Reliable, Distributed Scheduling and Rescheduling for Time-Critical, Multiagent Systems

Amanda Whitbrook, Qinggang Meng, *Member, IEEE*, and Paul W. H. Chung

***Abstract*— This paper addresses two main problems with many heuristic task allocation approaches—solution trapping in local minima and static structure. The existing distributed task allocation algorithm known as performance impact (PI) is used as the vehicle for developing solutions to these problems as it has been shown to outperform the state-of-the-art consensus-based bundle algorithm for time-critical problems with tight deadlines, but is both static and suboptimal with a tendency toward trapping in local minima. This paper describes two additional modules that are easily integrated with PI. The first extends the algorithm to permit dynamic online rescheduling in real time, and the second boosts performance by introducing an additional soft-max action-selection procedure that increases the algorithm's exploratory properties. This paper demonstrates the effectiveness of the dynamic rescheduling module and shows that the average time taken to perform tasks can be reduced by up to 9% when the soft-max module is used. In addition, the solution of some problems that baseline PI cannot handle is enabled by the second module. These developments represent a significant advance in the state of the art for multiagent, time-critical task assignment.**

***Note to Practitioners*—This work was motivated by the limitations of current agent-to-task allocation algorithms that do not use a central server for communication. In previously published work, the current state-of-the-art consensus-based bundle algorithm has demonstrated poor performance when applied to model task allocation problems with critical time limits, often failing to assign all of the tasks, especially when the deadlines are tight. The performance impact (PI) algorithm has a much better success rate with these model problems but would be flawed when applied to real missions because it has no mechanism for online replanning when new information becomes available. In addition, it is somewhat restricted in the way it searches for a problem solution, meaning that more efficient plans are often available but are not discovered. This paper tackles both of these shortcomings. The PI algorithm is extended to include a module that permits rescheduling when necessary, and a further module is introduced that widens the scope of the solution search. A third module that is able to offer robust plans, even for large-scaled missions involving many agents and tasks, has also been developed, although it is not discussed here. Implementation and testing of a version of PI that incorporates all three of these modules are the final goal of this research.**

***Index Terms*— Adaptive systems, auction-based scheduling, distributed task allocation, multiagent systems.**

Manuscript received August 18, 2016; revised January 6, 2017; accepted February 23, 2017. This paper was recommended for publication by Associate Editor l. Tang and Editor M. P. Fanti upon evaluation of the reviewers' comments. This work was supported by the Engineering and Physical Sciences Research Council under Grant ep/j011525/1 with BAE Systems as the leading industrial partner.

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).

Q. Meng and P. W. H. Chung are with the Department of Computer Science, Loughborough University, Loughborough, LE11 3TU, U.K. (e-mail: q.meng@lboro.ac.uk; p.w.h.chung@lboro.ac.uk).

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/TASE.2017.2679278

I. Introduction

SINGLE-TASK, single-robot (ST-SR) task allocation problems have been defined in [1] as consisting of agents capable of executing at most one task at a time, and tasks that require only one agent to complete them. In addition, the subclass of time-extended allocation problems (ST-SR-TA problems) has a temporal horizon and more tasks than agents to service them. These problems have many properties in common with manufacturing scheduling problems (see [2] and [3] for a full review) such as the well-known job shop scheduling problem, where a set of jobs is to be allocated to a set of machines that can handle only one job at a time. Each job needs to be completed during an uninterrupted time period of a given length on a given machine. If each job can be completed by a single machine, then the problems are equivalent. In both cases, the objective is to find an optimum assignment that satisfies particular constraints, for example an assignment that minimizes total duration to complete all tasks or jobs. Such problems are strongly NP-hard [4] as they are complex, combinatorial decision problems. For example, for the more general job shop scheduling problem (where jobs may require more than one machine), the number of possible solutions is equal to $(m!)^n$ where m is the number of jobs and n is the number of machines [5]. Solution methods for ST-SR-TA problems thus tend to scale very poorly as the computation time increases exponentially with the problem size [6]. This means that analytic methods such as linear programming (LP) or mixed integer LP are not suitable for large, complex problems of this type; the methods become intractable because of the exponential number of constraints in the model [1]. In addition, simple, uninformed search-based methods cannot deal with the large solution space. Much research effort has therefore been directed toward designing search-based methods that incorporate some sort of heuristic, for example Tabu-search [7], genetic algorithms [8], [9], simulated annealing [10], and artificial neural networks [11]. These algorithms are generally referred to as traditional approaches. There is also a rich literature on heuristic distributed agent-based strategies, for example Yet Another Manufacturing System (YAMS) (for manufacturing scheduling) [12] and auction-based approaches [13]. Although heuristic methods can produce good solutions in reduced time [2], a disadvantage with many is that they

often provide suboptimal solutions. Moreover, many heuristic algorithms used for ST-SR-TA task allocation problems (including the examples given) are static in nature with no dynamic rescheduling version available. This is an important consideration for many real-world task allocation problems, for example search-and-rescue missions (SAR), which take place in dynamically changing environments with situational awareness (SA) subject to change at any time.

The performance impact (PI) algorithm [14] is a distributed heuristic algorithm that has shown good performance for solving difficult, time-critical problems when compared to other auction-based methods, notably the consensus-based bundle algorithm (CBBA) [13], but it is a static program that needs to be run offline, and also suffers from suboptimality because the solution is prone to trapping in local minima. This paper aims to solve both of these problems and thus address the two main drawbacks of heuristic approaches—local minima traps and lack of a rescheduling mechanism. First, baseline PI is developed into a dynamic search algorithm, i.e., the online reassignment of tasks during the course of the mission is enabled. Thus, as soon as new information becomes available, the algorithm can dynamically reschedule. The modifications made enable the algorithm to handle new information concerning the locations of tasks, the addition of new tasks, the removal of tasks, and the addition and removal of agents from the team; this functionality was not available in PI previously. Numerous test scenarios have validated the dynamic capability and have demonstrated the benefits of reassignment compared to proceeding with the original plan. Second, baseline PI is extended to include an appropriate combination of PI task selection and soft-max task selection. This development improves performance and boosts the exploratory properties of the algorithm, meaning that escape from local minima is possible. For each problem, the parameter that determines the best action-selection combination is obtained by repeatedly solving between start and end values in suitable steps, until the best solution is found. Extensive testing under several different scenarios and network topologies is carried out to show empirically that the enhancement can improve the performance of the baseline PI algorithm by up to 9%, and enables solution of some problems that the baseline cannot handle. The search for an optimal soft-max parameter introduces a tradeoff between increasing solution time and boosting solution quality. A method for reducing the search time without compromising performance is thus put forward.

Development of PI to include dynamic rescheduling and escape from local minima is the main contribution of this paper. The enhancements are fully described in Sections IV-B and V-B, and the results for each additional module show that they represent an advance in the state of the art in multiagent task assignment for time-critical systems. In addition, specific details of the dynamic rescheduling aspects of distributed ST-SR-TA task allocation algorithms are not well-documented in the literature; this paper aims to fill the gap.

II. Literature Review

As stated earlier, the approaches to solving complex scheduling problems such as ST-SR-TA problems and manufacturing scheduling problems can be categorized into traditional methods and distributed agent-based methods. For a general discussion of the advantages and disadvantages of these categories see [2] and [3]. An example traditional method is the use of genetic or memetic algorithms to evolve a problem solution (see [8], [9], [15], and [16]). The main advantage of these methods is the ability to solve higher dimensioned problems considerably faster than some other search methods [17], but optimality is not guaranteed.

Agent-based approaches were first proposed in [18] for solving manufacturing scheduling problems and their application to general task-allocation problems is well documented (see [19]), where agent behavior is modeled on the division of labor in social insects. When designing such systems an important consideration is the choice of either a centralized or distributed communication architecture. Agents continually need to share information about their current task set, and centralized approaches (see [15] and [16]) incur a high communication overhead for larger systems, are vulnerable to single-point failure and have a limited range. However, centralized systems are generally simpler to implement and tend to run faster as no consensus processing stage is required to ensure that the agents have identical SA or identical solutions [13]. Alternatively, distributed systems (see [20]), where a scheduling algorithm is instantiated in each agent, require less communication bandwidth [21], allow an extended range, and have no single-point failure vulnerability. However, in real networks, where communication is sometimes limited, inconsistencies in the SA or the generation of different local solutions can lead to conflicting assignments [22], meaning that some form of consensus algorithm is necessary [23]. These consensus-before-planning algorithms provide an additional computational and data processing burden, which can slow down performance, but they have been shown to be robust to different network topologies [13]. For a full discussion of centralized and distributed auction methods see [24].

Many agent-based methods involve iterative task allocation, for example market-based decision strategies [25], where each agent is modeled as a self-interested party, and the whole fleet as an economy. The agents must maximize their own profit by making deals with others in the form of bidding for different tasks. Globally, the profit (revenue minus cost) must be maximized.

Auction-based algorithms (see [26]–[28]), which are a subset of market-based methods, have also been applied to ST-SR-TA task-allocation problems. In these algorithms, each agent bids on a task based on information from its own SA, and the highest bidder wins the task assignment. Either a central system or one of the bidders can act as the auctioneer. Auction-based methods are generally robust to inconsistencies in the SA, and have been shown to produce suboptimal solutions efficiently [13].

Choi *et al.* [13] have shown that their distributed CBBA algorithm effectively combines the positive properties of auction-based and consensus-before-planning approaches, producing conflict-free solutions independent of inconsistencies in the SA. Task selection is implemented via a decentralized auction phase, and agreement on the winning bids (rather

than the SA) is achieved through a consensus phase that also serves to release tasks that have been outbid. Application of the auction method to TA problems is made possible by grouping common tasks into bundles and allowing the agents to bid on the bundles rather than individual tasks, with the bundles continuously updating as the auction proceeds. This approach is known as a combinatorial auction. Choi *et al.* [30] show that the method produces similar solutions to some centralized sequential greedy procedures, and 50% optimality is guaranteed. Task bundling auction methods are also described in [29] and [30].

In CBBA, the bundles are formed by logically grouping similar tasks, as it would be too computationally costly to enumerate all possible bundles. The PI algorithm [14] is also a combinatorial auction method but builds its bundles iteratively using novel PI metrics designed to exploit the synergies between tasks to decrease global cost (see Section III-C for full details). The PI task swapping mechanism is similar to that used in Durfee and Lesser's partial global plan (PGP) algorithm for distributed problem solving [31]. Both techniques involve local planners that rate task order based on time costs and the effects on preceding and future tasks, and then use hill-climbing techniques [32] to generate a satisfactory ordering that is not necessarily optimal. One of the main differences is that the PGP does not iteratively test all possible combinations of task bundles and their ordering using specialized metrics. Instead, more highly rated tasks are simply moved to a slot earlier in the plan. In addition, instead of executing a separate consensus phase that takes the input from each agent into consideration, partial plans from each agent are periodically combined to create larger ones.

In [14], the distributed PI algorithm has been shown empirically to solve model task allocation problems with tight deadlines more effectively than the CBBA method. When solving a number of time-critical ST-SR-TA problems with different network topologies, different numbers of agents and tasks, and randomly generated locations for agents and tasks, the PI approach demonstrates a consistently lower mean time cost, and is able to solve many problems that the CBBA method cannot. However, neither PI nor CBBA can handle dynamic rescheduling, and PI's solutions are prone to trapping in local minima. It is clear that a rescheduling variant of PI is necessary, and the local minima problem needs addressing.

Traditional methods for solving task allocation problems are not generally suited for dynamic rescheduling as they are often too inflexible and too slow [33]. However, agent-based approaches have been reported for manufacturing scheduling (see [33] and [34]). In [33], a scheduling mechanism that evolves dynamically is used to combine centralized and distributed strategies. A global optimized schedule is initially implemented and a fast rescheduling solution is called for whenever changes occur as this is faster than waiting for an optimized schedule. In [34], a mediator mechanism is used to help agents dynamically find other agents that can contribute to a given task, and negotiation is through the contract net protocol [35]. However, these architectures are not suited to the problems of interest in this paper (ST-SR-TA problems) as here, each task requires only one agent for completion. Rescheduling algorithms for problems of this type are scarce in the literature.

III. Problem Definition, Scenario, and PI Architecture

A. Problem Summary

The set of problems of interest in this paper are formulated mathematically by defining a set of n heterogeneous agents $\boldsymbol{V} = [v_1, \ldots, v_n]^{\mathrm{T}}$, a set of m heterogeneous tasks $\boldsymbol{T} = [t_1, \ldots, t_m]^{\mathrm{T}}$, $m > n$, and a set of ordered task allocations $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n]^{\mathrm{T}}$, where $\boldsymbol{a}_i$, $i = 1, \ldots, n$, is the task list assigned to agent v_i. Note that the actual size ϱ_i of a task list $\boldsymbol{a}_i$ may vary between agents; for example, agent v_1 may be assigned a single task ($\varrho_1 = 1$), whereas agent v_2 may be assigned three tasks ($\varrho_2 = 3$). However, mathematically $|\boldsymbol{a}_i| \equiv m$ as the corresponding elements of any unassigned tasks are given value -1 in each set of allocations $\boldsymbol{a}_i$. A compatibility matrix $\boldsymbol{H}$ with entries $h_{i,j} \in [0, 1]$ defines whether agent v_i is able to perform task t_j as there may be different task types. (The value is 1 if it is able, 0 otherwise.) In addition, each task has a maximum (latest) start time $\boldsymbol{S} = [s_1, \ldots, s_m]^{\mathrm{T}}$ after which it cannot commence, i.e., the problem has the additional complexity of being time-limited. The time cost $c_{i,k}$ for a particular agent v_i and task t_k is defined as the time of arrival of agent v_i at the location of task t_k; the time spent in that location servicing task t_k is not included; instead, this contributes toward the arrival time of agent v_i at the next task t_{k+1}. In addition, note that time costs are cumulative so that the cost of servicing task t_k includes the cost of servicing all the tasks previous to it. Each task requires only one agent, and each agent can complete only one task at a time, although it can complete other tasks afterward, provided that there is enough time. The problem falls into the general category of ST-SR task allocation problems, as defined in [1], and since there are always more tasks than agents available to service them, the problem is also a time-extended assignment type, i.e., it is an ST-SR-TA system under the same taxonomy.

The solution requires a conflict-free assignment of agents to tasks that minimizes some global penalty. In this particular case, the global objective function is to minimize φ the mean single time cost (as defined above) over all tasks, that is

$$\varphi = \frac{1}{m} \sum_{i=1}^{n} \sum_{k=1}^{\varrho_i} c_{i,k}(\boldsymbol{a}_i) \tag{1}$$

where ϱ_i is the number of tasks assigned to agent v_i, and $c_{i,k}(\boldsymbol{a}_i)$ is the time cost incurred by agent v_i servicing the kth task in its task list (see Section III-C for a justification of the choice of objective function). Any solution that includes tasks that cannot commence in time is a failed solution; the time-cost in such cases is nonexistent, and the objective function cannot be calculated.

The constraints in the optimization problem are as follows:

$$\boldsymbol{a}_i \leq m \tag{2}$$

$$\bigcup_{i=1}^{n} \boldsymbol{a}_i = \boldsymbol{T}, \quad \boldsymbol{a}_i \cap \boldsymbol{a}_j = \emptyset \quad \forall i \neq j \tag{3}$$

$$h_{i,k} \in [0, 1] \tag{4}$$

$$c_{i,k}(\boldsymbol{a}_i) \leq s_k. \tag{5}$$

Equation (2) constrains the number of tasks assigned to a particular agent to be less than or equal to the total number of tasks. Equation (3) shows that each set of allocations is a subset of the whole set of tasks, that tasks cannot be assigned to multiple agents, and that all tasks must be assigned to some agent. Equation (4) constrains the elements of the compatibility matrix to either 0 or 1 in value, and (5) represents the task arrival time constraint.

B. Test Scenario

In this paper, the particular scenario selected for application of the problem type is based on the rescue aspect of urban SAR (USAR). The agents are vehicles [unmanned air vehicles (UAVs) carrying food and helicopters carrying medicine], and their mission is to rescue disaster survivors by delivering their supplies to them. Each survivor requires either food or medicine and their delivery constitutes the completion of a task. The start locations of the UAVs and helicopters are known in advance, as are the 3-D locations and requirements of the survivors, although note that task locations are assumed to be estimates that are subject to change in the reassignment work (Section IV). The allocation of a maximum start time s_k for each task represents the constraint that each survivor must be rescued before their health condition deteriorates completely. Note that the problem type described in Section III-A is not restricted to USAR applications; it applies to any distributed, time-critical scheduling problem.

C. PI Architecture

PI is a distributed task allocation algorithm that runs separately on each vehicle in the fleet. It uses a combinatorial auction-based approach, but introduces the concept of PI as a score function to build bundles iteratively by adding and removing tasks and testing the impact on global cost. As in the CBBA algorithm, there is a local task allocation phase in which each vehicle generates a bundle of tasks, and a task consensus phase that resolves conflicts through local communication between connected vehicles. The two phases are repeated until convergence, i.e., until a conflict-free task assignment is reached.

There are two types of PI, removal PI (RPI) and inclusion PI (IPI), which are now explained. The RPI $w_k(\boldsymbol{a}_i \ominus t_k)$ of task t_k to its assigned vehicle v_i is the cost of performing a removed task plus the difference in cost (with and without the removed task) of performing all subsequent tasks. It represents the contribution of a task to the local cost generated by a vehicle. RPI is defined as

$$w_k(\boldsymbol{a}_i \ominus t_k) = c_{i,b}(\boldsymbol{a}_i) + \sum_{r=b+1}^{\varrho_i} \{c_{i,r}(\boldsymbol{a}_i) - c_{i,r-1}(\boldsymbol{a}_i \ominus t_k)\} \tag{6}$$

where $\boldsymbol{a}_i \ominus t_k$ symbolizes removal of task t_k from the task list $\boldsymbol{a}_i$ of vehicle v_i, and b denotes the position of task t_k in the task list, i.e., $\boldsymbol{a}_{i,b} = t_k$. The summation term represents comparison of the time cost with the task t_k included in the task list (first term) and the time cost without it (second term). It is a summation since this is calculated for all the tasks following t_k in the task list. An RPI list $\boldsymbol{\gamma}_p = [w_1, \ldots, w_m]^{\mathrm{T}} p = 1, \ldots, n$ is thus compiled for each vehicle. To facilitate consensus, a vehicle list $\boldsymbol{\beta}_p = [\beta_1, \ldots, \beta_m]^{\mathrm{T}} p = 1, \ldots, n$ is also composed for each vehicle. This list records the local view of which vehicle is assigned to which task. Fig. 1 illustrates the layout of an example problem where vehicle v_6 is originally assigned tasks 8, 11, 9, and 10. The time taken to travel each path is shown. If v_6 starts at task 8's location, and each task's duration is 350 s, then the RPI of removing task 11 from the task list of v_6 is calculated as follows:

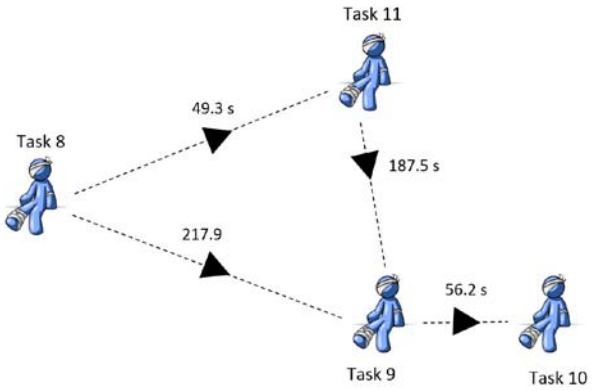

Fig. 1. Example problem showing the distances between tasks.

$$\begin{aligned}
c_{6,2}(\boldsymbol{a}_6) &= 350 + 49.3 = 399.3 \\
c_{6,3}(\boldsymbol{a}_6) &= 399.3 + 350 + 187.5 = 936.8 \\
c_{6,2}(\boldsymbol{a}_6 \ominus t_{11}) &= 350 + 217.9 = 567.9 \\
c_{6,3}(\boldsymbol{a}_6) - c_{6,2}(\boldsymbol{a}_6 \ominus t_{11}) &= 936.8 - 567.9 = 368.9 \\
c_{6,4}(\boldsymbol{a}_6) &= 936.8 + 350 + 56.2 = 1343 \\
c_{6,3}(\boldsymbol{a}_6 \ominus t_{11}) &= 567.9 + 350 + 56.2 = 974.1 \\
c_{6,4}(\boldsymbol{a}_6) - c_{6,3}(\boldsymbol{a}_6 \ominus t_{11}) &= 1343 - 974.1 = 368.9 \\
w_{11}(\boldsymbol{a}_6 \ominus t_{11}) &= 399.3 + 368.9 + 368.9 = 1137.1.
\end{aligned}$$

When a task is removed from a vehicle's task list it must be added to the task list of another. Thus, it is necessary to define IPI to measure the task's contribution to the local cost generated by the new vehicle. The IPI $w_k^*(\boldsymbol{a}_j \oplus t_k)$ of task t_k to vehicle v_j is the cost of performing the additional task plus the difference in cost (with and without the added task) of performing all subsequent tasks. It is defined as

$$w_k^*(\boldsymbol{a}_j \oplus t_k) = \min_{l=1}^{\varrho_j} \{w_{k,j,l}^{\Delta}\} \tag{7}$$

$$w_{k,j,l}^{\Delta} = \left\{ c_{j,l}(\boldsymbol{a}_j \oplus_l t_k) + \sum_{r=l}^{\varrho_j} \{c_{j,r+1}(\boldsymbol{a}_j \oplus_l t_k) - c_{j,r}(\boldsymbol{a}_j)\} \right\} \tag{8}$$

where $\boldsymbol{a}_j \oplus_l t_k$ symbolizes adding task t_k into the task list $\boldsymbol{a}_j$ of vehicle v_j, at the lth position. The value of $w_{k,j,l}^{\Delta}$ in (8) is calculated for each possible value of l and w_k^* is taken as the minimum of these. Again, the summation term represents comparison of the time cost with the task t_k now included in the task list (first term) and the time cost

without it (second term). This is calculated for all the tasks including and following position l in the task list. An IPI list $\boldsymbol{\gamma}_p^* = [w_1^*, \ldots, w_m^*]^{\mathrm{T}} p = 1, \ldots, n$ is thus compiled for each vehicle. Note that in the implementation of PI an infinity value is used for w_k^* when a task is already included in a vehicle's task list.

Intuitively, the RPI and $w_{k,j,l}^{\Delta}$ in (8) have the same value when a task is removed from a vehicle's task list and then is added back into the same task list in the same position, i.e., when $i = j$ and $b = l$. In the example, if task 11 is first removed from the task list of v_6 as before and then added back into it in position 2, $w_{k,j,l}^{\Delta}$ is calculated as follows:

$$\begin{aligned}
c_{6,2}(a_6 \oplus_2 t_{11}) &= 350 + 49.3 = 399.3 \\
c_{6,3}(a_6 \oplus_2 t_{11}) &= 399.3 + 350 + 187.5 = 936.8 \\
c_{6,2}(a_6) &= 350 + 217.9 = 567.9 \\
c_{6,3}(a_6 \oplus_2 t_{11}) - c_{6,2}(a_6) &= 936.8 - 567.9 = 368.9 \\
c_{6,4}(a_6 \oplus_2 t_{11}) &= 936.8 + 350 + 56.2 = 1343 \\
c_{6,3}(a_6) &= 567.9 + 350 + 56.2 = 974.1 \\
c_{6,4}(a_6 \oplus_2 t_{11}) - c_{6,3}(a_6) &= 1343 - 974.1 = 368.9 \\
w_{11,6,2}^{\Delta} &= 399.3 + 368.9 + 368.9 = 1137.1.
\end{aligned}$$

The actual value of $w_{11}^*(\boldsymbol{a}_6 \oplus t_{k11})$ is calculated by also evaluating $w_{11,6,l}^{\Delta}$ for $l = 1, 3, 4$ and taking the minimum value, i.e., the value obtained when adding the task at the position that yields the least additional cost.

When removing a task t_k from the task list $\boldsymbol{a}_i$ of v_i, it is obvious that there is benefit in adding it to the task list $\boldsymbol{a}_j$ of vehicle v_j if

$$w_k(\boldsymbol{a}_i \ominus t_k) > w_k^*(\boldsymbol{a}_j \oplus t_k)$$

as this will decrease the overall cost by the difference between the two values. The algorithm's full structure, which is written in MATLAB code, is now described.

At the start of the PI algorithm, the locations of the tasks and vehicles and the maximum start times are randomly generated, and the network topology is defined (see Section V-C). Also, the vehicle RPI lists and IPI lists are initialized to an m-sized vector holding the maximum MATLAB real number. The task lists, time cost lists, and vehicle lists are initialized to an m-sized vector of -1, -1, and 0 values, respectively (see Algorithm 3).

The consensus phase is a twofold process and is necessary as some tasks may be locally assigned to more than one vehicle. First, the vehicles exchange RPI lists, vehicle lists, and time stamps with all other vehicles in their range. The RPI and vehicle lists are then recomputed according to a consensus procedure first introduced in [13], which stipulates conditions for updating (adopting another vehicle's lists), leaving (keeping the same lists), and resetting. These rules are based on comparing RPI values and determining which vehicle has the most up-to-date information. For example, if vehicle j is the sender and vehicle i is the receiver, and both vehicles claim task k then if $w_{jk} < w_{ik}$ the receiver's action is to update so that $w_{ik} = w_{jk}$ and $\beta_{ik} = \beta_{jk}$. If the sender claims task k but the receiver credits it to a different vehicle p then, if either the time stamp for the information exchange between vehicles j and p is more recent than that between vehicles i and p, or if $w_{jk} < w_{ik}$, then the receiver's action is the same. This is the initial part of the consensus phase, which serves to resolve some of the conflict.

The task removal phase (see Algorithm 1) of the algorithm acts as the second part of the consensus phase. A lower RPI implies a more optimal assignment, so an agent with a higher RPI for a conflicting task should release it. Tasks are marked as candidates for removal from a vehicle's task list if there is disagreement between the vehicle list (computed in the initial part of the consensus phase) and the current task list, i.e., if a task is recorded on the task list, but that task is assigned to a different vehicle on the vehicle list. The RPI list $\boldsymbol{\gamma}_i^{\diamond} = [w_1, \ldots, w_m]^{\mathrm{T}}$ is then calculated according to (6) and iteratively compared with the previous RPI list $\boldsymbol{\gamma}_i = [w_1, \ldots, w_m]^T$ (that emerged from the initial part of the consensus phase), for all candidate tasks $\boldsymbol{d}_i$, i.e., the following is computed:

$$z = \max_{k=1}^{|d_i|} \{\gamma_{i,k}^{\diamond} - \gamma_{i,k}\}. \tag{9}$$

Algorithm 1: PI Task Removal

```
Compute candidate tasks for removal
while candidate list is not empty
  for each task in the task list
    if vehicle and task are compatible
      Set previous (and next) task and time cost
      Compute w_k from (6)
    end if
  next
  Compute z from (9)
  if z ≥ 0
    Remove task yielding max z from task list
    Remove task yielding max z from candidate list
    Re-calculate time cost list
  end if
end while
Put unremoved tasks back into vehicle list
```

If $z \geq 0$, then the task yielding the maximum z is removed from both the task list and the candidate list, and the time cost is then recalculated. In addition, $\boldsymbol{\gamma}_i^{\diamond}$ is computed again from (6) as its value changes following the removal of the task. Equation (9) is reevaluated, and the process repeats until $\boldsymbol{d}_i = \emptyset$. Any unremoved tasks are assigned to v_i in the vehicle list, i.e., $\boldsymbol{\beta}_i(\boldsymbol{d}_i) = v_i$ and the RPI list $\boldsymbol{\gamma}_i$ is set as the final $\boldsymbol{\gamma}_i^{\diamond}$.

The next phase is the task inclusion phase (see Algorithm 2), which is the part of the algorithm that builds the task bundles. The IPI list $\boldsymbol{\gamma}_i^* = [w_1^*, \ldots, w_m^*]^{\mathrm{T}}$ is computed according to (7) and (8), and compared with the RPI list $\boldsymbol{\gamma}_i = [w_1, \ldots, w_m]^{\mathrm{T}}$ computed in the task removal phase, i.e., the following is calculated:

$$q = \max_{k=1}^{m} \{\gamma_{i,k} - \gamma_{i,k}^*\}. \tag{10}$$

If $q > 0$, then the task t_ζ yielding the maximum q is added to the task list of v_i at the position l that returns the minimum w_ζ^*, and the time cost is recalculated. The RPI of

Algorithm 2: PI Task Inclusion

```
while tasks in task list not at upper limit
  for each task in problem
    if vehicle and task are compatible
      if task not already in task list
        for each insertion position
          Set previous task and time cost
          Compute w_k^* from (7) and (8)
        next
      end if
    end if
  next
  Compute q from (10) and l from (7) and (8)
  if q > 0
    Add task yielding max q to task list at position l
    Update vehicle list
    Set w_k = w_k^*
    Recalculate time cost list
  end if
end while
Recalculate γ_i
```

Algorithm 3: PI Main Program

```
Define World, Vehicles, Tasks, Network Topology
for each vehicle i
  Initialize a_i, c_i, w_i^*, w_i, β_i
next
while not converged
  Communicate w_i and β_i between vehicles
  Re-compute w_i and β_i according to CBBA rules
  for each vehicle i
    Carry out task removal
    Carry out task inclusion
  next
  if converged
    Mark as converged
  end if
end while
```

the task for v_i becomes the IPI of the task for v_i (so that the difference is zero), and the vehicle list $\boldsymbol{\beta}_i$ is adjusted accordingly. The IPI is recalculated, (10) is reevaluated, and the process repeats until there are no more tasks in the task list. Finally, the RPI $\boldsymbol{\gamma}_i = [w_1, \ldots, w_m]^{\mathrm{T}}$ is recalculated at the end of the phase. The communication exchange, initial consensus, task removal, and task inclusion phases continue iteratively until suitable stopping criteria are met (see Algorithm 3). PI is a heuristic algorithm that uses an iterative optimization principle; that is, each agent aims to decrease the overall cost at each iteration by recursively adding or removing tasks. The algorithm thus converges when no changes are made in the RPI values in both the task inclusion and removal phases (see [14]). However, the convergence rate slows down when the algorithm gets close to the solution; this helps to determine when to stop and accept the current solution without spending more time with little return.

It is important to note that optimization is on the average individual task cost for all the tasks, not average completion time for each vehicle. This is to take into account how many tasks benefit from the time saving. If there is a scenario where four tasks are completed earlier by swapping tasks, then this is preferable to a scenario where, for example, only two tasks are completed earlier. This is why the RPI takes account of the difference in time cost (with and without the removed task) for all subsequent tasks (and similar for the IPI calculation).

IV. Reassignment

A. Reassignment Problem

Reassigning the tasks when new information becomes available is not just a case of rerunning the algorithm, because some tasks may have already been serviced, or may be in the process of being serviced; thus some tasks need to be excluded from the reassignment. Additionally, vehicles are already acting on the schedules that they were previously given and so, in calculating their new position, one needs to know which task they were originally heading toward and how far they traveled along that path. As it stands, the baseline PI algorithm lacks the functionality to deal with these additional constraints. A rescheduling version of the algorithm is thus constructed to handle this functionality.

For the rescheduling version of the PI algorithm a time ψ is defined when new information first becomes available. This information may be in the form of updated task locations, new tasks added to or removed from the mission, new vehicles joining the fleet or deleted vehicles that have been recalled to base or to another mission. Any new vehicles and tasks will also have an estimated location associated with them. A set $\boldsymbol{K} = [\kappa_1, \ldots, \kappa_q]^{\mathbf{T}}$ of changed tasks is specified where q is the number of tasks that have updated locations or have been added or deleted. Each changed task κ_j has a set of associated x, y, and z coordinates. These are either the updated coordinate estimates for existing tasks or the given coordinates for newly added tasks. A set of tuples $\boldsymbol{\Theta} = [\theta_1, \ldots, \theta_q]^{\mathbf{T}}$ where $\theta_j = \langle k_j, x_j, y_j, z_j \rangle$ $j = 1, \ldots q$ represents this information. The status of the task is recognizable by the values in its tuple. Arbitrarily large numbers ϖ_1 and ϖ_2 can be used to represent a new task's nonexistent index and the nonexistent x coordinate of a deleted task, respectively. Thus, if $\kappa_j = \varpi_1$ and $\|x_j\| < \varpi_2$, then the task is a new task. If $\kappa_j \leq m$ and $x_j = \varpi_2$, then the task is a deleted task. If $\kappa_j \leq m$ and $\|x_j\| \leq x_{\max}$ where $x_{\max}$ is the maximum dimension of the world in the x direction, then the task is an existing one with an updated location. Similarly, a set $\boldsymbol{Z} = [\zeta_1, \ldots, \zeta_r]^{\mathbf{T}}$ of r deleted vehicles is stipulated, along with p, the number of new vehicles added to the fleet. The problem is to create an initial, conflict-free task-to-vehicle assignment before the new information becomes available, minimizing (1) and then to reassign appropriate tasks once the new data are received, taking account of the new intelligence and the updated locations of all the vehicles. The new assignment must minimize (1) as before, and must not conflict with the implemented tasks in the original assignment,

i.e., tasks that have already been completed or tasks that are in the process of being completed cannot be reassigned to a different vehicle unless they belong to the set $\boldsymbol{K}$. In addition, no task can be reassigned to vehicles in the set $\boldsymbol{Z}$. Note that if a vehicle is recalled to base while in the process of servicing a task, then it is assumed that it completes that task before returning to base. If a vehicle arrives at a task location and no survivor is present (because incorrect information on its location was initially supplied), then it is assumed that it moves onto the next task in its list until instructed otherwise via communication of a new task list.

B. Reassignment Methodology

The arrival of new information at time ψ is simulated by supplying values for ψ, q, p, and r and providing sets $\boldsymbol{K}$, $\boldsymbol{\Theta}$, and $\boldsymbol{Z}$ at the start of the main program. First, the algorithm executes as before ignoring this information, and generates an original assignment $\boldsymbol{A}_o$. Following this, the new vehicles, new tasks, and updated tasks are initialized and $\boldsymbol{V}$ and $\boldsymbol{T}$ can be updated. The new vehicles are assigned their type (helicopter or UAV) alternately, their locations are set randomly (using the same method as the existing vehicles), their availability is set to ψ, and the communication network is updated to include them. The new tasks are assigned their type (medicine or food) alternately, and their start times, along with the start times of updated tasks, are set to ψ. The new and updated locations of the tasks are assigned, and the old locations of the updated tasks are stored. A maximum (latest) start time s is also randomly generated for the new tasks.

The main program then generates a set of protected tasks $\boldsymbol{P}$ that cannot be reassigned. Tasks are protected if, under the original assignment, they have already been serviced by a vehicle (or have started to be serviced) at time ψ and there is no new information about their location. Algorithm 4 illustrates the task protection subroutine. For each task k in a vehicle's task list, it compares the minimum start time ω_k to ψ. If this is the first task in the list or the previous task is protected, then $\omega_k = c_{i,k}(\boldsymbol{a}_i)$, but if tasks have been completed or the previous task is unprotected, then the minimum start time also depends upon the task-type duration d_k and how many tasks have been completed. The protection element $P_{i,k}$ is set to 1 for protected tasks and 0 for unprotected tasks.

The original assignment $\boldsymbol{A}_o$ has an associated set of time costs $\boldsymbol{C}_o$. Before reassignment can take place the task lists and time cost lists must be updated to include the new vehicles and tasks. Thus, the following changes are made to $\boldsymbol{A}_o$ and $\boldsymbol{C}_o$.

1) New tasks are initially allocated to a vehicle that matches their type and this is done sequentially.
2) A task k is also swapped with its predecessor $k-1$ if k is protected and $k-1$ is unprotected and has not been removed, as this indicates that task $k-1$ has a changed location. The servicing vehicle would have arrived at the scene and would then have to proceed to task k without spending any time on task $k-1$.
3) Deleted tasks and their corresponding time costs are removed from $\boldsymbol{A}_o$ and $\boldsymbol{C}_o$.

Algorithm 4: Task Protection

```
Set P_{i,k} = 0
for all vehicles i
  for all tasks k in a_i
    if k > 1 AND P_{i,k-1} = 0
      ω_k = c_{i,k}(a_i) − (k − 1) d_k
    else
      ω_k = c_{i,k}(a_i)
    end
    if ω_k < ψ
      P_{i,k} = 1
    else
      P_{i,k} = 0
    end
    for all tasks j ∈ K
      P_{i,k} = 0
    end
  end
end
```

4) All unprotected tasks are removed from the task lists of deleted vehicles.
5) The time costs are reevaluated. If the start time of a task exceeds the maximum permitted, then the task and time cost are removed from the corresponding lists; this indicates a failure of the original solution to solve the reassignment problem.

The updated assignment and time cost sets are designated $\boldsymbol{A}_u$ and $\boldsymbol{C}_u$, respectively, and, provided the solution is viable for the reassignment problem, φ can be calculated from $\boldsymbol{C}_u$ generating φ_u. Additionally, $\varphi_{u,1}$ and $\varphi_{u,2}$ are determined as the mean task times for task types 1 and 2, respectively. The assignment $\boldsymbol{A}_u$ represents the solution of the reassignment problem using the original allocation. It is thus possible to use φ_u, $\varphi_{u,1}$, and $\varphi_{u,2}$ to determine the benefit of reassigning tasks after ψ rather than proceeding with the original solution.

The consensus and task allocation phases of the PI algorithm are run again with $\boldsymbol{A}_u$ and $\boldsymbol{C}_u$ as the starting points for the schedule. However, for this iteration, protected tasks cannot be candidates for removal during the task removal phase. In addition, during task inclusion, the initial insertion point in the task list is after the last protected task and tasks can only be inserted if they are not already in the list, are not protected, and have not been removed.

If a vehicle has not started out on a path toward a false task t_f at time ψ (either a missing task or the task it was originally assigned to that has now changed), then the time costs needed for the calculation of the RPI and IPI can be computed in the same way as for the initial schedule. However, if a vehicle is on its way to a false task at time ψ, then a path finder routine is called to determine the position of the vehicle and hence the time cost of servicing the next true task, which depends on distances between vehicles and tasks and the value of ψ. A vehicle may have undertaken many false paths, but there are only one or two key false tasks that matter for these calculations, since the location of the last

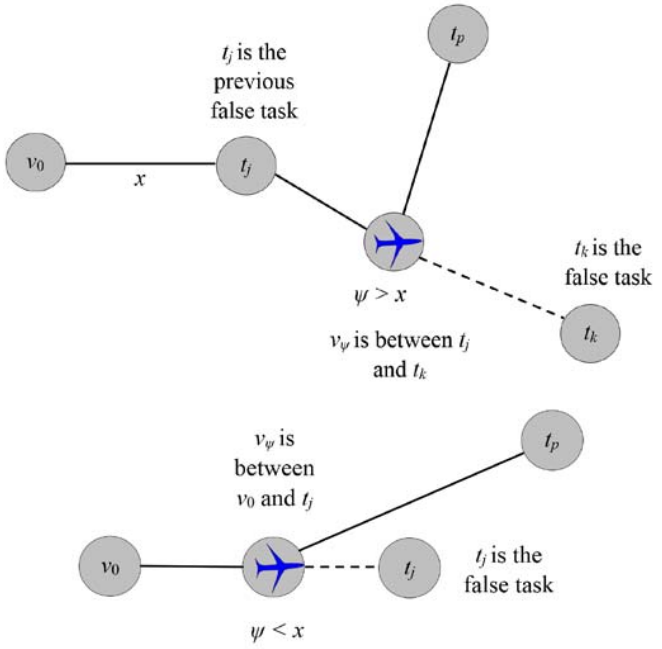

Fig. 2. Illustrating possible positions of key false tasks.

visited task (false or otherwise) is always known and the time that the vehicle deviates from its original path is always ψ. For example, assume a vehicle i is initially assigned tasks j and k and the location of task j has been estimated wrongly. If the time cost in getting from the vehicle's original position to the initial (false) location of task j is x, then vehicle i starts out on a false path toward task j. It either moves completely to the location of the false task j (if $\psi = x$), partially toward it (if $\psi < x$), or travels beyond it toward task k (if $\psi > x$) (see Fig. 2).

If i travels beyond task j, then task k is also a false task for vehicle i, even if task k's location is correct. This is because the new first-assigned task for i (following rescheduling) could be a different task completely, for example task p. The vehicle cannot travel from its initial location directly to task p. It must travel to task j and then head toward task k as per its original instructions before it can travel to task p. In this case, $t_f = k$ and task j is denoted as the previous false task $t_{\bar{f}}$. In all cases, the number of key false tasks $\Upsilon \in [0, 1, 2]$.

Even if there is no changed information about any of the tasks in a vehicle's task list, there may still be a false task because any vehicle's task list is subject to change following rescheduling and it must remain on its original path until the mission time has reached ψ. In such cases, if a task k is first in vehicle i's original task list and is unprotected then it is vehicle i's false task as protection only takes place when a task has been serviced at time ψ. If the task remains un-serviced and there is no changed information about the task, then the original servicing vehicle must be on its way to that task at time ψ. If task k is not first in vehicle i's original task list, then it is vehicle i's false task as long as $c_{i,k-1}(\boldsymbol{a}_i) + d_{k-1} \leq \psi$ where d_{k-1} represents the duration of the task preceding task k.

The path finder routine calculates the time cost $c(t_*)$ of task t_* based on the location $\Omega(v_\psi)$ of its assigned vehicle v at time ψ, which depends upon the false path or paths taken, the original locations and updated locations of tasks, the initial location $\Omega(v_0)$ of v, the velocity $\vartheta(v)$ of v, and the position l of t_* in the new task list. Task t_* is the task being added to the task list or the task immediately following the removed task. The following general relations are defined:

TABLE I
TIME COST CALCULATION DECISIONS

Υ	l	a	Γ
2	-	$\Omega(t_{\bar{f}})$	$\psi - c(t_{\bar{f}})$
1	> 1	$\Omega(t_{*-1})$	$\psi - (c(t_{*-1}) + d(t_{*-1}))$
1	1	$\Omega(v_0)$	ψ

$$\chi(t_f) = \frac{\odot\{a, \Omega(t_f)\}}{\vartheta(v)} \tag{11}$$

$$\Omega_x(v_\psi) = a_x + \frac{\Gamma}{\chi(t_f)}(\Omega(t_f)_x - a_x) \tag{12a}$$

$$\Omega_y(v_\psi) = a_y + \frac{\Gamma}{\chi(t_f)}(\Omega(t_f)_y - a_y) \tag{12b}$$

$$\Omega_z(v_\psi) = a_z + \frac{\Gamma}{\chi(t_f)}(\Omega(t_f)_z - a_z) \tag{12c}$$

$$\chi(t_*) = \frac{\odot\{\Omega(v_\psi), \Omega(t_*)\}}{\vartheta(v)} \tag{13}$$

$$c(t_*) = \psi + \chi(t_*). \tag{14}$$

In (11) and (12), the operational notation $\odot\{a, b\}$ is used to denote the distance between points a and b. Throughout (11) to (14), $\chi(t_f)$ represents the time taken to reach the false task, $\Omega(t_f)$ represents the location of the false task, $\chi(t_*)$ represents the time taken to travel from the position of the vehicle at ψ to the current task, and $\Omega(t_*)$ represents the location of the current task (which has been updated if $t_* \in \boldsymbol{K}$). Equation (11) is a simple distance-velocity relation used to define the theoretical time taken to reach the false task. Equation (12) defines the x, y, and z coordinates of the servicing vehicle at time ψ, and (13) is a simple distance–velocity relation used to define the time taken to reach the current task from $\Omega(v_\psi)$.

In (11), a represents the location of the servicing vehicle if the current task is the first in the task list ($l = 1$) and there is no previous false task ($\Upsilon = 1$). In this case, Γ in (12), which represents the time spent traveling on the false path, is simply ψ. If the current task is not the first in the list ($l > 1$) and there is one false task ($\Upsilon = 1$), then a in (11) represents the location of the previous task in the task list $\Omega(t_{*-1})$ and Γ in (12) is $\psi - (c(t_{*-1}) + d(t_{*-1}))$, where the last two terms represent the time cost and duration of the previous task, respectively. If there is a previous false task ($\Upsilon = 2$), then a in (7) represents the location of the previous false task $\Omega(t_{\bar{f}})$ and $\Gamma = \psi - c(t_{\bar{f}})$ in (12) where $c(t_{\bar{f}})$is the time cost of the previous false task. Table I summarizes the logic of the path finder routine for different combinations of Υ and l.

Apart from calls to the path finder routine (when there are false tasks in the path of the servicing vehicle), the algorithm proceeds in the same way as the original in terms of calculating

the RPI and IPI and converging to a solution. The final reassignment and associated time cost sets are designated $\boldsymbol{A}_r$ and $\boldsymbol{C}_r$, respectively, and the parameter φ is calculated from $\boldsymbol{C}_r$ for each task type generating $\varphi_{r,1}$ and $\varphi_{r,2}$, for the two task types 1 and 2, respectively. These are compared with $\varphi_{u,1}$ and $\varphi_{u,2}$ if $\boldsymbol{A}_u$ represents a viable solution. If $\varphi_{u,1} \leq \varphi_{r,1}$, then the algorithm reverts back to the original solution for type 1 vehicles, and if $\varphi_{u,2} \leq \varphi_{r,2}$, then the algorithm reverts back to the original solution for type 2 vehicles. Note that reversion is not possible if $\boldsymbol{A}_u$ is infeasible for the reassignment problem or if vehicles have been removed from the fleet. The final solution is represented by $\boldsymbol{A}_f$, and the associated time cost set $\boldsymbol{C}_f$ is used to calculate φ_f. If $\boldsymbol{A}_u$ represents a viable solution for the reassignment problem, then φ_f is compared to φ_u to assess the benefit of rescheduling. As long as

$$\varphi_f < \varphi_u \tag{15}$$

then it has been worthwhile building a new schedule in light of the new information.

C. Experimental Details

Different test conditions can be created by varying the numbers of tasks and vehicles and using different seed values to set the vehicle and task locations and the latest start times for the tasks. Two different world scenarios were used for testing the reassignment PI algorithm, Scenario A and Scenario B. In each scenario, the following experimental settings were adopted.

1) The world x and y coordinates ranged from -5000 to 5000 m in Scenario A and from -2500 to 2500 m in Scenario B. The z coordinates ranged from 0 to 1000 m in both scenarios. These settings provided sufficiently large and realistic rescue zones.
2) The helicopter and UAV velocities were arbitrarily set at 30 and 50 m/s, respectively.
3) All vehicles were available straight away at the start of the mission.
4) The mission time limit (the time window within which the mission must finish) ranged between 2000 and 3500 s in Scenario A and between 5000 and 6500 s in Scenario B.
5) The latest start time s was generated for each task using a random fraction of the overall mission time, with 1000 s as the lowest in Scenario A and 1500 s as the lowest in Scenario B.
6) The times to execute delivery of medicine and food were arbitrarily fixed at 300 and 350 s, respectively.

Note that the settings above were arbitrarily chosen and are not necessary for the algorithm to work; many other settings are possible. Forty different test problems (20 using Scenario A and 20 using Scenario B) were designed to validate the algorithm and measure the benefits of rescheduling rather than proceeding with the original plan. The test problems differed from each other in the seeds used to generate the 3-D worlds, the initial numbers of tasks and vehicles (limited to a maximum of 32 tasks, 16 vehicles in Scenario A, and 96 tasks, 16 vehicles in Scenario B), the task to vehicle ratio, the number of changed tasks, the identities of the changed tasks, the locations of the changed tasks, the positions of the changed tasks in the original schedule, the number of deleted vehicles, the number of additional vehicles, and the identities of the deleted and additional vehicles. Also, the effects of introducing the changes at different stages in the mission were investigated. Each of the 40 different test problems was run using ψ values of 140, 260, 340, 470, 530, and 700 s, generating 240 test cases in total. In Scenario A, task to vehicle ratios of 2, 4, 6, and 8 were used, and in Scenario B, the ratios used were 6, 8, and 10.

For each test case, φ_f was calculated and compared to φ_u if $\boldsymbol{A}_u$ represented a feasible solution. In addition, the $\boldsymbol{A}_f$ and $\boldsymbol{C}_f$ sets for 70 of the cases were examined and checked by hand to verify that they represented feasible solutions that did not conflict with the protected elements of the original solution including the paths already taken by the vehicles. The 240 test problems were also run using baseline CBBA for comparison, although this version of CBBA is unable to carry out reassignment.

D. Results

When PI was used, all 240 test cases were solved in their original form. When changes were introduced to the problem, PI was able to configure a new solution using its reassignment module in 86% (206) of the test cases. In 151 of the test cases (63%), the original assignment could be used to solve the reassignment problem; however, 73% of these 151 cases showed an improvement in mean task time after rescheduling took place. The remaining 27% demonstrated the same mean task time as the original solution after rescheduling. In 89 of the test cases, the original assignment could not solve the reassignment problem, but in 62% of these 89 cases, the reassignment module was able to configure a new solution.

When baseline CBBA was used, only 163 test cases (68%) were solved in their original form. When changes were introduced to the problem, CBBA was unable to configure a new solution in any of the cases as it lacks a reassignment module.

Four particular PI test cases 1 to 4 (each derived using the same seed and each using 8 vehicles and 16 tasks) are now provided to illustrate the results further. In test case 1, the locations of tasks 5 and 10 are updated, and in test case 2, tasks 5 and 10 are removed and two new tasks are introduced. The locations of these new tasks are the same as the changed locations in test case 1 to test the equivalence of these two different problem types. In test case 3, the locations of tasks 5 and 10 are updated and two new vehicles are also added. Test case 4 is the same as test case 3 except that vehicle 7 is also removed from the fleet. In each test, the new information arrives at $\psi = 260$ s.

Table II shows the assignments and task times corresponding to $\boldsymbol{A}_o$ (the solution of the original problem), which is the same for each test. Tables III–VI show the assignments and task times corresponding to $\boldsymbol{A}_u$ (the solution of the reassignment problem using the original plan), and $\boldsymbol{A}_f$ (the rescheduled solution) for each test.

TABLE II

TASK LIST, TASK COST, AND OBJECTIVE FUNCTION FOR ORIGINAL PROBLEM

v_i	$A_o(C_o)$	
	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$
1	1 (100.56)	5 (520.32)
2	6 (79.08)	4 (646.25)
3	7 (242.41)	2 (767.87)
4	3 (97.96)	8 (571.81)
5	13 (170.50)	15 (575.26)
6	10 (44.12)	11 (510.41)
7	9 (29.22)	14 (471.35)
8	12 (14.69)	16 (418.74)
φ_o	**328.79**	

Subscript o denotes the solution to the original problem using the original schedule

TABLE III

TASK LIST, TASK COST, AND OBJECTIVE FUNCTION FOR TEST CASE 1

v_i	$A_u(C_u)$		$A_f(C_f)$	
	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$
1	1 (100.56)	**5 (472.16)**	1 (100.56)	**4 (510.23)**
2	6 (79.08)	4 (646.25)	6 (79.08)	**5 (498.18)**
3	7 (242.41)	2 (767.87)	7 (242.41)	2 (767.87)
4	3 (97.96)	8 (571.81)	3 (97.96)	8 (571.81)
5	13 (170.50)	15 (575.26)	13 (170.50)	15 (575.26)
6	**11 (160.41)**	**10 (553.68)**	**11 (160.41)**	**10 (553.68)**
7	9 (29.22)	14 (471.35)	9 (29.22)	14 (471.35)
8	12 (14.69)	16 (418.74)	12 (14.69)	16 (418.74)
φ	φ_u = **335.74**		φ_f = **328.87**	

Subscripts u and f denote the solution to the reassignment problem using the original schedule and reassigned schedule respectively. Changes are highlighted in bold.

TABLE IV

TASK LIST, TASK COST, AND OBJECTIVE FUNCTION FOR TEST CASE 2

v_i	$A_u(C_u)$			$A_f(C_f)$	
	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$	$a_{i,3}\,(c_{i,3})$	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$
1	1 (100.56)	**17 (472.16)**		1 (100.56)	**4 (510.23)**
2	6 (79.08)	4 (646.25)		6 (79.08)	**17 (498.18)**
3	7 (242.41)	2 (767.87)		7 (242.41)	2 (767.87)
4	3 (97.96)	8 (571.81)		3 (97.96)	8 (571.81)
5	13 (170.50)	15 (575.26)	**18 (987.91)**	13 (170.50)	15 (575.26)
6	**11 (160.41)**			**11 (160.41)**	**18 (553.68)**
7	9 (29.22)	14 (471.35)		9 (29.22)	14 (471.35)
8	12 (14.69)	16 (418.74)		12 (14.69)	16 (418.74)
φ	φ_u = **362.89**			φ_f = **328.87**	

See footnote for Table III

In test case 1 (Table III), the original plan can be used to solve the reassignment problem, but it takes less time for vehicle 1 to reach task 5 as the location has changed. However, when vehicle 6 arrives at the original location of task 10, there is no task there to service and so it proceeds to the next task on its list, task 11. It receives the new location of task 10 while servicing task 11, and thus travels to task 10 upon completion of task 11. The mean task time increases to 335.47 s because of the changes. If rescheduling is used, the solution is almost the same except that tasks 4 and 5 are swapped so that vehicle 1 services task 4 and vehicle 2 services task 5 at less combined cost than using the original plan. Thus, (15) holds and time is saved by rescheduling.

In test case 2 (Table IV), tasks 5 and 10 are removed from the task list and tasks 17 and 18 are introduced. Task 17 is a type-1 task and is given to vehicle 1 by default; task 18 is a type-2 task and is thus assigned to vehicle 5. Although the problem is effectively the same as in test case 1 (since the positions of the new vehicles are the same as the updated positions in test case 1), the default vehicle assignment and task list position of the new type-2 task mean that φ_u has increased to 362.89 s. However, following rescheduling the same final solution is generated and (15) holds as before. Incidentally, it is not always true that the same solution will prevail when the updated tasks in one problem have the same new locations as substituted tasks in another. This is because the updated tasks already have a vehicle assignment and a task list position before rescheduling in the first problem, but are arbitrarily assigned in the second.

TABLE V

TASK LIST, TASK COST, AND OBJECTIVE FUNCTION FOR TEST CASE 3

v_i	$A_u(C_u)$		$A_f(C_f)$	
	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$
1	1 (100.56)	**5 (472.16)**	1 (100.56)	**4 (510.23)**
2	6 (79.08)	4 (646.25)	6 (79.08)	**5 (498.18)**
3	7 (242.41)	2 (767.87)	7 (242.41)	
4	3 (97.96)	8 (571.81)	3 (97.96)	8 (571.81)
5	13 (170.50)	15 (575.26)	13 (170.50)	
6	**11 (160.41)**	**10 (553.68)**	**11 (160.41)**	**10 (553.68)**
7	9 (29.22)	14 (471.35)	9 (29.22)	**16 (495.61)**
8	12 (14.69)	16 (418.74)	12 (14.69)	**15 (486.15)**
9			**2 (320.06)**	
10			**14 (300.36)**	
φ	φ_u = **335.74**		φ_f = **289.43**	

See footnote for Table III

TABLE VI

TASK LIST, TASK COST, AND OBJECTIVE FUNCTION FOR TEST CASE 4

v_i	$A_f(C_f)$	
	$a_{i,1}\,(c_{i,1})$	$a_{i,2}\,(c_{i,2})$
1	1 (100.56)	**4 (510.23)**
2	6 (79.08)	**5 (498.18)**
3	7 (242.41)	
4	3 (97.96)	8 (571.81)
5	13 (170.50)	**16 (607.40)**
6	**11 (160.41)**	**15 (544.70)**
7	9 (29.22)	
8	12 (14.69)	**10 (440.16)**
9	**2 (320.06)**	
10	**14 (300.36)**	
φ	φ_f = **292.98**	

Subscript f denotes the solution to the reassignment problem using the reassigned schedule. Changes are highlighted in bold

In test case 3, the mean task time and solution are the same as test case 1 when the original plan is used to solve the reassignment problem (see Table V). However, two new vehicles are introduced into the fleet, vehicle 9 (type 1), and vehicle 10 (type 2). When the problem is rescheduled, task 2, originally assigned to vehicle 3, is reassigned to vehicle 9, and task 14, originally assigned to vehicle 7, is reassigned to vehicle 10. In addition, tasks 15 and 16 are also reassigned to different vehicles. The reassignment results in a much more efficient solution with $\varphi_f = 289.43$ s.

In test case 4 (Table VI), the original plan cannot be used for the reassignment problem as vehicle 7 is recalled to base after 260 s. Although it is able to finish servicing task 9, it is unable to travel to its second task, task 14. The reassignment

problem can only be solved by rescheduling. The rescheduled solution is almost the same as in test case 3 except that tasks 10, 15, and 16 are now serviced by different vehicles. The reassignment results in a solution with $\varphi_f = 292.98$ s.

In timing trials for 8 vehicles and 16 tasks, the baseline PI algorithm and the rescheduling version solved 10 problems with no rescheduling demands in about 1 s each, although baseline PI was about 0.2 s faster in each case. When a single rescheduling demand was introduced, the run time increased to about 2 s for each trial. The rescheduling version of PI has thus proved both effective for reassignment and efficient in its computation time. It represents a contribution to the literature because distributed rescheduling algorithms that solve problems of this type are very scarce, and executable code for them is not generally available.

V. Soft-Max Action Selection

A. Soft-Max Action-Selection Mechanism

In the baseline PI algorithm, task allocation is governed only by comparing the calculated RPI and IPI values using (9) and (10). This approach can restrict the solution search space to local minima meaning that there is a need to provide an additional mechanism that permits further exploration. In the Boltzmann soft-max action-selection method [36], selection is based on a fitness score f for the various options. If there are m items, the fitness for item k is f_k and the fitness for each item $j = 1, \ldots, m$ is f_j, then the probability p_k of selecting item k is given by

$$p_k = \frac{e^{\frac{f_k}{\tau}}}{\sum_{j=1}^{m} e^{\frac{f_j}{\tau}}}. \tag{16}$$

By varying the parameter τ, it is possible to alter the selection strategy from picking a random item (τ infinite) to assigning higher probabilities for higher fitness (τ small and finite), or choosing only the item with the best fitness (τ tending to 0).

B. Integrating Soft-Max Selection Into the PI Algorithm

In the proposed approach, a Boltzmann soft-max action-selection routine is integrated into the PI algorithm and a loop is constructed around the main program. Within the loop different values of τ are trialed (in appropriate steps) until the best solution [i.e., that yielding the minimum φ value from (1)] is obtained.

The RPI and IPI are calculated for each task as in the baseline PI algorithm. For task removal, the arrays $\boldsymbol{\lambda}$, $\boldsymbol{\xi}$ (fitness) and $\boldsymbol{\sigma}$ [related to the top term in (16)] are then determined from

$$\boldsymbol{\lambda} = \boldsymbol{\gamma}^{\diamond}[\boldsymbol{d}] - \boldsymbol{\gamma}[\boldsymbol{d}] \tag{17}$$

$$\mu = \min\{\boldsymbol{\lambda}\} \tag{18}$$

$$\mu^* = |\mu| \forall \mu < 0 \tag{19}$$

$$\mu^* = 0 \quad \forall \mu \geq 0 \tag{20}$$

$$\boldsymbol{\xi} = \boldsymbol{\lambda} + \mu^* \tag{21}$$

$$\boldsymbol{\sigma} = e^{\frac{\boldsymbol{\xi}}{\tau}}. \tag{22}$$

For task inclusion, $\boldsymbol{\lambda}$ is calculated from

$$\boldsymbol{\lambda} = \boldsymbol{\gamma} - \boldsymbol{\gamma}^*. \tag{23}$$

Calculation of μ (an adjustment factor to remove negative values) is slightly more complex for task inclusion, as some of the members of the $\boldsymbol{\lambda}$ array may have values equal to MATLAB's largest possible value R from initialization. Thus, $\boldsymbol{\lambda}$ is first adjusted so that any such members have their values scaled by a factor R. If $\boldsymbol{\lambda}^*$ represents the adjusted $\boldsymbol{\lambda}$ array, then

$$\mu = \min\{\boldsymbol{\lambda}^*\} \tag{24}$$

and μ^* is given by (19) and (20) as before. The fitness is defined as

$$\boldsymbol{\xi} = \boldsymbol{\lambda}^* + \mu^* \tag{25}$$

and $\boldsymbol{\sigma}$ is given by (22) as before. For both task removal and task inclusion, the probability p_k of task k being selected is given by

$$p_k = \frac{\sigma_k}{\sum_{j=1}^{m} \sigma_j} \tag{26}$$

from (16). To facilitate this, a random number ρ is generated for each iteration of the task removal and task inclusion phases, and this number determines which task is selected for removal or inclusion according to (26). By varying τ in (22), it is possible to control the reliance of the strategy upon probability. Note that the value of z is still calculated from (9) for task removal, even if a different task is selected (i.e., if the task yielding the maximum value is not selected). For task inclusion q is not calculated from (10); instead, it is taken as λ_j where j represents the selected task. The position of insertion in the task list l is still taken as that yielding the minimum w_k^* from (7).

In theory, parameter reselection could be carried out at any time to cope with dynamic changes in the environment. Although this would impose an additional computational burden during run time, minimization of impact is possible by limiting the search to a region close to the original optimal parameter, assessing overall mission time, and imposing suitable stopping criteria.

C. Experimental Methodology

Two sets of experiments were conducted. In the first set (A), the world coordinates for Scenario A (in the rescheduling tests) were adopted and the task-to-vehicle ratio was maintained at 2:1. The mission completion time was restricted to 2000 s and the latest start time ranged randomly between 0 and 2000 s. Thus, in Scenario A, some of the problems were quite difficult to solve as the mission deadline is tight and it was possible to generate a number of tasks that needed attending to very early on in the mission. The number of tasks and vehicles was varied with a maximum of 32 survivors. This is in keeping with the work presented in [14] and [37], which was concerned with testing PI and CBBA on problems with tight time constraints, and allows easy comparison with this work. However, in more realistic scenarios the vehicle services

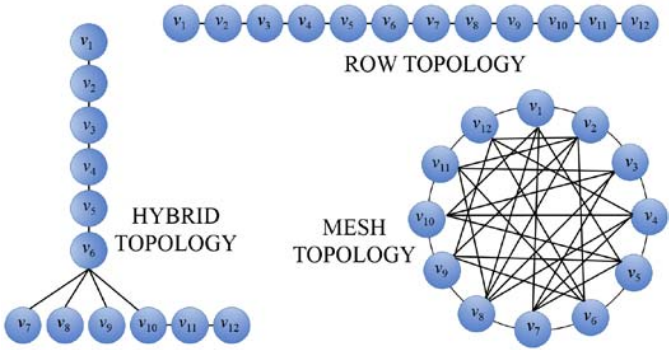

Fig. 3. Illustration of the different network topologies used in the experiments.

would be spread more thinly and the task deadlines would not be so tight. Thus, in the second set (B), the world coordinates from Scenario B were adopted and the number of tasks was maintained at 96 with variation of the number of vehicles so that more realistic task-to-vehicle ratios were tested. In addition, the mission completion time was restricted to 5000 s and the latest start time ranged randomly between 1500 and 5000 s. The smaller rescue zone and the more relaxed time constraints contributed to a more realistic scenario.

In experiment set A, nine seeds were used to create different 3-D cases and 20, 24, 28, and 32 tasks were trialed with $m = 2n$. In experiment set B, five seeds were used and 96 tasks were trialed with n values of 10, 12, 14, and 16. Each problem was solved using CBBA, the baseline PI algorithm, and the soft-max extension. If the problem was solved, i.e., each task was completed on time, then φ was calculated and recorded. If some tasks were not completed, then the number of failed tasks was recorded instead. Additionally, each problem in set A was solved using a row, mesh, and hybrid (row–tree) network topology (see Fig. 3, which illustrates the arrangement of the different network topologies for a 12 vehicle system).

In the row topology, the vehicles are connected in a line and communicate with the next and previous vehicle except that the first vehicle communicates only with the second and the last communicates only with the previous. The mesh topology has a circular communication arrangement where half of the other possible communication pairs are also randomly connected. The hybrid topology is a combination of a row and tree network; the vehicles begin in a row topology but the middle vehicle is connected to the next $\lfloor n/4 \rfloor + 1$ vehicles in a tree-like structure. The remaining vehicles continue in a row topology.

In experiment set A, τ values of between 1 and 50 were trialed in steps of 1. However, to avoid delays the stopping value was changed to $\tau = 20$ for 16 vehicles so that the run time never exceeded 3 min in the MATLAB implementations used here. In experiment set B, τ values between 10 and 30 were used throughout.

CBBA parameter values of 0.001 were used for λ and also for F (the vehicle fuel penalty) to match the scale of the worlds [38]. The CBBA score function was set at $\mathcal{H}e^{-\lambda t} - Fd$, where $\mathcal{H}$ is the reward associated with a task ($\mathcal{H} = 100$ for all tasks) and d is the distance between vehicle and task. All runs were executed in MATLAB R2013a on the same 64-b machine running Windows 7 Enterprise Edition, and using a 2.50-GHz Intel Core i5-2400S processor.

TABLE VII
SUMMARY OF ROW COMMUNICATION FOR SET A

Statistic	CBBA	PI	Soft max
% solved	6.25	90.63	100.00
% best solutions	0.00	9.38	93.75
Θ	0	N/A	3
Mean σ	-	-	2.83
Max σ	-	-	8.43

σ represents the % improvement in solution fitness when soft max selection is used. Θ represents the number of additional problems solved.

TABLE VIII
SUMMARY OF MESH COMMUNICATION FOR SET A

Statistic	CBBA	PI	Soft max
% solved	6.25	90.63	100.00
% best solutions	0.00	18.75	81.25
Θ	0	N/A	3
Mean σ	-	-	1.97
Max σ	-	-	6.29

See footnote for Table VII

TABLE IX
SUMMARY OF HYBRID COMMUNICATION FOR SET A

Statistic	CBBA	PI	Soft max
% solved	6.25	87.50	100.00
% best solutions	0.00	12.50	87.50
Θ	0	N/A	4
Mean σ	-	-	3.00
Max σ	-	-	8.84

See footnote for Table VII

Note that tests are carried out in simulation only as trials using PI under uncertain conditions have shown that a robust version of PI is necessary if the algorithm is to perform well in a real environment. Future work will thus aim to integrate the rescheduling and soft-max modules with a robust version of PI that has been developed [39] so that they can be tested in the real world.

D. Experimental Results

1) Experiment Set A: Table VII summarizes metadata for experiment set A using row communication, and Tables VIII and IX repeat these statistics for mesh and hybrid communication. In these tables, σ is the percentage improvement of the soft-max variant when compared to the solution generated by the baseline, and Θ is the number of additional problems that each algorithm could solve (i.e., the number unsolvable by the baseline but solvable by the algorithm in question). In calculating the percentage of problems solved, the number of problems was taken as 32, as 4 problems could not be solved by any method in any of the tests. The data are also depicted as a bar chart in Fig. 4 for ease of comparison. Note that if the sum of the percentages of best solutions is greater than 100% it is because two algorithms generated the same best solution.

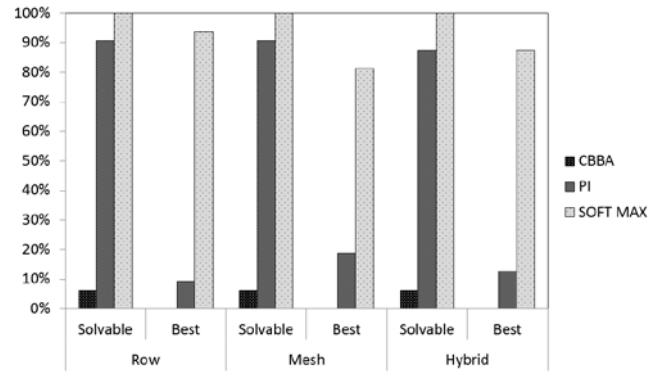

Fig. 4. Percentage of problems solved and percentage of best-solved problems in experiment set A.

TABLE X
ALGORITHM RUN TIMES FOR CASE 4 ROW COMMUNICATION (S) EXPERIMENT SET A

n	PI	PI*15	Soft max
4	0.1	1.7	1.0
6	0.2	3.5	3.0
8	0.6	9.1	9.2
10	0.9	13.4	18.8
12	1.5	22.5	29.8
14	3.2	48.5	54.9
16	F	-	83.5

Table X shows the time taken in seconds for 15 iterations of the soft-max PI variant when case 4 is solved with row communication. This is the time taken when the maximum τ is set to 15 with increments of size 1. These times are compared with the corresponding run times of the baseline, which only executes a single iteration. For comparison purposes, this value is also multiplied by 15 in the next column. If an "F" is shown, this indicates that the algorithm failed to solve the problem.

CBBA demonstrated a high failure rate; out of the 32 solvable problems trialed it was only able to solve two. In addition, it did not provide the best solution in any case and was not able to solve any additional problems. Its poor performance was consistent for all of the network topologies. These problems were chosen because the tight deadlines make them difficult to solve, and it seems that CBBA was simply unable to cope with these problems. It performed better in the experiments in Section IV because the constraints were somewhat relaxed.

The soft-max PI variant consistently outperformed the baseline in terms of the percentage of problems solved and the percentage of best solutions. In the row topology, baseline PI was able to solve about 91% of the problems, compared to 100% for the soft-max variant. These results were the same for mesh and hybrid communication, except that for hybrid communication baseline PI could only solve about 88% of the problems. This topology is weaker than the others as there is a heavy communication dependence upon the middle vehicle that is not present in the others. In the row communication experiments, the baseline PI algorithm produced the best solution in only about 9% of the problems compared to 94% for the soft-max variant. When mesh communication was trialed, these figures changed to about 19% and 81%, respectively. The mesh topology represents the most connected network, and thus the baseline algorithm was finding more optimal solutions than it would have under a sparser network. However, the soft-max variant still performed much better. When using the hybrid topology baseline PI solved about 13% of the problems best compared to about 88% for the soft-max variant.

TABLE XI
ALGORITHM RUN TIMES FOR CASE 4 ROW COMMUNICATION (S)

n	PI	Soft max	Soft max B
4	0.1	1.0	1.0
6	0.2	3.0	2.9
8	0.6	9.2	3.8
10	0.9	18.8	18.7
12	1.5	29.8	14.7
14	3.2	54.9	51.2
16	F	83.5	80.1

Up to about 8% improvement in φ was observed for row communication. This maximum decreased to about 6% for mesh communication and increased to about 9% for hybrid communication. The mean improvement σ was about 3% in the row and hybrid topologies, but dropped to about 2% for mesh communication.

The value of τ that produced the best solution varied somewhat. For example, for 14 vehicles, the best τ was 6 in one of the cases but was 47 in another. Given the wide variety in best τ values, it is not possible to narrow the scope of the search to save run time while still guaranteeing the best solution; if the scope of the search is reduced then the opportunity to find the best solution may be missed. Thus, there is a compromise between obtaining the best possible solution and minimizing run time. Further row-communication results for smaller numbers of vehicles are available in [37].

Table X shows that baseline PI runs almost instantaneously; for example, it takes only about a second when there are 10 vehicles, and the run time increases steadily as the number of vehicles rises. The run time of the soft-max variant is comparable with column 3 of the table, which multiples baseline PI's run time by 15. It suggests that running the soft-max variant is approximately equivalent to running the baseline the same number of times, i.e., softmax takes longer only because it executes a search; there are no additional complications in its architecture that slow it down. However, in order to reduce the run time a soft-max B variant was built. This algorithm terminates the search when a solution is reached that is a fixed percentage ε better than the best generated thus far, although it must execute at least δ searches. Table XI shows the results for 15 iterations of case 4 (row network) and the soft-max B variant with ε set at 2% and δ set at 5. The run times for PI and the original softmax are repeated here for comparison.

Table XI shows that imposing the additional stopping criteria provides benefit for some problems, but not all. There are benefits in stopping early for 8 and 12 vehicles. However, reduced execution times are only of value as long as the algorithm continues to demonstrate better solution quality than baseline PI.

TABLE XII
SOFT-MAX B VERSION RESULTS

n	PI	Soft max	τ	Soft max B	τ
10	312.05	298.74	5	298.74	5
12	268.95	267.08	9	267.08	9
14	264.79	257.13	17	257.13	17
16	1/0	278.25	10	278.25	10

TABLE XIII
SUMMARY OF ROW COMMUNICATION FOR SET B

Statistic	CBBA	PI	Soft max B
% solved	20.00	95.00	100.00
% best solutions	0.00	35.00	70.00
Θ	0	N/A	1
Mean σ	-	-	0.20
Max σ	-	-	0.83

See footnote for Table VII

Table XII shows the results of running the B version of the algorithm with case 4, row communication and the same number of maximum iterations as the earlier experiments. The B version produced the same results as the original soft-max variant, both in terms of the objective functions and the τ values that generated them, improving on the baseline PI result in each case. These results suggest that the B variant is the best option for these problem types; it is able to improve performance while maintaining a comparatively reasonable run time.

2) Experiment Set B: As the results for the different network topologies in experiment set A proved to be very similar, only the row topology was tested for experiment set B. The total number of experiments was 20, and each was repeated with CBBA, baseline PI, and the soft-max B variant of PI. The B variant was selected because the greater number of tasks in these experiments would have prevented the original soft-max variant from running efficiently. In addition, to try to reduce run time, the ε parameter was reduced to 0.2% and δ was set to 1. Table XIII and Fig. 5 summarize the metadata for this set of experiments, and Table XIV shows the run times in seconds for the five experiments using 16 vehicles. The best τ value for soft-max B is shown in column 4, and column 5 shows whether soft-max B terminated early (before $\tau = 30$ was reached).

This experiment set represents a more complex and more realistic collection of problems. CBBA did not solve any of the problems best and was, in fact, only able to solve four out of the 20 problems, performing slightly better than when tested on experiment set A. Although there were many more tasks to be completed in set B, leading to longer run times, the latest-start-time constraints were more relaxed than in set A, making the tasks slightly less prone to failure, although the constraints were tighter than in the experiments in Section IV, where CBBA performed much better. This supports the theory that it is the tight constraints inherent in these problems that makes them unsuited to CBBA. The soft-max variant outperformed baseline PI again in terms of both the number of problems it could solve and the percentage of best solutions, but the gap

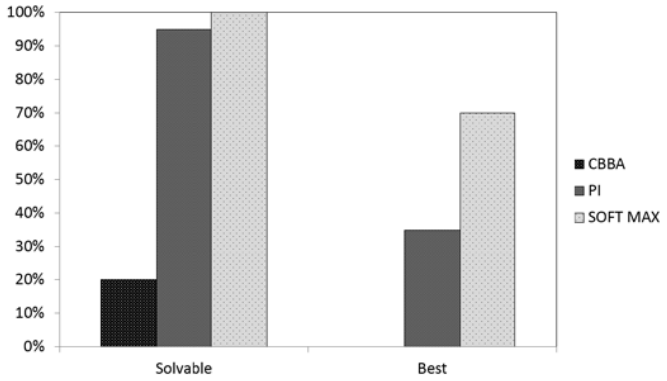

Fig. 5. Percentage of problems solved and percentage of best-solved problems in experiment set B.

TABLE XIV
ALGORITHM RUN TIMES IN SECONDS FOR 16 VEHICLES

Expt No.	PI	Soft max B	τ	Terminated Early
1	144	189	11	Yes
2	188	411	13	Yes
3	191	1295	25	Yes
4	132	355	12	Yes
5	196	279	12	Yes

in performance was not as large as in experiment set A. There was only one additional problem that soft-max B was able to solve, and the percentage of best solutions was 70% compared to 35% for the baseline.

The run time results in Table XIV show that soft-max B was able to complete in reasonable time compared to baseline PI in the cases where it was able to terminate quite early in the loop. However, when the best τ value was closer to the end value (experiment 3, with best $\tau = 25$), the run time was much longer (almost 7 times longer in this case). The benefits of employing soft-max B thus depend on the dimensionality and time scale of the problem and the stopping criteria imposed. This is discussed more fully in the following section.

E. Discussion

The tradeoff between solution quality and run-time efficiency is an important factor in both sets of experiments, and a key consideration is that the soft-max variant is not as efficient as the baseline in terms of execution time. This matters more for shorter mission times such as those considered in experiment set A, where the maximum mission time is 2000 s (about half an hour), and mean rescue times are at about 280 s (about four and a half minutes). In these problems, any saving in mean rescue time offered by the soft-max variant is counter-balanced by run-time losses. For example, if there is a 4% saving in mean rescue time, this equates to about 11 s for each task. If there are 28 tasks, the total saving is 308 s or about 5 min. However, if the search algorithm takes 5 min to run, then any gain is eliminated. For smaller time-scale problems, this means that it is important to terminate the algorithm execution early or limit the search space some other way as m increases, even if this means missing fitter solutions.

However, for more realistic larger-scale problems, for example, problems with a maximum mission time in terms of days and mean rescue start times in terms of hours, the algorithm's initial run-time efficiency is not as important and thus has much less impact on overall mean rescue time. In these cases, it would be possible to examine a wider range of the spectrum of possible τ values to be certain of finding the optimal or near-optimal solution without compromising the effectiveness or efficiency of the mission. In addition, a possible application is to reserve use of the soft-max module only for cases where baseline PI fails to solve.

These experiments have illustrated that PI is much more effective than CBBA in solving both lower dimensioned problems with tight constraints and higher dimensioned problems with more relaxed constraints. They have also demonstrated that PI's performance can be enhanced easily by making some adjustments to the action-selection mechanism within the task removal and task inclusion phases of the algorithm.

VI. Conclusion

This paper has confirmed that the distributed PI task-allocation algorithm [14] is easily enhanced to permit rescheduling when new information becomes available. The results of extensive testing of the rescheduling PI algorithm have validated its architecture and demonstrated benefits compared with proceeding with the original plan. The work represents an important extension to the baseline as real missions take place in dynamically changing environments where the SA is subject to alter rapidly. Task allocation strategies must therefore be able to adjust to new data and recompute new solutions in real time. The extended algorithm can handle new information concerning the locations of tasks, the addition of new tasks, the removal of tasks, and the addition and removal of vehicles. This paper represents a contribution to the literature because distributed rescheduling algorithms that solve problems of this type dynamically are very scarce, and executable code for them is not generally available.

The baseline PI algorithm has also been modified to solve the problem of solution trapping in local minima. The algorithm now includes a degree of soft-max action selection to introduce a level of exploration to its architecture. This variation allows new areas of the search space to be explored, generally improving solution fitness. In the experiments performed here, baseline PI's task allocation performance was increased by up to about 9%. In addition, the algorithm was able to solve some problems that failed using the baseline version. Although this enhancement increases run time quite substantially, especially for larger dimensions, some methods for reducing the extent of this limitation have been presented and have been reasonably successful.

In [14] and [37], and in this paper, baseline PI has been shown to outperform the popular state-of-the-art CBBA algorithm [13], especially for problems with tight time constraints. The introduction of rescheduling in PI widens the scope of its potential applications even further and makes it much more usable for real, time-critical task allocation problems. Furthermore, the additional action-selection mechanisms detailed in this paper enable improved performance at relatively low cost. This paper thus represents an advance in the state of the art in time-critical ST-SR-TA multiagent task planning. The integration of the rescheduling and the search exploration modules with the robust version of PI [39] to create a "Super-PI" is left as a subject for future work.

References

[1] B. P. Gerkey and M. J. M. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Intl. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, Sep. 2004.

[2] W. Shen, L. Wang, and Q. Hao, "Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 36, no. 4, pp. 563–577, Jul. 2006.

[3] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Eng. Appl. Artif. Intell.*, vol. 22, no. 7, pp. 979–991, Oct. 2009.

[4] J. L. Bruno, E. G. Coffman, and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Commun. ACM*, vol. 17, no. 7, pp. 382–387, 1974.

[5] T. P. Bagchi, *Multiobjective Scheduling by Genetic Algorithms*. Norwell, MA, USA: Kluwer, 1999.

[6] D. Bertsimas and R. Weismantel, *Optimization Over Integers*. Belmont, MA, USA: Dynamic Ideas, Jun. 2005.

[7] F. Glover and R. Marti, "Tabu search," in *Metaheuristic Procedures for Training Neural Networks*, E. Alba and R. Marti, Eds. New York, NY, USA: Springer, 2006, ch. 4, pp. 53–69.

[8] A. K. Khuntia, B. B. Choudhury, B. B. Biswal, and K. K. Dash, "A heuristics based multi-robot task allocation," in *Proc. IEEE Adv. Intell. Comput. Syst. (RAICS)*, Sep. 2011, pp. 407–410.

[9] T. Shima, S. J. Rasmussen, A. G. Sparks, and K. M. Passino, "Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms," *Comput. Oper. Res.*, vol. 33, no. 11, pp. 3252–3269, Nov. 2006.

[10] O. Catoni, "Solving scheduling problems by simulated annealing," *Siam J. Control Optim.*, vol. 36, no. 5, pp. 1539–1575, Sep. 1998.

[11] I. Sabuncuoglu and B. Gurgun, "A neural network model for scheduling problems," *Eur. J. Oper. Res.*, vol. 93, no. 2, pp. 288–299, 1996.

[12] V. D. Parunak, "Manufacturing experience with the contract net," in *Distributed Artificial Intelligence*, M. N. Huhns, Ed. New York, NY, USA: Pitman, 1987, pp. 285–310.

[13] 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.

[14] 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.

[15] C. Liu and A. Kroll, "Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks," *Soft Comput.*, vol. 19, no. 3, pp. 567–584, Apr. 2014.

[16] C. Liu and A. Kroll, "A centralized multi-robot task allocation for industrial plant inspection by using A* and genetic algorithms," in *Artificial Intelligence and Soft Computing* (Lecture Notes in Computer Science), vol. 7268, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds. Heidelberg, Germany: Springer, 2012.

[17] S. Forrest and M. Mitchell, "What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation," *Mach. Learn.*, vol. 13, no. 3, pp. 285–319, Nov. 1993.

[18] M. J. Shaw, "Dynamic scheduling in cellular manufacturing systems: A framework for networked decision making," *J. Manuf. Syst.*, vol. 7, no. 2, pp. 83–94, 1988.

[19] S. Nouyan, "Agent-based approach to dynamic task allocation," in *Proc. 3rd Int. Workshop Ant Algorithms*, 2002, pp. 28–39.

[20] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 876–888, Jul. 2015.

[21] P. B. Sujit and D. Ghose, "Self assessment-based decision making for multiagent cooperative search," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 4, pp. 705–719, Oct. 2011.
[22] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1465–1476, Sep. 2004.
[23] W. Ren, R. W. Beard, and D. B. Kingston, "Multi-agent Kalman consensus with relative uncertainty," in *Proc. Amer. Control Conf.*, Jun. 2006, pp. 1865–1870.
[24] K. Zhang, E. G. Collins, Jr., and D. Shi, "Centralized and distributed task allocation in multi-robot teams via a stochastic clustering auction," *ACM Trans. Autonom. Adapt. Syst.*, vol. 7, no. 2, Jul. 2012, Art. no. 21.
[25] M. B. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Lausanne, Switzerland, Sep./Oct. 2002, pp. 2714–2720.
[26] G. Oliver and J. Guerrero, "Auction and swarm multi-robot task allocation algorithms in real time scenarios," in *Multi-Robot Systems, Trends and Development*, T. Yasuda, Ed. Rijeka, Croatia: InTech, 2011, pp. 437–456.
[27] D. P. Bertsekas, "The auction algorithm for assignment and other network flow problems," Mass. Inst. Technol., Cambridge, MA, USA, Tech. Rep., 1989.
[28] A. Kalyanasundaram, R. A. K. Lalkhanwar, and S. Rao, "Fail-stop distributed combinatorial auctioning systems with fair resource allocation," in *Proc. IEEE Conf. Autom. Sci. Eng. (CASE)*, Trieste, Italy, Aug. 2011, pp. 181–188.
[29] S. Zaman and D. Grosu, "A combinatorial auction-based mechanism for dynamic VM provisioning and allocation in clouds," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 129–141, Jul./Dec. 2013.
[30] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt, "Simple auctions with performance guarantees for multi-robot task allocation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 1, Sep. 2004, pp. 698–705.
[31] E. H. Durfee and V. R. Lesser, "Using partial global plans to coordinate distributed problem solvers," in *Proc. IJCAI*, 1987, pp. 875–883.
[32] E. G. Talbi and T. Muntean, "Hill-climbing, simulated annealing and genetic algorithms: A comparative study and application to the mapping problem," in *Proc. HICSS*, Jan. 1993, pp. 565–573.
[33] P. Leitão and F. Restivo, "A holonic approach to dynamic manufacturing scheduling," *Robot. Comput.-Integr. Manuf.*, vol. 24, no. 5, pp. 625–634, 2008.
[34] W. Shen and D. H. Norrie, "An agent-based approach for dynamic manufacturing scheduling," in *Proc. Workshop Agent-Based Manuf.*, 1998, pp. 117–128.
[35] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1104–1113, Dec. 1980.
[36] R. S. Sutton and A. G. Barto, "Evaluative feedback," in *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998, p. 30.
[37] 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, Sep./Oct. 2015, pp. 6451–6488.
[38] L. B. Johnson, "private communication," Jul. 2014.
[39] A. Whitbrook, Q. Meng, and P. W. H. Chung, "A robust, distributed task allocation algorithm for time-critical, multi agent systems operating in uncertain environments," in *Proc. 30th Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst. (IEA/AIE)*, Arras, France, Jun. 2017.

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

She was a Research Fellow and a Teaching Associate at the University of Nottingham, a Senior Scientist in the Autonomous Systems Research Group, BAE Systems, Farnborough, U.K., and a Research Associate at 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 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.

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 Senior Lecturer 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, service and assistive robotics, robot learning and adaptation, multi-unmanned air vehicle cooperation, situation awareness and decision making for driverless vehicles, driver's distraction detection, human motion analysis and activity recognition, activity pattern detection, pattern recognition, artificial intelligence, and computer vision.

Paul W. H. Chung received the B.Sc. degree in computing science from Imperial College London, London, U.K., in 1981, and the Ph.D. degree in artificial intelligence from the University of Edinburgh, Edinburgh, U.K., in 1986.

From 1984 to 1991, he was with the Artificial Intelligence Applications Institute, University of Edinburgh. In 1991, he joined Loughborough University, Loughborough, U.K., where he has been a Professor of Computer Science since 1999, the Head of the Department of Computer Science from 2004 to 2008, and the Dean of the School of Science from 2011 to 2014. He was a Visiting Professor with the Beijing University of Posts and Telecommunications, Beijing, China, and the Institute Technology of Brunei, Gadong, Brunei. He was an Invitation Fellow with Okayama University, Okayama, Japan. He has successfully supervised 30 doctorate students and published over 200 papers. His current research interests include applying advanced computing techniques to solve novel complex problems.