This item was submitted to [Loughborough's Research Repository](#) by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

A novel distributed scheduling algorithm for time-critical multi-agent systems

PLEASE CITE THE PUBLISHED VERSION

<http://dx.doi.org/10.1109/IROS.2015.7354299>

PUBLISHER

IEEE

VERSION

AM (Accepted Manuscript)

PUBLISHER STATEMENT

This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at:
<https://creativecommons.org/licenses/by-nc-nd/4.0/>

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Whitbrook, Amanda, Qinggang Meng, and Paul Wai Hing Chung. 2019. "A Novel Distributed Scheduling Algorithm for Time-critical Multi-agent Systems". figshare. <https://hdl.handle.net/2134/18840>.

A Novel Distributed Scheduling Algorithm for Time-Critical Multi-Agent Systems

Amanda Whitbrook, Qinggang Meng, and Paul W. H. Chung

Abstract — This paper describes enhancements made to the distributed performance impact (PI) algorithm and presents the results of trials that show how the work advances the state-of-the-art in single-task, single-robot, time-extended, multi-agent task assignment for time-critical missions. The improvement boosts performance by integrating the architecture with additional action selection methods that increase the exploratory properties of the algorithm (either soft max or ϵ -greedy task selection). It is demonstrated empirically that the average time taken to perform rescue tasks can reduce by up to 8% and solution of some problems that baseline PI cannot handle is enabled. Comparison with the consensus-based bundle algorithm (CBBA) also shows that both the baseline PI algorithm and the enhanced versions are superior. All test problems center around a team of heterogeneous, autonomous vehicles conducting rescue missions in a 3-dimensional environment, where a number of different tasks must be carried out in order to rescue a known number of victims that is always more than the number of available vehicles.

I. INTRODUCTION

A. Motivation

Multi-agent task allocation problems in the Single-Task, Single-Robot, Time-Extended Assignment (ST-SR-TA) class are strongly NP-hard [4] as they represent complex, combinatorial decision problems, and, even when these problems are small, there is an exponential solution space [3]. These drawbacks mean that the Linear Programming (LP) approach, which guarantees optimality, is not suitable for problems of this type. Much research effort has thus been directed toward designing heuristic-based methods that can provide fitter solutions than those generated by greedy algorithms.

The contribution of this paper is the development and testing of extensions to the work carried out in [1] to enhance the performance of a distributed heuristic algorithm that uses the novel concept of performance impact (PI) to allocate time-critical tasks among a heterogeneous set of vehicles. The baseline PI algorithm is expanded to include an appropriate combination of PI task selection and either soft max or ϵ -greedy task selection. 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 is

carried out to show empirically that the enhancement can improve the performance of the baseline PI algorithm by up to 8%, and enables solution of some problems that the baseline cannot handle. Comparison with the state-of-the-art CBBA algorithm is also included and it is shown that the baseline PI is superior to it. The enhancements suggested here thus advance the state of the art in task assignment for multi-agent, time-critical systems even further. The increased effectiveness can be attributed to enabling escape from local minima by improving the exploration properties of the algorithm. However, the search for an optimal parameter introduces a trade-off between increasing solution time and boosting solution quality.

B. Related Work

The choice of either a centralized or distributed communication system is of paramount importance when designing task allocation algorithms, as robots continually need to share information about their current task set. Centralized approaches, for example [6] and [7], incur a high communication overhead for larger systems, and are vulnerable to single-point failure. In addition, the vehicles need to communicate with a central server in a fixed location, limiting the range of the mission. However, these systems are generally simpler to implement and tend to run faster as no consensus processing stage is required to ensure that the vehicles have identical situational awareness (SA) or identical solutions. Alternatively, distributed systems, where a planner is instantiated on each vehicle, require less communication bandwidth, allow extension of the mission 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 [8], meaning that some form of consensus algorithm is necessary [9]. 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 [2].

Many methods designed for the solution of ST-SR-TA problems involve iterative task allocation, for example market-based decision strategies [5], where each robot is modelled as a self-interested agent, and the whole fleet as an economy. The robots 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 [10], [11]), which are a subset of market-based methods, have also been applied to

A. M. Whitbrook, Qinggang Meng and Paul W. H. Chung are with the Department of Computer Science at Loughborough University, Loughborough, Leicestershire, LE11 3TU, United Kingdom (phone: +441509225913; e-mail: a.m.whitbrook@lboro.ac.uk, q.meng@lboro.ac.uk, p.w.h.chung@lboro.ac.uk).

ST-SR-TA problems. In these algorithms, each robot 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. A disadvantage with the method is that each agent needs to communicate directly with the auctioneer, limiting the choice of network topology that can be employed. To avoid this difficulty the auctions can be run within a set of direct neighbors only, although this can compromise mission performance. Auction-based methods are generally robust to inconsistencies in the SA, and have been shown to produce sub-optimal solutions efficiently [2]. For a full discussion of centralized and distributed auction methods see [12].

In [2] Choi et al. have shown that their distributed consensus-based bundle algorithm (CBBA, suitable for solving time-critical SR-ST-TA problems) 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 vehicles to bid on the bundles rather than individual tasks. Bundles are continuously updated as the auction proceeds. The authors show that the method produces the same solution as some centralized sequential greedy procedures, and 50% optimality is guaranteed. Task bundling auction methods are also described in [13] and [14].

In CBBA the bundles are formed by logically grouping similar tasks, as it would be too computationally costly to enumerate all possible bundles. The architecture in [1] uses a similar approach to CBBA, but considers a set of tasks to have a positive synergy for a vehicle if the combined cost of executing them together is less than the sum of the individual costs incurred by carrying them out separately (and vice versa for a negative synergy). The method uses a key novel concept called performance impact (PI) in order to exploit the synergies between tasks to increase optimality. This is a measure of the importance of a task to a vehicle's local cost. Global cost is decreased by satisfying certain criteria on the performance impact when switching tasks between vehicles. The distributed PI algorithm has been shown empirically to solve task allocation problems more effectively than the CBBA method. When solving a number of time-critical ST-SR-TA problems with different network topologies, different numbers of vehicles and tasks, and randomly generated locations for survivors and vehicles, the PI approach demonstrates a consistently lower average rescue time, and is able to solve many problems that the CBBA method cannot.

This paper develops the PI algorithm further and tests performance against both CBBA and baseline PI. It is arranged as follows; section II summarizes the problem domain and describes the baseline PI architecture. Section III details the additional action selection mechanisms and shows how they are integrated into baseline PI. Section IV provides the experimental methodology and presents a detailed result comparison for the algorithms using several different

scenarios. Section V concludes the paper and suggests possible extensions for the work.

II. PROBLEM DESCRIPTION AND PI ARCHITECTURE

A. Summary of the Problem

The problem of interest in this paper is formulated mathematically by defining a set of n heterogeneous rescue vehicles $\mathbf{V} = [v_1, \dots, v_n]^T$, a set of m tasks $\mathbf{T} = [t_1, \dots, t_m]^T$ $m > n$, and a set of ordered task allocations $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]^T$, where $\mathbf{a}_i$, $i = 1, \dots, n$, is the task list assigned to vehicle v_i . Note that the actual size of a task list α_i may vary between vehicles, for example vehicle v_1 may be assigned a single task, whereas vehicle v_2 may be assigned three tasks. However, mathematically $|\mathbf{a}_i| \equiv m$ as the remaining elements of each assignment vector are set to -1. A compatibility matrix $\mathbf{H}$ with entries $h_{i,j} \in [0,1]$ defines whether vehicle 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, a maximum start time $\mathbf{S} = [s_1, \dots, s_m]$ is randomly defined for each task in each test scenario. After this time has elapsed the task cannot commence, i.e. the problem has the additional complexity of being time-limited. Thus, the time cost for a particular task is defined as the time taken to arrive at the scene; the time taken to carry out the current task is fixed for each task type and is not included; only previous task times affect the current time cost. Each task requires only one vehicle, and each vehicle can complete only one task at a time, although it can complete other tasks afterwards, provided that there is enough time. The problem is to find a conflict-free assignment of vehicles to tasks that maximizes some global reward. It falls into the general category of Single Task–Single Robot (ST-SR) task allocation problems, as defined in [3], and since there are always more victims to find than vehicles available, the problem is also a Time-Extended Assignment (TA) type, i.e. it is an ST-SR-TA system under the same taxonomy.

In this particular case, the global objective function is to minimize ϕ the average single task time over all tasks, i.e.

$$\phi = \frac{1}{m} \sum_{i=1}^n \sum_{k=1}^{\alpha_i} c_{i,k}(\mathbf{a}_i), \quad (1)$$

where α_i is the number of tasks assigned to vehicle v_i , and $c_{i,k}(\mathbf{a}_i)$ is the time cost incurred by vehicle v_i servicing the k^{th} tasks in its task list.

The particular scenario used here is based on the rescue aspect of urban search-and-rescue (USAR). The vehicles are either Unmanned Air Vehicles (UAVs) supplying food or helicopters supplying medicine, and each vehicle must find its way to a victim that requires the supplies it is carrying. The start locations of the UAVs and helicopters are known in advance, as are the 3-dimensional locations and requirements of the victims. Different test scenarios can be created by using different seed values to set the vehicle and task locations randomly. The following problem details hold throughout this paper unless stated otherwise:

1. The number of tasks is always exactly double the number of vehicles, and the numbers of helicopters and

- UAVs are always equal.
- The world x and y coordinates range from -5000m to 5000m and the z coordinates range from 0m to 1000m.
 - The helicopters travel at 30m/s and the UAVs at 50m/s.
 - All vehicles are available straight away at the start of the mission.
 - The mission time limit (the time window within which the mission must finish) is set at 2000s and the earliest start time is always 0s for all tasks.
 - The maximum start time s is generated for each task using a random fraction of 2000s.
 - The times to execute delivery of medicine and food are fixed at 300s and 350s respectively.

Note that real USAR missions do not conform to the above assumptions; they are made merely to simplify the analysis and enable general conclusions to be drawn more easily.

B. The PI Architecture

PI is a distributed algorithm that runs independently on board each vehicle. It uses the basic framework of the distributed CBBA auction architecture [2], but introduces the novel concept of performance impact (PI) as a score function to determine the bundles and hence allocate the tasks. 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 performance impact, which are now explained. The removal performance impact (RPI) $w_k(\mathbf{a}_i \ominus t_k)$ 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 future tasks. It represents the contribution of a task to the local cost generated by a vehicle. RPI is defined as:

$$c_{i,b}(\mathbf{a}_i) + \sum_{r=b+1}^{\alpha_i} \{c_{i,r}(\mathbf{a}_i) - c_{i,r-1}(\mathbf{a}_i \ominus t_k)\}, \quad (2)$$

where $\mathbf{a}_i \ominus t_k$ symbolizes removal of task t_k from the task list $\mathbf{a}_i$ of vehicle v_i , and b denotes the position of task t_k in the task list, i.e. $a_{i,b} = t_k$. 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 $\boldsymbol{\gamma}_p = [w_1, \dots, w_m]^T$ $p = 1, \dots, n$ is thus compiled for each vehicle. To facilitate consensus, a vehicle list $\boldsymbol{\beta}_p = [\beta_1, \dots, \beta_m]^T$ $p = 1, \dots, n$ is also composed for each vehicle. This list records the local view of which vehicle is assigned to which task.

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 inclusion performance impact (IPI) to measure the task's contribution to the local cost generated by the new vehicle. The IPI $w_k^*(\mathbf{a}_j \oplus t_k)$ 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 future tasks. It is defined as:

$$w_k^*(\mathbf{a}_j \oplus t_k) = \min_{l=1}^{\alpha_j} \{w_{k,j,l}^\Delta\}, \quad (3)$$

$$w_{k,j,l}^\Delta = \left\{ c_{j,l}(\mathbf{a}_j \oplus t_k) + \sum_{r=l}^{\alpha_j} \{c_{j,r+1}(\mathbf{a}_j \oplus t_k) - c_{j,r}(\mathbf{a}_j)\} \right\}, \quad (4)$$

where $\mathbf{a}_j \oplus t_k$ symbolizes adding task t_k into the task list $\mathbf{a}_j$ of vehicle v_j , at the l^{th} position. The value of $w_{k,j,l}^\Delta$ in (4) 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 $\boldsymbol{\gamma}_p^* = [w_1^*, \dots, w_m^*]^T$ $p = 1, \dots, n$ 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 (4) have the same value when a task is removed from a vehicle's task list and is then added back into the same task list in the same position, i.e. when $i = j$ and $b = l$.

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

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

as this will decrease the overall cost by the difference between the two values. The novelty of the PI algorithm is the RPI and IPI concepts; its full structure, which is similar to the CBBA algorithm [2] and 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 (row, circular, mesh or hybrid) is defined. 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.

During the consensus phase the vehicles exchange RPI lists, vehicle lists and also time stamps with all other vehicles in their range. When all the lists have been received, the consensus takes place, i.e. the RPI and vehicle lists are re-computed according to an adaptation of the CBBA action rules specified in Table 1 of [2], 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.

After the consensus phase, the task removal phase of the algorithm begins. Tasks are marked as candidates for removal from a vehicle's task list if there is disagreement between the vehicle list computed in 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 $\gamma_i^\circ = [w_1, \dots, w_m]^T$ is then calculated according to (2) and iteratively compared with the previous RPI list $\gamma_i = [w_1, \dots, w_m]^T$ that emerged from the consensus phase, for all candidate tasks d_i , i.e. the following is computed:

$$z = \max_{k=1}^{|d_i|} \{\gamma_{i,k}^\circ - \gamma_{i,k}\}. \quad (5)$$

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 re-calculated. In addition, γ_i° is computed again from (2) as its value changes following the removal of the task. Equation (5) is re-evaluated, and the process repeats until $d_i = \emptyset$. Any unremoved tasks are assigned to v_i in the vehicle list, i.e. $\beta_i(d_i) = v_i$ and the RPI list γ_i is set as the final γ_i° .

The next phase is the task inclusion phase, where the IPI list $\gamma_i^* = [w_1^*, \dots, w_m^*]^T$ is computed according to (3) and (4), and compared with the RPI list $\gamma_i = [w_1, \dots, w_m]^T$ computed in the task removal phase, i.e. the following is calculated:

$$q = \max_{k=1}^m \{\gamma_{i,k} - \gamma_{i,k}^*\}. \quad (6)$$

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 re-calculated. The RPI of the task for v_i becomes the IPI of the task for v_i and the vehicle list β_i is adjusted accordingly. The IPI is recalculated, (6) is re-evaluated, and the process repeats until there are no more tasks in the task list. Finally, the RPI $\gamma_i = [w_1, \dots, w_m]^T$ is recalculated at the end of the phase. The communication exchange, consensus, task removal and task inclusion phases continue iteratively until suitable stopping criteria is met, for example, until no actions have been taken in the task removal and inclusion phases for more than two iterations. For further details of the PI algorithm see [1].

III. ACTION SELECTION MECHANISMS

A. Soft Max and ϵ -Greedy Action Selection

In the baseline PI algorithm task allocation is governed only by comparing the calculated RPI and IPI values using (5) and (6). This approach can restrict the solution search space to local minima. There is thus a need to provide an additional mechanism that permits further exploration of the solution space.

There are two variants of the modified PI algorithm; one that uses ϵ -greedy action selection, and one that uses Boltzmann soft max selection. In ϵ -greedy selection the best option is selected for a proportion $1 - \epsilon$ of trials, and a random option (with uniform probability) is selected for a proportion ϵ , ($0 \leq \epsilon \leq 1$). In the Boltzmann soft max method, selection is based on a fitness score f for the various options. If there are m items, and the fitness for item k is f_k , then the probability p_k of selecting item k is given by

$$p_k = \frac{e^{\frac{f_k}{\tau}}}{\sum_{j=1}^m e^{\frac{f_j}{\tau}}}. \quad (7)$$

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), to choosing only the item with the best fitness (τ tending to 0).

B. Integration with the PI Algorithm

In the proposed approach ϵ -greedy or Boltzmann soft max action selection routines are integrated into the PI algorithm and a loop is constructed around the main program. Within the loop different values of ϵ and τ respectively are trialed (in appropriate steps) until the best solution (i.e. that yielding the minimum φ value from (1)) is obtained. Pseudo code for the modified main program is presented in Algorithm 1.

Algorithm 1 Main Program

```

1: Initialize  $\hat{\varphi}$ ,  $\hat{\epsilon}$  ( $\epsilon$ -greedy) or  $\hat{\tau}$  (soft max) to large values, Set Seed
2: for  $\epsilon$  [ $\tau$ ] between start value and end value
3:   Reset Seed,
4:   Define World, Vehicles, Tasks, Network Topology
5:   for each vehicle  $i$ 
6:     Initialize  $a_i$ ,  $c_i$ ,  $w_i^*$ ,  $w_i$ ,  $\beta_i$ 
7:   next
8:   while not converged
9:     Communicate  $w_i$  and  $\beta_i$  between vehicles
10:    Re-compute  $w_i$  and  $\beta_i$  according to CBBA rules
11:    for each vehicle  $i$ 
12:      Carry out  $\epsilon$ -greedy [soft max] task removal
13:      Carry out  $\epsilon$ -greedy [soft max] task inclusion
14:    next
15:    Check convergence
16:    if converged
17:      Compute  $\varphi$  from (1)
18:      Results set =  $\mathcal{R}$ 
19:      Mark as converged
20:    end if
21:  end while
22:  if  $\varphi < \hat{\varphi}$  AND problem has not failed
23:     $\hat{\varphi} = \varphi$ ,  $\hat{\epsilon} = \epsilon$  [ $\hat{\tau} = \tau$ ],  $\hat{\mathcal{R}} = \mathcal{R}$ 
24:  end if
25: next
26: Show final  $\hat{\varphi}$ ,  $\hat{\epsilon}$  [ $\hat{\tau}$ ],  $\hat{\mathcal{R}}$ 
```

In the ϵ -greedy variant selection is modified when the RPI and IPI are calculated in the task removal and task inclusion phases respectively. In the task removal phase, after w_k has been calculated from (2) there is a probability ϵ , ($0 \leq \epsilon \leq 1$) of multiplication of w_k by random factor δ , ($1 \leq \delta \leq 1.5$). For example, if $\epsilon = 0.25$ there is a 25% probability that w_k is modified for each task in the task list and a 75% probability ($1 - \epsilon$) that w_k is unaffected. The modification means there

is a probability that the task yielding z in (5) will differ from the equivalent task in the baseline PI algorithm. Task inclusion works in a similar way. Pseudo codes for the ε -greedy task removal and task inclusion routines are presented in Algorithms 2 and 3 respectively. Note that the upper and lower limits for δ were determined from pre-trials. Use of a fixed δ represents a more common form of ε -greedy selection, but the pre-trials demonstrated advantages (in terms of solution quality) when using a variable δ with minimum value 1 and maximum 1.5.

Algorithm 2 ε - greedy Task Removal

```

1: Compute candidate tasks for removal
2: while candidate list is not empty
3:   for each task in the task list
4:     if vehicle and task are compatible
5:       Set previous (and next) task and time cost
6:       Compute  $w_k$  from (2)
7:       Set  $\rho, 0 \leq \rho \leq 1.0$  and  $\delta, 1 \leq \delta \leq 1.5$ 
8:       if  $\rho < \varepsilon$ 
9:         Set  $w_k = \delta w_k$ 
10:      end if
11:    end if
12:  next
13: Compute  $z$  from (5)
14: if  $z \geq 0$ 
15:   Remove task yielding max  $z$  from task list
16:   Remove task yielding max  $z$  from candidate list
17:   Re-calculate time cost list
18: end if
19: end while
20: Put unremoved tasks back into vehicle list

```

Algorithm 3 ε - greedy Task Inclusion

```

1: while tasks in task list not at upper limit
2:   for each task in problem
3:     if vehicle and task are compatible
4:       if task not already in task list
5:         for each insertion position
6:           Set previous task and time cost
7:           Compute  $w_k^*$  from (3) and (4)
8:           Set  $\rho, 0 \leq \rho \leq 1.0$  and  $\delta, 1 \leq \delta \leq 1.5$ 
9:           if  $\rho < \varepsilon$ 
10:            Set  $w_k^* = \delta w_k^*$ 
11:          end if
12:        next
13:      end if
14:    end if
15:  next
16: Compute  $q$  from (6) and  $l$  from (3) and (4)
17: if  $q > 0$ 
18:   Add task yielding max  $q$  to task list at position  $l$ 
19:   Update vehicle list
20:   Set  $w_k = w_k^*$ 
21:   Recalculate time cost list
22: end if
23: end while
24: Recalculate  $\gamma_i$ 

```

In the Boltzmann soft max version selection is modified in a different way. The RPI and IPI are calculated for each task as in the baseline PI algorithm. For task removal the arrays λ , ξ (fitness) and σ (related to the top term in (7)) are then determined from:

$$\lambda = \gamma^\circ[d] - \gamma[d], \quad (8)$$

$$\mu = \min\{\lambda\}, \quad (9)$$

$$\mu^* = |\mu| \forall \mu < 0, \quad (10)$$

$$\mu^* = 0 \forall \mu \geq 0, \quad (11)$$

$$\xi = \lambda + \mu^*, \quad (12)$$

$$\sigma = e^{\frac{\xi}{\tau}}. \quad (13)$$

For task inclusion λ is calculated from

$$\lambda = \gamma - \gamma^*. \quad (14)$$

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

$$\mu = \min\{\lambda^*\}, \quad (15)$$

and μ^* is given by (10) and (11) as before. The fitness is defined as

$$\xi = \lambda^* + \mu^*, \quad (16)$$

and σ is given by (13) as before. For both task removal and task inclusion, the probability p_k of task k being selected is given by

$$p_k = \frac{\sigma_k}{\sum_{j=1}^m \sigma_j} \quad (17)$$

from (7). 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 (17). By varying τ in (13) it is possible to control the reliance of the strategy upon probability. Pseudo codes for the Boltzmann soft max task removal and task inclusion routines are presented in Algorithms 4 and 5 respectively. Note that the value of z is still calculated from (5) 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 (6); 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 (3). Note that, 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.

Algorithm 4 Soft Max Task Removal

```
1: Compute candidate tasks for removal
2: while candidate list is not empty
3:   for each task in the task list
4:     if vehicle and task are compatible
5:       Set previous (and next) task and time cost
6:       Compute  $w_k$  from (2)
7:     end if
8:   next
9:   Set  $\lambda, \mu, \mu^*, \xi, \sigma$  from (8) to (13)
10:  Compute probabilities of task selection from (17)
11:  Generate  $\rho, 0 \leq \rho \leq 1.0$ 
12:  Select task for removal based on probabilities
13:  Compute  $z$  from (5)
14:  if  $z \geq 0$ 
15:    Remove selected task from task list
16:    Remove selected task from candidate list
17:    Re-calculate time cost list
18:  end if
19: end while
20: Put unremoved tasks back into vehicle list
```

Algorithm 5 Soft Max Task Inclusion

```
1: while tasks in task list not at upper limit
2:   for each task in problem
3:     if vehicle and task are compatible
4:       if task not already in task list
5:         for each insertion position
6:           Set previous task and time cost
7:           Compute  $w_k^*$  from (3) and (4)
8:         next
9:       end if
10:    end if
11:  next
12:  Set  $\lambda, \lambda^*, \mu, \mu^*$  from (14-15), (10-11)
13:  Set  $\xi, \sigma$  from (16) and (13)
14:  Compute probabilities of task selection from (17)
15:  Generate  $\rho, 0 \leq \rho \leq 1.0$ 
16:  Select task for inclusion based on probabilities
17:  Calculate  $\lambda_j$  for selected task  $j$ 
18:  Compute  $l$  from (3) and (4)
19:  if  $\lambda_j > 0$ 
20:    Add task  $j$  to task list at position  $l$ 
21:    Update vehicle list
22:    Set  $w_k = w_k^*$ 
23:    Recalculate time cost list
24:  end if
25: end while
26: Recalculate  $\gamma_i$ 
```

IV. EXPERIMENTS

A. Methodology

In the experiments nine seeds were used to create different 3-dimensional cases with 4, 6, 8, 10, 12, 14, and 16 vehicles, i.e. 63 different problems were tackled, each using a

row communication topology. The problems were solved using CBBA, the baseline PI algorithm, and the two proposed PI variants in this paper. 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 for each task type was recorded instead.

For ε -greedy selection, ε values of between 0.01 and 0.99 were trialed in steps of 0.01. However, pre-trials showed that the program execution time for all 99 ε values was becoming too large as n increased. To avoid delays the stopping values were changed to 0.35 and 0.15 for 10/12 and 14/16 vehicles respectively. For the same reason soft max selection τ values between 1 and 100 in steps of 1 were used for up to 8 vehicles, but the stopping value was adjusted to $\tau = 50$ for 10, 12 and 14 vehicles, and to $\tau = 20$ for 16 vehicles. These adjustments meant that run time never exceeded 7 minutes for ε -greedy selection (3 minutes for soft max selection, which proved to be a faster algorithm) in the MATLAB implementations used here.

In CBBA values of 0.001 were used for λ and also for F (the vehicle fuel penalty) to match the scale of the worlds generated. The CBBA score function used was:

$$\mathcal{H}e^{-\lambda t} - Fd,$$

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.

B. Results

Table I shows the calculated φ values for each algorithm and scenario and, in the case of the PI variants, the ε and τ values that produced the best solution are also displayed. The percentage improvement of the best algorithm (highlighted in bold) over baseline PI is also shown in the last column, where appropriate. If the best algorithm is able to solve when baseline PI cannot, then a '+' symbol is shown. If no algorithm can solve, then a '-' symbol is displayed.

Table II shows the time taken in seconds for 15 iterations of the ε -greedy and soft max PI variants when case 4 is solved. This is the time taken when the maximum ε is set to 0.15 and a step-size of 0.01 is used, and 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 PI algorithm, 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 in the table then the algorithm failed to solve the problem and, if a 'T' is shown, this indicates that the run was terminated because a 600 second limit was reached.

C. Discussion

CBBA demonstrated a high failure rate and out of the 63 problems trialed it was only able to solve 12. It offered the best solution in only 1 problem (for 4 vehicles, case 8), but in this case, the ε -greedy and soft max PI variants also produced the same solution.

The PI variants consistently out-performed the original in terms of the average rescue time; up to about 8%

improvement was observed. In addition, there were 3 and 4 problems respectively that the ϵ -greedy and soft max variants could solve that the baseline could not. They improved the average solution time (or were able to offer solutions when the baseline could not) for 92% of the problems; the remaining 8% of problems could not be solved by any algorithm.

TABLE I. ϕ VALUES

n	Case	CBBA	PI	ϵ -greedy	ϵ	Soft max	τ	% PI
4	1	1/0	1/0	1/0	-	1/0	-	-
	2	1/0	293.93	293.93	0.01	293.93	94	0.00
	3	0/1	309.30	302.99	0.43	309.30	2	2.04
	4	297.76	314.53	296.76	0.31	297.76	1	5.65
	5	0/1	263.83	261.57	0.43	263.83	1	0.86
	6	0/1	0/1	336.71	0.14	349.00	28	+
	7	338.03	342.75	332.12	0.19	337.78	41	3.10
	8	274.65	281.35	274.65	0.49	274.65	1	2.38
	9	313.20	312.72	298.65	0.05	298.65	1	4.50
6	1	1/1	320.83	298.65	0.91	294.42	28	8.23
	2	1/2	313.80	307.40	0.21	307.40	19	2.04
	3	0/1	289.50	288.84	0.61	289.50	2	0.23
	4	1/0	285.95	280.04	0.49	280.04	2	2.07
	5	0/1	270.88	255.66	0.08	256.82	1	5.62
	6	0/1	297.97	289.09	0.01	291.18	18	2.98
	7	307.61	303.01	291.45	0.31	291.45	5	3.81
	8	281.06	275.12	274.91	0.30	274.91	31	0.08
	9	270.23	288.95	269.10	0.44	270.23	2	6.87
8	1	3/0	297.76	294.85	0.76	297.77	2	0.98
	2	2/2	313.11	296.12	0.49	303.11	94	5.43
	3	1/2	316.31	292.05	0.91	292.23	7	7.67
	4	2/1	328.79	304.50	0.75	305.34	15	7.39
	5	274.96	270.33	257.95	0.77	266.53	63	4.58
	6	1/1	266.51	264.79	0.58	270.94	82	0.65
	7	0/1	288.89	278.54	0.97	283.99	10	3.58
	8	270.67	255.28	254.01	0.10	255.28	3	0.50
	9	275.92	281.69	265.38	0.88	267.33	2	5.79
10	1	3/2	294.15	284.44	0.32	289.12	6	4.00
	2	3/2	1/1	1/1	-	1/1	-	-
	3	2/1	296.75	282.23	0.26	281.24	3	5.23
	4	2/1	312.05	301.23	0.29	298.74	5	4.26
	5	1/1	1/0	274.27	0.33	288.94	4	+
	6	1/1	261.77	255.25	0.11	270.10	15	2.49
	7	0/1	276.20	268.32	0.16	271.44	2	2.86
	8	0/1	280.76	262.05	0.28	276.00	2	7.77
	9	273.67	272.68	263.15	0.27	263.48	10	3.92
12	1	2/2	287.77	276.28	0.33	278.19	3	3.99
	2	4/1	1/1	0/1	-	1/1	-	-
	3	2/0	303.77	278.44	0.16	283.27	4	8.34
	4	1/1	268.95	262.32	0.33	267.08	9	2.46
	5	2/1	274.55	262.40	0.18	262.27	19	4.47
	6	2/0	255.63	243.80	0.08	249.64	7	4.63
	7	0/1	272.94	264.13	0.32	261.54	9	4.17
	8	0/1	285.07	261.09	0.15	261.04	12	8.43
	9	1/0	271.86	250.68	0.35	256.28	12	7.79
14	1	4/1	278.68	271.30	0.14	277.71	15	2.65
	2	3/5	0/1	1/1	-	284.09	47	+
	3	3/0	281.63	275.13	0.11	279.52	42	4.33
	4	0/1	264.79	259.28	0.15	257.13	17	2.89
	5	1/1	269.45	262.10	0.15	260.55	31	3.31
	6	2/1	263.44	249.37	0.13	251.36	6	5.34
	7	2/1	272.09	261.76	0.08	259.26	12	4.72
	8	1/1	262.40	255.94	0.07	259.39	41	2.46
	9	1/0	245.47	241.52	0.15	245.47	14	1.61
16	1	2/3	0/2	0/2	-	0/2	-	-
	2	3/4	1/0	1/0	-	1/0	-	-
	3	3/4	274.48	267.32	0.06	266.85	20	2.78
	4	2/1	1/0	267.09	0.11	278.25	10	+
	5	2/1	265.43	263.93	0.03	259.56	10	2.21
	6	1/2	241.58	240.52	0.03	237.51	5	1.68
	7	2/0	238.49	241.87	0.06	237.02	12	0.62
	8	1/0	254.40	250.00	0.02	246.44	7	3.13
	9	251.23	239.81	237.23	0.11	240.01	13	1.08

In terms of average rescue time ϕ , the ϵ -greedy variant generally performed better than the soft max variant,

especially when vehicle and task numbers were low (less than 16 vehicles). However, for higher numbers of vehicles the search space had to be narrowed more for the ϵ -greedy variant to improve run time efficiency, which led to a decrease in its performance compared to the soft max variant. For example, for 8 vehicles the ϵ -greedy variant produced the best average rescue time in all cases. However, it is worth noting that, for this number of vehicles, the ϵ -greedy variant took much longer than the soft max variant to find the best solution, yet the soft max version was still able to outperform the baseline in most cases.

TABLE II. ALGORITHM RUN TIMES IN SECONDS

n	PI	PI*15	ϵ -greedy	Soft max
4	0.1	1.7	1.0	1.0
6	0.2	3.5	2.7	3.0
8	0.6	9.1	9.6	9.2
10	0.9	13.4	117.1	18.8
12	1.5	22.5	44.8	29.8
14	3.2	48.5	189.6	54.9
16	F	-	T	83.5

The values of ϵ and τ that produced the best solutions varied greatly. For example, for 6 vehicles, the best ϵ value was 0.01 in one case, but it was 0.91 in another. Similarly, for 8 vehicles the best τ was 2 in one of the cases but was 94 in another. Unfortunately, this means that it is not possible to narrow the scope of the search to save run time whilst 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. In these experiments as the number of vehicles increased, it was necessary to reduce the ranges of ϵ and τ values that were searched to preserve run-time efficiency. This meant that the best solutions were sometimes missed. For example, for 10 vehicles the maximum ϵ was set to 0.35, and this produced a best solution of 261.09 with $\epsilon = 0.15$ for case 8. However, if the maximum ϵ had remained at 0.99, then a better solution (258.95) would have been obtained when ϵ reached 0.89.

Table II shows that the baseline PI algorithm 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 multiplies 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. soft max takes longer only because it executes a search; there are no additional complications in its architecture that slow it down. This is not the case for the ϵ -greedy variant; for smaller numbers of vehicles (< 10) it behaves in a similar manner to the soft max variant, but begins to reveal much longer run times for higher numbers of vehicles (≥ 10). Moreover, these run times show erratic behavior, i.e. they do not rise steadily with the number of vehicles. For example, when 10 vehicles are used with case 4 the algorithm takes about 117s to run, but converges in about 45s when 12 vehicles are used. Further investigation has shown that the problem is caused by the number of inner iterations of the task allocation phase. For some problems this number can become very high for the ϵ -greedy variant because new probability parameters are

generated each time the RPI and IPI are calculated for each task in the task list. With the soft max variant, the RPI and IPI are calculated for each task in the list first before the probability parameters are created. Limiting the number of internal iterations in the ε -greedy variant should overcome this problem. Additionally, the time efficiency of both variants could be improved by terminating the search once the objective function has reached a given percentage of the best value computed so far.

An important consideration is that neither of the proposed algorithms is as efficient as the original PI algorithm in terms of actual run time. This matters for the problems considered here, where the maximum mission time is 2000s (about half an hour), and mean rescue times are at about 280s (about four and a half minutes). In these problems any savings in mean rescue time offered by the amended algorithms are counter-balanced by run-time losses. For example, if there is a 4% saving in mean rescue time, this equates to about 11s for each task. If there are 28 tasks the total saving is 308s or about 5 minutes. However, if the modified algorithm takes 5 minutes to run, then any gain is eliminated. For smaller time-scale problems this means that it is important to reduce the search space accordingly as n increases, even if this means missing fitter solutions.

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 as it runs in terms of minutes and seconds, 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 ε and τ values to be certain of finding the optimal or near optimal solution without compromising the effectiveness or efficiency of the mission. In addition, if a new parameter search is required because new information has become available then the computational burden can be reduced by using prior knowledge of the original optimal parameters, i.e. the search can be conducted within a much smaller range that centers on the previously selected parameter. This means that problem solution using modified PI is possible online, and can be applied to dynamically changing problems.

V. CONCLUSIONS AND FUTURE WORK

In this paper, the distributed Performance Impact (PI) task-allocation algorithm has been enhanced to include a degree of either ε -greedy or soft max action selection, thus introducing a level of exploration to the architecture, much like a genetic algorithm, which uses random cross-over and mutation to obtain more variation in the solution. These variations can permit new areas of the search space to be explored and hence can improve solution fitness. Indeed, in the experiments performed here the introduction of more exploration improved baseline PI's task allocation performance by up to about 8%, and enabled the algorithm to solve additional problems that failed using the baseline version. Baseline PI has been shown to outperform the popular state-of-the-art CBBA algorithm [2] in this paper and also in [1]. The additional action selection mechanisms detailed in this paper enable improved performance at

relatively low cost. This work thus represents an advance in the state-of-the-art in ST-SR-TA multi-agent task planning.

Future work will aim to improve the time efficiency of the ε -greedy and soft max PI variants. This will be achieved by limiting the number of inner iterations in the task allocation phase in the ε -greedy variant and stopping the search in both variants when the objective function has already improved by a given percentage of its best value. The tests carried out in this paper will also be repeated using different network topologies and the effects of introducing uncertainty into key parameters will also be explored. Finally, the baseline PI algorithm will be adapted to allow it to reschedule tasks as soon as new information becomes available.

This work was supported by EPSRC (grant number EP/J011525/1) with BAE Systems as the leading industrial partner.

REFERENCES

- [1] 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 Transactions on Cybernetics*, DOI: 10.1109/TCYB.2015.2418052, April 2015
- [2] H-L Choi, L. Brunet, and J. P. How, "Consensus-Based Decentralized Auctions for Robust Task Allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, Aug. 2009, pp. 912–926
- [3] B. P. Gerkey, and M. J. M. Mataric, "A Formal Analysis and Taxonomy of Task Allocation in Multi-robot Systems," *Intl. J. of Robotics Research*, vol. 23, no. 9, Sept. 2004, pp. 939–954
- [4] J. L. Bruno, E. G. Coffman, and R. Sethi "Scheduling Independent Tasks to Reduce Mean Finishing Time," *Communications of the ACM*, vol. 17, no. 7, 1974, pp. 382–387
- [5] M. B. Dias, and A. Stentz, "Opportunistic Optimization for Market-Based Multi-robot Control", in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland 2002, pp. 2714–2720.
- [6] C. Liu, and A. Kroll, "Memetic Algorithms for Optimal Task Allocation in Multi-robot Systems for Inspection Problems with Cooperative Tasks," *Soft Comput.*, April 2014
- [7] C. Liu, and A. Kroll, "A Centralized Multi-robot Task Allocation for Industrial Plant Inspection by Using A* and Genetic Algorithms", *L. Rutkowski et al. (Eds.): ICAISC 2012, Part II*, LNCS 7268, 2012, pp. 466–474
- [8] J. A. Fax, and R. M. Murray, "Information Flow and Cooperative Control of Vehicle Formations", *IEEE Trans. Autom. Control*, vol. 49, no. 9, Sept. 2004, pp. 1465–1476
- [9] W. Ren, R. Beard, and D. Kingston, "Multi-agent Kalman Consensus with Relative Uncertainty", in *Proc. Amer. Control Conf.*, 2006, pp. 1865–1870
- [10] G. Oliver, and J. Guerrero, "Auction and Swarm Multi-Robot Task Allocation Algorithms in Real Time Scenarios", *Toshiyuki Yasuda (Ed.): Multi-Robot Systems, Trends and Development, InTech*, ISBN: 978-953-307-425-2, 2011, pp. 437–456
- [11] D. P. Bertsekas, "The Auction Algorithm for Assignment and Other Network Flow Problems", Mass. Inst. Technol., Cambridge, MA, Tech. Rep., 1989
- [12] K. Zhang, E. G. Jr. Collins, 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, July 2012, Article 21
- [13] S. Zaman, and D. Grosu, "A Combinatorial Auction-based Mechanism for Dynamic VM Provisioning and Allocation in Clouds", *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, 2013, pp. 129–141
- [14] 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. of the 2004 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1, 2004, pp. 698–705