Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Assignment and Prioritization of Tasks with Uncertain Durations for Satisfying Makespans in Decentralized Execution

Sriram Gopalakrishnan[1] and **Daniel Borrajo**[2,*]

[1] Arizona State University
[2] J.P. Morgan AI Research
sgopal28@asu.edu, daniel.borrajo@jpmchase.com

Abstract

Task assignment under execution uncertainty and temporal/resource constraints is a standard problem for many organizations. Existing approaches in the AI planning & scheduling and operations research literature predominantly focus on dynamic controllability, and non-preemptive execution of tasks. Such solutions are appropriate for teams of agents under tight control requirements. However, in most organizations with human teams, once tasks have been assigned, humans tend to execute their assignments without a constant central oversight (which is needed for dynamic controllability).

In this paper we define a problem in which execution of tasks is distributed (without central oversight), and assumes humans can preempt their tasks when other tasks of higher priority are ready to be worked on. We present two algorithms based on Tabu search and Monte Carlo Tree Search to assign and prioritize tasks for such problems. Experimental results show the improved efficacy of these approaches for this problem setting over non-preemptive strategies.

Introduction

Organizations manage human teams to accomplish their goals by planning to complete a set of tasks within time and resource constraints. As an example, in the Agile methodology for software development, tasks are assigned to team members typically every two weeks (Abrahamsson et al. 2017). An example goal would be to build a website, and the associated tasks would include designing the interface, programming different pages, and testing. There would be dependencies among tasks, and goal priorities. Key characteristics of these type of planning&scheduling (P&S) tasks are: (1) uncertainty in task durations; (2) availability of team members; (3) task-specific qualifications (e.g., only a developer trained on Javascript programming can develop the front-end interface); and (4) there may be limited or no central-control during the execution, as people's tasks are assigned at the start of the planned period, and then control is local i.e. each person decides what to work on. In relation to this last characteristic, if a team member can work on a blocking task that is a prerequisite for other tasks, then they might preempt their current task to work on it. Once that is done, then they can resume working on the preempted task. In this way, that team member would unblock others who might be idly waiting because they are not qualified to do it.

The objective of a P&S solution in our setting is to compute a task-to-human assignment that considers the following: tasks dependencies, goal prioritization, makespan (deadline), resource constraints (agent availability and qualifications), uncertain tasks duration, distributed control, and preemption capabilities of agents. In this paper, tasks are low-level activities like actions in a plan. A good solution to the type of P&S problems herein, would increase the likelihood of task completion within the makespan.

The contributions of this paper include the formalization of a real-world P&S problem, and the adaptation of two search algorithms to solve it, viz. Tabu search and Monte Carlo Tree Search (MCTS). We then evaluate these approaches on diverse and randomly generated problem instances. The experimental results show the ability of the algorithms to effectively compute task assignments and task priorities to leverage preemption and improve outcomes in this problem setting.

Related Work

An approach to execution control under uncertainty is Dynamic Controlability (DC) (Morris et al. 2001) for Simple Temporal Networks with Uncertainty (STNU) (Vidal and Ghallab 1996; Shah et al. 2007). However, such DC approaches require a central controller for dispatching, and continuous monitoring of the execution state which is not available in our problem setting. Distributed Multi-agent STNU (MaSTNU (Casanova et al. 2016)) sought to address the problem of distributed control when communication with a central agent is absent or intermittent. It computes multiple STNUs, one for each agent.

However, without enabling preemption of tasks (context switching), these approaches cannot satisfy the makespans in some problems. Figure 1 shows an example, where there is no solution even without uncertainty (contingent links). In the example, we use a disjunctive condition using the representation used in (Cimatti, Micheli, and Roveri 2016) which is there to indicate an agent can only do one task at a time and no-preemption. We also have an external edge that enforces synchronization between agents (using the represen-

[*] On leave from Universidad Carlos III de Madrid

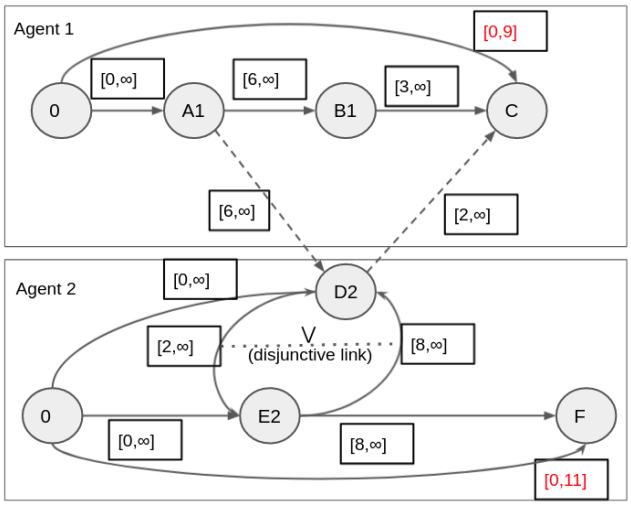

Figure 1: Example of a multi-agent STN that cannot be solved without preemption.

tation from (Casanova et al. 2016)). In this example, there is no assignment to D2 or E2 that allows satisfying all constraints. However, if we allow preemption, it lets us ignore the disjunctive constraint. Then E2 can be started at t=0, preempted at t=6 to finish D2, and then return to E2. This would satisfy all makespans (displayed in red).

Another framework is probabilistic STN (PSTN), where the uncertain durations follow a probability distribution, which is an assumption we make in our work as well. There is recent work on computing a dynamic control schedule for PSTNs (Gao, Popowski, and Boerkoel 2020), and another one for using Monte Carlo Tree Search (Saint-Guillain, Vaquero, and Chien 2021). These methods still require a central controller.

From operations research (OR) literature, our problem maps to Resource Constrained Project Scheduling Problems (RCPSP). There are different dimensions to RCPSP problems. In a recent survey on RCPSP problems (Habibi, Barzinpour, and Sadjadi 2018), the dimensions were categorized into (1)resource type, (2) activity concepts, (3)objective function, (4) availability of information. With respect to these dimensions, our problem is defined as (1)renewable resources (human agents)(2) preemptable activities (3) time-based objectives (4) Stochastic durations. We have not found any RCPSP work that considers both stochastic durations and *preemption* with a makespan objective. The approaches we found in RCPSP literature that consider preemption have deterministic durations. These approaches either have no penalty for interruption (Moukrim, Quilliot, and Toussaint 2015) or impose a hard limitation on the number of interruptions (Zhu, Li, and Shen 2011) to account for preemption cost without explicitly considering it in the constraint satisfaction problem.

The last branch of relevant literature is from Operating Systems (OS), specifically from Real Time Operating Systems (RTOS) (Stankovic and Rajkumar 2004). In such systems, process priorities are used to interrupt/preempt the current activity on a processing unit, and switch control. Priorities become very useful when most of the tasks have uncertain durations, or are affected by exogenous events (interrupts), and the execution is distributed across different processes. The closest analogy to our problem we found in RTOS literature is Worst Case Execution Time (WCET)(Sharma, Elmiligi, and Gebali 2015). WCET is especially used to certify safety critical systems like automotive controllers; the RTOS response to various test-loads is evaluated to make sure the processing time for critical tasks is within a safety cutoff. This is different from our problem in that we do not tune our approach to satisfy a fixed makespan for certain tasks. Rather, ours is a general purpose (assignment) algorithm to handle variable makespans, and constraints over agent availability, and qualifications.

Problem Definition

Our problem of Assignment and Prioritization of Tasks (APT) for Distributed Execution is given by the tuple $< \pi, T, A, P, G, \gamma, w, m, \delta, q, a, \beta >$ where:

- π: Partial Order Plan represented by a directed acyclic graph (DAG = $< T, E >$) where T is the set of vertices of the DAG, and correspond to the tasks in the plan, and the directed edges capture the ordering dependencies.
- T: set of tasks
- A: set of agents.
- P: fixed set of priority levels *assignable* to the actions. $P \subset \mathbb{Z}^+$
- G: set of goals.
- $\gamma : G \rightarrow 2^T$: tasks associated to each goal. A goal is considered complete when all its tasks are done.
- $w : G \rightarrow \mathbb{R}^+$: importance weight (priority) of each goal. *This is different from the* ***assignable*** *task priority from* P.
- $m \in \mathbb{R}^+$ is the maximum makespan within which the goals ought to be completed.
- $\delta : T \times \mathbb{R}^+ \rightarrow [0, 1]$ defines the probability of completing a task in a given amount of time.
- $q : A \times T \rightarrow \{0, 1\}$: qualification function; $q(a_i, t_j) = 1$ if agent a_i can perform task t_j.
- $a : A \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \{0, 1\}$: availability function; $a(a_i, t_1, t_2) = 1$ if a_i is available between t_1 and t_2.
- $\beta : A \times T \rightarrow \mathbb{R}^+$ is the preemption cost function that tells how much additional time a task will take because it has been preempted.

A solution (output) to the APT problem consists of finding an assignment function $\alpha : T \rightarrow A$ and prioritization function $\rho : T \rightarrow P$ that fulfill the temporal and resource constraints (given by π, q and a). In order to compare different solutions, we define a scoring function for goal completion. The score of a plan is the weighted probability of goal completion within the maximum makespan (m). This is defined in Equation 1.

$$\sum_{g \in G} \frac{w(g)}{\sum_{g' \in G} w(g')} \times prob(c(g, \alpha, \rho) < m) \qquad (1)$$

where $prob(.)$ returns the probability of an event, and $c(.)$ is a stochastic function that returns the completion time of goal g for a given assignment and prioritization. $c(.)$ is determined using a simulator of the execution process that we will describe next in the following section.

Solving APT Problems

We use a search-based approach to solving APT. To estimate the goodness of solutions –given the stochasticity of task durations– we developed a simulator to sample and rollout execution trajectories for a given assignment and prioritization of tasks. In the following sections we will describe the simulator and the search algorithms that use it to solve APT.

Simulator

The simulator takes in the problem components and a potential solution (given by α and ρ) and returns a sampled value of the $c(.)$ function. The simulator samples durations for each task, and simulates execution. We use a discrete-time simulator. A continuous-time simulator would be faster, but a discrete simulator implicitly considers the slack from human behavior like reading emails. Thus, it better fits the problem setting we are interested in. At each step, the simulator determines which task an agent works on based on their backlog; backlog is the set of tasks ready and incomplete at that step. Tasks assigned to an agent as per $\alpha(.)$ will appear on their backlog when task dependencies are completed. The simulator includes certain key dynamics:

- Completion time: when a task is assigned to an agent's backlog, it comes with its priority level (given by ρ) and the time required to complete the task (sampled using δ).
- FIFO: when two tasks arrive to an agent's backlog with equal task priority (set by ρ), the agent works on the first task that arrived. If they arrive at the same time, then the order is arbitrary.
- Agent availability: when an agent is working on a task, progress is made on the task only during the time when the agent is available (given by $a(.)$). When the agent is available, the time required to complete the task is decremented by one time-step of the simulator.
- Preemption: when a higher-priority task enters the backlog, it will preempt the current task, which will return to the backlog. The time needed for completion of the preempted task will increase (as per β).

The simulator returns the completion time of each goal in G. Completion times are used by the search algorithms to find good assignments and prioritizations of tasks. In this work, we consider all tasks as preemptible. Non-preemptible tasks can be easily supported by checks in the simulator. The rest of the methodology remains unchanged.

Search Algorithms

We present two stochastic search approaches to solve APT problems: Tabu search and MCTS. Both use simulations to evaluate the value of a solution using Equation 1. In Tabu search, we start with a random initial solution, and take local improvement steps until no more improvements greater than a specified threshold are possible. We iterate the process (random restart) for as many times as possible within a time limit. The tabu list in tabu search only keeps the solutions obtained at the end of each iteration.

For MCTS, the nodes at each level in the tree correspond to a partial assignment of tasks with priorities (a partial solution). Each successor node adds an assignment and a priority for a task. During MCTS search, the process steps through each node and recursively selects the best-child node based on the average reward of the node, (initialized to 0); this is only *if* the node is already fully expanded. If the node is not yet fully expanded, i.e. only a subset of children have been evaluated, then a random node from the remaining children is selected to be opened. Each node is evaluated by random rollout, i.e. random assignments and prioritizations for the remaining tasks (subject to the qualification constraints). The reward at the leaf node (end of a rollout) is the average score returned by 30 simulations of the complete solution obtained at the leaf node. This reward is backpropagated to each of the parent nodes, and the average reward is updated. When the allowed time for search has expired, MCTS may *not* have opened a terminal node of the search tree (which would have all tasks assigned). In that case, we take the best leaf-node from amongst the nodes opened so far; this would only have a partial assignment of tasks. We then complete the assignment using a hill-climbing search for the remainder with no random restarts or tabu list. This is needed because unlike MCTS for the games of Chess or Go, we need a complete solution and not just the next decision in the search tree. In our experiments, we did *not* utilize exploration with upper confidence bounds (Fürnkranz and Scheffer 2006). We found that exploration resulted in worse solutions when limited by the cutoff time; we attribute this to MCTS taking more time to explore than exploit/discern between good partial solutions that were found early.

One of the algorithmic decisions that helped improve results in MCTS was to order the task assignment decisions intelligently. We run a topological sort on the partial-order plans DAG and use the topology levels to order tasks in the search tree. The rationale is that decisions made for tasks (nodes) at lower topological levels would impact downstream decisions, so it makes more sense to decide these first. Additionally, since goals have importance weights associated to them, we further order task assignments in MCTS based on the highest weight of the goals that require them. Lastly, our MCTS code was built upon the standard implementation of the algorithm (Coulom 2006) as implemented in the python `mcts` library.

Experiments

We evaluate the Tabu and MCTS algorithms on 6 randomly generated APT problems. We give each algorithm a time limit of 30 minutes. The number of tasks in each plan is fixed to 30. So, a plan DAG with fewer levels/depth will have more tasks per topology level resulting in a wider graph. The number of goals was randomly set between [3, 5], and each goal weight was randomly set between [1,3]. Each task from the last topological level was randomly assigned to one of

				Goals						
H	a	p	m	g_0	g_1	g_2	g_3	g_4	MCTS	Tabu
3	4	1	163	1	1	2	1	1	0.86	0.83
3	4	3	163	1	1	2	1	1	0.99	0.93
3	6	1	163	2	2	1	3	3	0.99	1.0
3	6	3	163	2	2	1	3	3	1.0	1.0
3	4	1	156	2	3	1			0.04	0.0
3	4	3	156	2	3	1			0.48	0.4
3	6	1	156	1	2	1	2	2	1.0	1.0
3	6	3	156	1	2	1	2	2	1.0	1.0
3	4	1	165	3	3	3	1	3	1.0	1.0
3	4	3	165	3	3	3	1	3	0.93	0.99
3	6	1	165	3	3	2			1.0	1.0
3	6	3	165	3	3	2			1.0	1.0
6	4	1	169	2	1	3			0.32	0.33
6	4	3	169	2	1	3			0.34	0.33
6	6	1	169	1	3	3			1.0	1.0
6	6	3	169	1	3	3			1.0	1.0
6	4	1	174	2	1	1			1.0	1.0
6	4	3	174	2	1	1			1.0	0.99
6	6	1	174	3	1	2			1.0	1.0
6	6	3	174	3	1	2			1.0	1.0
6	4	1	154	1	2	3	3		0.51	0.59
6	4	3	154	1	2	3	3		0.7	0.76
6	6	1	154	3	2	3	3		1.0	0.84
6	6	3	154	3	2	3	3		1.0	1.0

Table 1: Likelihood of finishing execution by maximum makespan for MCTS and Tabu search given different configurations of topological depth (H), number of agents ($a = |A|$), priority levels ($p = |P|$)) and maximum makespan (m).

the goals. The simulation time-step was 1 hour. The following problem features were randomly generated:

Task Time: The time required per task is stochastic, and follows a uniform distribution between two limits. The limits are sampled from a normal distribution with a mean of 8 (hours), and standard deviation of 3; the smaller sampled value is the lower bound.

Agent Availability: Each agent is available during random intervals of time to make the search more challenging. We start with the agent being available for the max duration of a problem. Then, for each hour, the likelihood of an agent taking time off starting from any given hour is 0.05. If an agent takes time off, the duration is sampled from a Gaussian distribution with mean 8 and standard deviation 4.

Likelihood of Task Dependency: Each task has a 5% chance of being connected to any other task. This is on-top of the single edge needed to enforce the depth of the DAG underlying the partial plan.

Qualifications: Each task requires a qualification to perform it. Each agent is assigned a subset of the possible qualifications, with at least one agent having each qualification. The likelihood of an agent having an additional qualification is 0.25. We fixed the number of qualifications to 4.

The makespan m was set to 60% of the sum of all the tasks' duration upper bound. We set this value empirically based on results from using a team of 3 agents and 1 priority level; the score using Equation 1 was often below 50% with either search algorithm, so we chose it as a challenging makespan. Given these variations in problem parameters, we posit that our problem generation is sufficiently parameterized to produce diverse, and challenging problems.

During simulation of an assignment on a plan, when an agent's task is preempted by a higher priority task, we set the penalty to a fixed amount; 0.5 hours additional time to complete the preempted task. For our experiments, we vary the following: topology of the underlying DAG - we set the depth to 3 or 6, which affects the longest sequence of dependencies; number of agents - we used teams of 4 and 6 people; priority levels - we used 1 and 3 priority levels to evaluate the effects of preemption. All code was written in python and experiments were run on a PC with Intel® Core™ i7-6700 CPU, running at 3.40GHz on Ubuntu 20.04 with 32 GB of memory. All random elements are controlled by a seed.

Results

In Table 1, we present the results on 6 randomly generated problems; each problem's data is separated into a sub-table. The score under the MCTS and Tabu columns is computed as per Equation 1, and is the averaged result of 100 simulation runs. For 4 agents, when the number of priorities increases from 1 to 3, the success rate increases appreciably for both algorithms, except in one anomalous case which we attribute to the stochastic nature of the search. Having more priority levels alleviates the agent resource constraint. An example of this is when only one agent has a necessary qualification for many tasks, that agent becomes the bottleneck. By allowing preemption, that agent can switch tasks and improve outcomes. For example, in the second graph we see an increase in the score from 4 agents with 1 priority level, to 4 agents and 3 priority levels. The reason for that is only one agent had the qualification required for 8 longer tasks. When the same partial plan was run with 6 agents, there were two more agents who had that qualification, and that helped the success rate jump to 1.0. Both MCTS and Tabu search performed comparably well, and so we cannot say one is better. It is unsurprising that with 6 agents (even with just 1 priority) both methods tend to find seemingly optimal score solutions, except for the last graph when Tabu search only finds a 1.0 score solution with 3 priority levels and not with 1 priority level.

Conclusion

In this work, we presented a new type of P&S problem (APT) that resembles a set of real world problems. The main differences with prior work is the combination of distributed control during execution and the agent's ability to preempt tasks. We also contribute the adaptation of two search algorithms, MCTS and Tabu search, to solve these problems. The experimental results show that both algorithms provide promising results for APT problems. The results also show that when agents are able to preempt their current tasks, the goal completion score improves appreciably.

Acknowledgements

This paper was prepared for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Abrahamsson, P.; Salo, O.; Ronkainen, J.; and Warsta, J. 2017. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Casanova, G.; Pralet, C.; Lesire, C.; and Vidal, T. 2016. Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 930–938.

Cimatti, A.; Micheli, A.; and Roveri, M. 2016. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, 72–83. Springer.

Fürnkranz, J.; and Scheffer, T. 2006. *Machine Learning: ECML 2006: 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212. Springer Science & Business Media.

Gao, M.; Popowski, L.; and Boerkoel, J. 2020. Dynamic control of probabilistic simple temporal networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9851–9858.

Habibi, F.; Barzinpour, F.; and Sadjadi, S. 2018. Resource-constrained project scheduling problem: review of past and recent developments. *Journal of project management*, 3(2): 55–88.

Morris, P.; Muscettola, N.; Vidal, T.; et al. 2001. Dynamic control of plans with temporal uncertainty. In *IJCAI*, volume 1, 494–502.

Moukrim, A.; Quilliot, A.; and Toussaint, H. 2015. An effective branch-and-price algorithm for the preemptive resource constrained project scheduling problem based on minimal interval order enumeration. *European Journal of Operational Research*, 244(2): 360–368.

Saint-Guillain, M.; Vaquero, T. S.; and Chien, S. A. 2021. Lila: Optimal Dispatching in Probabilistic Temporal Networks using Monte Carlo Tree Search. In *31st International Conference on Automated Planning and Scheduling*, 38.

Shah, J. A.; Stedl, J.; Williams, B. C.; and Robertson, P. 2007. A Fast Incremental Algorithm for Maintaining Dispatchability of Partially Controllable Plans. In *ICAPS*, 296–303.

Sharma, M.; Elmiligi, H.; and Gebali, F. 2015. Performance evaluation of real-time systems. *International Journal of Computing and Digital Systems*, 4(01).

Stankovic, J. A.; and Rajkumar, R. 2004. Real-time operating systems. *Real-Time Systems*, 28(2-3): 237–253.

Vidal, T.; and Ghallab, M. 1996. Dealing with Uncertain Durations In Temporal Constraint Networks dedicated to Planning'. In *ECAI*, 48–54. PITMAN.

Zhu, J.; Li, X.; and Shen, W. 2011. Effective genetic algorithm for resource-constrained project scheduling with limited preemptions. *International Journal of Machine Learning and Cybernetics*, 2(2): 55–65.