Multiagent Task Allocation and Planning with Multi-Objective Requirements

Thomas Robinson
University of Wollongong
Wollongong, Australia
tmr463@uowmail.edu.au

Guoxin Su
University of Wollongong
Wollongong, Australia
guoxin@uow.edu.au

Minjie Zhang
University of Wollongong
Wollongong, Australia
minjie@uow.edu.au

ABSTRACT
In service robot applications, planning is often integrated with task allocation. Linear Temporal Logic (LTL) as an expressive high-level formalism is widely used for task specification, and allows for formalised restrictions on temporal sequences of tasks. In multiagent planning, a Multi-Objective Markov Decision Process extends the standard model with vector rewards capturing possibly conflicting planning objectives. Such objectives include the success rates of accomplishing individual tasks, and the cost budgets for individual agents. In this paper, we consider the problem of concurrently allocating LTL task sequences to a team of agents and calculating optimal task schedulers simultaneously, satisfying cost and probability thresholds. We reduce this problem to multi-objective scheduler synthesis for a team MDP structure, whose size is linear in the number of agents. Our preliminary experiment demonstrates the scalability of our approach.

KEYWORDS
Multiagent Planning; Multi-Objective Planning; Task Allocation

2 APPROACH
Tasks as Automata: LTL formulas \( \varphi \) over a finite set \( AP \) of atomic propositions are defined using the following grammar:

\[
\varphi ::= \top \mid a \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid X \varphi \mid \varphi_1 \lor \varphi_2
\]

where \( a \in AP \). The operators \( X \) and \( U \) stand for “next” and “until”, respectively.

It is natural to consider task allocation which completes in finite time, especially in the circumstance where we consider allocating sequences of tasks to a single agent. Task formulation which completes in finite time requires a subset of LTL formulas which are guaranteed to have a finite fragment. We then only want to consider the co-safe language \( L^c = \Sigma^* \setminus L_{safe} \), where if \( x \in \Sigma^* \) and an accepting prefix, and \( y \in \Sigma^o \) then \( x \cdot y \in L \). It has been shown that any LTL formula containing only the temporal operators \( X \) (next), \( U \) (until), an \( F \) (eventually) in positive normal form (PNF) always results in a co-safe property satisfying \( L^c \) [12]. Further, because the infinite part of a co-safe LTL formula does not affect the outcome of any sequence [12], we may define a non-deterministic finite automaton (NFA) which is always accepting for the finite sequence [2]. It is well known that deterministic finite automaton (DFA) and NFA are equally expressive so then we may construct a DFA, \( A_j = (Q_j, q_{j,0}, Q_j, F, \Sigma, \delta) \) for any co-safe LTL task formula.

Agent models: An agent with some expectation of uncertainty in the outcome of its actions is represented by a labelled Markov decision process (MDP) defined as a tuple \( M = (S, \sigma, A, P, L) \) where \( S \) is a finite state space, \( s_0 \in S \) is an initial state, \( A \) is as set of actions, \( P : S \times A \times S \rightarrow [0, 1] \) is a transition function, \( L : S \rightarrow \Sigma \) is a labelling function. We want to verify that the
We add another new state which some combination of tasks are initial or completed/rejected.

Table 1: Model size and runtime comparison between the team MDP and MAMDP in Experiment 1, where $t$ - runtime in seconds, $|S|$ - state space, $|P|$ - number of transitions, $|S'|$ - reachable states, $|P'|$ - number of reachable transitions.

| #Agents | Team MDP | | | | | | MAMDP | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 3       | 9       | 4520    | 9504    | 9       | 26091   | 25985   | 1000    | 9000    |         |
| 4       | 12      | 7488    | 21488   | 77      | 260901  | 3.3x10^6| 10000   | 120000  |         |
| 5       | 17      | 9360    | 34992   | 105     | 7627    | 2.1x10^6| 2.9x10^6| 100000  | 1.5x10^6|
| 6       | 23      | 11232   | 59776   | 7585    | 2x10^7  | 3.6x10^6| 1x10^6  | 1.8x10^6|
| 7       | 30      | 13104   | 69120   | 1000000 | 1.5x10^7| 2.1x10^6|

(a) A pre-modified self-loop
(b) A modified self-loop
(c) A pre-modified self-loop
(d) A modified failure

Figure 1: Modification diagram for self-loops and failures to accurately evaluate task rewards.

traces generated on a labelled MDP is exactly those accepted by the collection of task DFA. To do this we can iteratively construct a product MDP $M_i = (M_1 \otimes A_1) \otimes A_2 \otimes \ldots \otimes A_m$ for agent $i$ using the collection of task DFA $(A_1, \ldots, A_m)$ according to [2] for which the resulting state space is $S_i \times Q$. A standard product MDP is unsuitable for optimising task allocation and requires a number of modifying steps, the resulting structure is distinguished as $M_i \otimes A = (S_i \times Q, (s_0, \varnothing), A_i, P_i, L_i)$ for agent $i$. The goal of an agent is to minimise the paths of an allocated task leading to failure and checking that the failure probability is within some acceptable threshold. Therefore, it is natural that once a task has been initiated by an agent it is carried out to completion. To account for this we make the following modification to self-loops, for each $\alpha \in A'(s, \varnothing)$ redirect the self-loop of $(s, \varnothing)$ under the action $\alpha$ to a pair $(s_{\text{new}}, \varnothing)$ as long as $q_i$ is not accepted or rejected, demonstrated in Figures (1a, 1b). Further, rewards (penalties) should only be assigned for task failure when a task has reached a failure state for the first time. We add another new state $s^*$ if there is a non-zero probability of transitioning from a non-failure state to one that satisfies failure for task $j$, shown in Figures (1c, 1d). States in the local product for which some combination of tasks are initial or completed/rejected are labelled Stoppable($A_j$).

Rewards Structures: A rewards function maps state-action pairs to an $n$-agents $+ m$-tasks rewards vector $\mathcal{R}(s, a) = (c_1, \ldots, c_n, t_i, \ldots, t_m)$ where $c_i = \rho_i(s, \varnothing, a)$ inherited from the agent MDP when a task is initial or in-progress for agent $i$ and $0$ everywhere else. The elements $t_i \in \mathcal{R}(s, \varnothing, a)$ are task rewards derived from Equation 2 below. Computing optimal schedulers relies on optimising convex combinations of expected total rewards using $\mathcal{R}(s, \varnothing, a)$.

$$\bar{\rho}_{j+n}(s, \varnothing, a) = \begin{cases} 1 & \text{if } s, \varnothing \models \text{justFail}(A_j) \\ 0 & \text{otherwise.} \end{cases}$$  (2)

Problem Definition: A collection of tasks is a valid mission if three conditions hold: (i) only one task is active at any given time, (ii) there is only one action relevant to a task progress in any reachable state, (iii) after a task is completed the MDP returns back to its initial state. Then given a set of agents and rewards structures, a multi-objective task allocation and planning (MOTAP) problem is to find a randomised allocation function which satisfies task completion probability thresholds and each agents cost constraint. A randomised allocation function is derived from the pareto optimal schedulers, and optimally allocates tasks to agents in a memoryless way. Therefore, tasks can be concurrently distributed to agents, who carry out tasks according to an optimal plan. To solve the MOTAP problem we require a team MDP, $M_i = (S_i, s_0, A_i, P_i, L_i)$ constructed in a similar way to [5], in the sense that a team MDP is a collection of local products with virtual switch transitions $b \in A_i$. In our team MDP, switch transitions exist between any state labelled with Stoppable($A_j$) to an initial state of the $i + 1 \in I$ agent, $P_I(s, \varnothing, b, (s_{i+1,0}, \varnothing))$.

3 EXPERIMENTS

Our approach was evaluated for model size, number of states $|S|$ and $|P|$ transitions, and runtime $t$. The aim of the experiment was to determine how our approach scaled in the number of agents compared to a traditional implementation of an asynchronous multiagent MDP (MAMDP) in PRISM [13] for the same set of agents and LTL formulation.

The outcome of the experiment can be observed in Table 1. When scaling in the number of agents, there was a significant difference in the rate of change between team MDP and the MAMDP, with the former having a much lower model checking time and a much smaller model size. Scaling the multiagent model to agents caused PRISM to experience an out of memory exception. As the number of agents increased, the total model checking time for the team MDP grew linearly as expected, while that for MAMDP grew exponentially. For the team MDP there is a slight divergence between state-space growth and number of transitions (which is not observed in the MAMDP multi-objective model checking). That is due to the additional switch transitions handing over control of tasks to the next agent, which increased the number of transitions, but that increment was small compared with the size of state space. Our findings in this experiment support the claim that MOTAP is more suitable for handling larger multiagent systems than conventional models.

4 CONCLUSION AND FUTURE WORK

This paper proposed an efficient approach for allocating tasks, specified as co-safe LTL, that synthesised randomised schedulers for MDP agents meeting multi-objective requirements involving both cost and probability thresholds. Several future directions of the current work exist. While the team MDP allows decoupling of agents, there is possibly a further extension to decouple tasks. Exploration of the framework to remove the condition of an MDP returning to its initial state after task completion. Last, incorporating multi-objective reinforcement learning [21] into our framework to perform task allocation and planning in the model-free setting.