Heuristics for the Stochastic Dynamic Task-Resource Allocation Problem with Retry Opportunities

Nalan Gülpınar

Warwick Business School, The University of Warwick, Coventry, CV4 7AL, UK

Ethem Çanakoğlu

Bahçeşehir University, Industrial Engineering, Istanbul, Turkey

Juergen Branke

Warwick Business School, The University of Warwick, Coventry, CV4 7AL, UK

Abstract

This paper deals with a stochastic multi-period task-resource allocation problem. A team of agents with a set of resources is to be deployed on a multi-period mission with the goal to successfully complete as many tasks as possible. The success probability of an agent assigned to a task depends on the resources available to the agent. Unsuccessful tasks can be tried again at later periods. While the problem can in principle be solved by dynamic programming, in practice this is computationally prohibitive except for tiny problem sizes. To be able to tackle also larger problems, we propose a construction heuristic that assigns agents and resources to tasks sequentially, based on the estimated marginal utility. Based on this heuristic, we furthermore propose various Approximate Dynamic Programming approaches and an Evolutionary Algorithm. All suggested approaches are empirically compared on a number of randomly generated problem instances. We show that the construction heuristic is very fast and provides good results. For even better results, at the expense of longer computational time, Approximate Dynamic Programming seems a suitable alternative.

Keywords: Task-resource allocation, approximate dynamic programming, heuristics, retry opportunities.

1. Introduction

Resource management is one of the fundamental problems in operations research and involves dynamically assigning tasks and allocating discrete resources over time. The task assignment prob-

Email addresses: Nalan.Gulpinar@wbs.ac.uk (Nalan Gülpınar), ethem.canakoglu@bahsehir.edu.tr (Ethem Çanakoğlu), Juergen.Branke@wbs.ac.uk (Juergen Branke)

lem is about deciding which agent should perform which task and at what time, whereas the resource allocation problem determines the level and type of resources to be used for attempting each task. The resources are consumed for the accomplishment of the tasks and include for instance materials, energy, ammunition, and man hours.

The task assignment problem (e.g. Alighanbari and How (2008)) and resource allocation problem (e.g. Tharumarajah (2001), Ernst et al. (2006), Nwozo and Nkeki (2012), Angalakudati et al. (2014), and Zhen (2015)) have been extensively studied independently in the literature and have various applications in production planning, freight transportation, job scheduling and financial planning. In particular, dynamic multi-resource allocation models have been developed for infrastructure management (Pekka and Ahti, 2009), project scheduling (Wiesemann et al., 2012), service capacity management (Liu and Truong, 2013) and healthcare (Chao et al., 2003).

There are also real life applications in team management and multi-agent planning in military missions such as reconnaissance and surveillance of unmanned vehicles; for instance, see Gulpinar et al. (2010), Nygard et al. (2001), Chandler et al. (2002), Koenig et al. (2007), Tovey et al. (2005), Zheng and Koenig (2009), Samuel and Guikema (2012) and Hong and Gordon (2015). For these cases, task assignment and resource allocation decisions cannot be made in isolation for a successful mission planning and effective team deployment (de Weerdt and Clement, 2009). In a multi-agent environment, the coordination of tasks and resources results in real time problems that cannot be solved in polynomial time (Bellingham et al., 2003). As Haslum and Geffner (2014) pointed out, the optimal solution is not tractable for the scheduling of tasks using both renewable and consumable resources. In general, the dynamic and discrete nature of task assignment and resource allocation decisions increase the problem complexity; thus, the underlying optimization problems are NP-hard; for instance, see Farias and Roy (2006). Moreover, for real life problems, uncertainty due to noisy data and unexpected events needs to be taken into account during the modelling stage.

In some problems, the accomplishment of tasks cannot be assumed for sure since agents may be unsuccessful. Ahuja et al. (2007) assumed that multiple weapons can be assigned to a single target and considered a static environment as all decisions are made at the beginning. They proposed exact branch-and-bound algorithms for the network flow counterpart of the problem and also used a neighborhood search based heuristic. Alighanbari and How (2008) extended the weapon task assignment problem where the targets can be shot down by the agents. They formulated the problem using a Markov decision process and introduced one-step lookahead heuristics to compare with the performance of mixed integer linear programming. Chen et al. (2009) introduced an asset-based dynamic weapon-target assignment optimization model subject to capability, strategy, resource and engagement feasibility constraints. In order to solve this problem, evolutionary algorithms were developed. Wacholder (1989) applied neural networks to solve a weapon-task assignment problem

efficiently within a static setting. Deng et al. (2013) also applied genetic algorithms to solve an integer programming formulation of a static task assignment problem where heterogeneous unmanned aerial vehicles can cooperate to accomplish multiple tasks to minimize total mission time. Recently, Davis et al. (2017) considered an asset-based defensive variant of the dynamic weapon-task assignment problem. They applied an approximate dynamic programming approach to find an optimal fire control policy for a defensive missile system.

Stochastic programming approaches have been developed to take into account uncertain parameters. For instance, Murphey (2000) developed a two-stage nonlinear integer stochastic programming formulation of the dynamic weapon assignment problem where the numbers and locations of targets are unknown a priori. Although the model allows arrivals of new tasks over time, it does not take into account observations of past allocation outcomes. A cutting plane optimization method was proposed to solve the underlying integer program. Castanon and Wohletz (2002) also studied a dynamic task-resource allocation problem where unsuccessful tasks are assigned further resources over two stages. The outcome of the first stage resource allocation is observed before making the second stage allocations to multiple tasks. They formulated the problem as a two-stage stochastic control problem and introduced an approximation for admissible control space to be used in a model-predictive control algorithm. Ahner and Parson (2015) considered a dynamic programming formulation of the two-stage stochastic programming problem. They assumed that the number of target arrivals in the first stage is known, but in the second stage, it is assumed to be stochastic and following a known distribution. They solved the approximate weapon-task assignment problem using an adaptive dynamic programming method.

It is well known that dynamic problems with large state spaces and action sets suffer from the curse of dimensionality. Therefore, different tractable approaches have been proposed and efficient heuristics and approximation algorithms have been developed to solve the task assignment and resource allocation problem (e.g. Calinescu et al. (2002) and Bertsimas et al. (2014)). Among those approaches, the simulation optimization and approximate dynamic programming methods are worthwhile to mention. In particular, approximate dynamic programming based algorithms are introduced for the dynamic task assignment problem with various applications in transportation (Powell (1996), Spivey and Powell (2003), and Spivey and Powell (2004)) and dynamic resource allocation (Farias and Roy (2006), Powell et al. (2002), and Powell and Topaloglu (2006)). Powell et al. (2002) applied an adaptive dynamic programming algorithm to the resource allocation problem. A constructive rule based heuristic (Xin et al., 2011) and ant colony optimization (Pendharkar, 2015) are other models applied to dynamic task assignment and resource allocation problems.

In this paper, we consider a stochastic multi-period task-resource allocation problem where a team of agents and a pool of different kinds of resources need to be deployed on a given set of tasks.

The underlying task-resource allocation problem is formulated using a Markov decision process. At the beginning of each period, based on the current system state, the agents are allocated some of the resources and assigned to specific tasks. An agent's success probability for a task depends on the resources employed. The overall goal is to maximize the utility of the successfully completed tasks by the end of the planning horizon. Our contribution in this paper is threefold;

- First, we model the joint task-resource allocation problem by taking into account opportunities to retry the same task in the future. If an agent has not been successful in completing the task, this task may be re-tried at a later period. In this way, the team of agents may increase the overall performance and utilise available resources efficiently over time. To the best of our knowledge, reallocation of tasks and resources during the planning horizon has not been considered in previous resource management applications developed in the literature.

- Secondly, we introduce a new constructive heuristic that assigns agents and resources to tasks sequentially using the estimated marginal utility in view of retry option. Based on this heuristic, we furthermore propose forward (approximate) dynamic programming algorithms and an evolutionary algorithm. We introduce various value function approximations based on single and multiple features.

- Thirdly, we perform computational experiments to evaluate the performance of all approaches. All suggested approaches are empirically compared on a number of randomly generated problem instances. The numerical results show that the construction heuristic is very fast and provides good results. For even better results, at the expense of longer computational time, approximate dynamic programming seems a suitable alternative.

The rest of the paper is organised as follows. Section 2 presents the formulation of the underlying task-resource allocation problem. In Section 3, we introduce the heuristic approach and explain basic steps using an illustrative example. Section 4 briefly describes the simulation based approaches in view of different value function approximations. The computational experiments and results are reported in Section 5. Section 6 concludes the paper and points out some ideas for future work.

2. Problem Statement

In this section, we first describe the dynamic stochastic task-resource allocation problem and then provide a dynamic programming formulation via a Markov Decision Process (MDP). The MDP model assumes that the decision-making process has the Markovian property and applies a dynamic programming principle to solve the underlying planning problem.

The dynamic stochastic task-resource allocation problem consists of a team of agents to be deployed to accomplish a given set of tasks using the available resources over a fixed planning horizon. The success probability of an agent assigned to a task in any period depends on the resources allocated. Decisions on the task and resources allocated to each agent for each time period are made at the beginning of each period, and take into account the remaining resources and tasks that have not yet been successfully completed. The overall objective is to maximize the expected utility of the successfully completed tasks by the end of the planning horizon.

2.1. Notation and Assumptions

Table 1 briefly describes the notation used for the problem formulation. Let I and J denote the number of available agents and tasks (indexed by i and j), respectively. We assume that there exist K types of resources (indexed by k) to accomplish those tasks during a planning horizon. The decisions are made at discrete time points $t = 1, \cdots, T$ during the planning horizon. Each period involves the same time length and may represent a week, a day, or an hour.

Table 1: Description of notation.

Notation	Description
T	planning horizon that is discretised by $t = 1, \cdots, T$ time periods
I, J	number of agents and tasks (indexed by $i = 1, \cdots, I$, and $j = 1, \cdots, J$), respectively
K	number of resource types (indexed by $k = 1, \cdots, K$)
u_j	utility (reward) to be gained by successfully completing task j
N_t	index set of tasks available at time t
$L_{k,t}$	total amount of resources of type k available at time t
S_t, A_t	set of all possible states and actions at time t, respectively
$\mathcal{P}(s, a, s')$	state transition function that defines probability from state s to state s' given action a
$V_{\pi,t}(s)$	value function evaluated under policy π at state s of time t
ν_{ijt}	success probability of task j if assigned to agent i at time t
x_{ijt}	binary variable set to 1 if agent i is assigned to task j at t; and zero otherwise.
r_{ijkt}	amount of resource type k allocated to agent i to attempt task j at time t
w_{ijk}	weight representing impact of allocating resource type k to agent i to accomplish task j
$G_v(j, t, k)$	marginal utility at iteration v obtained by allocating one additional unit of resource type k to task j at time t
$\mathcal{B}_t$	index set of all successfully accomplished tasks in time period t

We assume that there are finitely many (reusable) resources and tasks available over the planning horizon. Moreover, each agent can be assigned to only one task during a time period. For simplicity

of the problem statement, we make further assumptions; however, they can be relaxed for a general problem description.

- During the planning horizon, the number of agents and tasks remains the same and all types of resources are available at the beginning of the planning horizon.
- Each task is assumed to be completed within the same duration (one time period).

Note that while in this paper we focus on problems where the agents have all the same capabilities and skills as well as are homogeneous, for the sake of generality, the problem definition in this section allows also for heterogeneous agents.

For a given time t, we define decision variables in terms of resource allocation. Let r_{ijkt} represent the amount (units) of resource type k to be allocated to agent i to accomplish task j at time t. Similarly, x_{ijt} denotes a binary variable that takes a value of 1 if task j is assigned to agent i at time t; and zero otherwise. In addition, u_j denotes the utility gathered by successfully completing task j.

The amount of resources to be used to accomplish task j by agent i at time t determines the success probability, ν_{ijt}. At the end of time period t, agent i who attempted to accomplish task j is reported as successful or not. We assume that the success probability of task j to be accomplished by an agent i is monotonically increasing with the amount of resources allocated. In particular, in this paper, we assume the following success probability:

$$\nu_{ijt} = 1 - \frac{1}{1 + \sum_{k=1}^{K} w_{ijk} \times r_{ijkt}} \tag{1}$$

where parameter w_{ijk} represents impact of using resource type k to accomplish task j by agent i.

2.2. System Dynamics and Dynamic Programming Formulation

The dynamic nature of the problem comes from the coordination of tasks and resources over time, and the probabilistic success. We assume that for each task, the resource specifications in terms of types and usage requirements at each time period are known before the assignment takes place. Over time we maintain knowledge of (uncompleted) tasks and the total amount of resource available for each type k. This information is used to specify the system dynamics as follows.

Let N_t represent the set of all tasks available at the beginning of period t, $\mathcal{B}_t$ the set of all successfully completed tasks in time period t. Let $L_{k,t}$ denote the level of available resource type k at time t. In particular, at the beginning of the planning horizon ($t = 0$), N_0 involves all available tasks, $\mathcal{B}_0 = \emptyset$ and $L_{k,0}$ represents the initial level of resources for all types.

The set of tasks at $t+1$ is updated according to the information gathered at the end of period t as follows;

$$N_{t+1} = N_t \setminus \mathcal{B}_t. \tag{2}$$

The resource level at time t is coordinated through the following dynamic equation

$$L_{k,t+1} = L_{k,t} - \sum_{i=1}^{I} \sum_{j=1}^{J} r_{ijkt}, \quad \text{for} \quad k = 1, \cdots, K. \tag{3}$$

We assume that there are no intermediate rewards to be gained, and the utility is gathered at the terminal time period from all tasks being completed successfully. In addition, no discount factor for the rewards is taken into account in the dynamic programming formulation. Given complete dynamics of the system and reward structure, we can now formulate the multi-period task-resource allocation problem using the MDP based approach that takes into account the dynamic stochastic nature of the planning problem and provides the optimal policies at each time period.

We define a state s at time t as a combination of the set of all remaining tasks N_t and the level of resources available $L_{k,t}$, and denote it by $s = (N_t, L_{k,t})$. So s actually is an ordered pair consisting of N_t and $L_{k,t}$. Let S_t represent the set of all possible states at time t. Then it is defined as follows;

$$S_t = \{s \mid \ s = (N_t, L_{k,t}), \ \ N_t \subseteq N_0, \ \ 0 \leq L_{k,t} \leq L_{k,0}\}.$$

An action to be taken at each state of time period t involves both allocation of resources (r_{ijkt}) and assignment of tasks (x_{ijt}) to agents. We denote a set of such actions by A. The admissible actions defined by the decision variables at time t have to satisfy the following linear constraints;

$$\sum_{j \in N_t} x_{ijt} \leq 1, \qquad i = 1, \cdots, I \tag{4}$$

$$\sum_{i=1}^{I} x_{ijt} \leq 1, \qquad j = 1, \cdots, J \tag{5}$$

$$\sum_{k=1}^{K} r_{ijkt} \geq x_{ijt}, \qquad j = 1, \cdots, J, \ i = 1, \cdots, I \tag{6}$$

$$\sum_{k=1}^{K} r_{ijkt} \leq M x_{ijt}, \qquad j = 1, \cdots, J, \ i = 1, \cdots, I \tag{7}$$

$$\sum_{i=1}^{I} \sum_{j=1}^{J} r_{ijkt} \leq L_{k,t}, \qquad k = 1, \cdots, K. \tag{8}$$

The constraints (4) and (5) ensure that each agent and task can be assigned to only one task and agent, respectively. This is because an agent can engage with only one task at a time. On the other hand, the constraints (6) and (7) establish the dependence between decisions of resource allocation and task assignment. Note that M is chosen as a sufficiently large number. If agent i is assigned to task j at time t (i.e., $x_{ijt} = 1$), then at least one unit of resource from type k is required to be assigned (i.e. $r_{ijkt} \neq 0$). However, if no task is assigned to agent i (i.e., $x_{ijt} = 0$), then obviously no resource needs to be allocated (i.e. $r_{ijkt} = 0$). The conditions in (8) impose that no more than the total available resources from each type k can be allocated to agents to conduct the assigned tasks at each time period.

A policy consists of admissible actions $\pi_t(s)$ at state s of time t. Given a state transition function $\mathcal{P}$, the decision maker may choose any action that is available in state $s \in S_t$ and the system then stochastically moves into the next state $s' \in S_{t+1}$. Since the state transition process displays a Markov property, the next state s' depends only on the current state s and the chosen action.

The state transition function is defined in terms of the joint success probability of agents conducting the assigned tasks. A reachable state $s' = (N_{t+1}, L_{k,t+1})$ from the state $s = (N_t, L_{k,t})$ in view of an admissible policy $\pi_t(s) = (x, r)$ is considered if it is possible to reach state N_{t+1} using an admissible assignment. If there is no task assignment (i.e. $x_{ijt} = 0$), the only reachable state is $N_{t+1} = N_t$ and the transition probability function becomes trivial as $\mathcal{P}(s, \pi_t(s), s') = 1$ when s' is the same as the state s; otherwise, $\mathcal{P}(s, \pi_t(s), s') = 0$. When there exists at least one assignment of task j to agent i at time t (i.e. $\sum_{i,j} x_{ijt} \geq 1$), the reachable states can be constructed as follows;

- the set N_{t+1} at $t+1$ is obtained from the set of tasks N_t at t so that $N_{t+1} \subset N_t$,
- the difference between the sets N_t and N_{t+1} should only include the assigned tasks, defined as $N_{t+1} \supset N_t \backslash \{\ j \mid x_{ijt} = 1, \exists i\}$, and
- the required level of resource for each type k at $t+1$ is determined by satisfying the condition $L_{k,t+1} = L_{k,t} - \sum_{i \in I, j \in J} r_{ijkt}$.

Then the transition probability function for each reachable state s' from state s in view of an admissible policy $\pi_t(s)$ can be computed as follows;

$$\mathcal{P}(s, \pi_t(s), s') = \prod_{\{i,j|\ x_{ijt}=1,\ j \in N_t \backslash N_{t+1}\}} \nu_{ijt} \prod_{\{i,j|\ x_{ijt}=1,\ j \in N_{t+1}\}} (1 - \nu_{ijt}).$$

The transition probability function for all non-reachable states is defined as zero.

As mentioned before, the dynamic task-resource allocation problem aims to find an optimal policy that maximizes the total expected utility gained from all the completed tasks by the end of the planning horizon. Therefore, the boundary value function for state $s = (N_{T+1}, L_{k,T+1})$ of the

system is determined as

$$V_{\pi,T+1}(s) = \sum_{j \in N_0 \setminus N_{T+1}} u_j.$$

The recursive backward dynamic equation for value function $V_{\pi,t}(s)$ for state $s = (N_t, L_{k,t})$ at time t (for $1 \leq t \leq T$) can then be formulated as

$$V_{\pi,t}(s) = \max_{\pi_t(s)} \sum_{s' \in S_{t+1}} \mathcal{P}(s, \pi_t(s), s') V_{\pi,t+1}(s'). \tag{9}$$

Notice that the value function at time t is computed in terms of the total expected value functions for the upcoming time periods. In other words, all possible resulting states $s' \in S_{t+1}$, the likelihood of their occurrence $\mathcal{P}(s, \pi_t(s), s')$, and their $(t+1)$-step value $V_{\pi,t+1}(s')$ under policy π are considered to evaluate the future. In summary, given a starting state s, the state transition $\mathcal{P}$ and terminal value function $V_{\pi,T}(s)$, a standard family of algorithms calculates the optimal policy π^* that maximises the expected rewards and is achieved at the maximum value function as follows;

$$\pi_t^*(s) = \arg\max_{\pi_t(s)} \left\{ \sum_{s' \in S_{t+1}} \mathcal{P}(s, \pi_t(s), s') V_{\pi,t+1}(s') \right\}. \tag{10}$$

Thus, the optimal policy is constructed to maximize the expected utility of the successfully completed tasks by the end of the planning horizon.

The dynamic task-resource allocation model (10) can be solved by the traditional backward dynamic programming algorithm where the optimal decisions and value functions are calculated iteratively starting from the terminal time and stepping backwards in time. As in most real life dynamic programming problems, the task-resource allocation problem suffers from the curse of dimensionality arising in state and action spaces. For instance, for J tasks, the highest resource level l for each type and K different types of resources, the MDP formulation of the task-resource allocation problem involves up to $2^J \times (l+1)^K$ possible states at any time period. Note that this also includes the cases when zero remaining resource level left from any type as states. The number of possible actions at each state is determined by all possible combinations of the tasks to be attempted times all combinations of the possible resource allocations. This makes the problem intractable even for small instances, as shown by the results of numerical experiments in Section 5. For example, a problem instance for 3 resource types (with 20 units each) to be allocated among 10 tasks and 3 agents consists of more than 9 million states and allows for more than 200 million possible actions. This motivates the use of approximation methods to reduce state and action spaces as well as the need for heuristics to solve the underlying dynamic task-resource allocation problem efficiently.

3. A Heuristic for Task-Resource Allocation with Retry Option

In this section, we will develop a construction heuristic for the dynamic stochastic task-resource allocation problem. We'll start with a simple heuristic that ignores the possibility that agents may fail and that a task may be attempted again. We then extend the heuristic to take into account the retry option. Note that given all agents are homogeneous, the heuristic does not specify which agent is assigned a particular task, but just ensures that the number of agents used in each period is less than or equal the available number of agents. As a consequence, we drop the subscript i from r_{ijtk} and w_{ijk} unlike the general problem statement given in Section 2.

3.1. Construction Heuristic Ignoring Retry (HW-NR)

The basic idea of this heuristic is to allocate resources to tasks iteratively and in a greedy way, wherever the marginal gain in utility is maximal. It assumes that the success probability of an agent on a task increases monotonically with the amount of allocated resources, with decreasing returns. Monotonicity seems a natural assumption. The heuristic could be adapted to cases where returns are not decreasing. In this case, allocating multiple units (the number that has the largest increase in probability per unit of resource) would have to be considered.

The marginal utility gain that can be achieved by allocating one additional unit of resource from type k to task j at time t can be calculated as

$$G_v(j,t,k) = 0.99^{(t-1)} \left((1 - \frac{1}{1 + w_{jk} + \sum_{k'=1}^{K} r_{jtk'} \times w_{jk'}}) - (1 - \frac{1}{1 + \sum_{k'=1}^{K} r_{jtk'} \times w_{jk'}}) \right) u_j \quad (11)$$

where v is the iteration counter. This basically calculates the increase in success probability of the task, multiplied by the task's utility. The factor $0.99^{(t-1)}$ ensures that the resource usage at an early stage (if possible) is more valuable than at later stages.

The heuristic computes the initial matrix $G_0(j,t,k)$ and then iteratively identifies the maximum element $(j^*, t^*, k^*) = \text{argmax}\{G_v(j,t,k)\}$. It then allocates one additional unit of resource k^* to task j^* in time period t^* and updates the matrix. In particular it computes matrix G_{v+1} from G_v by

- updating element (j^*, t^*, k^*) according to Eq. (11).
- setting all other entries relating to task j^* in times $t' \neq t^*$ to zero, as it assumes retry is not possible and thus the value of allocating resources to taks j^* in any other time period is zero.
- checking how many agents have already been assigned a task in time period t^*. If all agents have been allocated a task, then no new task can be attempted in this time period. Thus all marginal utilities of tasks that are not already allocated are set to zero for this time period.

- checking whether resource type k^* has been fully allocated. Then obviously it is not possible to allocate any additional units of this resource to any task, and thus all entries in the matrix relating to this resource type are set to zero.

This is repeated until all available resources have been assigned. Note that because we assume the success probability increases monotonically with the amount of resources, and because there is no limit to the number of resources assigned to a task, some $G_v(j,t,k)$ will always be positive and all resources will be allocated within the planning horizon.

3.2. Construction Heuristic with Retry (HWR)

The heuristic described in this section is a more elaborate version of the above construction heuristic, taking into account the possibility of retry for an attempted but unsuccessful task when constructing the plan. If a task is allocated in a particular time period, the marginal utility of allocating resources to the same task in later time periods is not set to zero, but discounted. Basically, two situations are considered. If the task has not yet been completed, then attempting it again and assigning resources has the usual marginal utility. If the task has been successfully completed before, the allocated resources can still be re-allocated to other tasks, and we assume the resources would be used in the best possible way, i.e., to capture the next highest marginal utility. The heuristic with retry option weights these two cases by the probability that the task has been completed before time t. To this end, a matrix $H(j,t,k)$ is introduced and computed as follows.

Let us denote by γ_{jt} the probability of task j not being successfully accomplished before time t. The availability of each task j is initially assumed to be one ($\gamma_{j,t} = 1$). Then it is updated at each iteration of the algorithm as

$$\gamma_{jt} = \frac{1}{1 + \sum_{k=1}^{K} r_{j,t-1,k} \times w_{jk}} \gamma_{j,t-1}. \tag{12}$$

For any task that has been attempted before but not successfully accomplished yet, there are retry possibilities at the next stages of the algorithm. The marginal value $H_v(j,t,k)$ for task j at time period t using matrices H_{v-1} and G_v is updated as;

$$H_v(j^*,t,k) = \gamma_{j^*t} G_v(j^*,t,k) + (1-\gamma_{j^*t}) \ \max\{ H_{v-1}(j',t,k) \mid j' = 1,\cdots,J,\ j' \neq j^* \} \tag{13}$$

Notice that the matrix H_v consists of marginal values of the allocated resources for task j^* as well as the value of possible use of that resource in other tasks with availability factor γ_{j^*t}. If task j^* is still active and is in set N_t, then $\gamma_{j^*t} = 1$. In this case, its expected marginal value will be gathered by the corresponding value in $G_v(j^*,t,k)$.

Algorithm 1 Pseudo code of the HWR algorithm

Given model specifications (I, J, K, T) and resource levels L_k

– Set iteration number $v = 1$

– Initialize $\gamma_{jt} = 1$, $r_{jtk} = 0$, $x_{jt} = 0$ for $j = 1, \cdots, J$, $t = 1, \cdots, T$, $k = 1, \cdots, K$.

– Compute initial marginal matrices for $j = 1, \cdots, J$, $t = 1, \cdots, T$, $k = 1, \cdots, K$

$$G_0(j,t,k) = 0.99^{(t-1)} \left(1 - \tfrac{1}{1+w_{jk}}\right) u_j, \text{ and } H_0(j,t,k) = G_0(j,t,k).$$

while $\sum_{k=1}^{K} L_k \neq 0$, **do** ▷ Repeat task-resource allocation

Allocate one unit of resource to task with the highest marginal worth.

– Find task j^* with the highest marginal value.

$$(j^*, t^*, k^*) = \arg\max \{ H_{v-1}(j,t,k) \mid j = 1, \cdots, J,\ t = 1, \cdots, T,\ k = 1, \cdots, K \}$$

– Allocate one unit of resource type k^* to task j^* at time t^*.

– Set $r_{j^*t^*k^*} := r_{j^*t^*k^*} + 1$, and $x_{j^*t^*} := 1$.

Update the availability factors, marginal values and resource levels:

– Compute $\gamma_{j^*,t'} = \left(\dfrac{1}{1 + \sum_{k'=1}^{K} r_{j^*,t'-1,k'} \times w_{j^*k'}} \right) \gamma_{j^*,t'-1}$ for $t' = t^* + 1, \cdots, T$.

– Update marginal values for task j at time t using resource types $k = 1, \cdots, K$ as

$$G_v(j,t,k) := \begin{cases} 0.99^{(t-1)} \left(\frac{1}{1+\sum_{k'=1,k'\neq k}^{K} r_{jtk'} \times w_{jk'}} - \frac{1}{1+w_{jk}+\sum_{k'=1,k'\neq k}^{K} r_{jtk'} \times w_{jk'}} \right) u_j \text{ if } j = j^*,\ t = t^* \\ G_{v-1}(j,t,k), \text{ otherwise} \end{cases}$$

$$H_v(j,t,k) := \begin{cases} \gamma_{jt} G_v(j,t,k) + (1-\gamma_{jt}) \max\limits_{j'=1,\cdots,J, j'\neq j} \{H_{v-1}(j',t,k)\} \text{ if } j = j^*, t = t^*, t^*+1, \cdots, T \\ H_{v-1}(j,t,k), \text{ otherwise} \end{cases}$$

– Update $L_{k^*} := L_{k^*} - 1$. If $L_{k^*} = 0$, then set $H_v(j,t,k^*) := 0$ for $j = 1, \cdots, J,\ t = 1, \cdots, T$.

Check whether all agents are allocated at t^*:

If $\sum_{j=1}^{J} x_{jt^*} = I$, then for all j and k update $H_v(j,t^*,k) := \begin{cases} 0, & \text{if } x_{jt^*} = 0 \\ H_v(j,t^*,k) & \text{if } x_{jt^*} = 1 \end{cases}$

Increase the iteration number $v := v + 1$

end while

The heuristic then works with matrix H in the same way as the heuristic ignoring retry using matrix G, i.e., iteratively one unit of resource is assigned to a task where the marginal utility is maximal. The plan is re-computed in every time period to reflect the information (success or failure of attempted tasks) that became available. The pseudo code of the HWR algorithm is presented as Algorithm 1.

3.3. Numerical Example

In order to illustrate the main steps of the HWR algorithm, we consider a toy numerical example. Suppose that there are two agents ($I = 2$) that can use two resource types ($K = 2$) to accomplish three tasks ($J = 3$) during the remaining three time periods ($T = 3$). To simplify calculations, we assume that each agent gains the same (unit) utility $u_j = 1$ for successfully completing any job j. Total units of resources in both types are given as $L_1 = 3$ and $L_2 = 2$. The following matrix W consists of weights w_{jk} associated with an impact of allocating one unit of resource type k to task j.

$$W = [w_{jk}] = \left[\begin{array}{c|cc} & k=1 & k=2 \\ \hline j=1 & 2.0 & 1.0 \\ j=2 & 1.6 & 1.4 \\ j=3 & 1.2 & 1.8 \end{array}\right].$$

Iteration 0**:** We initialize parameters and compute marginal matrices as

$$G_0(:,:,1) = \left[\begin{array}{c|ccc} & t=1 & t=2 & t=3 \\ \hline j=1 & 0.667 & 0.660 & 0.653 \\ j=2 & 0.615 & 0.609 & 0.603 \\ j=3 & 0.545 & 0.540 & 0.535 \end{array}\right], \textit{ and } \quad G_0(:,:,2) = \left[\begin{array}{c|ccc} & t=1 & t=2 & t=3 \\ \hline j=1 & 0.500 & 0.495 & 0.490 \\ j=2 & 0.583 & 0.578 & 0.572 \\ j=3 & 0.643 & 0.636 & 0.630 \end{array}\right].$$

We then set up $H_0(:,:,1) = G_0(:,:,1)$, and $H_0(:,:,2) = G_0(:,:,2)$. The main steps of the algorithm are then applied until all available resources are allocated as follows.

Iteration 1**:** The maximum marginal value is found as 0.667 for task $j^* = 1$, at time $t^* = 1$, and resource type $k^* = 1$. This suggests to allocate one unit of resource from type $k = 1$ to task 1. Therefore, we set $x_{11} = 1$ and $r_{111} = 1$. The availability factor $\gamma_{12} = \gamma_{13} = 0.33$ shows that task 1 has equal probability of 33% to still be available in time periods 2 and 3. We then compute the marginal values of matrix G for $j^* = t^* = 1$ and $k = 1, 2$ as $G_1(1, 1, 1) = 0.133$ and $G_1(1, 1, 2) = 0.083$ from Eq. (11). The other entries for $j, t = 2, 3$ and $k = 1, 2$ remain the same as those in $G_0(j, t, k)$. We then update the values of matrix H_1 associated with task j^* at time $t = 1, 2, 3$ for resource types $k = 1, 2$ because probabilities of each task being available in later periods and marginal values of each task change. For example, $H_1(1, 2, 1) = 0.33 \times G_1(1, 2, 1) + (1 - 0.33) \times \max\{0.609, 0.540\} = 0.626$.

Overall, we obtain

$$H_1(:,:,1) = \begin{bmatrix} 0.133 & 0.626 & 0.620 \\ 0.615 & 0.609 & 0.603 \\ 0.545 & 0.540 & 0.535 \end{bmatrix}, \quad H_1(:,:,2) = \begin{bmatrix} 0.083 & 0.589 & 0.583 \\ 0.583 & 0.578 & 0.572 \\ \mathbf{0.643} & 0.636 & 0.630 \end{bmatrix}.$$

Since there remain resources available of both types and not all agents at time 1 have been assigned, we carry on with the next iteration in the same manner.

Iteration 2: The maximum marginal 0.643 is obtained for task $j^* = 3$, at time $t^* = 1$, using resource type $k^* = 2$. Therefore, one unit of resource from type 2 will be assigned to task 3. Then, we update decisions of task assignment and resource allocation ($x_{31} = 1$ and $r_{312} = 1$) and resource levels ($L_2 = 1$). The availability factors γ_{32} and γ_{33} are 0.357. We compute $G_2(3,1,1) = 0.107$ and $G_2(3,1,2) = 0.140$ while the other entries of G_2 remain the same as in G_1.

$$G_2(:,:,1) = \begin{bmatrix} 0.133 & 0.660 & 0.653 \\ 0.615 & 0.609 & 0.603 \\ 0.107 & 0.540 & 0.535 \end{bmatrix}, \quad G_2(:,:,2) = \begin{bmatrix} 0.083 & 0.495 & 0.490 \\ 0.583 & 0.578 & 0.572 \\ 0.140 & 0.636 & 0.630 \end{bmatrix}.$$

The marginal values in matrix H_2 are also updated. Since tasks 1 and 3 are already assigned ($x_{11} = x_{31} = 1, x_{21} = 0$), no agent is available for task 2 at time 1. In order to enforce this condition, both marginal values $H_2(2,1,1)$ and $H_2(2,1,2)$ are fixed at zero.

$$H_2(:,:,1) = \begin{bmatrix} 0.133 & \mathbf{0.626} & 0.620 \\ 0.000 & 0.609 & 0.603 \\ 0.107 & 0.595 & 0.589 \end{bmatrix}, \quad H_2(:,:,2) = \begin{bmatrix} 0.083 & 0.589 & 0.583 \\ 0.000 & 0.578 & 0.572 \\ 0.140 & 0.606 & 0.600 \end{bmatrix}.$$

The current resource levels are $L_1 = 2$ and $L_2 = 1$. The next iteration follows.

Iteration 3: The highest marginal value is 0.626 and achieved at $t^* = 2$ for task $j^* = 1$ and resource type $k^* = 1$. Therefore, we set $x_{12} = 1$ and $r_{121} = 1$. The resource levels are updated as $L_1 = 1$ and $L_2 = 1$, and $\gamma_{13} = 0.111$. We reflect changes in matrices G_3 and H_3.

Iteration 4: At this stage, maximum marginal is 0.609 when one unit of resource type 1 ($k^* = 1$) is assigned to task $j^* = 2$ at time $t^* = 2$. Thus $x_{22} = 1$, $r_{221} = 1$ and $\gamma_{23} = 0.384$. Then the marginal matrices become

$$G_4(:,:,1) = \begin{bmatrix} 0.133 & 0.132 & 0.653 \\ 0.615 & 0.145 & 0.603 \\ 0.107 & 0.540 & 0.535 \end{bmatrix}, \quad G_4(:,:,2) = \begin{bmatrix} 0.083 & 0.082 & 0.490 \\ 0.583 & 0.133 & 0.572 \\ 0.140 & 0.636 & 0.630 \end{bmatrix}.$$

At time 2, no agent is available ($x_{12} = x_{22} = 1, x_{32} = 0$). Therefore, we fix $H_4(3,2,1) = H_4(3,2,2) = 0$. The resource levels become $L_1 = 0$ and $L_2 = 1$. Then we set $H_4(j,t,1) = 0$ for each j and t due

to a lack of resource type 1.

$$H_4(:,:,1) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad H_4(:,:,2) = \begin{bmatrix} 0.083 & 0.432 & 0.588 \\ 0.000 & 0.133 & 0.589 \\ 0.140 & 0.000 & \mathbf{0.600} \end{bmatrix}.$$

The last resource level in type 2 ($k^* = 2$) is allocated to task 3 at time ($t^* = 3$). Then we set $x_{33} = 1$, $r_{332} = 1$. At time 3, not all agents have been assigned ($x_{13} = x_{23} = 0, x_{33} = 1$) but no resource is left ($L_1 = L_2 = 0$). Therefore, the algorithm terminates with the best task-resource allocation as displayed in the following table.

Resource type	$k=1$			$k=2$		
Time / Tasks	$j=1$	$j=2$	$j=3$	$j=1$	$j=2$	$j=3$
$t=1$	1	–	–	–	–	1
$t=2$	1	1	–	–	–	–
$t=3$	–	–	–	–	–	1

Once this plan has been constructed, the assignment for the first period is implemented, and data about which tasks have been successfully completed is collected. Then, we move on to the next iteration, and re-compute a new plan as described above based on the new information for all remaining time periods.

3.4. Computational Complexity

We will report on runtimes of all approaches in Section 5.2. To determine the computational complexity of the heuristics in big-Oh notation, as before let J be the number of jobs, K the number of resource types, T the number of time periods, and Z the total number of resource units available. The heuristic with retry maintains two tables G and H of size $J \times T \times K$. To generate a plan, it iteratively assigns one resource at a time, i.e., it goes through Z iterations. To allocate one resource, in each iteration it has to identify the maximum entry in table H, which has complexity $O(J \times T \times K)$ (Step 8 in Algorithm 1). It then updates all data, with the most time consuming step being the update of H in Step 16, because it involves updating $O(K \times T)$ elements, and for each update we need to find the maximum of $J-1$ entries, so this operation is again $O(J \times T \times K)$. Overall, constructing a plan has thus complexity $O(J \times T \times K \times Z)$.

The heuristic without retry can be implemented more efficiently, because any job is attempted in at most one time step, and in any iteration, if it has not yet been scheduled, only the earliest feasible time step (with still a free agent) needs to be considered. So in effect, with some clever bookkeeping, one only has to maintain and search a $J \times K$ matrix in each iteration, yielding an overall computational complexity of $O(J \times K \times Z)$ to construct one plan.

4. Simulation-based Dynamic Programming Approach

In order to solve the dynamic task-resource allocation problem, we also consider simulation based stochastic dynamic programming methods (namely forward dynamic programming (FDP) and approximate dynamic programming (ADP)) as alternative approaches to the heuristic introduced in Section 3. In this section, we provide a brief introduction to FDP and ADP with various value function approximation methods; the interested reader is referred to Powell (2011) and Topaloglu and Powell (2005) for further information.

The traditional dynamic programming algorithm uses the backward dynamic programming principle where the optimal decisions and value functions are calculated iteratively starting from the terminal time and stepping backwards in time. Although this produces the exact solution, it is affected by the curse of dimensionality since the value function is computed at each state and all possible actions are evaluated. On the other hand, approximate dynamic programming (ADP), which is founded on the basis of the forward dynamic programming algorithm, is designed to reduce action and state spaces by adopting an approximation technique for the value function. Forward dynamic programming (FDP) is a simulation based algorithmic framework and solves the underlying MDP problem using a strategy that steps forward through time starting from an initial state. Each sample path is generated using a Monte Carlo simulation from the same initial state. The value functions are evaluated for all states (a look-up table) or updated at the states on a random path (reached from the initial state) using aggregation of states or regression models.

4.1. Forward Dynamic Programming Algorithm

At the m-th iteration of the FDP algorithm, as presented in its pseudo code in Algorithm 2, the value function $\bar{V}_t^m(s')$ is updated for some or all states $s' \in S_t^m$ to approximate the real value function $V_t(s')$. In this sense, the FDP algorithm differs from the backward dynamic programming algorithm that computes the value function at every state. The performance of the FDP algorithm depends on how the value function is initialized and the decisions can be suboptimal. In our case, we initialize the value function to be "optimistic" by assigning a state a value equal to the sum of the utilities of all uncompleted tasks, plus something for available resources (Step 2 of Algorithm 2. This will encourage FDP to explore different states. The final value function is an approximate solution to the problem since the FDP algorithm does not compute the value function at every state, but only those reached from the initial state.

In order to solve the MDP formulation of the task-resource allocation problem, we implement the FDP algorithm using the heuristic HWR as well as various value function approximations with features of the system. At the m-th iteration ($1 \leq m \leq O$) of the simulation procedure, the FDP algorithm starts from the same state and applies the heuristic HWR. At the m-th iteration of the

Algorithm 2 Pseudo code of the FDP algorithm

1: Initilize simulation number O and set initial state s_0.

2: Compute initial value function:

$$\bar{V}_t^0(s) = \sum_{j \in N_t} u_j + 0.1 \sum_{k=1}^{K} L_{kt}, \quad \forall s, \quad t = 1, \cdots, T.$$

$$\bar{V}_{T+1}^0(s) = \sum_{t=1}^{T} \sum_{j \in \mathcal{B}_t} u_j, \quad \forall s.$$

3: **for** $m = 1, \cdots, O$ **do** ▷ Do O iterations

4: Set initial state of iteration $s_0^m = s_0$.

5: **for** $t = 1, \cdots, T$ **do** ▷ Simulate a path through time

6: − Determine $(x^*, r^*) = \text{argmax}_{(x,r)} \sum_{s' \in S_{t+1}^m} P(s'|s_t^m, (x,r)) \bar{V}_{t+1}^{m-1}(s')$.

7: − Update $\bar{V}_t^m(s_t^m) = \sum_{s' \in S_{t+1}^m} P(s'|s_t^m, (x^*, r^*)) \bar{V}_{t+1}^{m-1}(s')$.

8: − Generate success or failure of attempted tasks by Monte Carlo simulation.

9: − Compute new state s_{t+1}^m based on s_t^m, x^*, r^* and generated random outcome.

10: **end for**

11: **end for** ▷ Terminate the algorithm.

simulation procedure, the approximate value function at state s_t^m of time t is

$$\bar{V}_t^m = \max_{(x,r)} \sum_{s' \in S_{t+1}^m} P(s'|s_t^m, (x,r)) \bar{V}_{t+1}^{m-1}(s'). \tag{14}$$

Then Monte Carlo simulation is used to determine whether an agent was successful or not (based on success probability ν_{ijt}). If task j has been completed successfully, it is removed from the set N_t not to be reattempted. For the next iteration at $(t+1)$, we update the set of uncompleted tasks and total resource level for each type.

4.2. ADP with Various Approximations

Approximate Dynamic Programming (ADP) mitigates the curse of dimensionality of FDP by introducing approximations. The ADP approach may be considered as an extension of the simulation based methods of stochastic dynamic programming. The estimates of the value function are generated from Monte-Carlo simulation of state trajectories and accordingly, the value function is updated at each iteration. The ADP method follows the main steps of the FDP algorithm with some modifications in Steps 2, 6 and 7. Instead of representing the value function as a lookup table for all

possible states, we approximate the state of the system by only one or two features, and integrate the above heuristics (with or without retry option) in the estimation of the state's value.

4.2.1. A Value Function Approximation with Single Feature

Here, we consider a value function approximation using a single feature of time in conjunction with the heuristic algorithms (with and without retry option). We assume that the value function at iteration m can be approximated by

$$\bar{V}_t^m\left(s_t^m\right) = \rho^m\left(t\right)\Omega\left(s_t^m\right), \tag{15}$$

where $\Omega\left(s_t^m\right)$ is a heuristic value of state s_t^m that is computed by allocating all available resources using one of the heuristics (with or without retry option) from Section 3 and adding up all the obtained marginal utilities. The factor $\rho^m\left(t\right)$ is a factor depending only on time and is learned by ADP in the following way.

Initially, $\rho^0\left(t\right)$ is set to 1, and then updated in each iteration (replacing Step 7 in Algorithm 2) by

$$\rho^m\left(t\right) = \left(1-\lambda\right) \times \rho^{m-1}\left(t\right) + \lambda \times \bar{\rho}, \tag{16}$$

where λ represents the learning rate (which we set to 0.1 in the experiments in Section 5) and m is the iteration counter. $\bar{\rho}$ is the factor computed in the current iteration that is derived from the decision step. First, the value of the best decision is computed as

$$\hat{v}_t^m(s_t^m) = \max_{(x,r)} \sum_{s' \in S_{t+1}^m} P\left(s' \mid s_t^m, (x,r)\right) \rho^{m-1}\left(t+1\right) \Omega\left(s'\right). \tag{17}$$

Then, the value of $\bar{\rho}$ is used to balance the computed value of the current state $\hat{v}_t^m(s_t^m)$ with the value predicted by the heuristic $\Omega\left(s_t^m\right)$, and is updated as

$$\bar{\rho} = \frac{\hat{v}_t^m(s_t^m)}{\Omega\left(s_t^m\right)}. \tag{18}$$

4.2.2. A Value Function Approximation with Multiple Features

We also consider the value function approximation approach using two features, namely time and level of resources. Because the number of possible resource levels would be too large for a lookup

table, we cluster the level of resources for each type $k = 1, \cdots, K$ into the following four categories:

$$\tau_k = \begin{cases} a, & \text{if } L_{kt} = 0 \\ b, & \text{if } L_{kt} = 1 \\ c, & \text{if } L_{kt} \in \{2, 3\} \\ d, & \text{if } L_{kt} \geq 4. \end{cases} \tag{19}$$

Thus, τ_k can take one of the values a, b, c, and d for each category k of resource. Similar to the single feature case described above, we assume the value of a state can be approximated as

$$\bar{V}_t^m(s_t^m) = \rho^m\left(t, \tau_1, \ldots, \tau_K\right) \Omega\left(s_t^m\right). \tag{20}$$

The factor $\rho^m\left(t, \tau_1, \ldots, \tau_K\right)$ is updated in every iteration according to

$$\rho^m\left(t, \tau_1, \ldots, \tau_K\right) = (1 - \lambda) \times \rho^{m-1}\left(t, \tau_1, \ldots, \tau_K\right) + \lambda \times \bar{\rho}, \tag{21}$$

where $\bar{\rho}$ is determined by first calculating the best estimated obtainable value given we are in state s_t^m

$$\hat{v}_t^m(s_t^m) = \max_{(x,r)} \sum_{s' \in S_{t+1}^m} P\left(s' | s_t^m, (x, r)\right) \rho^{m-1}\left(t, \tau_1, \ldots, \tau_K\right) \Omega\left(s'\right). \tag{22}$$

and then setting

$$\bar{\rho} = \frac{\hat{v}_t^m(s_t^m)}{\Omega\left(s_t^m\right)}. \tag{23}$$

4.3. Evolutionary Algorithm

Evolutionary Algorithms (EAs) are general purpose meta-heuristics based on the principles of natural evolution. EAs maintain a population of solution candidates throughout the search. After initialization, new solutions are iteratively generated by selecting good solutions from the population (parents), crossover (combination of information in parents) and mutation (small random alterations). The new individuals (children) are then also evaluated and inserted into the population, usually replacing the worst solutions. The algorithm stops usually after a given number of iterations, returning the best solution found. For more details on EAs, the interested reader is referred to Eiben and Smith (2007).

A key design decision with respect to EAs is how a solution is represented. In our case, the EA only decides on the amount of resources to be allocated in each time period, whereas the heuristic without retry option is used in each time period to determine the task and resource allocation.

For the computational experiments, we apply the generic EA implemented in Matlab's Global Optimization Toolbox. There is one variable (gene) for each resource type and time period. For example, for a case with 6 time periods and 2 types of resources, an EA solution consists of 12 variables. The EA tool used requires genes to be real numbers. Therefore, we normalize and round the values of all genes so that they sum up to the total available resources. Then given the number of resources at each period, we allocate the resources to the tasks at that period using a one period heuristic. The fitness of the individual is calculated by the expected value of the task-resource allocation from running 1000 forward simulations. The EA uses a population size of 100, linear ranking selection with stochastic uniform sampling, and an elite of size 2. We used intermediate crossover with ratio of 1 and Gaussian mutation with standard deviation of 0.1.

5. Computational Experiments

In this section, we first describe the design and data structure used for numerical experiments and then present the computational results of different approaches studied for the dynamic task-resource allocation problem.

5.1. Design of Experiments and Data

We design a series of computational experiments in order to illustrate the performance of different algorithms. The various approaches are abbreviated as follows;

- Traditional MDP approach applies a *backward dynamic programming algorithm (BDP)* to find a full optimal policy.
- *Heuristics with retry option (HWR) and with no retry option (HW-NR)* analyse the remaining jobs at every state by allocating one unit of resources from various types according to their expected marginal utility. After each resource is allocated, the immediate strategy is applied.
- *Forward Dynamic Programming (FDP)* is a simulation based dynamic programming technique to evaluate the value function at each state of a path starting from the initial state.
- *SF-NR and MF-NR* are based on approximate dynamic programming approach with single and multi factors, respectively. The heuristic (HW-NR) algorithm with no retry option is implemented.
- *SF-WR and MF-WR* consider an approximate dynamic programming approach with single and multi factors, respectively, and apply the heuristic (HWR) in view of retry option to compute the feature to describe a state.

- *EV-AL* is the evolutionary algorithm that is implemented in the standard toolbox of MATLAB.

These algorithms were implemented in MATLAB and all computational experiments were run in a laptop with Intel Core 2 duo 2.2 Ghz, 2 GB RAM.

For the numerical experiments, we randomly generated a set of problem instances. We classified these problems according to their relative sizes into small, medium and relatively larger size problems. Description of problem instances (in terms of resource type, units of resources, number of agents and tasks and time periods) is presented in Table 2. We assume that one, two and three types of resources are to be allocated to two, three and five agents. The small and medium size problems involve eight tasks to be assigned to two and three agents during six time periods. The large size problems have twenty and thirty tasks to be accomplished by five agents during five time periods. We assume that there are 7, 10 and 15 units of resources at various problem types.

Table 2: Description of problem instances.

Problem Instances	Resource Types	Number of Agents	Number of Tasks	Time Periods	Units of Resources (in each type)
R1-A2	1	2	8	6	(15)
R1-A3	1	3	8	6	
R2-A2	2	2	8	6	(10)
R2-A3	2	3	8	6	
R3-A2	3	2	8	6	(7)
R3-A3	3	3	8	6	
R2-T20	2	5	20	5	(15)
R3-T20-L	3	5	20	5	$w_{ijk} \in [0.25, 0.45]$
R3-T20	3	5	20	5	
R3-T20-H	3	5	20	5	$w_{ijk} \in [0.55, 0.75]$
R3-T30	3	5	30	5	

The utility gained from each task is randomly generated by a uniform random distribution between 10 and 20.

Finally, weights (w_{ijk}) representing impact of assigning one unit of resource to an agent to accomplish a task are generated from a uniform distribution between 0.4 and 0.6. Only for problems R3-T20-L and R3-T20-H, we consider parameters w_{ijk} with low and high values varying within intervals [0.25, 0.45] and [0.55, 0.75], respectively.

5.2. Computational Results

The performance of each algorithm is measured in terms of total expected utility achieved at the end of planning horizon and the CPU time taken to solve each problem instance.

The simulation-based FDP algorithms are run with 1000 iterations twenty times independently to measure the effect of path randomness on the total utility. From the empirical studies, we observe that there is no substantial improvement on the total utility when more than 1000 iterations are considered. We present the results of each algorithm in terms of average and standard errors of team performance in Table 3. On the other hand, the traditional backward dynamic programming algorithm is run once and provides the optimal policy but only for small size instances. In order to compute the expected team performance using the HWR and HW-NR algorithms, we first solve the underlying problem and then evaluate the same task-resource allocation plan with twenty different realisations of random events. The expected value and standard errors of evaluated total utilities are computed for each problem and displayed in Table 3 as well.

Table 3: Team performance obtained by different approaches, mean ± *std. err.*

Methods	BDP	HW-NR	HWR	FDP	SF-NR	SF-WR	MF-NR	MF-WR	EV-AL
Problems	Average Team Performance								
R1-A2	**60.31**	42.34 *±0.0*	60.20 *±0.0022*	**60.31** *±0.0*	**60.31** *±0.0*	**60.31** *±0.0*	**60.31** *±0.0*	**60.31** *±0.0*	60.30 *±0.0022*
R1-A3	**88.70**	61.64 *± 0.0022*	88.43 *± 0.0089*	88.55 *± 0.0045*	87.63 *± 0.0089*	87.86 *± 0.0045*	88.04 *± 0.0157*	88.48 *± 0.0291*	85.72 *± 0.0224*
R2-A2	**45.41**	28.28 *± 0.0022*	45.13 *± 0.0112*	45.22 *± 0.0067*	44.19 *± 0.0157*	44.40 *± 0.0201*	44.83 *± 0.0291*	45.04 *± 0.0201*	43.13 *± 0.0224*
R2-A3	**69.86**	37.92 *± 0.0067*	69.53 *± 0.0291*	68.97 *± 0.0291*	69.03 *± 0.0291*	69.34 *± 0.0358*	69.66 *± 0.0224*	69.81 *± 0.0626*	68.89 *± 0.0559*
R3-A2	**85.32**	51.26 *± 0.0112*	81.32 *± 0.0425*	83.31 *± 0.0470*	83.00 *± 0.0335*	83.32 *± 0.0514*	84.04 *± 0.0716*	84.74 *± 0.0425*	81.67 *± 0.0648*
R3-A3	TL	63.38 *±0.0112*	107.23 *± 0.0380*	108.10 *± 0.0537*	107.41 *± 0.0224*	108.44 *± 0.0604*	109.36 *± 0.0470*	**109.90** *± 0.0827*	107.00 *± 0.0760*
R2-T20	TL	122.71 *± 0.0201*	242.58 *± 0.0738*	240.86 *± 0.1945*	241.27 *± 0.0537*	243.81 *± 0.0850*	244.99 *± 0.0917*	**245.18** *± 0.1185*	239.47 *± 0.1207*
R3-T20-L	TL	103.68 *± 0.0224*	198.10 *± 0.1029*	197.31 *± 0.2594*	198.05 *± 0.0760*	198.94 *± 0.1319*	199.14 *± 0.1364*	**199.68** *± 0.1789*	196.07 *± 0.1409*
R3-T20	TL	146.70 *± 0.0201*	290.88 *± 0.1096*	289.65 *± 0.2504*	289.07 *± 0.0850*	292.11 *± 0.1275*	293.75 *± 0.1275*	**294.00** *± 0.1856*	286.89 *± 0.1453*
R3-T20-H	TL	209.11 *± 0.0179*	363.88 *± 0.1073*	358.54 *± 0.2706*	362.04 *± 0.0850*	365.94 *± 0.1409*	366.22 *± 0.1386*	**367.36** *± 0.1677*	359.27 *± 0.1342*
R3-T30	TL	157.71 *± 0.0246*	313.62 *± 0.1431*	308.11 *± 0.3287*	311.99 *± 0.1051*	315.00 *± 0.1677*	315.35 *± 0.2012*	**317.02** *± 0.2795*	309.49 *± 0.1655*

The efficiency of each algorithm is measured by CPU time (seconds). The results of the heuristics (HWR and HW-NR) and simulation-based FDP algorithms, presented in Table 4, represent the total time (in terms of seconds) taken to find the task-resource allocation and to compute the total utility for a single run. For the BDP algorithm, we imposed a time limit of 1.000.000 seconds (about 12

days) to solve a problem instance. As mentioned before, the FDP algorithm is also repeated twenty times. However, it could not solve the larger problem instances like R3-A3, R2-T20, R3-T20 and R3-T30 within the given time limit (such cases are listed as "TL" in Table 4). For those cases, we report the average and standard errors of total utilities achieved at the end of the time limit in Table 3. The highest average and the lowest standard errors of team performance (in Table 3) as well as the lowest solution time (in Table 4 but excluding HW-NR due to its poor performance) are highlighted in bold.

From the results of computational experiments (in Tables 3 and 4), we can make the following observations.

- As it was explained before, a traditional backward dynamic programming algorithm produces an optimal policy by evaluating the value function at each possible state. However, it suffers from the curse of dimensionality. This was also the case for the dynamic task-resource allocation problem. As we see from the numerical experiments, BDP can only solve small and medium size problems (namely, R1-A2, R1-A3, R2-A2, R2-A3, and R3-A2) and requires the highest CPU time in comparison with other approaches. For instance, the optimal policy for R3-A2 (with 2 agents, 3 resource types during 6 time periods) was obtained in approximately 77 hours. Moreover, we could not find a solution for large size problem instances within the preset time limit. We conclude that performance of the BDP algorithm heavily depends on the problem size.

- The ADP algorithm with multi-features (in terms of time and resource levels) outperforms the single-featured ADP approaches regardless of the choice of the heuristics HWR and HW-NR. Among all approaches apart from BDP, the best approximate team performance is achieved by the MF-WR algorithm (i.e. multi-featured approximate dynamic programming with heuristic in view of retry possibilities), but it also requires the highest CPU time. The overall utility obtained by the MF-WR algorithm for large size problems constitutes the best expected team performance while the solutions of small and medium size problems are very close to optimality. Furthermore, we observe that the standard error of team performance obtained by the MF-WR (FDP) algorithm is larger than those obtained by other algorithms for small (large) size problem instances (see Table 3).

- Retry opportunities play an important role in improving the overall quality of the expected utility as can be seen from the numerical results. The heuristic with retry option (HWR) outperforms the heuristic ignoring retry possibilities (HW-NR). In fact, the HW-NR algorithm provides the worst team performance. In comparison of the HWR with other approaches, we observe that the HWR algorithm produces a comparable expected team performance at

Table 4: CPU time (seconds) taken to solve the problem by various approaches.

Problem Types	Solution Methods								
	BDP	HW-NR	HWR	FDP	SF-NR	SF-WR	MF-NR	MF-WR	EV-AL
R1-A2	804.81	0.13	1.12	345.56	**0.56**	3.56	2.91	18.43	425.36
R1-A3	1504.85	0.33	2.34	567.10	**1.02**	7.12	6.11	53.98	613.65
R2-A2	5548.63	0.65	**5.43**	1630.39	140.32	432.21	1098.06	9845.77	887.36
R2-A3	9566.61	1.01	**9.23**	2504.11	108.12	407.58	987.86	3642.21	1020.36
R3-A2	277432.60	4.13	**41.23**	15245.32	245.54	532.23	2098.36	4065.98	1214.33
R3-A3	TL	15.43	**114.13**	TL	545.15	1342.12	4608.63	9562.12	1587.38
R2-T20	TL	27.01	**234.84**	TL	837.42	2018.66	4096.56	15226.31	1281.38
R3-T20-L	TL	58.19	**579.81**	TL	1634.59	5044.03	8838.05	38564.05	1610.45
R3-T20	TL	65.31	**587.31**	TL	1674.72	5045.73	9103.75	38450.28	1630.94
R3-T20-H	TL	78.09	**591.35**	TL	1667.21	5144.91	9014.79	39335.32	1639.24
R3-T30	TL	147.48	**1174.27**	TL	3181.92	12109.72	22759.34	107660.70	2621.14

minimum CPU solution time. In particular, the quality of the solution for small and medium size problem instances is as good as the optimal solution (obtained by the BDP algorithm). The SF-WR and MF-WR methods perform better in terms of the quality of the solution, but are much slower than the SF-NR and MF-NR algorithms, respectively.

- As displayed in Table 3, large (small and medium) problem instances have high (relatively low) standard errors of team performance.

  The higher the standard error, the less representative the sample will be of the overall population. In comparison to other problem instances, the highest variability is realised for problem instance R3-T30 regardless the choice of the algorithm. The forward dynamic programming and variant of ADP approximation algorithms except MF-WR provide the highest standard errors for problem instance R3-T20-H (where the weights w_{ijk} are generated with high values). We can conclude that model parameters strongly affect the performance of the dynamic programming algorithms.

- The FDP algorithm performs better than the HWR algorithm on small size problems (R1-A2, R1-A3, and R2-A2) in terms of team performance. However, it quickly becomes computationally demanding as problem size increases, and often could not complete the set 1000 iterations within the given time limit. Table 3 reports the best approximate solution obtained when the maximum time limit was reached. For most cases, the expected team performance of the FDP algorithm is still comparable with the HWR algorithm and definitely better than the HW-NR algorithm.

- Finally, we can see that performance of the evolutionary algorithm in terms of team utility

is close to the performance of the FDP algorithm. However, it is much faster especially for large problems. On the other hand, it performs worse than other approaches apart from the HW-NR algorithm.

5.3. Influence of Model Parameters

We also investigate the impact of model parameters such as utilities and weights on the team performance. For this purpose, we generated both parameters following a uniform distribution with mean values varying between 9 and 25 and spread of 10 for utilities (u_j) and between 0.45 and 0.65 with spread of 0.2 for weights (w_{ij}). For illustrative purposes we consider problem R1-A3 (involves one resource type and three agents) and solve it with different combinations of model parameters by the BDP algorithm as well as the heuristic approaches with and without considering retry opportunities. Table 5 displays the ratio (%) of team performance achieved by the heuristics HWR and HW-NR to the BDP algorithm.

Table 5: Performance of heuristics with and without retry option in percent relative to BDP

	Performance of BDP vs HWR					Performance of BDP vs HW-NR				
	Utilities					Utilities				
Weights	$[4,14]$	$[8,18]$	$[12,22]$	$[16,26]$	$[20,30]$	$[4,14]$	$[8,18]$	$[12,22]$	$[16,26]$	$[20,30]$
$[0.55,0.75]$	100	100	100	99.85	99.88	75.44	75.96	70.23	69.01	68.06
$[0.5,0.7]$	100	100	99.12	99.34	100	72.43	70.79	68.11	63.77	65.35
$[0.45,0.65]$	100	99.16	100	100	99.16	65.35	65.44	65.25	59.79	61.09
$[0.4,0.6]$	100	99.50	99.14	98.61	99.44	62.20	57.43	55.88	57.69	55.51
$[0.35,0.55]$	100	99.14	98.75	100	99.41	54.83	54.15	51.79	53.85	50.24

As can be seen from Table 5, when the utilities are generated within $[4,14]$, HWR produces the optimal solution (as the BDP algorithm does) regardless of the choice of the mean weights. In most of other cases, the gap between solutions obtained by the HWR and BDP algorithms is less than 1% (i.e. the ratio varies between 99.14% and 99.88%). There exist two cases (having weights in $[0.35,0.55]$ and $[0.4,0.6]$ and utilities in $[12,22]$ and $[16,26]$, respectively) where the gap is around 1.29% and 1.25%. These results basically confirm that the heuristic HWR is robust with respect to the changes in model parameters. On the other hand, for HW-NR, the choice of mean utilities and resource usage weights plays an important role in performance of the heuristic with no retry option in comparison with the BDP algorithm.

5.4. Learned Value Functions in ADP

Next, we illustrate the evolution of learning parameters used for different value function approximations within the forward dynamic programming algorithm in Figure 1. In this case, the problem

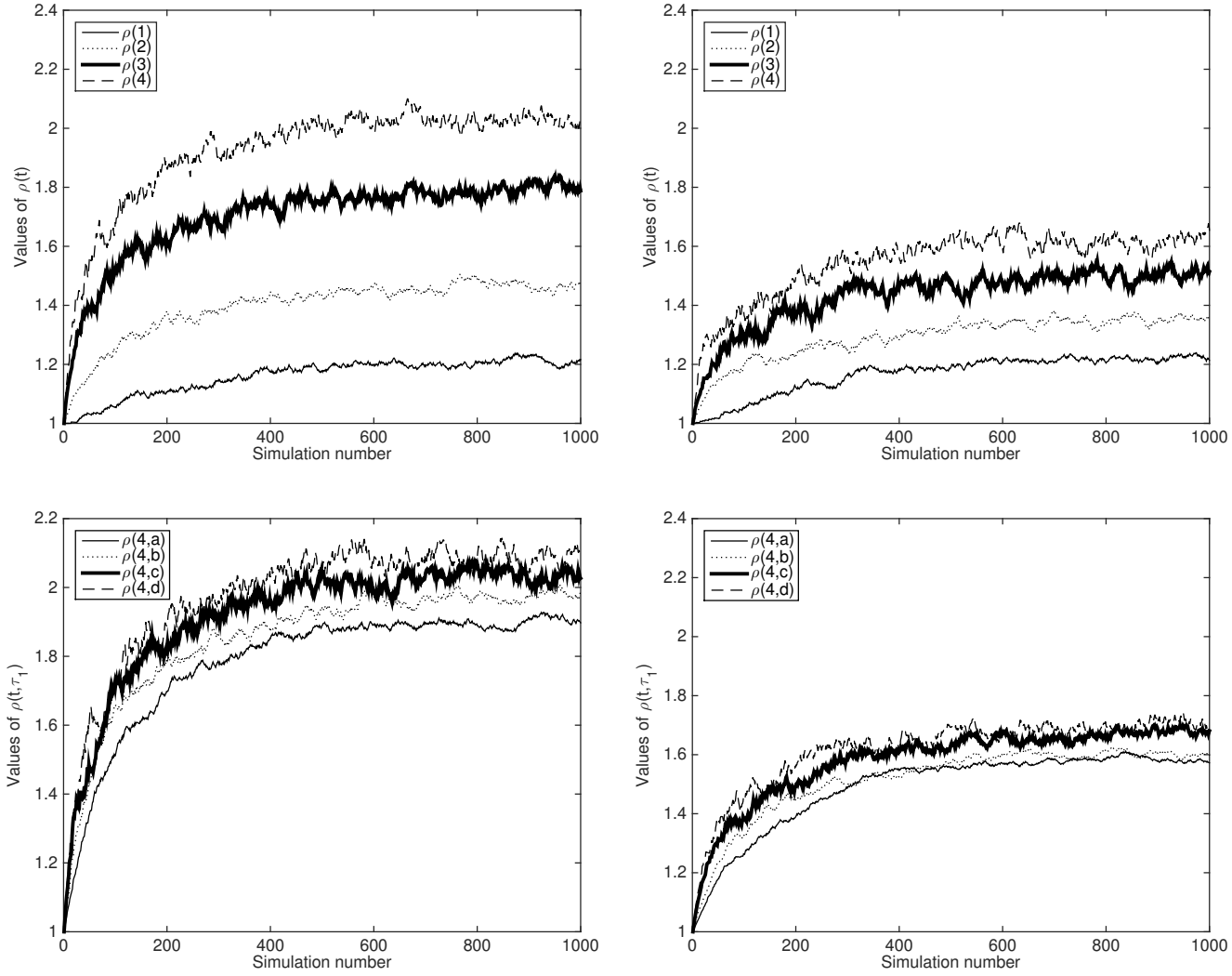

Figure 1: Evoluation of learning parameters using single-feature (top) and multi-feature (bottom) value function approximations with HW-NR (left) and HWR (right).

R1-A3 is solved by the value function approximations with single-feature (at the top panel) and multi-feature (at the bottom panel) using the HW-NR (on the left side) and HWR (on the right side) heuristics. The value of $\rho(t)$ for $t = 1, 2, 3, 4$ remaining time periods is obtained over 1000 simulations. Similarly, we display values of $\rho(t, \tau_k)$ for $t = 4$ remaining time periods and changing resource levels $\tau_k = a, b, c, d$ (see Section 4) in this figure. One can easily observe that both $\rho(t)$ and $\rho(t, \tau_k)$, regardless the choice of heuristics with and without retry possibilities, converge to values higher than unity.

We also present the values of learning parameters achieved after 1000 simulation experiments in Figure 2. The values of learning parameters used for the heuristic with no-retry option are

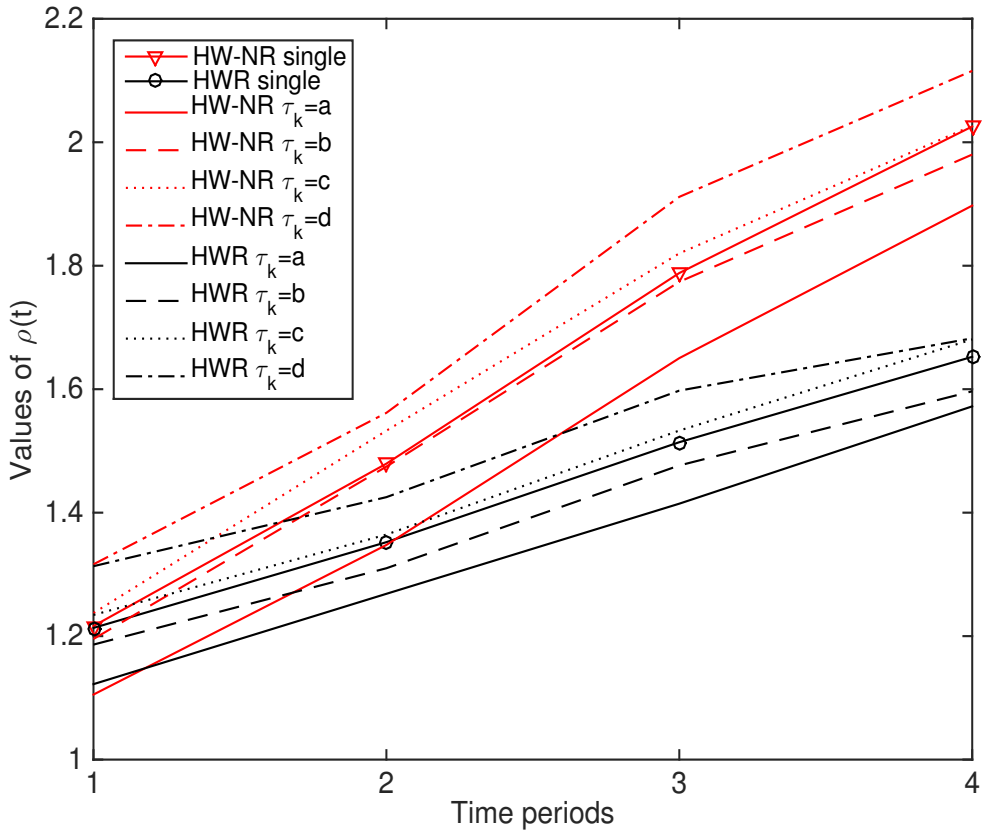

Figure 2: Learning parameters obtained by single-feature and multi-feature heuristics

always higher than the ones obtained for the heuristic with retry option. The empirical experiments show that the estimated value function with no retry is always low; thus, those parameters help to compensate for the difference. We also observe that the remaining time periods and the resource levels play an important role on the values of learning parameters. In case of the single-feature value function approximation with both HWR and HW-NR, less remaining time towards the end of the planning horizon provides higher $\rho(t)$ values since there is less opportunity to re-try.

6. Concluding Remarks

In this paper, we investigated the dynamic task-resource allocation problem with retry opportunities for failed attempts. This problem has various real world applications, including in military where one has to allocate troops and weapons to combat missions over time, or repair service, where one has to allocate engineers and equipment to repair tasks. We proposed exact and approximate approaches for solving this problem, including novel heuristic algorithms that attempt to take into

account the future worth of each resource allocation during the remaining time periods. Features of the underlying stochastic system are identified to define value function approximations. We designed computational experiments using randomly generated problem instances and compared performance of the construction heuristics with more traditional dynamic programming approaches.

The computational results show that the traditional dynamic programming algorithm is only capable of solving small size problems. The retry possibilities increase the team performance and are thus an important aspect to consider. The heuristic with retry option HWR is robust in comparison with other approaches. It produces a total utility close to the optimal or best solution while being comparatively fast. The numerical experiments show that features of time and resource levels play an important role on the performance of the approximate dynamic programming approach. In particular, the multi-featured value function approximation based on HWR outperforms other approximate dynamic programming approaches in terms of team performance, albeit with relatively high computational time.

As future work one may consider to extend the underlying problem setting within cooperative or collaborative multi-agent systems. In this case, it would be possible to assign more than one agent to the same task. As in many real-life applications, task duration may vary. There is also potential research on how to model and solve the task-resource allocation problem for variable success probabilities. For instance in defence applications, the success probability of hitting a target at the first attempt will be different to the ones at the next attempts because of moving targets, changing environment and increased security systems.

7. References

Ahner, D. K., Parson, C. R., 2015. Optimal multi-stage allocation of weapons to targets using adaptive dynamic programming. Optimization Letters 9 (8), 1689–1701.

Ahuja, R. K., Kumar, A., Jha, K. C., Orlin, J. B., 2007. Exact and heuristic algorithms for the weapon-target assignment problem. Operations Research 55 (6), 1136–1146.

Alighanbari, M., How, J. P., 2008. A robust approach to the UAV task assignment problem. International Journal of Robust and Nonlinear Control 18 (2), 118–134.

Angalakudati, M., Balwani, S., Calzada, J., Chatterjee, B., Perakis, G., Raad, N., Uichanco, J., 2014. Business analytics for flexible resource allocation under random emergencies. Management Science 60 (6), 1552–1573.

Bellingham, J., Tillerson, M., Richards, A., How, J. P., 2003. Multi-task allocation and path planning for cooperating UAVs, Cooperative Control: Models, Applications and Algorithms. Springer, pp. 23–41.

Bertsimas, D., Gupta, S., Lullic, G., 2014. Dynamic resource allocation: a flexible and tractable modeling framework. European Journal of Operational Research 236 (1), 14–26.

Calinescu, G., Chakrabarti, A., Karloff, H., Rabani, Y., 2002. Improved approximation algorithms for resource allocation, Integer Programming and Combinatorial Optimization. Springer, pp. 401–414.

Castanon, D. A., Wohletz, J. M., 2002. Model predictive control for dynamic unreliable resource allocation. In: Proceedings of the 41st IEEE Conference on Decision and Control. Las Vegas, Nevada USA.

Chandler, P., Pachter, M., Rasmussen, S., Schumacher, C., 2002. Multiple task assignment for a UAV team. In: Proceedings of the AIAA Guidance, Navigation, and Control Conference.

Chao, X., Liu, L., Zheng, S., 2003. Resource allocation in multisite service systems with intersite customer flows. Management Science 49 (12), 1739–1752.

Chen, J., Xin, B., Peng, Z., Dou, L., Zhang, J., 2009. Evolutionary decision-making for the dynamic weapon-target assignment problem. Science in China Series F: Information Sciences 52 (11), 2006–2018.

Davis, M. T., Robbins, M. J., Lunday, B. J., 2017. Approximate dynamic programming for missile defense interceptor fire control. European Journal of Operational Research 259 (3), 873–886.

de Weerdt, M. M., Clement, B. J., 2009. Introduction to planning in multiagent systems. Multiagent and Grid Systems 5 (3), 1359–1365.

Deng, Q., Yu, J., Wang, N., 2013. Cooperative task assignment of multiple heterogeneous unmanned aerial vehicles using a modified genetic algorithm with multi-type genes. Chinese Journal of Aeronautics 26 (5), 1238–1250.

Eiben, A. E., Smith, J. E., 2007. Introduction to Evolutionary Computing. Springer.

Ernst, A., Jiang, H., Krishnamoorthy, M., 2006. Exact solutions to task allocation problems. Management Science 52 (10), 1634–1646.

Farias, V., Roy, B. V., 2006. Approximation algorithms for dynamic resource allocation. Operations Research Letters 34 (2), 180–190.

Gulpinar, N., Canakoglu, E., Thoms, J., 2010. Robust team decision making under uncertainty. International Journal of Applied Decision Sciences 3 (3), 206–220.

Haslum, P., Geffner, H., 2014. Heuristic planning with time and resources. In: European Conference on Planning. pp. 107–112.

Hong, S. A., Gordon, G. J., 2015. Decomposition-based optimal market-based planning for multiagent systems with shared resources. In: Workshop and Conference Proceedings. Vol. 15. pp. 351–360.

Koenig, S., Tovey, C., Zheng, X., Sungur, I., 2007. Sequential bundle-bid single-sale auction algorithms for decentralized control. In: The International Joint Conference on Artificial Intelligence. pp. 1359–1365.

Liu, N., Truong, V. A., 2013. Multi-resource allocation scheduling in dynamic environments. Manufacturing and Service Operations Management 15 (2), 280–291.

Murphey, R. A., 2000. Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems. Vol. 42. Kluwer Academic Publishers, Ch. An approximate algorithm for a weapon target assignment stochastic program, pp. 406–421.

Nwozo, C. R., Nkeki, C. I., 2012. On a dynamic optimization technique for resource allocation problems in a production company. American Journal of Operations Research 2, 357–363.

Nygard, K., Chandler, P., Pachter, M., 2001. Dynamic network flow optimization models for air vehicle resource allocation. In: American Control Conference, Arlington, Virginia, USA. pp. 1853–1858.

Pekka, M., Ahti, S., 2009. Combining a multiattribute value function with an optimization model: An application to dynamic resource allocation for infrastructure maintenance. Decision Analysis, 139 – 152.

Pendharkar, C., 2015. An ant colony optimization heuristic for constrained task allocation problem. Journal of Computational Science 7, 37–47.

Powell, W., 1996. A stochastic formulation of the dynamic assignment problem with an application to truckload motor carriers. Transportation Science, 30 (3), 195–219.

Powell, W., 2011. Approximate dynamic programming. John Wiley and Sons.

Powell, W., Shapiro, J. A., Simão, H. P., 2002. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. Transportation Science 36 (2), 231–249.

Powell, W., Topaloglu, H., 2006. Approximate dynamic programming for large-scale resource allocation problems. INFORMS Tutorials in Operations Research, 123–147.

Samuel, A., Guikema, S. D., 2012. Resource allocation for homeland defense: Dealing with the team effect. Decision Analysis, 238 – 252.

Spivey, M., Powell, W., 2004. The dynamic assignment problem. Transportation Science 38 (4), 399–419.

Spivey, M., Powell, W. B., 2003. Some fixed-point results for the dynamic assignment problem. Annals of Operations Research 124, 15–33.

Tharumarajah, A., 2001. Survey of resource allocation methods for distributed manufacturing systems. Production Planning and Control 12 (1), 58–68.

Topaloglu, H., Powell, W., 2005. A distributed decision-making structure for dynamic resource allocation using nonlinear functional approximations. Operations Research 53 (2), 281–297.

Tovey, C., Lagoudakis, M. G., Jain, S., Koenig, S., 2005. The generation of bidding rules for auction-based robot coordination. In: Parker, L. E., Schneider, F. E., Schultz, A. C. (Eds.), Multi-Robot Systems: From Swarms to Intelligent Automata. Vol. 3. Springer, pp. 3–14.

Wacholder, E., 1989. A neural network-based optimization algorithm for the static weapon-target assignment problem. ORSA Journal on Computing 1 (4), 232–246.

Wiesemann, W., Kuhn, D., Rustem, B., 2012. Multi-resource allocation in stochastic project scheduling. Annals of Operations Research 193 (1), 193–220.

Xin, B., Chen, J., Peng, Z., Dou, L., Zhang, J., 2011. An efficient rule-based constructive heuristic to solve dynamic weapon-target assignment problem. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans 41 (3), 598–606.

Zhen, L., 2015. Task assignment under uncertainty: stochastic programming and robust optimisation approaches. International Journal of Production Research 53 (5), 1487–1502.

Zheng, X., Koenig, S., 2009. K-swaps: cooperative negotiation for solving task-allocation problems. In: The International Joint Conference on Artificial Intelligence. pp. 373–379.