Decentralised Online Planning for Multi-Robot Warehouse Commissioning

Daniel Claes
smARTLab, Department of Computer Science
University of Liverpool, UK
daniel.claes@liverpool.ac.uk

Frans Oliehoek
smARTLab, University of Liverpool
AMLab, University of Amsterdam
fao@liverpool.ac.uk

Hendrik Baier*
Digital Creativity Labs, Department of Computer Science
University of York, UK
hendrik.baier@york.ac.uk

Karl Tuyls
smARTLab, Department of Computer Science
University of Liverpool, UK
k.tuyls@liverpool.ac.uk

ABSTRACT

Warehouse commissioning is a complex task in which a team of robots needs to gather and deliver items as fast and efficiently as possible while adhering to the constraint capacity of the robots. Typical centralised control approaches can quickly become infeasible when dealing with many robots. Instead, we tackle this spatial task allocation problem via distributed planning on each robot in the system. State of the art distributed planning approaches suffer from a number of limiting assumptions and ad-hoc approximations. This paper demonstrates how to use Monte Carlo Tree Search (MCTS) to overcome these limitations and provide scalability in a more principled manner. Our simulation-based evaluation demonstrates that this translates to higher task performance, especially when tasks get more complex. Moreover, this higher performance does not come at the cost of scalability: in fact, the proposed approach scales better than the previous best approach, demonstrating excellent performance on an 8-robot team servicing a warehouse comprised of over 200 locations.

Keywords

Decentralised Online Planning; SPATAPs; Multi-Robot Systems; MMDP; Warehouse Commissioning

1. INTRODUCTION

Multi-robot systems are becoming more common in industry. It is expected that technology will revolutionise the *Factories of the Future* (*Industry 4.0*), leading to factories with networked devices and mobile manipulation platforms. Particularly important will be a paradigm shift from fixed robots to more flexible general-purpose mobile robots. An important problem in *Industry 4.0* is multi-robot warehouse commissioning, where robots fetch and deliver items.

A crucial aspect of warehouse commissioning is the coordination of which worker delivers and fetches which item. Currently, with human pickers, this task is tackled using pre-computed pick lists [14]. These pick lists are generated centrally by the warehouse management system and then executed statically, which means that the pickers cannot be re-assigned when en-route.

In robotics, this problem has been framed as a *Multi-Robot Task Allocation problem (MRTA)* [13]. MRTA approaches assign a set of tasks to the robots, which then execute these from start to finish. A common solution to MRTA are auctions where each robot bids for the tasks according to its own evaluation. While well studied [6, 11] and conceptually clear, these approaches can lead to two shortcomings: First, in a highly adaptive and changing environment, new tasks are likely to appear while the robots are on their way. However, the static assignments mean that the robots cannot respond accordingly. Second, these approaches typically require a centralised task allocation component, which may hinder scalability, flexibility and robustness.

In order to address these drawbacks, a new formal framework of *spatial task allocation problems (SPATAPs)* was introduced that describes how a team of agents interacts with a dynamically changing set of tasks, where the tasks are spatially distributed over a set of locations in the environment [9]. SPATAPs form a sub-class of Multi-Agent MDP (MMDP) [4] problems, which themselves are an extension of Markov Decision Processes (MDP) [20] to multiple agents. These models have the advantage of providing principled solutions under uncertainty of action outcomes and provide clear definitions of optimality. In particular, the SPATAPs framework is 'action-based' in the sense that is treats moving and working on a task alike, as actions.[1]. This means that its reasoning takes place at a much more fine-grained level than MRTA approaches; the SPATAPS framework facilitates reasoning about all such actions in order to maximize expected utility. For instance, a longer route to a currently targeted

*Hendrik Baier was previously affiliated with the Advanced Concepts Team, European Space Agency, The Netherlands

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

[1]If desired, the acronym can also be interpreted as *spatial action-based task allocation problems*

task might be preferable if it brings the robot closer to locations where it is likely that high-priority task appear in the near future, or robots may position themselves in good locations even without the presence of any tasks. Even if it would be possible to extend MRTA approaches to allow for dynamical reallocation in case that new tasks arrive, they will be inherently incapable of *planning* for such contingencies, which is exactly what SPATAPS do allow.

As a SPATAP is an MMDP, it can be solved using MMDP solution methods. However, such methods are centralised with mentioned drawbacks. Moreover, these solutions do not scale well, due to the exponential explosion of state and action spaces when introducing more agents and tasks. An alternative approach is to tackle these problems using distributed online planning. That is, rather than centrally computing an optimal plan for all agents, each agent in parallel tries to compute a plan for itself. Such a decentralised approach has the benefits that it is robust against failure since there is no centralised planner, and flexible in that it allows to easily introduce more robots if required. An empirical study showed that such a decentralised approach for SPATAPs yields promising results in comparison to solving the MMDP and other baselines [9]. Still, some limitations remain: the approach was demonstrated to scale to 60 nodes, which —while still resulting in more than 6×10^{30} states— is far from the scalability desired in real-life warehouse commissioning tasks. Moreover, it relies on a number of rather ad-hoc approximations to keep the state space over which is actually planned small.

Since Monte Carlo Tree Search (MCTS) [17] methods allow for an effective treatment of problems with large numbers of states, this paper investigates how such methods can be used to overcome the mentioned limitation. In particular, we propose a number of computationally cheap rollout strategies that are specific for SPATAPs, and discuss how these relate to prior work [25, 6]. We also investigate how certain domain-specific modifications of MCTS used in the Scotland Yard game [19] can be transferred to SPATAPs.

Additionally, we extend SPATAPs to facilitate more realistic modelling of commissioning warehouse tasks by including a drop-off point, i.e tasks have to delivered to a depot location. We further impose a maximum capacity constraint on the robots, such that they can only perform a limited amount of pick-ups before they have to return to the depot location to drop off their load.

Extensive empirical evaluations show that our MCTS approach leads to significantly higher task performance, *especially for more complex problems*. Moreover, we also show that the approach scales better than the previous state of the art, demonstrating excellent performance on an 8-robot team servicing a warehouse comprised of over 200 locations.

2. RELATED WORK

The work of [14] relates to this research w.r.t. order picking and batching. Such approaches usually only generate static *pick lists*, which are then statically executed. They do not allow for online re-planning or changing the task allocations during execution. Furthermore, they do not run in a decentralised fashion.

Auction-based approaches can be found in [1, 7]. These do not address the sequential and changing nature of the tasks, since they assume tasks to be static. Auctioning also generally relies heavily on communication, while our approach only needs the locations of the other agents. Summarizing, [22] found "that the advantages of the best auction-based methods are much reduced when the robots are physically dispersed throughout the task space and when the tasks themselves are allocated over time".

Another related topic are general resource allocation problems [26]. The problem we address differs from that line of work in that we allow re-allocation at every time step and consider spatially distributed tasks and travel times.

Some related work, that is also using MCTS for solving task allocation problems is described in [15]. However, they deal with MRTA problems with time constraints, i.e. a static task set. A complete plan is computed beforehand in a centralised fashion and then executed, while with SPATAPs we are able to recompute the plan online at every change of the global state.

Generally, most competing approaches based on task allocation methods suffer from the problem that agents do not anticipate future tasks, and/or rely on communication between the agents.

3. PROBLEM DESCRIPTION

In warehouse commissioning, a team of agents needs to service a set of tasks distributed in the warehouse. These tasks are single item orders that have to be fetched from within the warehouse and brought back to the depot. Orders can be placed at any given time, so that new tasks appear over time.

A typical warehouse layout is a *rope ladder* [23], depicted in Figure 1. In this layout, the items are stored in shelves that are organised in aisles with some cross aisles. The shelves can have multiple compartments at different heights. This environment can be represented as a graph $\mathcal{G}$ with a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$ as overlaid in Figure 1. Each node represents a position in the warehouse where the agent can reach a number of storage compartments, on both sides of the aisle and at different heights. This limits the planning complexity without a loss in accuracy, since we assume that the agent can pick from different sides and heights equally well. One node is defined for the depot to which the orders have to be brought, marked in red in Figure 1. The agents have a limited inventory, i.e. they can pick and store a limited number of items, before they have to return to the depot to clear their load.

The agents can move along the edges in the graph. Each edge can have different costs, e.g. edges moving through free space could be quicker than moving along a shelf. There is also the possibility that movement is unsuccessful for the agent, due to wheel slip or other uncertainties.

Orders are mapped to the locations in the graph and appear as tasks at the corresponding nodes. We assume that the distribution of the orders is known, e.g. by analyzing order histories. Therefore, we can model the probabilities of tasks appearing for each location. More specifically, each node has a given chance p that a task appears in the next time step t, and p is known to the agents. The tasks can have different costs c.

3.1 SPATAPs

The commissioning problem can be modeled as a SPATAP, which is a special case of *factored* MMDPs, i.e. the state space is spanned by a set of state variables. We expand the model with inventory states, leading to the following defini-

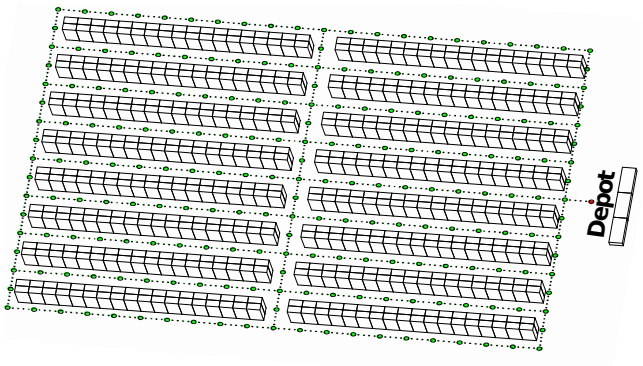


Figure 1: A rope ladder warehouse, with two blocks of shelves and a drop off depot. Overlaid is the resulting navigation graph from the warehouse model.

tion of a *commissioning SPATAP*. For more details, please refer to [9].

Definition 1. A *commissioning SPATAP* is defined as a tuple $\langle \mathcal{D}, \mathcal{G}, \mathcal{I}, \mathcal{T}, \mathcal{S}, \mathcal{A}^M, \mathcal{A}^T, \mathcal{A}, P^M, P^T, R^M, R^T \rangle$, where $\mathcal{D}$ is the set of n agents, $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ is a graph comprised of a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$. Further, $\mathcal{I} = \{\mathfrak{I}_{empty}, \ldots, \mathfrak{I}_{full}\}$ describes the set of inventory states per agent and $\mathcal{T} = \{\mathfrak{T}_0, \ldots, \mathfrak{T}_{|\mathcal{T}|}\}$ is the set of task states per location n_x. $\mathcal{S}$ defines the set of states s, which can be factored as $s = \langle \lambda, \iota, \tau \rangle$, where λ and ι are the respective vectors of the locations and inventory states of all agents, and τ is the vector of the task status for all locations. $\mathcal{A}^M$ and $\mathcal{A}^T$ are the sets of movement and task actions respectively, i.e. each agent i can choose an action $a_i \in \mathcal{A}^M \cup \mathcal{A}^T$ and $a = a_1 \ldots a_n$ is the combined action for all agents. $\mathcal{A}$ is the set of all possible joint actions. $P^T = Pr(\tau', \iota' | \tau, \iota, \lambda, a)$ define the task and inventory transition probabilities, and $P^M = Pr(\lambda' | \lambda, a)$ models the changes in agent locations. $R^T(\tau, \lambda, \iota, a)$ and $R^M(\lambda, a)$ define the rewards for the tasks and the movements.

In our setting, the possible movement actions $\mathcal{A}^M$ are either staying or choosing any outgoing edge to go to the next node. The task related actions are defined as $\mathcal{A}^T = \{PERFORM\_TASK, CLEAR\_LOAD\}$. The task transition function models the execution of a task, e.g., when an agent performs a task at a node and the inventory of the agent is not yet exceeded, then the task state of the node and the inventory of the agent are updated, and it also models the exogenous events of new tasks appearing, i.e. by orders being placed.

3.2 Solving SPATAPs

While using the factored components enables us to relatively easily simulate the environment, already enumerating all possible states may prove difficult, since the resulting state space is equal to $|\mathcal{S}| = |\mathcal{N}|^{|\mathcal{D}|} \cdot |\mathcal{I}|^{|\mathcal{D}|} \cdot |\mathcal{T}|^{|\mathcal{N}|}$. The size depends exponentially on the number of agents and the number of locations.

The approach *Empathy by fixed weight discounting (EFWD)* [9] introduces two orthogonal approximations to enable online planning in the class of SPATAPs.

The first approximation in EFWD is an aggregation of the effect of other robots in the area by using their locations as a proxy to predict their future actions. More specifically, a *presence mass* is calculated over the locations for each timestep into the future. This is the probability distribution of any other agent being at a given location at that timestep. It is calculated using a *self-absorbed (SA)* policy, i.e. planning while ignoring the presence of the other agents. Thus, we can generate the reward function and transition functions such that they only depend on the actions of the planning agent. As a second approximation, the state space is reduced by a *subjective phase myopic approximation*, i.e. the set of nodes where tasks can appear in the planning process of each agent is limited to the k-nearest nodes which already have tasks present.

4. MONTE CARLO TREE SEARCH FOR SPATAPS

A shortcoming of the previous approach is that limiting the state space can lead to unwanted outcomes. For instance, using a phase-myopic approach that only considers the nodes with currently active tasks might lead to the agents not moving at all if there is currently no task present. This could lead to large losses in performance compared to already moving towards the most likely positions where the next tasks may appear.

In this paper, we adopt the basic principle of distributed planning, including the need to (approximately) model other agents, but we seek a more principled way of dealing with the complexity of the individual planning tasks. In particular, rather than limiting the size of the state space of these problems with ad-hoc approximations, we build on principled methods to deal with large state spaces. We build upon sample-based planning methods whose performance is *independent* of the size of the state space [16].

4.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a simulation based search algorithm [17, 10]. It has been very successful, e.g., in general game playing [5] and computer Go—it is the basis for AlphaGo [24]. The main intuition behind MCTS is that by using Monte Carlo simulations to quickly sample thousands of possible trajectories, we can achieve good approximations of the values of possible actions. While doing these Monte Carlo simulations, a search tree, which stores statistics used to guide the search, is built incrementally starting from just a root node. When 'inside' the search tree, the statistics are used to select the most promising actions, when 'outside' the tree, action selection is guided by (computationally cheap) rollout policies.

We base our work on UCT [17], which uses UCB1 [2] to select actions inside the tree. UCT, however, is not directly amenable to distributed planning for SPATAPs. There are a number of challenges that need to be overcome: First, there is an issue with the extremely large number of states we want to consider. Second, it turns out that it can be important to incentivize agents to do tasks themselves, rather than relying on other agents to do them. Finally, a major difficulty is how the planning agent can predict the behaviour of other agents. Previous work based such predictions on full solutions of the 'self-absorbed' MDP, but this is not feasible for the problem sizes we want to consider, and is too slow for application in a MCTS rollout policy. In the remainder of this section, we propose a number of techniques addressing these issues.

4.2 Huge State Spaces: Sparse UCT

While MCTS methods can deal with fairly large state spaces, huge state spaces are problematic. If the number of states reachable from a given state by a given action is very large for example, it can lead to the algorithm always being 'outside the tree' and never building up meaningful statistics. This is a problem in SPATAPs: due to tasks appearing at random nodes, it is very unlikely in our problem setting that choosing the same action from the same state would ever lead to the same successor state twice.

To deal with this problem, we propose to use Sparse UCT [3] for SPATAPs, which builds upon 'Sparse Sampling', an online planning method whose performance is *independent* of the size of the state space [16]. The main idea behind Sparse Sampling is that it is sufficient to sample, for each visited node and action, only some constant number w of successor nodes. We therefore keep track of how many successor states have already been created for the same state and action, and if the sample width w is exceeded, we return a random existing successor state instead of sampling a new one.

4.3 Incentivizing Agents: DIY Bonus

Accurately modelling the behaviour of the other agents is a fundamentally difficult problem (which we treat in the next sub-section), and it is unlikely that an agent will be able to make perfect predictions about what tasks will be addressed by the other agents. However, incorrectly assuming that an important task will be addressed by a team mate can lead to high costs. Therefore, in the face of such uncertainties, it might be good if agents have a slight preference to do tasks themselves.

The inspiration from this comes from the cooperative game of Scotland Yard, where it has been shown that the planning agent sometimes relies too much on other agents fulfilling their task of catching the 'hider' [19]. The authors propose to discount the value of a successful rollout by $r \in [0, 1]$ if a different agent than the planning agent caught the hider. This idea cannot be directly translated to our approach, since we have no definition of 'successful rollout'. However, since we also observed the behaviour that the planning agents were relying too much on the other agents to perform a task, we propose a *do-it-yourself (DIY)* bonus if a task is performed by the planning agent. The action *PERFORM_TASK* is awarded an additional bonus $DIY_r \in [0, 1]$ if it is performed by the planning agent. If the bonus is too small, especially in symmetric configurations, there can be a tendency for no agent to do the task; if it is too high, both agents tend to try to perform the task.

4.4 Modelling Other Agents: Greedy Rollout Policies

Good predictions of team mates are critical for the success of any distributed planning approach. For an approach based on MCTS this is even more difficult, since action predictions for other agents are needed in every step of every simulation. This means that the predictions should not only be sufficiently accurate, but also computationally cheap.

For this purpose we introduce three greedy heuristics for SPATAPs. We do not only use them to predict the actions of other agents (both 'inside' and 'outside' of the tree), we also use them as the rollout policy for the planning agent (only 'outside' the tree), which performs much better than a random rollout policy as we show empirically.

Algorithm 1: Reverse Greedy

Input : s: state $< \lambda, \iota, \tau >$
Output: a: actions for all agent

Let V be a vector of size $|\mathcal{D}|$
Assign $V[i] \leftarrow \infty \; \forall i \in \mathcal{D}$
foreach $n_x \in \mathcal{N}$ **do**
  $v_{best} \leftarrow \max_{i \in \mathcal{D}} \texttt{NV}(n_x, i)$
  $a_{best} \leftarrow \texttt{Index}(\max_{i \in \mathcal{D}} \texttt{NV}(n_x, i))$
  if $v_{best} > V[a_{best}]$ **then**
    $a[a_{best}] \leftarrow \texttt{GoTo}(n_x, a_{best})$
    $V[a_{best}] \leftarrow v_{best}$
return a

All three approaches are based on heuristic valuations of how much value each agent i can generate for each task location n_x. The evaluation function we use is defined as follows:

$$\mathrm{NV}(n_x, i) = \begin{cases} -\infty & \text{if } \tau_{n_x} = \mathfrak{T}_{empty} \\ \frac{\texttt{TV}(\tau_{n_x}, \iota_i)}{\texttt{dist}(\lambda_i, n_x)} & \text{otherwise} \end{cases} \quad (1)$$

where $\mathfrak{T}_{empty}$ means that there is no task present, $\texttt{dist}(\lambda_i, n_x)$ is the length of the shortest path from the agent i to the node n_x and $\texttt{TV}(\tau_{n_x}, \iota_i)$ denotes an evaluation of the task status at node τ_{n_x}, given the current inventory status of the agent ι_i. In our case this is the largest sum of the cost values of the tasks at the node that can still fit in the inventory of the agent, i.e. we keep adding the task with the highest available cost until either the inventory is full, or there is no task left. This function TV can be used to influence the agents' behaviours. It could for instance also include other factors such as the time the task is already active, if older tasks should be valued higher. As NV is a reward function, we aim to maximise its value.

For all three heuristics, we assume that if an agent is at its capacity limit, it will choose the shortest path to the depot and unload. These agents are also not further considered in the planning process, until their inventory is empty again.

4.4.1 Greedy with Social Law

A very simple heuristic is for all agents i to always move towards the node that has the highest evaluation according to our NV function, i.e. $a_i = \texttt{GoTo}(\max_{n_x \in \mathcal{N}} \texttt{NV}(n_x, i), i)$, where the function $\texttt{GoTo}(n_x, i)$ returns the next action on the shortest path of agent i to node n_x. If the agent is already at that node, it returns the action *PERFORM_TASK*.

In order to overcome symmetries, we can apply a social law similarly to [9]: If two agents have the same node with the highest reward, the one with the higher id will go to the best evaluated node, while the next ranked agents will go to the next ranked node. This heuristic will be used as baseline for our evaluation.

4.4.2 Reverse Greedy Allocation

The general idea behind the Reverse Greedy heuristic is to look at the problem from the perspective of the nodes. The idea is that each location in the graph is assigned to the agent that has the best evaluation for this node. This is also comparable to a partition organization as in [25], since all the tasks are distributed between the robots according to their

Algorithm 2: Iterative Greedy

Input : s: state $< \lambda, \iota, \tau >$
Output: a: actions for all agent

Let V be a matrix of size $|\mathcal{D}||\mathcal{N}|$
$V[i][n_x] \leftarrow$ `NV`(n_x, i) $\forall n_x \in \mathcal{N}\ \forall\ i \in \mathcal{D}$
$A_{ass} \leftarrow \emptyset$; $N_{ass} \leftarrow \emptyset$
for *iteration* $i < |\mathcal{D}|$ **do**
  $n_{best} \leftarrow$ None; $a_{best} \leftarrow -1$; $v_{best} \leftarrow \infty$
  foreach *Agent* $i \in \mathcal{D}$ *and* $i \notin A_{ass}$ **do**
    $n_{sel}, v_{sel} \leftarrow$ `GetBest`(V, i)
    while $n_{sel} \in N_{ass}$ **do**
      $V[i] \setminus n_{sel}$
      $n_{sel}, v_{sel} \leftarrow$ `GetBest`(V, i)
    if $v_{sel} > v_{best}$ **then**
      $v_{best} = v_{sel}$;$a_{best} = i$; $n_{best} = n_x$
  $a[a_{best}] \leftarrow$ `GoTo`(n_x, a_{best})
  $A_{ass} \cup a_{best}$; $N_{ass} \cup n_{best}$
return a

evaluations. However, this approach adaptively changes the partition according to the changing locations of the agents and tasks.

Algorithm 1 shows pseudocode for this policy. The `Index` function returns the agent that corresponds to the minimum node value v_{best}. This approach intrinsically takes care of a social law if the agents are always iterated in the same order by the `Index` function, e.g. decreasing by id.

4.4.3 Iterative Greedy Allocation

For the final rollout policy that we propose, we can take inspiration from auctioning approaches (as commonly used in MRTA, i.e. in [6]). We evaluate all locations for all agents and iteratively assign the currently best evaluated location to the highest ranked agent. More specifically, we compute `NV` for all agents and all locations. The agent that has the node with the best evaluation over all agents and locations, gets the location assigned and both, the agent and location are removed for the future allocations. In the next iteration, the agent which has the best evaluation for the remaining node is selected and so on. Algorithm 2 shows pseudocode for this approach. The function `GetBest`(V, i) returns the currently best evaluated location for agent i and its value.

Using the heuristics.

Note that both Reverse and Iterative Greedy are centralized algorithms that suggest actions for all agents for a given state. However, these algorithms can be run decentralised on the robots, which is possible, since the global state is known. During the planning process, each agent needs to predict actions for all other agents, thus these heuristics are an efficient way to calculate these estimated actions all at once. More specifically, during the UCT simulations: when 'inside' the tree, the planning agent overrides the action prescribed by these algorithms with the action prescribed by UCB1. In the rollout phase (i.e., outside the tree) all agents follow the prescribed actions. Additionally, these heuristics serve as baselines. In this case, they are used without any MCTS search. Each agent computes the heuristic until his action is assigned and acts accordingly.

5. EXPERIMENTS & RESULTS

To empirically evaluate our proposed MCTS approach, we have implemented a warehouse simulation and our rollout policies in ROS [21]. Each agent is simulated independently. This is realized by running each agent's planning approach in a different process. Only the global state (λ, ι, τ) is communicated from the simulator to the agents, and there is no direct communication between agents. The depot location is marked in red and the normal task nodes in green. The agents always start at the depot. Movement actions have a 90% chance of succeeding, which simulates the uncertainty in the real world, e.g. due to wheel slip or other sensor noise. The probability of tasks appearing at a node is drawn randomly from a set of three probabilities ($p_{low} = \frac{0.2}{|\mathcal{N}|}, p_{mid} = \frac{0.4}{|\mathcal{N}|}, p_{high} = \frac{1}{|\mathcal{N}|}$). This simulates that certain areas in the warehouse may store items that are ordered at a higher frequency and vice-versa. Each task that appears can have a different priority, i.e. higher costs of not servicing this task. These are sampled according from the set of $(c_1 = 1, c_2 = 2, c_3 = 5)$ with probabilities $(p_{c_1} = 0.8, p_{c_2} = 0.1, p_{c_3} = 0.1)$.

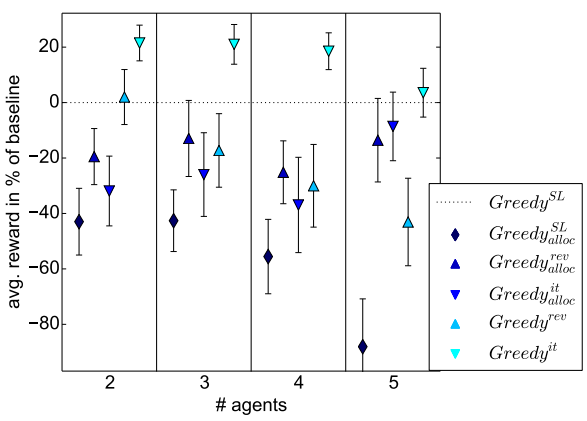
The seeds for creating the probability distribution and for tasks appearing are synchronised between the different runs, such that for all policies, the same number of tasks appear at the same locations during the run. We run 30 simulations of 100 time steps each, unless reported differently. As a baseline we use the proposed rollout policy *greedy with social law* ($Greedy^{SL}$) without any MCTS search. We compare our approach against the state-of-the-art approach *EFWD* with $k = 5$ nearest tasks, and additionally, we compare against using the two heuristics, Reverse Greedy ($Greedy^{rev}$) and Iterative Greedy ($Greedy^{it}$), without any MCTS search. For this evaluation, each planning agents computes the centralised policy until its own action is assigned.

For our approach, the MCTS search runs 20,000 simulations up to a depth of 60 steps, while choosing the actions during the rollouts by one of the rollout policies introduced in the previous section, i.e. Greedy with Social Law ($MCTS^{SL}_{Greedy}$), Reverse Greedy ($MCTS^{rev}_{Greedy}$) and Iterative Greedy ($MCTS^{it}_{Greedy}$) or purely random selection ($MCTS_{random}$). We implemented all rollouts in an ε-greedy manner, i.e. uniformly random actions are performed with a probability of $\varepsilon = 0.05$. These settings were determined empirically.

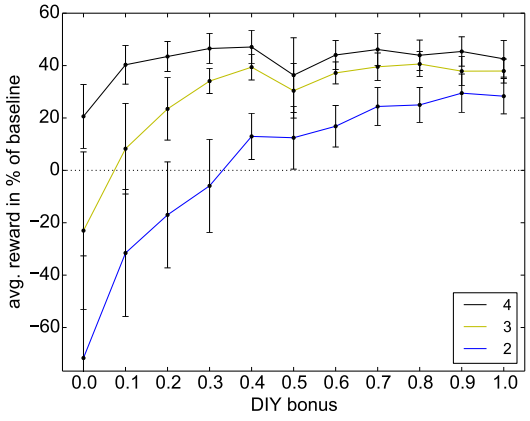
Fixed allocation vs online re-allocation.

We compared the heuristics using fixed allocation, i.e. the robots choose their tasks and do not change it until it is performed, against online re-allocation, meaning that the robots replan at every time step. Thus, the fixed allocation methods (defined as $Greedy^{SL}_{alloc}$, $Greedy^{rev}_{alloc}$ and $Greedy^{it}_{alloc}$) simulate behaviour as following the MRTA framework, while online re-allocation is according to the SPATAPs definition. The results are presented in Figure 2a, where the whiskers show 95% confidence intervals. $Greedy^{it}$ outperforms the baseline $Greedy^{SL}$ and all other approaches with up to four significantly, while with five agents the the fixed allocation methods, $Greedy^{rev}_{alloc}$ and $Greedy^{it}_{alloc}$ are catching up.

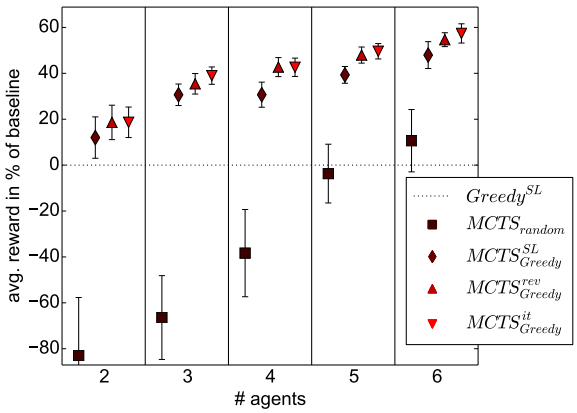
When using fixed allocations, the two proposed heuristic, Reverse and Iterative Greedy, are almost on par. However, using $Greedy^{rev}$ with online planning actually decreases its performance, especially with more agents. The decreasing



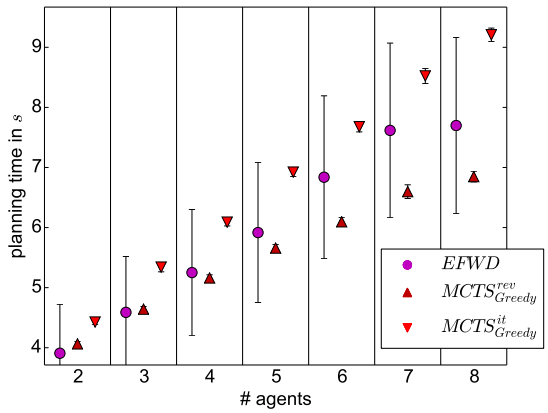
(a) Fixed vs online (re)-allocation.



(b) DIY Bonus.



(c) Different rollout heuristics.



(d) Planning times.

Figure 2: Comparing (a) fixed allocation against online re-allocation, (b) the effect of the DIY bonus and (c) the different rollout policies using the world *warehouse-small* with 30 nodes and different number of agents against the baseline. (d) Comparing the planning time for different policies using the world *warehouse-medium* with 66 nodes and different number of agents. The whiskers show the 95% confidence intervals.

performance can be explained by the nature of the simulation. As the agents all start at the depot, all potential tasks are allocated only to the highest ranked agent. Thus the agents will not spread out. When the allocation is fixed, the agents keep moving out as soon as a task is assigned, thus they spread more making the partitioning of the $Greedy^{rev}$ more effective. Additionally, that all methods (with the exception of $Greedy^{rev}$) move closer together with more agents, is the result of having relatively less tasks per agent. Thus the effect of online planning is smaller. With larger worlds, thus relatively more tasks, this effect is less.

To summarise, the online re-allocation yields a big advantage especially for $Greedy^{it}$ and the $Greedy^{SL}$ baseline, while for $Greedy^{rev}$ is actually a disadvantage.

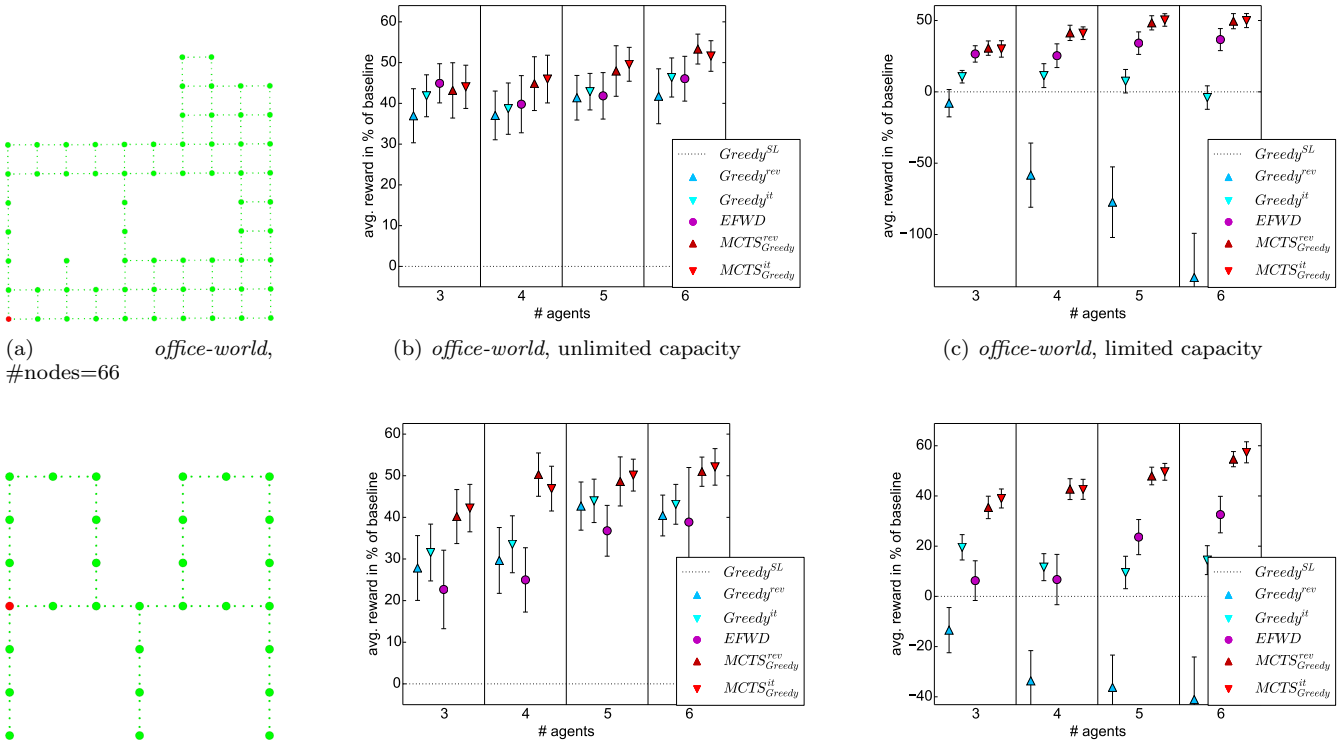
DIY-Bonus.

To evaluate which value for DIY_r yields the best performance, we run the simulation for different number of agents and increasing values of DIY_r. Figure 2b shows the resulting rewards with their 95% confidence intervals. As can be seen, the value of DIY_r has a high influence on the reward, especially for 2 agents. With only 2 agents, there are more situations in which there is a tie for a certain location, so it is better when both head towards it, instead of no one. With more agents these situations occur less frequently. For the remaining experiments we set the DIY bonus to 0.7 as a tradeoff, since especially in smaller worlds we experienced suboptimal behaviour due to multiple robots moving towards the same task location.

Different rollout policies.

When comparing the different rollout policies, the results in Figure 2c show that the $MCTS^{SL}_{Greedy}$ outperforms the $Greedy^{SL}$ baseline significantly. Thus adding MCTS search to the heuristics significantly improves the performance. However, MCTS in itself with using the random rollout policy ($MCTS_{random}$), performs less than the baseline, showing that the rollout policies have a large influence on the performance.

Using Iterative and Reverse Greedy for the rollouts yields a significantly higher performance than the other policies, especially for more agents. Both of these strategies already



(a) *office-world*, #nodes=66

(b) *office-world*, unlimited capacity

(c) *office-world*, limited capacity

(d) *warehouse-small*, #nodes=30

(e) *warehouse-small*, unlimited capacity

(f) *warehouse-small*, limited capacity

Figure 3: Comparing the different approaches in the worlds *office* (a-c) and *warehouse-small* (d-f) with infinite capacities and uniformly distributed task appearances and with limited capacities and distributed task appearances. The whiskers show the 95% confidence intervals.

take care of some task allocation, which helps to improve the action selection. Therefore, we focus on these two approaches for further evaluation.

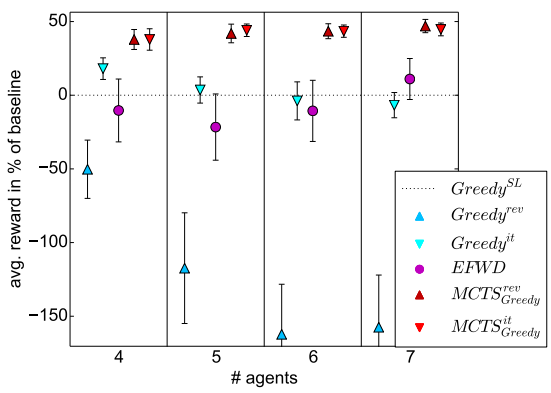
Planning times.

To investigate the planning times, we randomly sampled 200 different states and averaged the time it takes to plan in these states. We also included the state-of-the-art *EFWD* approach. The results are summarised in Figure 2d. We can see that the search times of all approaches increase roughly linearly with the number of agents. However, while the MCTS-based approaches have a very consistent planning time, the *EFWD* approach varies greatly. This is due to the k-nearest task approximation. When there is no task active, the planning time is nearly 0. However, as soon as there are one or more tasks active, the planning time increases greatly. We can also see that $MCTS^{rev}_{Greedy}$ is significantly faster than $MCTS^{it}_{Greedy}$, since it does not need to iteratively assign the tasks after the evaluation.

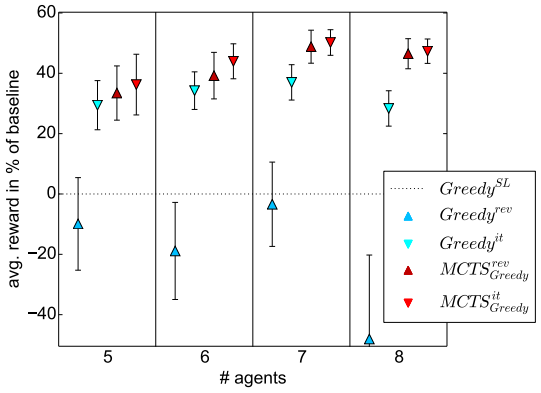
Limited vs unlimited capacities.

We have evaluated the approaches in settings where the agents have unlimited capacity and the tasks appear uniformly distributed over the warehouse. Essentially, when the agents have unlimited capacities, the depot node is obsolete, since the agents will never return to unload. Figure 3 shows the result for the *warehouse-small* environment and the *office* environment (the *office* is the same as in [9]). In comparison, with unlimited capacities (cf. Figure 3e and 3b), the resulting performances are closer to one another than with limited capacities (cf. Figure 3f and 3c). The MCTS based approaches still outperform all other approaches. The proposed heuristics without MCTS ($Greedy^{rev}$ and $Greedy^{it}$) perform as good as the *EFWD* approach. Most significantly, the $Greedy^{rev}$ policy performs almost as good as the $Greedy^{it}$ policy, which is in great contrast to the limited setting, where it yields even less reward than the $Greedy^{SL}$ baseline. This can be explained by the structure of the unlimited setting. The adaptive partitioning of $Greedy^{rev}$ works a lot better, since the agents spread due to the appearing tasks, and do not have to come back to the depot again. Thus assigning the tasks based on their locations works well. *EFWD* performs much better in the *office-world* in comparison to the *warehouse-small*. This is most likely due to the structure of the worlds. The *office-world* is much more interconnected, with almost no dead ends. Since *EFWD* does no positioning when there are no tasks present, it helps that the average shortest path length between nodes is a lot shorter in *office-world*.

To conclude, while the presented heuristics work really well in simple environments already by themselves, adding MCTS search still improves their performance. *EFWD* yields good performance in highly connected worlds.



(a) *warehouse-medium*, #nodes=66



(b) *warehouse-large*, #nodes=214

Figure 4: Comparing the different approaches in larger warehouse sizes and with different numbers of agents. The whiskers show the 95% confidence intervals.

Larger warehouses.

Additionally, we compared the different approaches in two larger sized warehouse models, *warehouse-medium* with $n = 66$ (cf. Figure 5) and *warehouse-large* with $n = 214$ (cf. Figure 1). For the large warehouse, we increased the number of simulated steps to 250 and the number of repetitions was decreased to 15.

EFWD was not able to complete any run in *warehouse-large* due excessive planning times, i.e. more than 1000 seconds for one step. The results are shown in Figure 4. *EFWD* shows a generally increasing performance for more agents, but the relative rewards against the baseline are varying. Only for larger numbers of agents it can outperform the baseline and standalone rollout strategies. We can see that the two MCTS approaches perform nearly identically for *warehouse-small* and *warehouse-medium*. The good performance of $Greedy^{it}$ in *warehouse-large* is due to the high chance that there are many tasks active in a larger world. Therefore, iteratively assigning the best tasks yields a very good result. As soon as the number of agents increases however, MCTS search improves the result again, since there are relatively fewer tasks to distribute, and positioning becomes more important again.

Positioning.

To show the effect of positioning, we let $MCTS^{rev}_{Greedy}$ run for 50 steps, while we disabled the appearance of new tasks assumed by the agents' world model. Figure 5 shows that the robots are nicely spread out. This is in stark contrast when using the heuristics without any MCTS search and also the state-of-the-art *EFWD* approach. These remain in the same position if no tasks is present.

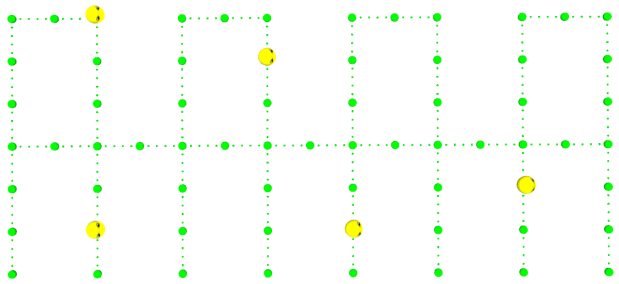


Figure 5: Positioning of $MCTS^{rev}_{Greedy}$ in the world *warehouse-medium* after 50 steps. All agents started in the top left corner.

6. CONCLUSIONS & FUTURE WORK

In this paper, we introduced a Monte Carlo Tree Search approach for the problem of multi-robot warehouse commissioning. The problem is modeled as an extended SPATAP to include the capacity constraints of robots. Our empirical evaluation shows that we can greatly improve the current state-of-the-art, while also being able to solve much larger problems.

Some possible routes for future work include tuning the node evaluation function `NV`. For instance, we can include a weighted connectivity of the nodes. More specifically, we can compute a force-field, based on the task appearance probabilities and the locations of the agents. Other possibilities are to improve the MCTS search, e.g. by introducing node priors as for instance shown in [12].

Currently, we are working on deploying the approach on real robots. We are setting up a multi-robot approach that is inspired by the RoboCup@Work [18] competition. Multiple robots have to pick up items in the environment and bring them to a common depot node. A further idea is to use our work from [8] as a local collision avoidance mechanism that allows the robots to *share* the edges and nodes on a graph. As each node represents various compartments, we intent to rely on such a local low-level conflict resolution if multiple robots have to be at the same node. Additionally, we will investigate to what extent adding large penalties for occupying the same node will mitigate the problem of sharing nodes.

Acknowledgements.

F.O. is supported by NWO Innovational Research Incentives Scheme Veni #639.021.336.

REFERENCES

[1] S. Amador, S. Okamoto, and R. Zivan. Dynamic multi-agent task allocation with spatial and temporal constraints. In *Proc. of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1384–1390, 2014.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, May 2002.

[3] R. Bjarnason, A. Fern, and P. Tadepalli. Lower bounding klondike solitaire with monte-carlo planning. In *Proc. of the International Conference on Automated Planning and Scheduling*, pages 26–33, 2009.

[4] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proc. of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210. Morgan Kaufmann, 1996.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] J. Capitán, M. T. J. Spaan, L. Merino, and A. Ollero. Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research*, 32(6):650–671, 2013.

[7] H.-L. Choi, L. Brunet, and J. P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. on Robotics*, 25(4):912–926, 2009.

[8] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen. Collision avoidance under bounded localization uncertainty. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 1192–1198, 2012.

[9] D. Claes, P. Robbel, F. A. Oliehoek, K. Tuyls, D. Hennes, and W. van der Hoek. Effective approximations for multi-robot coordination in spatially distributed tasks. In *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 881–890. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[10] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, pages 72–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[11] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proc. of the IEEE*, 94(7):1257–1270, 2006.

[12] S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *Proc. of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[13] B. P. Gerkey and M. J. Mataric. Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3862–3868. IEEE, 2003.

[14] S. Henn, S. Koch, and G. Wäscher. Order batching in order picking warehouses: a survey of solution approaches. In *Warehousing in the Global Supply Chain*, pages 105–137. Springer, 2012.

[15] B. Kartal, E. Nunes, J. Godoy, and M. Gini. Monte carlo tree search with branch and bound for multi-robot task allocation. In *The IJCAI-16 Workshop on Autonomous Mobile Service Robots*, 2016.

[16] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

[17] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin / Heidelberg, 2006.

[18] G. K. Kraetzschmar, N. Hochgeschwender, W. Nowak, F. Hegger, S. Schneider, R. Dwiputra, J. Berghofer, and R. Bischof. RoboCup@Work: Competing for the Factory of the Future. In *RoboCup Symposium*, 2014.

[19] P. Nijssen and M. H. Winands. Monte carlo tree search for the hide-and-seek game scotland yard. *IEEE Trans. on Computational Intelligence and AI in Games*, 4(4):282–294, 2012.

[20] M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[21] M. Quigley et al. ROS: An open-source Robot Operating System. In *Proc. of the Open-Source Software Workshop (ICRA)*, 2009.

[22] E. Schneider, E. I. Sklar, S. Parsons, and A. T. Özgelen. Auction-based task allocation for multi-robot teams in dynamic environments. In *Conference Towards Autonomous Robotic Systems*, pages 246–257. Springer, 2015.

[23] C. Seward. Accelerating warehouse operations with neural networks. `https://tech.zalando.de/blog/accelerating-warehouse-operations-with-neural-networks/`, 2015.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[25] J. Sleight and E. H. Durfee. A decision-theoretic characterization of organizational influences. In *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 323–330, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[26] J. Wu and E. H. Durfee. Resource-driven mission-phasing techniques for constrained agents in stochastic environments. *Journal of Artificial Intelligence Research*, 38:415–473, 2010.