Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Agile Planning for Real-World Disaster Response

Feng Wu[*] **Sarvapali D. Ramchurn**[†] **Wenchao Jiang**[‡] **Jeol E. Fischer**[‡]
Tom Rodden[‡] **Nicholas R. Jennings**[†]
[*]Computer Science and Technology, University of Science and Technology of China, Hefei, China
[†]Electronics and Computer Science, University of Southampton, Southampton, UK
[‡]Mixed Reality Lab, University of Nottingham, Nottingham, UK

Abstract

We consider a setting where an agent-based planner instructs teams of human emergency responders to perform tasks in the real world. Due to uncertainty in the environment and the inability of the planner to consider all human preferences and all attributes of the real-world, humans may reject plans computed by the agent. A naïve solution that replans given a rejection is inefficient and does not guarantee the new plan will be acceptable. Hence, we propose a new model re-planning problem using a Multi-agent Markov Decision Process that integrates potential rejections as part of the planning process and propose a novel algorithm to efficiently solve this new model. We empirically evaluate our algorithm and show that it outperforms current benchmarks. Our algorithm is also shown to perform better in pilot studies with real humans.

1 Introduction

In the aftermath of major disasters, First Responders (FRs), such as medics, security personnel and search and rescue teams, are rapidly dispatched to help save lives and infrastructure. In particular, FRs with different skills, capabilities and experience may be required for the different tasks that need to be performed (e.g., medics and volunteers to assist casualties, security personnel and fire-fighters to save buildings on fire during riots). While performing such tasks, FRs often operate in a very dynamic and uncertain environment, where, for example, fires spread, riots start, or the environment floods. Crucially, such environments require FRs to devise complex schedules for sub-teams to perform tasks across spatially distributed locations as quickly and efficiently as possible.

Given this, a number of algorithms have been developed to empower software agents to instruct humans to form teams and carry out specific tasks. For example, [Chapman *et al.*, 2009; Ramchurn *et al.*, 2010] provide both centralized and decentralized solutions to the problem of task allocation to teams of FRs with different capabilities. However, they typically assume that the environment is completely *deterministic* (e.g., task deadlines are known and humans move at a fixed speed) and that humans *always* obey instructions. Thus, humans may be expected to team up with people they have never worked or will always be fit to work on all possible tasks. However, as pointed out by [Moran *et al.*, 2013], such assumptions simply do not hold in reality. The environment is typically prone to significant uncertainties and humans may reject plans suggested by a software agent if they are tired or prefer to work with specific partners. Now, a naïve solution to this would involve re-planning every time a rejection is received. However, this may instead result in a high computational cost (as a whole new plan needs to be computed for the whole team), may generate a plan that is still not acceptable, and, following multiple rejection/replanning cycles (as all individual team members need to accept the new plan), may lead the teams to suboptimal solutions.

Against this background, we propose a model for agile planning, where plans sent by a software agent to humans may be rejected. Humans may do so because they are tired, or more importantly, because they prefer to stick to specific team members. Specifically, inspired by results from the DARPA Coordinators programme [Musliner *et al.*, 2006; Maheswaran *et al.*, 2008] we build upon the framework of Multi-agent Markov Decision Processes (MMDP) [Boutilier, 1996] to integrate the notion of plan rejection as one of the possible failures that may happen when actions are performed by the actors in the system. Thus, if the humans reject a plan, the system simply transitions to another state and returns a new plan. Therefore, re-planning is an *implicit* process whereby a new plan is selected by a pre-computed policy leant from our model. Now, to compute such a policy, we develop an algorithm using Monte-Carlo Tree Search (MCTS), and in particular, the UCT method [Kocsis and Szepesvári, 2006] due to its scalability to large problems. In particular, we show that the UCT method is inefficient for our problem as it can very easily get stuck in local optima due to rejections. To address this, our algorithm applies a Two-Pass Planning (TPP) process where, in the first pass, we compute the best policy for the underlying MMDP without rejections, and, in the second pass, we handle the rejections using the policy computed by the first pass. By so doing, we decouple the normal states (where humans are allocated tasks) from the rejection states (where they await new plans).

In more detail, this paper advances the state of the art and contribute to the community in the following ways:

1. We develop a new model for task allocation with rejections, k-RMMDPs, that is informed from real-world

field trials of planning agents [Wagner *et al.*, 2004; Fischer *et al.*, 2014; Maheswaran *et al.*, 2008]. Our model is the first to explicitly capture rejections as part of the planning process.

2. We develop a novel algorithm to solve k-RMMDPs. Specifically, we show how our TPP algorithm allows us to cope with the increased state-space induced by the k-RMMDP model (compared to pure MMDPs).
3. We empirically evaluate our TPP algorithm in simulations and show that it outperforms the state-of-the art. Crucially, we deploy our algorithm in a *real-world* pilot study, shown to improve performance significantly.

The rest of the paper is organized as follows. We first briefly review the related work and provide basic definitions. Then, we propose the k-RMMDP model and our algorithm to solve this model as well as a method to reduce the search space. We then report the simulation results of our algorithm and the results from our pilot studies.

2 Related Work

The problem of planning under uncertainty is usually modeled by Markov Decision Process (MDP). In the presence of multiple agents, the MDP model has been extended to Multi-agent MDP (MMDP) with joint action space. In the past decades, many algorithms have been proposed to solve large MDPs such as RTDP [Barto *et al.*, 1995], LAO* [Hansen and Zilberstein, 2001], and UCT [Kocsis and Szepesvári, 2006]. However, they all assume that the agents completely follow the policy during execution time. With agents rejecting plans, plan repair [Fox *et al.*, 2006] or strengthening [Hiatt *et al.*, 2009] may be useful for deterministic domains but very difficult for MDPs because the best action for one state may depends on the actions selected in all the other states. Our idea is to pre-plan for the rejections, motivated by fault-tolerant planning [Jensen *et al.*, 2004]. This is nontrivial and the key challenges that we try to address here are: 1) *how to define the rejection model for a team of agents* and 2) *how to handle the huge augmented state space given rejections*.

Among the early effort in the DARPA Coordinators program, Musliner *et al.* [2006] is most related to our work, where they also use MMDPs to compute a joint policy for human participants. However, they require the transition probabilities being fully specified (they use the Bellman backup for *informed unrolling*), which is difficult for our problem that involves complex physical processes (e.g., spreading of radioactive cloud). Furthermore, they do not model people's preferences and therefore do not explicitly handle rejections. This is crucial especially when some tasks may be harmful to the participants (e.g., entering radioactive cloud). Therefore, novel solutions are needed to tackle our problem.

3 Formal Model

We consider the problem where a team of FRs (e.g., medics, security personnel, logistics experts, or search and rescue teams) need to perform a number of tasks (e.g., save casualties, control access to resources) in the aftermath of a disaster. For example, rescuing a victim of the disaster may require a search and rescue team to dig out a victim from under the rubble and medics to provide life support, while moving food reserves may require security personnel to protect the trucks provided by logistics experts. Hence, the FRs need to decide on the *sequence of actions* to execute given that the tasks are located in different parts of the disaster space and where the effect of their actions may be liable to some *uncertainty* (e.g., due to fires spreading and preventing access to targets, or the FRs getting tired and taking longer to complete the tasks).

Given the complexity of computing the allocation of tasks to FRs, such a problem is typically given to a planning agent that can advise the human FRs on what to do. To this end, we develop a system that such an agent can run, by computationally modeling the behavior of FRs in terms of the actions they take and the teams they form to complete their tasks. Hence, in what follows, we model this problem using a *Multi-Agent Markov Decision Process* (MMDP). Moreover, given that the planner agent may not be able to model all aspects of the real-world, its plans may be rejected (as shown by [Maheswaran *et al.*, 2008; Moran *et al.*, 2013; Wagner *et al.*, 2004]), and therefore, we also describe how the rejection of plans can be modeled.

3.1 Task Allocation under Uncertainty

Formally, an MMDP is defined as tuple $\langle I, S, \{A_i\}, T, R, s^0, \gamma \rangle$, where: $I = \{1, 2, \cdots, n\}$ is the set of n FRs as described above; S is a set of system states (e.g., where the FRs are positioned, their current task); A_i is the action set of FR $i \in I$; $T : S \times \vec{A} \times S \rightarrow [0, 1]$ is the transition function where $\vec{A} = \times_{i \in I} A_i$ is the set of joint actions; $R : S \times \vec{A} \rightarrow \Re$ is the reward function (e.g., the level of completion of a rescue mission or the time it takes to distribute vital resources); $s^0 \in S$ is the initial state; and $\gamma \in (0, 1]$ is the discount factor as in the standard MMDPs (γ is 0.95 in our experiments). Here, an action $a_i \in A_i$ is what an FR can do in one step in a fixed amount of time so all FRs complete their actions at the same time as commonly assumed [Musliner *et al.*, 2006]. Tasks with long duration can be accommodated by adding extra states (e.g., states about tasks in process) and the FRs concerned repeat their action until the task is completed. The goal of solving MMDPs is to find the optimal policy that maximizes the number of completed tasks with minimum costs.

3.2 The Rejection Model

As aforementioned, human FRs tend to reject plans when they may find tasks too hard (e.g., too far if they are tired, or not meeting their capabilities) or prefer to pair up with key partners. Hence, we define a model to capture such rejections.

Definition 1. A rejection model $\Delta_N : S \times \vec{A} \rightarrow [0, 1]$ for a team of FRs $N \in 2^I$ is defined as a probability distribution given states and joint actions, where $\Delta_N(s, \vec{a})$ specifies the probability of all FRs in N rejecting action $\vec{a}$ while the other FRs that are not in N accept $\vec{a}$ in state s.

In particular, $\Delta_\emptyset(s, \vec{a})$ is the probability that all FRs accept action $\vec{a}$ in state s (i.e., no FRs reject $\vec{a}$) while $\Delta_I(s, \vec{a})$ is the probability that all FRs reject $\vec{a}$ in s. Thus, the sum of all the rejection probabilities $\forall s, \vec{a}$: $\sum_{N \in 2^I} \Delta_N(s, \vec{a}) = 1$ because $\vec{a}$ may either be accepted or rejected by some of the FRs. For example, given a team of three FRs, $I = \{1, 2, 3\}$, all possible cases are: $\forall N \in \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$, in which $\Delta_{\{1,3\}}(s, \vec{a})$ denotes the probability that FRs 1 and 3 jointly reject $\vec{a}$ while FR 2 solely accept it in state s. In other words, our rejection model is defined over sub-teams of the FRs. By so doing, we model the rejection behavior of the whole team instead of a single FR because our goal is to generate plans that are acceptable for all the FRs. Although the rejection model has many parameters, it is often very *sparse* in practice because FRs reject a plan only when necessary.

Here, we assume that each FR has only a *bounded* number of chances to reject a plan in a single decision step (based on the observations in [Ramchurn *et al.*, 2015], humans do not reject plans infinitely many times). In other words, we only allow each FR k rounds in a decision step to reject the plan and request a new plan. The upper limit k is necessary to ensure an agreement can be reached by all the FRs during a limited amount of time. Given these assumptions, we next extend the MMDP to consider k rejections in the model.

4 The k-RMMDP Model

A *k-rejection MMDP* (k-RMMDP) has the same action space $\vec{A}$ as the MMDP but has an augmented state space $\mathbb{S}$ that additionally captures rejection events, a transition function $\mathbb{T}$ that incorporates our rejection model, and a reward function $\mathbb{R}$ that captures the cost of rejections as described below.

The State Space. $\mathbb{S}$ is defined as follows:

$$(s, \mathbf{j}) \in S \times \{1, 2, \cdots, k\}_1 \times \cdots \times \{1, 2, \cdots, k\}_n$$

where $s \in S$ is a state of the underlying MMDP model and $\mathbf{j} = (j_1, \cdots, j_n)$ is the joint rejection count of the FRs with $0 \le j_i \le k$ the rejection count for FR i. Here, $(s, \mathbf{j}) \in \mathbb{S}$ is called a *rejection state* with $\mathbf{j} \neq \mathbf{0}$ where $\mathbf{0} = (0, 0, \cdots, 0)$ means no rejections, and $(s, \mathbf{0}) \in \mathbb{S}$ is called a *normal state*.

The Transition Function. $\mathbb{T}$ is defined as follows:

1. $\forall s, \vec{a}, s', \mathbf{0} \le \mathbf{j} < \mathbf{k} : \mathbb{T}((s', \mathbf{0})|(s, \mathbf{j}), \vec{a}) = \Delta_{\emptyset}(s, \vec{a}) \cdot T(s'|s, \vec{a})$ where $\Delta_{\emptyset}(s, \vec{a})$ is the probability that joint action $\vec{a}$ is accepted by all the FRs in state s (No rejection is raised by any FRs). In this case, state s transitions to the next state s' according to the transition function T, and the joint rejection count $\mathbf{j}$ is reset to $\mathbf{0}$.
2. $\forall s, \vec{a}, \mathbf{j}, N \in 2^I \setminus \{\emptyset\}$ and $\forall i \in N, j_i \in \mathbf{j}, 0 \le j_i < k$: $\mathbb{T}((s, \mathbf{j} + \mathbf{1}_N)|(s, \mathbf{j}), \vec{a}) = \Delta_N(s, \vec{a})$ where N is the set of FRs who reject the joint action $\vec{a}$ and $\mathbf{1}_N = \langle \mathbf{1}_N(i) \rangle$ is a length-n 0-1 vector with $\mathbf{1}_N(i) = 1$ if $i \in N$ otherwise 0. When $\vec{a}$ is rejected by the FRs in N, rejection count $\mathbf{j}$ increases by 1 while system state s remains unchanged. If an FR has reached k rejections, it is excluded in this case due to: $\forall i \in N, j_i \in \mathbf{j} : 0 \le j_i < k$.
3. $\forall s, \vec{a}, s' : \mathbb{T}((s', \mathbf{0})|(s, \mathbf{k}), \vec{a}) = T(s'|s, \vec{a})$ where $\mathbf{k} = \langle k, k, \cdots, k \rangle$ is the joint rejection count meaning that all the FRs reach the limit so that joint action $\vec{a}$ is executed and system state s transitions to the next state s'.
4. $\forall s, \vec{a}, \mathbf{j}, s', \mathbf{j}' : \mathbb{T}((s', \mathbf{j}')|(s, \mathbf{j}), \vec{a}) = 0$ otherwise.

To sum up, if not all FRs reach the k rejection limit (i.e., $\mathbf{j} \neq \mathbf{k}$), the FRs can either accept $\vec{a}$ and the state transits from $(s, \mathbf{j})$ to $\forall s' \in S, (s', \mathbf{0})$, or reject $\vec{a}$ by $\forall N \in 2^I \setminus \{\emptyset\}$ and the system state s remains unchanged while the rejection counts increase from $\mathbf{j}$ to $\mathbf{j} + \mathbf{1}_N$. Otherwise (i.e., $\mathbf{j} = \mathbf{k}$), the FRs must accept $\vec{a}$ and the state transits from $(s, \mathbf{k})$ to $\forall s' \in S, (s', \mathbf{0})$. In our pilot studies, we observed that rejections can be resolved in a limited amount of time, during which we assume that the state remains unchanged.

The Reward Function. We assign a cost to each rejection so that the policy computed by the planner minimizes the number of rejections of the FRs (i.e., it attempts to find the most acceptable plan). Specifically, we define the reward function $\mathbb{R}$ of the k-RMMDP as $\forall s, \vec{a}, \mathbf{j} : \mathbb{R}((s, \mathbf{j}), \vec{a}) = R(s, \vec{a})$ if $\mathbf{j} = \mathbf{0}$ and $-C$ otherwise where $C \in \Re^+$ is the rejection cost specifying how costly a rejection is to the team performance. By so doing, the FRs are rewarded with the original reward function only when the joint action is accepted by all the FRs and penalized with a fixed value C when any rejection is raised. Indeed, the choice of the rejection cost depends on the specific domain and our model can accommodate more complex functions. The goal of solving k-RMMDP is to find the best policy that maximizes the reward of the underlying MMDP with minimum rejections.

Now, the policy π for the k-RMMDP is a mapping from augmented states $(s, \mathbf{j})$ to joint actions. At execution time, a joint action $\vec{a}$ is selected by $\pi(s, \mathbf{0})$ in state s. If no rejection is raised, $\vec{a}$ is executed by the FRs. Otherwise, another joint action is selected by $\pi(s, \mathbf{j})$. This process terminates until a joint action is accepted or all the FRs reach the k limits.

Complexity and Analysis. It is worth noting that our model is a variation of the MMDP where the rejection model is *explicitly* represented. In other words, k-RMMDP is a special case of MMDPs with the states, the transition function, and the reward function of considering rejections. The rejection model is necessary to specify when a plan may be rejected by some FRs during planning time. Comparing to the pure MMDP representation, the main advantage of having such explicit representation is that we are able to exploit the problem structure (as detailed next) and develop planning algorithms to more efficiently solve our problem.

Specifically, for each system state $s \in S$, there are $(k^n - 1)$ rejection states. Totally, there are $|S| \cdot (k^n - 1)$ rejection states. Therefore, the total number of states in k-RMMDPs is $\mathcal{O}(|S| \cdot k^n)$ (rejection states plus normal states), much larger than the state size of the underlying MMDP (i.e., $|S|$). For example, if the rejection limit is $k = 3$ and the number of FRs is $n = 8$, the state size of k-MMDPs is $|S| \cdot 3^8 = |S| \cdot 6561 \gg |S|$. Note that this complexity result is directly inherent from the setting that each FR can independently reject plans k times.

Given the huge state space, it is computationally intractable for optimal MDP algorithms (e.g., VI, PI, LP, etc.) to solve k-RMMDPs because they will quickly run out of time or space given that $|S|$ is already very large. Anytime algorithms such as RTDP [Barto *et al.*, 1995] and UCT [Kocsis and Szepesvári, 2006] can handle large state space but will easily get trapped in local optima where very long sequences of pure rejection states such as $(s, \mathbf{0}) \to \cdots \to (s, \mathbf{k})$ are

Algorithm 1: Two-Pass UCT Planning

```
Input: The $k$-RMMDP Model: $\mathcal{M}$, The Initial State: $(s^0, \mathbf{0})$.
// The first pass:
//   Compute the policy for $\forall s, (s, \mathbf{0})$.
Solve the underlying MMDP of $\mathcal{M}$ and build a search tree:
    Run UCT on the underlying MMDP: $\langle S, \vec{A}, T, R \rangle$
    Update all the $Q$-values: $Q(s, \vec{a}) \leftarrow Q(s, \vec{a}) \cdot \Delta_{\emptyset}(s, \vec{a})$
// The second pass:
//   Compute the policy for $\forall s, \mathbf{j} \neq \mathbf{0}, (s, \mathbf{j})$.
foreach state node $(s, \mathbf{0})$ in the tree do
    Create a sub MMDP:
        Initial state: $(s, \mathbf{0})$
        Normal states: $\forall \mathbf{j}, (s, \mathbf{j})$
        Terminal states: $\forall s', (s', \mathbf{0})$
        Actions: $\vec{A}$, Transition: $\mathbb{T}$, Reward: $\mathbb{R}$
    Run UCT on the sub MMDP:
        Build a subtree with states $\forall \mathbf{j}, (s, \mathbf{j})$
        Propagate values of $\forall s', (s', \mathbf{0})$ to the subtree
        Propagate values of the subtree to node $(s, \mathbf{0})$
    Update all the $Q$-values of $(s, \mathbf{0})$'s ancestors in the tree
// The policy:  $\pi(s^0, \mathbf{j}) = \arg\max_{\vec{a}} Q((s^0, \mathbf{j}), \vec{a})$.
return the policy computed with $Q$-values in the search tree
```

unfolded and evaluated. Note that the process of completing tasks requires transitions among normal states. However, the number of normal states is only a fraction of the overall states, i.e., $|S|/(|S| \cdot k^n) = 1/k^n$ (it is about 0.0001 for the aforementioned example). Therefore, it is very inefficient for the state-of-the-art MDP approaches to find a good policy in such a huge state space (as shown in our experiments).

Having defined k-RMMDP, we next proceed to describe our algorithm to compute the policy.

5 Solving the k-RMMDP Model

In this section, we first provide a brief overview of MCTS as the building block of our algorithm and then go on to specify the TPP algorithm that we apply to address the challenge of huge state space and solve k-RMMDPs.

5.1 Monte-Carlo Tree Search

Given that k-RMMDP augments MMDP, the solutions that apply to the MMDP framework will also apply to our framework. Here, we focus on MCTS as it has been shown to scale to large problem domains [Gelly and Silver, 2011] and is also an *anytime* algorithm (which makes it particularly attractive when fast response times are required in deployments). Moreover, it can compute the policies using only simulations of future events (a.k.a. *generative model*). This is particularly useful for our problem where the transition between states is very difficult to model explicitly. For example, in our experiments, we have 8 FRs and 17 tasks in a 50×55 grid. The total number of possible system states is more than 2×10^{400}. Hence, it is intractable to represent the transition and reward functions in a tabular form. It is also difficult to have a compact representation because the state transition involves complex physical processes (e.g., the spreading of radioactive cloud). Fortunately, it is fairly straightforward to develop a simulator (i.e., generative model) that takes a state as input and (stochastically) outputs the next state given the joint actions of the FRs as what we did for our problem.

Although LRTDP [Bonet and Geffner, 2003] and Anytime AO* [Bonet and Geffner, 2012] are also anytime algorithms, both of them require the *transition function* (not available in our problem). Therefore, MCTS is more suitable for our domain as it only relies on *simulations*. In MCTS, the key operation is the action selection in each state node of the search tree. One common choice to implement it is using the UCB1 [Auer *et al.*, 2002] heuristic. Here, MCTS with UCB1 is usually called UCT [Kocsis and Szepesvári, 2006]. In other words, UCT is the leading implementation of MCTS using the UCB1 heuristic to balance the exploration and exploitation. Although UCT can converge to the optimal solution given sufficient amount of simulations, directly applying UCT to solve k-RMMDPs turns out to be very inefficient (may require huge number of simulations) because the state space of k-RMMDPs is very large (see discussion in the previous section). Therefore, the planner can easily get trapped in local optima when all the actions are rejected by the FRs. Hence, we need a procedure to improve the search.

5.2 Two-Pass UCT Planning

We propose the TPP (Two-Pass Planning) algorithm to more efficiently solve k-RMMDPs. Our key observation is that the Q-value function of each state-action pair can be recursively split into two components given the state structure as:

$$\begin{aligned}
Q((s,\mathbf{j}),\vec{a}) &= \mathbb{R}((s,\mathbf{j}),\vec{a}) + \sum_{s',\,\mathbf{j}'} \mathbb{T}((s',\mathbf{j}')|(s,\mathbf{j}),\vec{a})V(s',\mathbf{j}') \\
&= \mathbb{R}((s,\mathbf{j}),\vec{a}) + \sum_{s' \in S} \mathbb{T}((s',\mathbf{0})|(s,\mathbf{j}),\vec{a})V(s',\mathbf{0}) \\
&\quad + \sum_{\mathbf{j} \neq \mathbf{k}, N \in 2^I \setminus \emptyset} \mathbb{T}((s,\mathbf{j}+\mathbf{1}_N)|(s,\mathbf{j}),\vec{a})V(s,\mathbf{j}+\mathbf{1}_N) \\
&= \mathbb{R}((s,\mathbf{j}),\vec{a}) + \Delta_{\emptyset}(s,\vec{a}) \sum_{s' \in S} T(s'|s,\vec{a})V(s',\mathbf{0}) \\
&\quad + \sum_{\mathbf{j} \neq \mathbf{k}, N \in 2^I \setminus \emptyset} \Delta_N(s,\vec{a})V(s,\mathbf{j}+\mathbf{1}_N)
\end{aligned}$$

where $V(s',\mathbf{j}') = \max_{\vec{a} \in \vec{A}} Q((s',\mathbf{j}'),\vec{a})$ is the value function. Here, the first component specifies the transitions to the normal states $\forall s' \in S, (s', \mathbf{0})$ and the second component is for the transitions to the rejection states $\forall N \in 2^I \setminus \emptyset, (s, \mathbf{j} + \mathbf{1}_N)$. To exploit this structure, we solve k-RMMDPs by first focusing on the transitions only to $\forall_{s' \in S}(s', \mathbf{0})$ and then considering the transitions to the rejection states. The main procedure of our TPP method is outlined in Algorithm 1.

In the first pass, we compute the policies and build a search tree without considering the rejection states. In other words, we only consider the normal states $\forall_{s \in S}, (s, \mathbf{0})$ in this pass. This is equivalent to solving the underlying MMDP, which can be done by UCT. The outcome of this pass is a search tree with state nodes and action nodes. Intuitively, this pass computes the best policy in the case that no FR rejects the selected actions. From the system's point of view, this policy is the most efficient one for the team of FRs to complete the

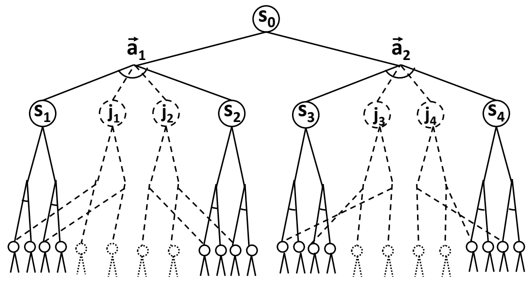


Figure 1: Search tree built by the TPP algorithm.

tasks (ignoring human preferences to stick to certain teams or to avoid certain tasks). Moreover, ignoring the rejection states makes the UCT search deeper in the tree and, hence, allows it to better explore the policy space. In this way, we avoid local optima caused by the rejection states (i.e., very long sequences of rejections with few tasks being completed).

In the second pass, we compute the policy for the rejection states. Specifically, we traverse the search tree in reverse order, that is, from the leaves to the root, and expand each action node along the path with the rejection states by drawing from the rejection model. In other words, we consider the case when the action is rejected by the agents and the state transitions from $(s, \mathbf{0})$ to $(s, \mathbf{j})$ where the joint rejection count $\mathbf{j} \neq \mathbf{0}$. To expand each action node, we create a sub MMDP for each node $(s, \mathbf{0})$ in the search tree as follows. This sub MMDP starts with state $(s, \mathbf{0})$. Given a joint action $\vec{a}$, it transitions to another state $(s, \mathbf{j})$ where $\mathbf{j} \neq \mathbf{0}$ if $\vec{a}$ is rejected by the agents. It terminates with state $(s', \mathbf{0})$ where $s' \in S$ if $\vec{a}$ is accepted. Notice that the joint action space $\vec{A}$, the transition function $\mathbb{T}$, and the reward function $\mathbb{R}$ of the sub MMDP are identical to the original k-RMMDP model.

Each sub MMDP is also solved by UCT. Starting with $(s, \mathbf{0})$, we build a subtree with joint actions and rejection states. The leaf nodes of the subtree are terminal states $(s', \mathbf{0})$ where $s' \in S$. Note that we do not need to expand the terminal states in the subtree. Instead, we link them to the nodes in the search tree computed by the first pass. Then, the Q-values of the nodes $(s', \mathbf{0})$ are propagated to the subtree. Intuitively, these Q-values are the future rewards after the joint action is *accepted* by the FRs. By so doing, we limit the search space in the second pass only to the rejection states. If $(s', \mathbf{0})$ is not sampled in the first pass, we recursively expand it by UCT until the leaf nodes are all in the search tree or the maximal search depth is reached. We also propagate the Q-values of the subtree back to the search tree computed in the first pass and update the Q-values of the search tree. The goal is to bias the Q-values of $(s, \mathbf{0})$ so that the joint action that is preferable to the FRs has a larger value than the other joint actions that have a higher risk of being rejected by the FRs.

Figure 1 graphically illustrates the search tree of our TPP algorithm where the tree nodes with solid lines are generated in the first pass while the nodes with dotted lines are expanded in the second pass. As seen from the figure, the search tree interleaves state nodes with action nodes. In the first pass, the state node $(s_0, \mathbf{0})$ (the rejections count $\mathbf{0}$ is omitted in the graph) is expanded with two action nodes $\vec{a}_1$ and $\vec{a}_2$ and then transits to states $(s_1, \mathbf{0}), (s_2, \mathbf{0})$ and $(s_3, \mathbf{0}), (s_4, \mathbf{0})$ respectively assuming $\vec{a}_1$ and $\vec{a}_2$ are accepted by all FRs. This process repeats for the newly expanded state nodes until the maximal search depth is reached. This is equivalent to solving the underlying MMDP with UCT. In the second pass, each action node is expanded with the rejection states considering the case that the action may be rejected by some FRs. For example, two nodes of rejection states $(s_0, \mathbf{j}_1), (s_0, \mathbf{j}_2)$ (the system state s_0 is omitted in the graph) are appended to action node $\vec{a}_1$ depending on who reject $\vec{a}_1$. Then, $(s_0, \mathbf{j}_1), (s_0, \mathbf{j}_2)$ are expanded with new action nodes. This is equivalent to proposing new actions to the FRs after they reject $\vec{a}_1$. If the new actions are accepted, the state $(s_0, \mathbf{0})$ transits to new normal states so the subtree with dotted lines is linked with some state nodes generated in the first pass (if such nodes do not exist, new nodes are created and expanded as in the first pass). If the new actions are also rejected, new nodes of rejection states are generated and the whole process repeats down to the maximal depth. After that, the values propagate from the leaf nodes up to nodes $(s_0, \mathbf{j}_1), (s_0, \mathbf{j}_2)$ and then to node $\vec{a}_1$ and node $(s_0, \mathbf{0})$ with their values being updated accordingly. Finally, the best policy is computed based on the Q-values.

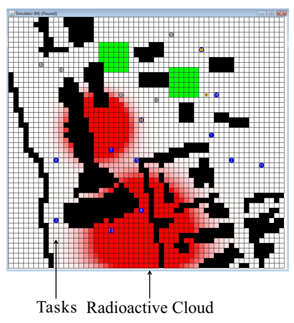
6 Empirical Evaluation

We consider a disaster scenario in which a satellite, powered by radioactive fuel, has crashed in a sub-urban area. Debris is strewn around a large area, damaging buildings and causing accidents and injuring civilians. Moreover, radioactive particles discharged from the debris are gradually spreading over the area, threatening to contaminate food reserves and people. Hence, emergency services are deployed to evacuate the casualties and key assets before they are engulfed by the radioactive cloud. To test the performance of our algorithm, we built an MMDP simulator for this scenario. As shown in Figure 2(a), there are 8 FRs and 17 response tasks located in a 50×55 grid (i.e., the map). The FRs are assigned a specific role as: medic, fire-fighter, soldier, or transporter. Their mission is to evacuate all four types of targets as: victim (requires medic and fire-fighter), animal (requires medic and transporter), fuel (requires soldier and fire-fighter), or other resource (requires soldier and transporter).

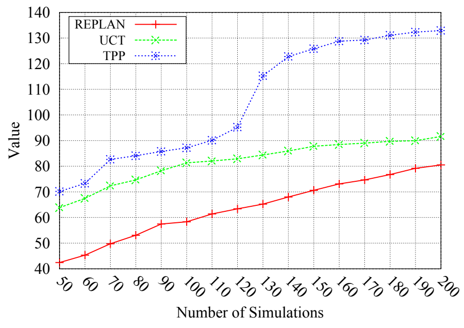
The state contains information about the FRs' locations in the grid and their health levels, the status of all tasks (i.e., todo, working, and completed), and the coverage of the radioactive cloud shown as the red region in Figure 2(a). At every time step, each FR can move to a neighboring cell or stay in the current cell and the radioactive cloud spreads stochastically in the grid (see [Ramchurn *et al.*, 2015] for more detail). A task must be completed by FRs with the required roles before it is contaminated by the radioactive cloud. The FRs get a reward if a task is completed. If a FR enters the radioactive cloud, her health level will drop based on the dose received from the cloud and is "killed" if her health level is 0.

6.1 Simulation Results

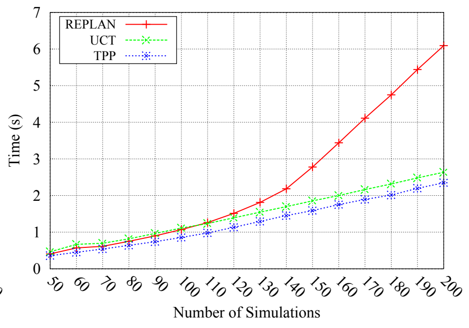
We benchmark our algorithm against several state-of-the-art methods. Specifically, we compared our TPP algorithm with two baselines where: REPLAN is the replanning approach



(a) Simulator



(b) Values



(c) Runtime

Figure 2: Simulation of disaster response scenario.

for the underlying MMDP (It tries replanning when a plan is rejected); UCT is the flat UCT planner using our k-RMMDP model. As aforementioned, our transition model is not available in a closed form and therefore most of existing algorithms cannot solve our problem. All the three algorithms were run online and called only when the policy for a given state was not computed. By so doing, we avoided computing the policy for all possible states off-line, which is intractable for our problem. We tested the problem instance with each algorithm 1000 times and report the average results. Specifically, we report the average value and runtime of each algorithm as the key performance metrics, where the values are the accumulated rewards achieved using the policies computed by the algorithms and the runtime is the planning time of each algorithm per decision step. In the experiments, we initialize the rejection model by randomly generating the preference of each FR and set the discount factor $\gamma = 0.95$, the rejection limit $k = 3$, and the rejection cost $C = 1.0$.

Figure 2(b) and 2(c) summarize our results with different numbers of simulations from 50 to 200. As shown in Figure 2(b), TPP outperforms UCT and REPLAN with higher values. This indicates that TPP produces better policies that can complete more tasks and get fewer rejections from FRs. The gaps between TPP and REPLAN are always large ($\geq \frac{28.3}{170} \approx 16.6\%$ improvement[1]) regardless of the number of simulations. This is because TPP explicitly considers rejections in the k-RMMDP model while REPLAN simply generates another plan when a plan is rejected. Moreover, as shown in Figure 2(c), REPLAN takes significantly more time than TPP because replanning is very costly, especially when more simulations (e.g., 200) are used. This further justifies our k-RMMDP model. Given small number of simulations (< 100), TPP performs slightly better than UCT. However, when the number of simulations is sufficiently large (e.g., 200), TPP outperforms UCT by more than $\frac{52.4}{170} \approx 30.8\%$ improvement. We can also see from Figure 2(b) that TPP nearly converges at 200. This also shows that TPP converges faster than UCT with fewer simulations.

Apart from the results shown in the figure, we also ran UCT on the same problem instance with 10000 simulations for 1000 times and got the averaged value of 133.76, which is very close to what TPP achieved with 200 simulations. Note that it has been proved that UCT will converge to the optimal policy with sufficient simulations [Kocsis and Szepesvári, 2006]. Therefore, this value can be viewed as the approximate upper-bound value for TPP. In terms of runtime, as shown in Figure 2(c), REPLAN took much more time for replanning as expected. Surprisingly, TPP did not take more time than UCT though we call UCT twice in TPP. This is because TPP found the best plan that is acceptable for all FRs faster without extensively expanding the rejection states, as what may happen in UCT. Therefore, the time required by the second pass of TPP is minimal given the search tree computed in the first pass. In summary, our simulation results demonstrate that: 1) explicitly modeling rejections using our k-MMDP model is a better solution than replanning; 2) when the number of simulations is sufficient, our TPP algorithm outperforms UCT with much better value but takes less time.

6.2 Pilot Studies

We deployed our algorithm in a real-world disaster response simulation exercise, whereby a planner agent instructs human participants, acting as FRs, to complete a number of tasks. The scenario was similar to our simulation except that we tested our system on people. Specifically, the planner agent takes the game status (i.e., positions of FRs, known status of the cloud, and messages received from FRs) as input and produces a plan for each FR for the current state. Once a plan is computed, the plan is split into individual task allocations and sends them to each FRs. In more detail, 2 pilot runs of our planner agent, one with REPLAN and the other with TPP, were carried out with distinct sets of 8 participants in each run lasting 30 minutes (i.e., a deadline). A video of our pilot runs can be viewed at: `http://bit.ly/1ebNYty`.

During the trials, as participants, and as tasks were completed, the planner (that knows where the tasks are and has estimates of the environmental hazard) instructed the players to team up with specific partners and complete tasks in the simulated disaster space [Fischer *et al.*, 2014]. Crucially, a mobile app allowed them to reject the plan suggested by the planner agent. As there is no well defined rejection model for humans, we trained the rejection model with the data from our previous trials and update our model online based on the acceptance and rejection of plans (i.e., the FRs' preferences)

[1]The maximal value of the problem with 17 tasks is 170.

as they are received. Specifically, we observed from the previous trials that all of the rejections happened when following the allocation would have split existing teams, or instructed players to team up with physically more distant responders. Other than the distances, FRs may reject a plan when they think the assigned tasks are too risky for them. They also tend to reject a plan when they are not in good health condition.

In the field trial with REPLAN, it was found that participants rejected plans many more times than for TPP (8 times for REPLAN and none for TPP). Moreover, it was found that, with REPLAN, the participants only completed 75% of the tasks in 19 mins while with TPP, they completed 80% of the tasks in 17 mins (i.e., more tasks faster than REPLAN). Practically, *none* of the participants had depleted any of their health points in the run with TPP while, with REPLAN, the average health points was 80%. This demonstrates the benefit of explicitly considering the FRs' rejections. While these pilot studies are not intended to be statistically conclusive, they do indicate that the plans computed by TPP (that tends to keep existing teams together and prevent rejections) allowed FRs to form more effective teams that implement (possibly better and more importantly safer) plans faster. Crucially, they validate our approach in the real-world.

7 Conclusions

In this paper, we defined k-RMMDP, a novel model of task allocation for emergency responders, that is informed by real-world field trials that showed that humans may reject plans computed by a planner agent. Our model is the first to implicitly consider such rejections. Moreover, we provide a two-pass algorithm, based on UCT, to improve the search for a solution in a large policy space. Finally, we both simulate our algorithm and deploy it in pilot studies to demonstrate how it significantly outperforms the standard replanning approach as well as the naïve UCT method and the benefit of explicitly modeling FRs' preferences when generating plans for people in the real-world. By so doing, we establish a novel benchmark for agile planning. Future work will look at running more field trials to further investigate the effectiveness of our planning algorithm and develop more effective models and methods to learn the preferences from human input.

Acknowledgments

Feng Wu was supported in part by the Fundamental Research Funds for the Central Universities and the Youth Innovation Promotion Association, CAS.

References

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi-armed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[Barto *et al.*, 1995] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Bonet and Geffner, 2003] B. Bonet and H Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. of ICAPS*, pages 12–21, 2003.

[Bonet and Geffner, 2012] Blai Bonet and Hector Geffner. Action selection for MDPs: Anytime AO* vs. UCT. In *Proc. of AAAI*, pages 1749–1755, 2012.

[Boutilier, 1996] Craig Boutilier. Planning, learning and coordination in multi-agent decision processes. In *Proc. of TARK*, pages 195–210, 1996.

[Chapman *et al.*, 2009] A. Chapman, R. A. Micillo, R. Kota, and N. R. Jennings. Decentralised dynamic task allocation: A practical game-theoretic approach. In *Proc. of AAMAS*, pages 915–922, 2009.

[Fischer *et al.*, 2014] Joel E Fischer, Wenchao Jiang, Andruid Kerne, Chris Greenhalgh, Sarvapali D Ramchurn, Steven Reece, Nadia Pantidi, and Tom Rodden. Supporting team coordination on the ground: Requirements from a mixed reality game. In *Proc. of COOP*, pages 49–67, 2014.

[Fox *et al.*, 2006] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS*, volume 6, pages 212–221, 2006.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[Hiatt *et al.*, 2009] L. M. Hiatt, T. L. Zimmermanz, S. F. Smithz, and R. Simmons. Strengthening schedules through uncertainty analysis. In *Proc. of IJCAI*, pages 175–180, 2009.

[Jensen *et al.*, 2004] Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic nondeterministic planning. In *Proc. of ICAPS*, pages 335–344, 2004.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proc. of ECML*, pages 282–293, 2006.

[Maheswaran *et al.*, 2008] Rajiv T. Maheswaran, Pedro Szekely, G. Gati, J. Jin, K. Smyth, and C. Vanbuskirk. Predictability and criticality metrics for coordination in complex environments. In *Proc. of AAMAS*, pages 647–654, 2008.

[Moran *et al.*, 2013] Stuart Moran, Nadia Pantidi, Khaled Bachour, Joel E. Fischer, Martin Flintham, Tom Rodden, Simon Evans, and Simon Johnson. Team reactions to voiced agent instructions in a pervasive game. In *Proc. of IUI*, pages 371–382, 2013.

[Musliner *et al.*, 2006] David J Musliner, Edmund H Durfee, Jianhui Wu, Dmitri A Dolgov, Robert P Goldman, and Mark S Boddy. Coordinated plan management using multiagent MDPs. In *AAAI Spring Symposium: Distributed Plan and Schedule Management*, pages 73–80, 2006.

[Ramchurn *et al.*, 2010] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *The Computer Journal*, 53(9):1447–1461, 2010.

[Ramchurn *et al.*, 2015] S. D. Ramchurn, F. Wu, J. E. Fischer, S. Reece, W. Jiang, S. J. Roberts, T. Rodden, and N. R. Jennings. Human-agent collaboration for disaster response. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 1–30, 2015.

[Wagner *et al.*, 2004] Thomas Wagner, John Phelps, Valerie Guralnik, and Ryan VanRiper. An application view of coordinators: Coordination managers for first responders. In *Proc. of AAAI*, pages 908–915, 2004.