-MaxAss was presented in [21] and is extended here to include better cost scoring, convergence guarantee, and extended simulations including scenarios with battery limits only, and scenarios with task deadlines and battery limits.

Starting from a suboptimal assignment in which additional tasks cannot be directly included without violating time constraints, the extension PI-MaxAss presented in this paper is able to reassign tasks to increase the total number of allocated tasks simply through a change in the computation of IPIs and RPIs. The idea introduced in this paper is to attribute a high cost (RPI) to an assigned task when the release of this task can permit an additional task to be inserted within the free time created. An assignment is considered optimal and without cost if the release of any task does not permit another task to be assigned within the free time created. Likewise, a task's IPI is set to be without cost if it can be included into a task list and satisfy time constraints. During the conflict resolution phase, conflicts resolve in favor of vehicles offering the lowest RPI. Vehicles that can create a time slot for candidate tasks through the release of an assigned task therefore release that task during a conflict. The result is that tasks are reassigned and feasible time slots are created for unassigned tasks.

To illustrate the limitation of previous methods and the proposed solution, consider a simple scenario shown in Fig. 1(a)

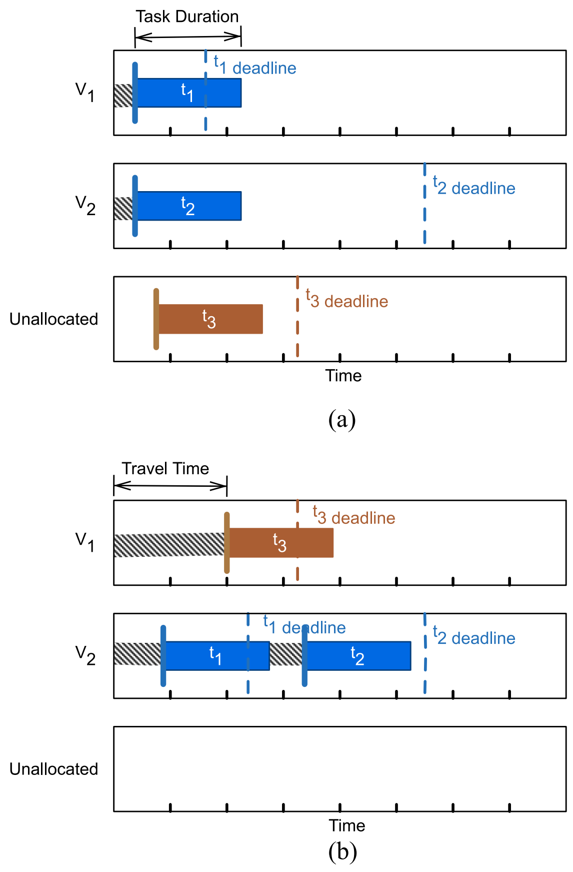


Fig. 2. Task schedules for v_1 and v_2. A travel time is assumed between the vehicles' initial locations and between different task locations, based on the distance and speed that they can travel. A fixed task duration is also assumed. A task must be started before the deadline in order to rescue that survivor, but may end after the deadline. v_1 is the only vehicle close enough to reach t_3 in time. (a) t_1 and t_2 are optimized to minimize waiting time but t_3 is unallocated. v_1 cannot feasibly include t_3 into its schedule given t_1. (b) If t_1 is reassigned from v_1 to v_2, this creates the time slot for v_1 to include the unallocated task t_3.

and the associated schedule on a timeline in Fig. 2(a). With the PI algorithm, the vehicles include tasks into their lists starting with the lowest IPI. With PI-MinAvg, v_1 first includes t_1 and v_2 first includes t_2 into their task lists. Once included, t_1 cannot be released from v_1 unless v_2 includes t_1 with a lower RPI. Likewise, t_2 cannot be released from v_2 unless v_1 includes t_2 with a lower RPI. For t_3 to be serviced before its deadline, v_1 must go to t_3 directly. However, v_1 is incapable of servicing both t_1 and t_3 and meet both of their time constraints. Task t_1 does not get reassigned to v_2 because the RPI of t_1 is lower in v_1's task list than in v_2's task list. Therefore t_3 does not get assigned. The suboptimal task allocation is due to the minimization of waiting time performed by PI-MinAvg. The novelty in PI-MaxAss is that the cost of t_1 in v_1's task list is higher than in v_2's task list, causing t_1 to be reassigned to v_2. This creates a time slot in v_1's schedule for t_3. Therefore, PI-MaxAss achieves the optimal allocation illustrated in Figs. 1(b) and 2(b). Although the waiting time for t_1 and t_2 has increased in Fig. 2(b), this reassignment has enabled an additional task to be assigned.

Algorithm 3 Computing RPI-MaxAsses for Tasks in v_i's Task List

```
1:  Set RPI of tasks in $\mathbf{a}_i$ to 0: $\gamma_{i,k} \leftarrow 0, \quad t_k \in \mathbf{a}_i$
2:  Identify Candidate Tasks: $\bar{\boldsymbol{\psi}}_i$
3:  for each task $k$ in $\mathbf{a}_i$ do
4:      $\mathbf{a}_i^{\ominus k} = \mathbf{a}_i \ominus t_k$
5:      Update times $c_{i,z}(\mathbf{a}_i^{\ominus k})$ for tasks after $t_k$
6:      for each task $q$ in $\bar{\boldsymbol{\psi}}_i$ do
7:          if $\gamma_{i,q} - r > \gamma_{i,k}$ then
8:              for each position $l$ in $\mathbf{a}_i^{\ominus k}$ do
9:                  if $\mathbf{a}_i^{\ominus k} \oplus_l t_q$ is feasible then
10:                     $\gamma_{i,k} = \gamma_{i,q} - r$
11:                     break
12:                 end if
13:             end for
14:         end if
15:     end for
16: end for
```

A. Formal Description

With PI-MaxAss, unallocated tasks are set initially to have a fixed highest RPI-MaxAss, a constant defined as U, such that if t_q is unassigned then $w_q^{\ominus} = U$. The RPIs of assigned tasks t_k are initially set to 0, such that $w_k^{\ominus} = 0$.

The steps of PI-MaxAss follow the two phases depicted in Algorithm 1. During the task inclusion phase shown in Algorithm 2, as with PI-MinAvg, the PI-MaxAss candidate tasks for inclusion into $\mathbf{a}_i$ are those compatible with v_i's capabilities and not already in $\mathbf{a}_i$, and with an RPI-MaxAss greater than 0. The candidate tasks for inclusion into $\mathbf{a}_i$ are formally defined as

$$\boldsymbol{\psi}_i = \left[t_1, \ldots, t_\zeta\right], \quad t_q \notin \mathbf{a}_i, 0 < w_q^{\ominus}. \tag{8}$$

The IPI-MaxAss of t_q in $\mathbf{a}_i$ is formally defined as

$$w_q^{\oplus\star}(\mathbf{a}_i, t_q) = 0, \exists l \quad \forall t_z \in \left\{\mathbf{a}_i \oplus_l t_q\right\} \\ : c_{i,z}\left(\mathbf{a}_i \oplus_l t_q\right) \leq \min(s_z, f_i), t_q \in \boldsymbol{\psi}_i. \tag{9}$$

In other words, the IPI-MaxAss of the candidate task t_q is set to 0 if there exists a position l in $\mathbf{a}_i$ where the task t_q is inserted and all time constraints are met. On line 10 in Algorithm 2, IPI-MaxAss is recorded in place of IPI-MinAvg such that $w_q^{\oplus\star} = 0$ if the condition on line 6 returns true for at least one position l. The optimal position l is computed as it is for IPI-MinAvg, according to (4) and (5).

Lines 13–21 in Algorithm 2 remain the same for PI-MaxAss. As the RPI-MaxAss of assigned tasks were initialized to 0, only unassigned tasks are candidates for inclusion in the first round of the task inclusion phase. RPI-MaxAss is computed on line 22 in the place of RPI-MinAvg. The steps for computing RPI-MaxAss are shown in Algorithm 3.

Candidate tasks in the computation of RPI-MaxAss follow the same constraints as the candidates in (8) with the added constraint that the candidate task's RPI-MaxAss is greater than δ. This constraint is used to limit the number of reassignments permissible to allocate an additional task (see Section III-B). Candidate tasks used in the computation of

RPI-MaxAss for a task t_k are formally defined as

$$\bar{\boldsymbol{\psi}}_i = [t_1, \ldots, t_\zeta], \quad t_q \notin \mathbf{a}_i, \quad 0 < \delta < \boldsymbol{\gamma}_{i,q}. \tag{10}$$

The identification of candidate tasks occurs on line 2 in Algorithm 3. To compute the RPI-MaxAss of a task t_k in $\mathbf{a}_i$, first, a temporary task list $\mathbf{a}_i^{\ominus k}$ is created that is equivalent to $\mathbf{a}_i$ with t_k removed and is formally defined as

$$\mathbf{a}_i^{\ominus k} = \mathbf{a}_i \ominus t_k, \quad t_k \in \mathbf{a}_i. \tag{11}$$

The creation of $\mathbf{a}_i^{\ominus k}$ occurs on line 4 in Algorithm 3. Next, a candidate task t_q is inserted into each position l in $\mathbf{a}_i^{\ominus k}$ to determine if there exists a position l in $\mathbf{a}_i^{\ominus k}$ in which t_q is inserted and all time constraints are met. If such a position l exists then t_k can feasibly be replaced by t_q in $\mathbf{a}_i$ and the RPI-MaxAss of t_k is computed as the RPI-MaxAss of t_q reduced by r. This computation is repeated for each task t_q in $\bar{\boldsymbol{\psi}}_i$. The list of tasks $\boldsymbol{\mho}_{i,k}$ that can replace t_k in $\mathbf{a}_i$ while respecting time constraints is formally defined as

$$\boldsymbol{\mho}_{i,k} = \left\{ t_q \in \bar{\boldsymbol{\psi}}_i \mid \exists l \quad \forall t_z \in \left\{ \mathbf{a}_i^{\ominus k} \oplus_l t_q \right\} \right. \\ \left. : \left(c_{i,z}\left(\mathbf{a}_i^{\ominus k} \oplus_l t_q \right) \le \min(s_z, f_i) \right) \right\}. \tag{12}$$

If a task t_k in $\mathbf{a}_i$ can be replaced by two or more candidate tasks t_q with different RPI-MaxAsses, the highest RPI-MaxAss is recorded. The RPI-MaxAss of a task is formally defined as

$$w_k^{\ominus\star}(\mathbf{a}_i, t_k) = \max_{q=1}^{|\mho_{i,k}|} \left\{ w_q^{\ominus\star} - r \right\}, \quad t_q \in \boldsymbol{\mho}_{i,k}, r \in \mathbb{R}_+. \tag{13}$$

The condition on line 7 in Algorithm 3 ensures that the feasibility of inserting t_q into $\mathbf{a}_i^{\ominus k}$ is not computed if the resulting RPI-MaxAss of t_k is not higher than its current value. This condition reduces unnecessary computation and satisfies finding the maximum RPI-MaxAss according to (13). The condition on line 9 checks the feasibility of inserting t_q in position l in $\mathbf{a}_i^{\ominus k}$ so that the computation of RPI-MaxAss on line 10 is performed only with candidate tasks that satisfy (12).

Fig. 3 illustrates how the computation of a decreasing RPI-MaxAss allows for multiple reassignments to create a time slot for an unassigned task, and signposts the path with the fewest reassignments. Fewer reassignments minimizes the time to reach consensus and better maintains the original solution's optimization for minimizing average waiting time.

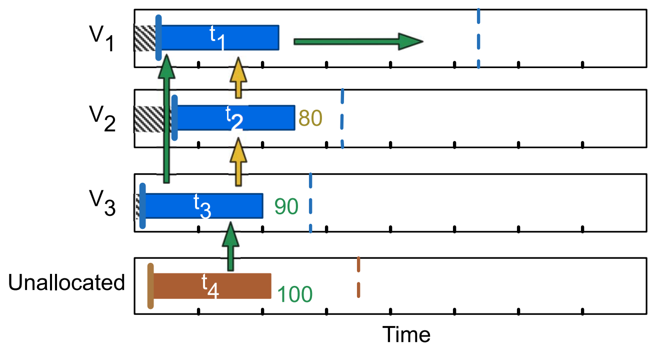


Fig. 3. RPI-MaxAss minimizes the number of changes to existing task assignments to create a time slot for an unallocated task. In this scenario it is assumed that v_3 is the only vehicle near enough to t_4 to service it in time. t_4 is unallocated and takes RPI-MaxAss $= U = 100$. r is set as 10. t_3 can be replaced by t_4 according to (12) therefore t_3's RPI-MaxAss is $100 - 10 = 90$ according to (13). t_2 can be replaced by t_3 therefore t_2's RPI-MaxAss is $90 - 10 = 80$. During the task inclusion phase, v_1 can include t_3 or t_2 (without removing t_1) therefore t_3 and t_2's IPI-MaxAss are 0 according to (9). Given (6), v_1 selects t_3 for inclusion as t_3 yields the greatest difference between RPI and IPI. During the communication and conflict resolution phase, v_3 releases t_3 due to having a higher RPI-MaxAss for t_3 than v_1. During the task inclusion phase, v_3 includes t_4. The decreasing RPI-MaxAss ensures that the minimal number of reassignments is selected when different options are available for the inclusion of an unassigned task.

B. Swap Distance

In a time critical scenario such as search and rescue, it may be necessary to limit the time it takes for the distributed system to converge to a task allocation. The time to converge partly depends on the number of iterations until consensus. Depending on the network topology, propagating new assignments across the network may require multiple iterations affecting the total time to consensus. Therefore, with PI-MaxAss, limiting the number of reassignments permissible to assign an unassigned task is required. A maximum number of reassignments, expressed as "Swap Distance" SD is defined. SD is a new parameter, not present in CBBA or PI-MinAvg, introduced in PI-MaxAss to limit the maximum number of reassignments. As defined by (10), a candidate task in $\bar{\boldsymbol{\psi}}_i$ must have an RPI-MaxAss greater than δ which limits the number of reassignments to SD; δ is defined as

$$\delta = U - (r * \text{SD}), \quad r < \frac{U}{\text{SD}}, \text{SD} \in \mathbb{R}_+, U \in \mathbb{R}_+. \tag{14}$$

In Fig. 3, $U = 100$ and $r = 10$. If $\text{SD} = 0$ then $\delta = 100$ resulting in no candidates for the computation of RPI, according to (10). As a consequence only unassigned tasks have an RPI greater than 0 and can therefore be included in the task inclusion phase according to (8). If $\text{SD} = 1$ then $\delta = 90$ and one reassignment is permissible for the inclusion of an unassigned task. In Fig. 3, the path that requires two reassignments in which the RPI-MaxAss of t_2 is 80 is not permissible when $\text{SD} = 1$. When $\text{SD} = 1$, t_3 does not satisfy the constraints to be in $\bar{\boldsymbol{\psi}}_2$ because its RPI-MaxAss is not greater than δ, therefore the RPI-MaxAss of t_2 remains as 0. The path with two reassignments is only possible with $\text{SD} = 2$ (or higher). SD therefore restricts the tasks eligible to be candidates so that the number of reassignments is less than or equal to SD. Guidance on setting SD is discussed in Section V.

C. Convergence

Preliminary experiments running PI showed that two or more vehicles occasionally get caught in an infinite cycle exchanging the same tasks. In order to avoid infinite cycles and to guarantee convergence, the proposed solution is to limit the number of times that a vehicle can remove the same task from its list before it no longer attempts to include it. A maximum limit on removals Υ where $\Upsilon \in \mathbb{Z}_+$ can be set. This precaution may prevent those tasks that are being repeatedly exchanged from being allocated optimally, however, it ensures that the system can converge. A vector $\boldsymbol{\varpi}_i$ is used to store the number of times each task has been removed from a vehicle v_i's task list. During the conflict resolution phase when a task t_k has been removed from v_i's task list: $\varpi_{i,k} = \varpi_{i,k} + 1$. During the task inclusion phase, a task t_k is considered a candidate in $\boldsymbol{\psi}_i$ for inclusion if $\varpi_{i,k} < \Upsilon$ is satisfied.

D. Complexity

To assess the computational complexity of running PI-MaxAss on one vehicle, the method used in [22] is followed. In [22], the computational complexity of PI-MinAvg is determined to be polynomial. The complexity is dominated by the computation of IPI-MinAvg during the task inclusion phase and it is defined in [22] as

$$O\Big((m_i - |\mathbf{a}_i|)|\mathbf{a}_i|^2\Big)\vartheta_y\sigma \tag{15}$$

where $|\mathbf{a}_i|$ represents the cardinality of the task list $\mathbf{a}_i$. m_i is the capacity of vehicle v_i. A maximum number $m_i - |\mathbf{a}_i|$ tasks can be added into a vehicle's task list during each iteration of the algorithm. σ denotes the complexity of computing the time cost of a task. ϑ_y denotes the number of tasks that are not yet in the task list and meet the compatibility constraints. ϑ_y is equivalent to the cardinality of candidate tasks $|\boldsymbol{\psi}_i|$ as defined in this paper. In the experiments conducted in this paper, no hard limit was imposed on the number of candidate tasks. However, such a parameter could be introduced to limit the computational cost of the task inclusion phase.

The complexity of PI-MaxAss is dominated by the computation of each task's RPI-MaxAss in vehicle v_i's task list, as shown in Algorithm 3. The first step in the outer loop (for each task in vehicle v_i's task list) is to remove a task and adjust the times of the remaining tasks in the temporary task list $\mathbf{a}_i^{\ominus k}$; the complexity is $|\mathbf{a}_i^{\ominus k}|(|\mathbf{a}_i^{\ominus k}| + 1)\sigma/2$. Within the inner loop, the task times of each task starting from the position of the included task are computed: $|\mathbf{a}_i||\bar{\boldsymbol{\psi}}_i|(|\mathbf{a}_i^{\ominus k}|+1)((|\mathbf{a}_i^{\ominus k}|+1)+1)\sigma/2$. Altogether this equates to $|\mathbf{a}_i^{\ominus k}|(|\mathbf{a}_i^{\ominus k}|+1)\sigma/2 + |\mathbf{a}_i||\bar{\boldsymbol{\psi}}_i||\mathbf{a}_i|(|\mathbf{a}_i|+1)\sigma/2$. This simplifies to

$$O\Big(|\mathbf{a}_i|^3|\bar{\boldsymbol{\psi}}_i|\sigma/2\Big). \tag{16}$$

The RPI-MaxAss computation has a higher complexity than the RPI-MinAvg computation, but is equivalent to the complexity of computing IPI-MinAvg.

IV. Numerical Results

This section presents the results of numerical simulations conducted to test the performance of the proposed PI-MaxAss compared with the performance of PI-MinAvg and CBBA when maximizing allocated tasks in scenarios with time constraints. CBBA is an established benchmark for comparison in distributed task allocation problems and therefore provides a useful metric for general comparisons with similar algorithms. Thus, the evaluation of the proposed method is performed by comparison with CBBA using a range of parameter settings.

A. Scenario and Simulation Setup

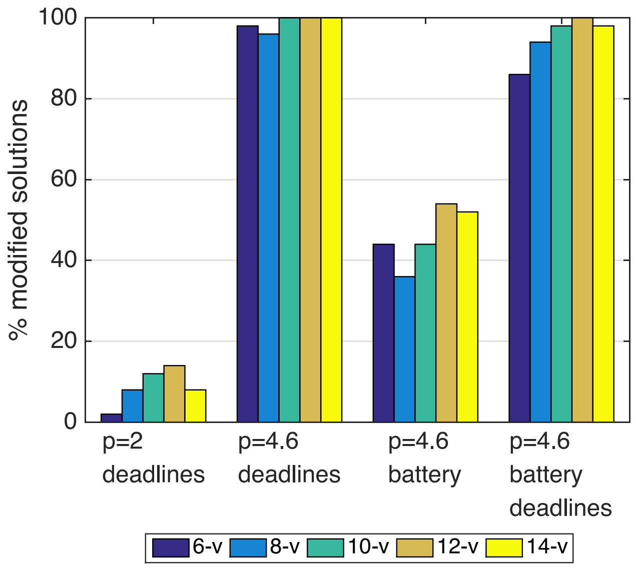
To test the robustness of the proposed approach, the same types of scenarios as in [20] and [22] were used. These include scenarios with a variety of different parameters including task and vehicle numbers, and network topologies. Moreover, the parameter settings are extended in this paper to include a more challenging high task-to-vehicle ratio, and to include fuel constraints on vehicles. Preliminary experiments revealed that changing other parameter settings such as the starting positions of the vehicles, e.g., all vehicles starting from the same position, did not significantly affect the number of task allocations. The setup uses a rescue team equally split into two vehicle types with different functions. One vehicle type provides medicine, the other provides food. All tasks are considered to have equal priority to facilitate a clearer analysis of the task allocation maximization process. However, a range of priorities could be introduced in future extensions of the algorithm through an ordering of candidate tasks. The scenario specification, summarized in Table II, is as follows: the vehicles' speeds are assumed to be constant and are set to 30 and 50 m/s, respectively. The survivors are likewise equally split into those requiring food and those requiring medicine. The medicine tasks last for a duration of 300 s and the food tasks last 350 s. The deadlines for starting each rescue are uniformly distributed on a timeline between 0 and 2000 s. The mission takes place in a 3-D space spanning 10 000 m × 10 000 m × 1000 m. The tasks are randomly placed in a 3-D space, and vehicles on the 2-D ground space, with coordinates drawn from uniform distributions. The battery limit of each vehicle is set randomly between 1000 and 2000 s. Given the random initialization of task and vehicle locations and deadlines, it is sometimes impossible for some tasks to be started by any vehicle before their deadline. In these simulations, all task information is available to all vehicles up front. The task allocation procedure is performed before any tasks are executed, although previous studies have demonstrated a version of the PI algorithm that is effective at allocating new tasks online [50].

TABLE II
SCENARIO SPECIFICATION

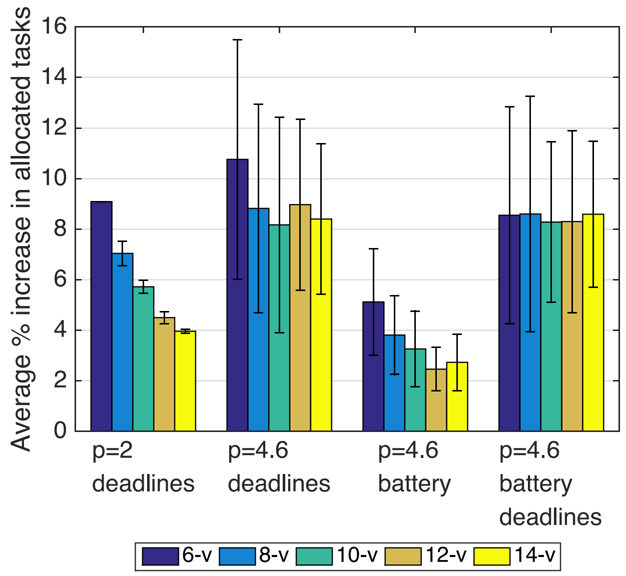
	Medicine	Food
Vehicle Speed	30m/s	50m/s
Vehicle Battery	Between 1000 and 2000 seconds	
Vehicle Start Position	10 000m x 10 000m x 0m ground space	
Task Duration	300 seconds	350 seconds
Task Deadline	Between 0 and 2000 seconds	
Task Location	10 000m x 10 000m x 1000m 3D space	

B. Simulation Results

1) PI-MinAvg Versus PI-MaxAss: Fig. 4 compares the PI-MinAvg solutions with the PI-MaxAss solutions that are initialized with the PI-MinAvg solution. A row formation was used for these experiments and a swap distance of 2 (SD = 2) was set. Fig. 4(a) shows the percentage of runs where PI-MaxAss increased the number of allocated tasks from the PI-MinAvg solution. Fig. 4(b) shows the corresponding average percentage change and standard deviation of number of allocated tasks when PI-MaxAss changed the number of allocated tasks.

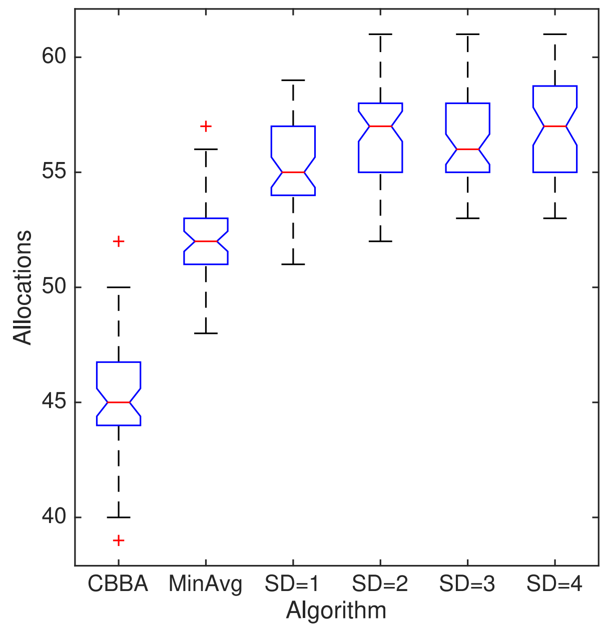


(a)

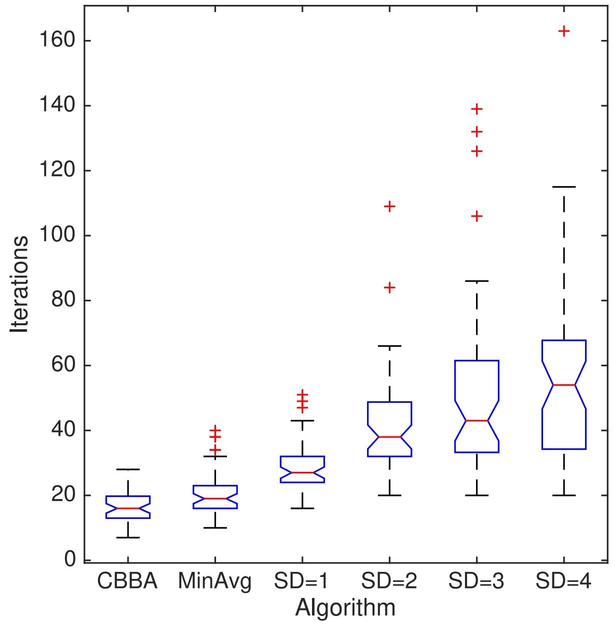


(b)

Fig. 4. For each scenario, the simulations were tested for an increasing number of vehicles and tasks. Vehicle numbers were 6, 8, 10, 12, and 14. For ratio $p = 2$ the number of tasks were 12, 16, 20, 24, and 28, and with ratio $p = 4.6$ task numbers were 28, 36, 46, 56, and 64. Ratio $p = 2$ was tested with task deadlines, ratio $p = 4.6$ was tested with task deadlines, with battery limits only, and with battery limits and task deadlines, respectively. In (a) each bar shows the percentage of solutions over 50 runs that PI-MaxAss assigned additional tasks starting from PI-MinAvg solution. (b) Corresponding average percentage change and standard deviation in number of allocated tasks when PI-MaxAss changed the number of allocated tasks.



(a)



(b)

Fig. 5. Comparison of CBBA, PI-MinAvg, PI-MaxAss with swap distance 1, 2, 3, and 4 on number of allocated tasks and iterations for the 14-vehicle 64-tasks scenario with battery limits and task deadlines. The plus symbols represent outliers. (a) Box plot of the total number of allocated tasks for each of the 50 runs for each algorithm. (b) Box plot of the number of total iterations for each of the 50 runs for each algorithm.

Fig. 4 shows both the results using the same experimental setup as in [20] and [22] with a task-to-vehicle ratio of 2 to 1 (ratio $p = 2$), deadlines for each task and without battery limit time constraints, and results using a task-to-vehicle ratio $p = 4.6$ with task deadlines only, vehicle battery limits only, and combined task deadlines and battery limits, respectively. Ratio $p = 4.6$ was selected to test the system approaching maximum capacity. In [20] and [22] experimental results showed that PI-MinAvg was capable of finding a solution that maximized the number of allocated tasks in most cases. The ratio $p = 2$ results in Fig. 4(a) reflect these findings. For each of the five setups with ratio $p = 2$, PI-MaxAss increased the number of allocated tasks from the PI-MinAvg solution; in the best case 14% of the runs were improved upon. In each run that PI-MaxAss increased the number of allocations (starting from PI-MinAvg with $p = 2$), one extra task was allocated. The results for ratio $p = 4.6$ show that when the system is approaching maximum capacity, i.e., when the order and allocation of tasks is critical to optimize number of allocated tasks, PI-MaxAss increased the number of task allocations in approximately half the runs with battery only time constraints and in up to 100% of runs with task deadlines. Up to three extra tasks were assigned in runs with battery only time constraints. Up to eight extra tasks were assigned in runs with task deadlines with ratio $p = 4.6$. In one such instance, PI-MaxAss increased the number of allocated tasks from 44 to 52 out of 56 tasks, where four tasks were impossible to allocate from the outset due to their relative positions and deadlines. In other words, PI-MaxAss facilitated an 18% increase in allocated tasks achieving the maximum allocation. In another instance, a 20% increase was achieved by increasing the number of allocations from 35 to 42 out of 46 tasks.

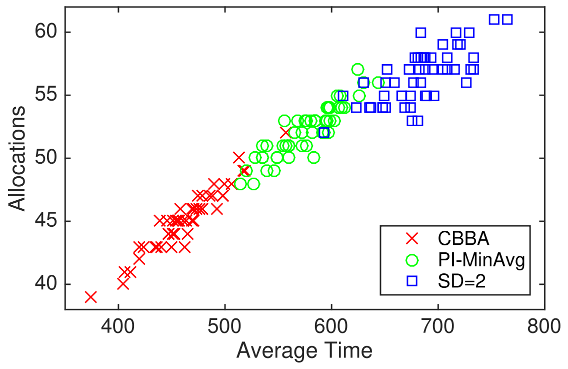
Over the 2000 runs, in six cases PI-MaxAss modified the solution by reassigning tasks without increasing the total number of assigned tasks. In all other instances that the solution was modified, the number of allocations was increased.

2) Swap Distance Parameter Comparison: Fig. 5 shows the results of a comparison between the performance of CBBA, PI-MinAvg, and PI-MaxAss with swap distance set between 1 and 4. The performance with regards to number of allocated tasks and number of iterations until convergence is presented. The total iterations for one simulation is determined by the last time an allocation change was made, either through inclusion or removal. As the PI-MaxAss solutions are initialized with the solutions from PI-MinAvg, the number of iterations for a run of PI-MaxAss is the sum of iterations taken for PI-MinAvg and PI-MaxAss, so PI-MaxAss will necessarily be at least as high as PI-MinAvg in all instances. Fig. 5(a) is a box plot [51] that shows the total number of allocated tasks for each algorithm. Fig. 5(b) is a box plot that shows the corresponding total number of iterations for each algorithm. The notches in the plots show that increases in allocated tasks between CBBA, PI-MinAvg, PI-MaxAss with SD = 1 and SD = 2 are statistically significant, and are correlated with an increase in iterations. For SD = 3 and SD = 4 there is an increase in iterations without a significant increase in task allocations compared with SD = 2. Table III shows that when the swap distance is limited to 1, an average of 3 extra tasks are allocated from the PI-MinAvg solution (shown in the table in the supplementary material) and the number of iterations has 95% confidence of being between the intervals 7.86 and 9.42 (not counting the iterations for PI-MinAvg). The tradeoff is just over 1 fewer allocated tasks on average compared with SD = 2. As the swap distance increases, the confidence intervals for the number of iterations also widen.

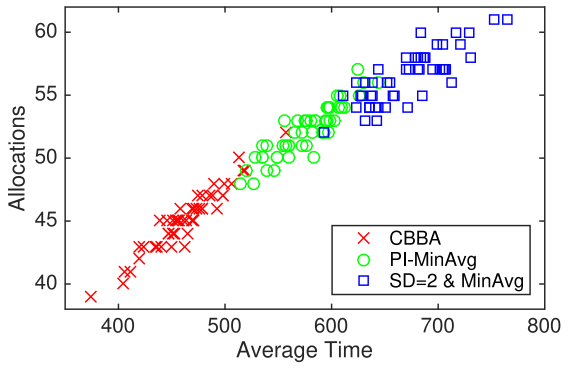
3) Average Time Comparison: Fig. 6(a) plots a comparison of the average waiting time and allocations for each run using SD = 2. Fig. 6(b) plots the same results using the starting solution of PI-MaxAss and shows the effect of switching back to optimizing waiting time after increasing allocated tasks with PI-MaxAss. Here, PI-MinAvg was initialized with the solution of PI-MaxAss. Average waiting time logically increases as more tasks are performed. This increase is reflected in the graphs that show a proportional increase in average waiting time between CBBA, PI-MinAvg, and PI-MaxAss. Fig. 6(b) shows that average waiting time can be optimized with PI-MinAvg after allocations have increased with PI-MaxAss. In 32 out of the 50 runs, the average waiting time is reduced, in the best case by 63 s. In this instance four extra tasks had been allocated with PI-MaxAss. The improvement in waiting time was achieved with nine iterations of PI-MinAvg. The average iterations for the second round of PI-MinAvg was 6.1 over the 50 runs.

TABLE III
AVERAGE TASK ALLOCATIONS AND ITERATIONS PERFORMANCE OF PI-MAXASS OVER 50 SIMULATIONS, WITH STANDARD DEVIATION AND CONFIDENCE INTERVALS FOR ITERATIONS

	Deadlines			
14-v 64-t	$SD = 1$	$SD = 2$	$SD = 3$	$SD = 4$
Allocations	57.7	58.8	59	59.2
Iterations	8.9	25.7	40.3	47.2
Std dev	3.0	23.8	27.0	26.0
95% confidence	8.1-9.8	19.0-32.5	32.6-48.0	39.8-54.6
	Batteries and Deadlines			
14-v 64-t	$SD = 1$	$SD = 2$	$SD = 3$	$SD = 4$
Allocations	55.1	56.4	56.7	56.8
Iterations	8.6	20.4	30.3	34.5
Std dev	2.7	15.2	25.8	26.1
95% confidence	7.9-9.4	16.0-24.7	23.0-37.7	27.1-41.9



(a)



(b)

Fig. 6. Scatter graphs comparing the performance of CBBA, PI-MinAvg, and PI-MaxAss with swap distance = 2, with respect to average waiting time for 50 runs. Each plot represents the final average waiting time of all assigned tasks for one run. In (b) PI-MinAvg was run starting from the solution of PI-MaxAss to show that average waiting time can be further optimized once additional tasks have been assigned. An improved average waiting time is indicated by points shifted to the left for SD = 2 & MinAvg compared with SD = 2.

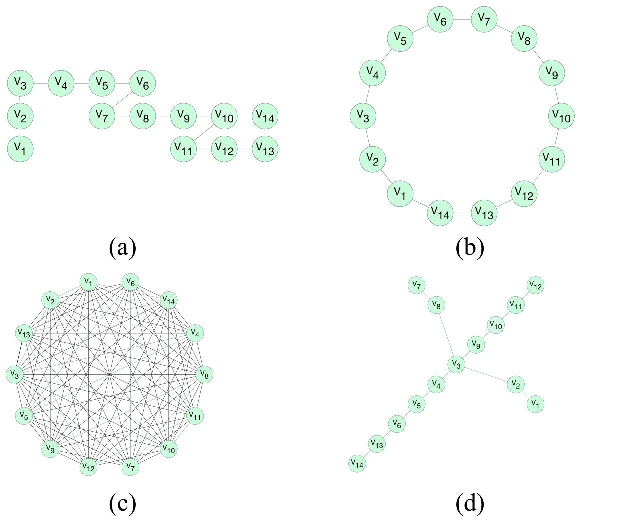


Fig. 7. Network topologies that the system was tested with. (a) Row topology. (b) Circular topology. (c) Mesh topology. (d) Star topology.

4) Topology Comparison: Changing topologies are inherent to dynamic environments with moving vehicles. It is therefore informative to assess how the proposed method performs across different topologies [12]. Fig. 7 illustrates with nondirected graphs the different network topologies under which the system was tested.

Fig. 8 shows the results of comparing different vehicle formation topologies in terms of the number of allocated tasks and iterations. The row topology, circular topology, the fully connected topology and the star topology illustrated in Fig. 7 are compared. The number of allocated tasks is consistent across topologies for CBBA and similar across topologies for PI-MinAvg and PI-MaxAss with SD = 2. Notable differences are the reduced number of iterations for each algorithm with the fully connected topology and the relative increase in iterations for the star topology for each algorithm.

V. Discussion

The results show that PI-MaxAss can significantly improve the total number of allocated tasks starting from a suboptimal solution. There is a tradeoff between computation time and solution quality that should be considered depending on the application [15]. Note that computation time here is represented by the number of iterations, while in practice the processing speed and the communication speed of the agents will determine how long an iteration lasts. If extra computation time is available, the results show that switching optimization objectives from minimizing average waiting time to maximizing task allocations can break the solution out of local optima and further optimize the task allocation without reducing the quality of the solution. After more tasks have been included, the quality of the solution can then be optimized further with few iterations by switching back to the time minimization method. This switching strategy as described in [15] exploits the high optimization performance of single-objective search algorithms for a bi-objective problem, while remaining flexible and modular.

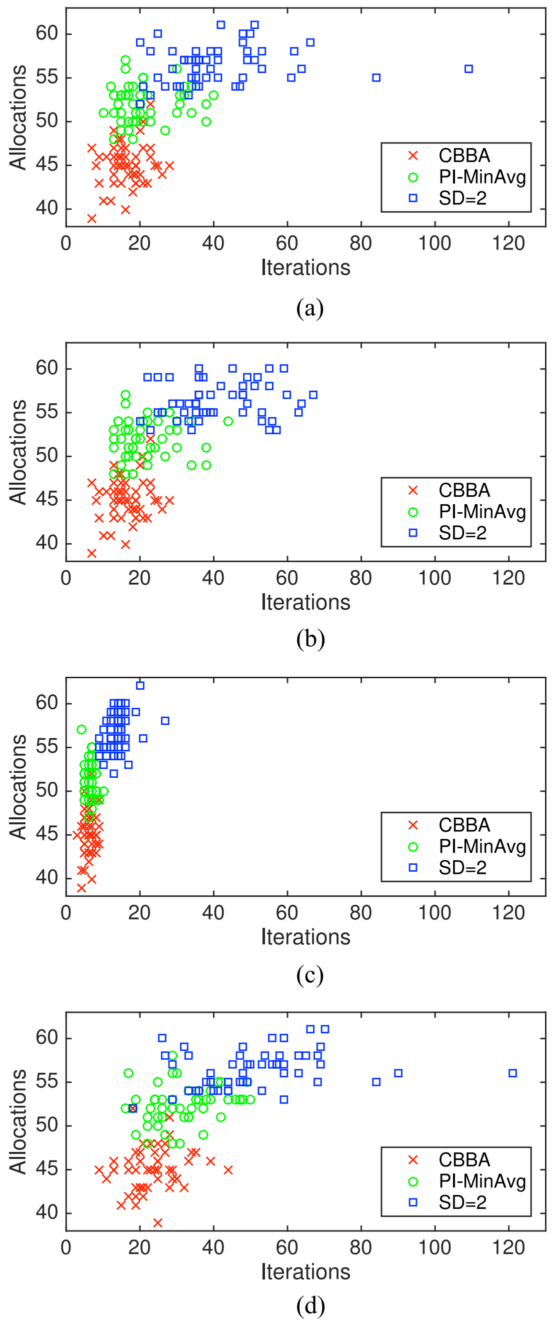


Fig. 8. Comparison of the performance over 50 runs of CBBA, PI-MinAvg, PI-MaxAss with swap distance 2 for 14-vehicle 64-task battery and deadlines scenario, with respect to number of allocated tasks and iterations over different network topologies, (a) row, (b) circular, (c) mesh, and (d) star topologies.

In the cases where PI-MinAvg was able to reach an optimal or near optimal solution with regards to the number of allocated tasks, such as the two tasks-per-vehicle scenario, PI-MaxAss made few or no improvements on the PI-MinAvg solution and accordingly the computation time was not unnecessarily increased. These results further support the switching strategy [15] which increased computation time only when the solution could be improved by the proposed method PI-MaxAss.

The results show that a swap distance limited to 1 is preferable when a reliably low number of iterations is required while still providing a significantly higher number of allocated tasks. A higher swap distance can be used if the extra computation time is available to increase the likeliness of finding a better solution. On the other hand, although PI-MaxAss is guaranteed not to decrease allocations starting from an initial task allocation, it cannot be guaranteed that PI-MaxAss with a higher swap distance finds an equal or higher task allocation than a lower swap distance.

For the scenarios tested, a swap distance of 3 or 4 did not significantly increase the allocations despite the correlated increase in iterations. For each additional task reassignment, the new task allocations are propagated through the network of vehicles, and this can take several iterations depending on the network topology meaning that, as the number of reassignments increases, so do the number of iterations. It is also likely that the number of instances where 3 or 4 reassignments are required are fewer than those requiring 1 or 2 reassignments. This may result in an insignificant increase in task allocations along with a relatively high increase in number of iterations.

PI-MaxAss was shown to be effective at increasing allocated tasks when the time constraint was on vehicle battery limits only. In these cases, the extra flexibility in the possible ordering of task allocations meant that PI-MinAvg was more likely to find an optimal solution, however, PI-MaxAss increased the allocations in about half of the runs, a noteworthy proportion.

In 0.3% of 2000 runs, PI-MaxAss modified the solution by reassigning tasks without increasing the total number of assigned tasks. This may happen because an additional task allocation attempt may be inhibited if a time slot created to assign a new task is instead filled by a task later in that reassignment sequence.

Tests with different topologies provided strong evidence that the number of allocated tasks is independent of the specific topology. The number of iterations required to reach consensus, on the contrary, appears to vary according to the type of topology. The increase in iterations is due to information requiring multiple iterations or "hops" to reach all vehicles when the network is not fully connected. In general, the longer the network diameter, i.e., the shortest path between the two most distant vehicles, the longer the system takes to reach consensus.

The task shifting effect of PI-MaxAss is similar to the theoretical task swap loop methods described and analyzed in [35]–[38] and [52]. Compared with these methods, PI-MaxAss has the advantage that it does not require distinguishing roles. Furthermore, PI-MaxAss does not require finding a complete swap loop to reassign tasks. As opposed to the task swap loop methods, with PI-MaxAss the last task reassignment in the sequence need not be assigned to the vehicle that started the sequence. By following the task swap loop strategy, the created time slot is more likely to be filled by the task being reassigned from another vehicle, inhibiting the assignment of an additional unassigned task. A final distinction is that the objective of PI-MaxAss is to increase the number of task assignments within vehicles' schedules, whereas the costs being minimized in [36]–[38] are nonspecific, and the problem being addressed considers vehicles that can be assigned one task each, at most.

VI. Conclusion

In a search and rescue mission, optimal task allocation for available vehicles is crucial. In this paper, an effective algorithm that allows for simple and efficient reassignment of allocated tasks is proposed and analyzed to improve the task allocation solution of a previous method for task allocation. The novel idea is to allow vehicles to reallocate tasks to create a feasible space for unallocated tasks by taking advantage of existing schedule space. Simulations showed a noteworthy increase in performance, measured as the total number of allocated tasks, making the method appealing when this objective is a priority. An increment in the number of iterations appeared proportionate to the gain in performance. Experimental results confirmed that the proposed algorithm can be applied beneficially to an existing scheduling method, thus opening the possibility of integration to other implementations.

Future work will look at restricting the number of iterations used to reach consensus while maintaining the solution quality, as well as implementation in more realistic testing scenarios that could include having the vehicles returning to a base to refuel.

Acknowledgment

The authors would like to thank Dr. D. Sibley for his insightful feedback.

References

[1] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *J. Intell. Robot. Syst.*, vol. 72, no. 2, pp. 147–165, 2013.
[2] A. Q. Li, R. Cipolleschi, M. Giusto, and F. Amigoni, "A semantically-informed multirobot system for exploration of relevant areas in search and rescue settings," *Auton. Robots*, vol. 40, no. 4, pp. 581–597, 2016.
[3] J. Parker, E. Nunes, J. Godoy, and M. Gini, "Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork," *J. Field Robot.*, vol. 33, no. 7, pp. 877–900, 2016.
[4] R. R. Murphy, *Disaster Robotics*. Cambridge, MA, USA: MIT Press, 2014.
[5] A. C. Kapoutsis *et al.*, "Real-time adaptive multi-robot exploration with application to underwater map construction," *Auton. Robots*, vol. 40, no. 6, pp. 987–1015, 2016.
[6] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," *J. Intell. Robot. Syst.*, vol. 80, no. 1, pp. 33–58, 2015.
[7] G. Binetti, D. Naso, and B. Turchiano, "Decentralized task allocation for surveillance systems with critical tasks," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1653–1664, 2013.
[8] C. Robin and S. Lacroix, "Multi-robot target detection and tracking: Taxonomy and survey," *Auton. Robots*, vol. 40, no. 4, pp. 729–760, 2015.
[9] L. Wang, S. Keshavarzmanesh, H.-Y. Feng, and R. O. Buchal, "Assembly process planning and its future in collaborative manufacturing: A review," *Int. J. Adv. Manuf. Technol.*, vol. 41, nos. 1–2, pp. 132–144, 2009.
[10] H. Fazlollahtabar, M. Saidi-Mehrabad, and J. Balakrishnan, "Mathematical optimization for earliness/tardiness minimization in a multiple automated guided vehicle manufacturing system via integrated heuristic algorithms," *Robot. Auton. Syst.*, vol. 72, pp. 131–138, Oct. 2015.
[11] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proc. IEEE*, vol. 94, no. 7, pp. 1257–1270, Jul. 2006.
[12] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 427–438, Feb. 2013.
[13] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, 2004.

[14] S. Alatartsev, S. Stellmacher, and F. Ortmeier, "Robotic task sequencing problem: A survey," *J. Intell. Robot. Syst.*, vol. 80, no. 2, pp. 279–298, 2015.

[15] L. Paquete and T. Stützle, "A two-phase local search for the biobjective traveling salesman problem," in *Evolutionary Multi-Criterion Optimization* (LNCS 2632). Heidelberg, Germany: Springer, 2003, pp. 479–493.

[16] N. Jozefowiez, F. Semet, and E.-G. Talbi, "Multi-objective vehicle routing problems," *Eur. J. Oper. Res.*, vol. 189, no. 2, pp. 293–309, 2008.

[17] A. T. Tolmidis and L. Petrou, "Multi-objective optimization for dynamic task allocation in a multi-robot system," *Eng. Appl. Artif. Intell.*, vol. 26, nos. 5–6, pp. 1458–1468, May 2013.

[18] M. Statheropoulos *et al.*, "Factors that affect rescue time in urban search and rescue (USAR) operations," *Nat. Hazards*, vol. 75, no. 1, pp. 57–69, 2015.

[19] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *Int. J. Robot. Res.*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.

[20] A. Whitbrook, Q. Meng, and P. W. H. Chung, "A novel distributed scheduling algorithm for time-critical multi-agent systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Hamburg, Germany, 2015, pp. 6451–6458.

[21] J. Turner, Q. Meng, and G. Schaefer, "Increasing allocated tasks with a time minimization algorithm for a search and rescue scenario," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Seattle, WA, USA, 2015, pp. 3401–3407.

[22] W. Zhao, Q. Meng, and P. W. H. Chung, "A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[23] M. M. Flood, "The traveling salesman problem," *Oper. Res.*, vol. 4, no. 1, pp. 61–75, 1956.

[24] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *Eur. J. Oper. Res.*, vol. 209, no. 1, pp. 1–10, 2011.

[25] J. Fakcharoenphol, C. Harrelson, and S. Rao, "The K-traveling repairmen problem," *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007, Art. no. 40.

[26] A. Lee and M. Savelsbergh, "Dynamic ridesharing: Is there a role for dedicated drivers?" *Transp. Res. B Methodol.*, vol. 81, pp. 483–497, Nov. 2015.

[27] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, 1973.

[28] H. Tang and E. Miller-Hooks, "A TABU search heuristic for the team orienteering problem," *Comput. Oper. Res.*, vol. 32, no. 6, pp. 1379–1407, 2005.

[29] S.-W. Lin and V. F. Yu, "A simulated annealing heuristic for the team orienteering problem with time windows," *Eur. J. Oper. Res.*, vol. 217, no. 1, pp. 94–107, Feb. 2012.

[30] Z. Luo, H. Qin, and A. Lim, "Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints," *Eur. J. Oper. Res.*, vol. 234, no. 1, pp. 49–60, Apr. 2014.

[31] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*, vol. 604. Cham, Switzerland: Springer, 2015, pp. 31–51.

[32] S. E. Butt and T. M. Cavalier, "A heuristic for the multiple tour maximum collection problem," *Comput. Oper. Res.*, vol. 21, no. 1, pp. 101–111, 1994.

[33] M. G. Lagoudakis *et al.*, "Auction-based multi-robot routing," *Robot. Sci. Syst.*, vol. 5, p. 98, Jun. 2005.

[34] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robot. Auton. Syst.*, vol. 58, no. 7, pp. 900–909, Jul. 2010.

[35] X. Zheng and S. Koenig, "K-swaps: Cooperative negotiation for solving task-allocation problems," in *Proc. Int. Joint Conf. Artif. Intell.*, Pasadena, CA, USA, 2009, pp. 373–379.

[36] L. Liu and D. A. Shell, "An anytime assignment algorithm: From local task swapping to global optimality," *Auton. Robots*, vol. 35, no. 4, pp. 271–286, 2013.

[37] L. Liu, N. Michael, and D. A. Shell, "Fully decentralized task swaps with optimized local searching," in *Proc. Robot. Sci. Syst.*, Berkeley, CA, USA, 2014, pp. 429–444.

[38] L. Liu, N. Michael, and D. A. Shell, "Communication constrained task allocation with optimized local task swaps," *Auton. Robots*, vol. 39, no. 3, pp. 429–444, 2015.

[39] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[40] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *Proc. Amer. Control Conf.*, Baltimore, MD, USA, 2010, pp. 3057–3062.

[41] S. Hunt, Q. Meng, and C. J. Hinde, "An extension of the consensus-based bundle algorithm for multi-agent tasks with task based requirements," in *Proc. 11th Int. Conf. Mach. Learn. Appl.*, Boca Raton, FL, USA, 2012, pp. 451–456.

[42] S. Ponda *et al.*, "Decentralized planning for complex missions with dynamic communication constraints," in *Proc. Amer. Control Conf.*, Baltimore, MD, USA, 2010, pp. 3998–4003.

[43] L. B. Johnson, S. S. Ponda, H.-L. Choi, and J. P. How, "Improving the efficiency of a decentralized tasking algorithm for UAV teams with asynchronous communications," in *Proc. AIAA Guid. Navig. Control Conf.*, 2010, pp. 7572–7593.

[44] S. S. Ponda, L. B. Johnson, A. N. Kopeikin, H.-L. Choi, and J. P. How, "Distributed planning strategies to ensure network connectivity for dynamic heterogeneous teams," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 5, pp. 861–869, Jun. 2012.

[45] D. Di Paola, D. Naso, and B. Turchiano, "Consensus-based robust decentralized task assignment for heterogeneous robot networks," in *Proc. Amer. Control Conf.*, San Francisco, CA, USA, 2011, pp. 4711–4716.

[46] D. Di Paola, A. Gasparri, D. Naso, and F. L. Lewis, "Decentralized dynamic task planning for heterogeneous robotic networks," *Auton. Robots*, vol. 38, no. 1, pp. 31–48, 2014.

[47] G. Binetti, D. Naso, and B. Turchiano, "Decentralized task allocation for heterogeneous agent systems with constraints on agent capacity and critical tasks," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, Guangzhou, China, 2012, pp. 1627–1632.

[48] R. Cui, J. Guo, and B. Gao, "Game theory-based negotiation for multiple robots task allocation," *Robotica*, vol. 31, no. 6, pp. 923–934, 2013.

[49] D. Smith, J. Wetherall, S. Woodhead, and A. Adekunle, "A cluster-based approach to consensus based distributed task allocation," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Netw. Based Process.*, Turin, Italy, 2014, pp. 428–431.

[50] A. Whitbrook, Q. Meng, and P. W. H. Chung, "Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems," *IEEE Trans. Autom. Sci. Eng.*, to be published, doi: 10.1109/TASE.2017.2679278.

[51] R. McGill, J. W. Tukey, and W. A. Larsen, "Variations of box plots," *Amer. Stat.*, vol. 32, no. 1, pp. 12–16, 1978.

[52] C. Sung, N. Ayanian, and D. Rus, "Improving the performance of multi-robot systems by task switching," in *Proc. IEEE Int. Conf. Robot. Autom.*, Karlsruhe, Germany, 2013, pp. 2999–3006.

Joanna Turner received the B.Sc. degree in computer science from Loughborough University, Loughborough, U.K., in 2013, where she is currently pursuing the Ph.D. degree.

Her current research interests include multiagent task allocation, machine learning, and biologically inspired artificial intelligence.

Qinggang Meng (M'06) received the B.Sc. and M.Sc. degrees in electronic engineering from Tianjin University, Tianjin, China, and the Ph.D. degree in computer science from Aberystwyth University, Aberystwyth, U.K.

He is currently a Reader in robotics and autonomous systems with the Department of Computer Science, Loughborough University, Loughborough, U.K. His current research interests include biologically and psychologically inspired learning algorithms and developmental robotics, robot learning and adaptation, autonomous vehicles/systems, multi-UAV/UGV cooperation, service and assistive robotics, situation awareness and decision making for driverless vehicles, verification and validation of autonomous systems, driver's distraction detection, human motion analysis and activity recognition, activity pattern detection, pattern recognition, artificial intelligence, machine learning, deep learning, and computer vision.

Gerald Schaefer (M'04) received the Ph.D. degree in computer vision from the University of East Anglia, Norwich, U.K.

He was with the Colour and Imaging Institute, University of Derby, Derby, U.K., from 1997 to 1999, the School of Information Systems, University of East Anglia, Norwich, U.K., from 2000 to 2001, the School of Computing and Informatics, Nottingham Trent University, Nottingham, U.K., from 2001 to 2006, and the School of Engineering and Applied Science, Aston University, Birmingham, U.K., from 2006 to 2009, before joining the Department of Computer Science, Loughborough University, Loughborough, U.K. His current research interests include color image analysis, image retrieval, physics-based vision, medical imaging, and computational intelligence. He has published extensively in these areas with a total publication count exceeding 450.

Dr. Schaefer is/was a member of the editorial board of over 20 international journals, has reviewed for over 120 journals and served on the programme committee of over 400 conferences. He has been invited as a Keynote or Tutorial Speaker to numerous conferences, is the Organizer of various international workshops and special sessions at conferences, and is the Editor of several books, conference proceedings, and special journal issues.

Amanda Whitbrook received the B.Sc. (Hons.) degree in mathematics and physics from Nottingham Trent University, Nottingham, U.K., in 1993, the M.Sc. degree in management of information technology from the University of Nottingham, Nottingham, in 2005, and the Ph.D. degree in applied mathematics (numerical analysis) from Nottingham Trent University in 1998.

She was a Research Fellow and a Teaching Associate with the University of Nottingham, a Senior Scientist with the Autonomous Systems Research Group, BAE Systems, Farnborough, U.K., and a Research Associate with Loughborough University, Loughborough, U.K. She is currently a Lecturer in computer science with the Department of Electronics, Computing and Mathematics, University of Derby, Derby, U.K. Her current research interests include biologically inspired artificial intelligence, artificial immune systems, genetic algorithms, swarm optimization, mobile robot navigation, artificial intelligence for computer games, and heuristic methods for multiagent task allocation.

Andrea Soltoggio received the B.Sc. and M.Sc. equivalent degrees in computer engineering from the Norwegian University of Science and Technology, Trondheim, Norway, and the Politecnico di Milano, Milan, Italy, in 2004, and the Ph.D. degree in computer science from the University of Birmingham, Birmingham, U.K., in 2009.

He was with the Laboratory of Intelligent Systems, EPFL, Lausanne, Switzerland, and was a Visiting Researcher with the University of Central Florida, Orlando, FL, USA. From 2010 to 2014, he was a Technical Coordinator of the EU-funded FP7-IP project AMARSi with the Research Institute for Cognition and Robotics, Bielefeld University, Bielefeld, Germany. Since 2014, he has been a Lecturer in computer science with Loughborough University, Loughborough, U.K. His current research interests encompass evolutionary computation, learning and plasticity in neural networks, and broader aspects of cognition and intelligence.