T-HTN: Timeline based HTN Planning for Multi-Agent Systems

Viraj Parimi

CMU-RI-TR-21-64

August 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Stephen Smith, *chair*
Prof. Zachary Rubinstein, *co-chair*
Prof. George Kantor
Anahita Mohseni Kabir

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my family and friends.

Abstract

Planning in mission-critical systems like deep-space habitats with onboard robotic systems must be robust to unforeseen circumstances. Such systems are expected to complete a set of goals with different deadlines each day for routine maintenance while also accounting for emergencies. With the presence of humans within the habitat, the robotic systems can be required to perform specific tasks while possibly collaborating with the humans. Further, since the habitat can support multiple robots, this becomes a source of contention as they have to share the limited set of onboard resources. This dynamic between the humans, robots, and the habitat generates a complex system where failures at any level can cause significant delays leading to temporal uncertainty. Such delays can have huge implications depending on whether or not the delay causes the system to miss a goal's deadline. Hence, it becomes crucial for the planner to address the overall schedule within the context of the current temporal deadlines of the goals and the resource constraints within the environment. Assuming a known map of the environment and a fixed horizon time, one can develop a schedule for the robotic systems that accounts for such temporal uncertainty and resource constraints by leveraging the timeline-based planning framework. To this end, this thesis proposes `T-HTN`, a novel planner that extends the Hierarchical Task Networks (HTN) model by incorporating temporal reasoning via flexible timeline structures to produce plans that respect the goal's deadlines and the complex resource constraints introduced in multi-robot scenarios. `T-HTN` is a robust extension to a timeline-based planner whose efficacy has been tested on multiple example scenarios within a simulation environment.

Acknowledgments

First, I am grateful to Prof. Stephen Smith and Prof. Zachary Rubinstein for their constant support during the past two years. I am incredibly thankful to them for showing immense patience, guidance, and faith in me, especially during the challenging pandemic over the past year. I immensely enjoyed our thought-provoking weekly conversations. Their support and ideas helped advance my understanding of relevant concepts and provided me with a strong background, without which this work would not have been possible. I am also grateful to them for giving me an opportunity and funding to collaborate with researchers across the country. I am also thankful to Prof. George Kantor for serving my thesis committee and asking insightful and vital questions. I would also like to thank Dr. Anahita Mohseni Kabir for acting on my thesis committee and providing critical feedback.

Second, I would like to thank Abhinav Khattar for being the best friend one could ask for. Our time spent together helped me push through my work. I would also like to thank Jayanth Mogali for being a supporting lab mate and for our long and insightful research discussions. Further, I would like to thank Isaac Isukapati and Rachel Burcin for their constant support during my time at CMU and for making this experience a lot more enriching. I have learned a lot from them, and their continued encouragement helped me improve as a researcher. Last but not least, I would like to thank my friend, Mononito Goswami, for being there towards the end of my master's thesis and helping me graduate to the next phase of my academic life.

Funding

This work was supported by views of the National Aeronautics and Space Administration (NASA) under grant number 80NSSC19K1052. Any opinions, findings, conclusions, or recommendations expressed in this thesis are those of the author(s) and do not necessarily reflect the views of NASA.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Overall management of spacecraft activities on board near-earth habitats (ISS) has always been a manual, human-driven process supported by ground-based crew scheduling tools. Similarly, management of any onboard robotic systems has typically been planned, rehearsed, and controlled from the ground with the assistance of astronauts on board the habitat. The lack of real-time communications mandates greater autonomy in spacecraft planning and control in moving to deep space environments. With the prospect of deep space habitats being unoccupied by humans for significant periods, the need for mechanisms to coordinate the actions of multiple robotic and autonomous systems becomes increasingly important to achieve the safe, robust, and efficient maintenance of spacecraft subsystems.

From a multi-agent planning perspective, the problem of coordinating a team of collaborative agents has typically been treated in one of three ways. One approach is to pre-enumerate a set of local roles and protocols to govern the interaction between agents in the accomplishment of a given task (e.g., [30]). A second has been by to define extensions to single-agent planning frameworks that manage selection from the set of pending goals for a given agent (e.g., [1]). A third has been to focus on the development of solutions to more specialized planning problems, e.g., the recent flurry of activity on the Multi-Agent Path Finding (MAPF) problem (e.g., [16, 25, 29]). In the context of deep space habitats, none of these approaches are satisfactory on their own. Specification of agent roles and interaction protocols can be an appropriate tactic for accomplishing joint tasks. Still, it is difficult for such decentralized approaches to assure safe and collision-free physical actions of robotic systems in constrained spaces or to satisfy temporal constraints involving multiple independent actors (e.g., no robotic movement during a spacecraft maneuver) without introducing other mechanisms. Alternatively, goal management extensions to single-agent planning frameworks, as well as to decentralized multi-agent planning approaches [31], tend to be self-interested and are also not equipped to handle shared global constraints (e.g., access of a given robot to a particular segment of a rail network at a given point in time) or limitations in the availability of requisite robotic systems over specific temporal windows in any principled way. Specialized multi-agent solutions such as MAPF can be helpful in particular circumstances but do not offer an overall answer to the problem of interest. We view goal selection and management across multiple robotic and autonomous systems as more of a scheduling and resource allocation problem than a planning problem. Specialized solutions such as MAPF can provide additional leverage by adopting this perspective.

In the area of autonomous space mission planning, EO-1 [23] demonstrated an impressive ability to dynamically re-plan its observation schedule based on image processing results obtained from previous observations in a closed-loop fashion. However, the requirements of deep space habitat operations control offer more significant challenges. Whereas coordination of onboard image processing constitutes a relatively predictable sequence of computational tasks, management of robotic system tasks is fraught with uncertainty in how long tasks will take, whether or not they will achieve the desired result, and how to recover if the desired result is not achieved. An effective approach to autonomous control in deep space environments must be capable of anticipating uncertainty and giving greater flexibility to execution processes. Alternatively, another NASA success has been the MAPGEN science planner for the Spirit and Opportunity Mars rovers [6], which was built on the earlier timeline-based planning technology of the NASA Remote Agent mission [19] that does anticipate and plan to accommodate some amount of temporal uncertainty. However, MAPGEN was designed principally to provide an interactive constraint-checking capability for science planners operating on the ground (planning the next day's science activities). Like other timeline-based planners, its underlying planning engine relies on the low level, state-variable specific heuristics for search control and does not exploit the power and structure of a global scheduling and resource allocation perspective. We aim to develop a timeline-based planning framework that integrates the power of EO-1's hierarchical, scheduling perspective on search control with the complementary strength of MAPGEN to produce and maintain plans with temporal flexibility.

1.2 Contributions

We present `T-HTN`, a novel HTN-based timeline planner/scheduler that enables global coordination of multiple robotic and autonomous systems (as well as human agents) in carrying out inspection, maintenance, and repair tasks on board a deep space habitat. We designed `T-HTN` to effectively account for complex temporal and spatial constraints on task execution and extract significant inter-agent coordination constraints while providing individual agents with the flexibility to cope with some amount of execution-time uncertainty locally. Further, we conducted initial testing in multi-robot scenarios involving two UR5 robot arms mounted on a common rail to evaluate effectiveness in managing resources that must be shared across multiple agents.

we present the following core contributions:

- A hybrid planner/scheduler `T-HTN` that combines the structural power of an HTN representation with the expressivity and flexibility of timeline-based planning frameworks from a goal regression and resource allocation perspective.
- A comprehensive plan search procedure that provides ease of integration for specialized planners for optimized plan generation via specialized functional constructs.
- A modular and easy-to-use system that generates plans in a standard output specification, which connects to existing motion planners and yields comprehensive information that aids in building a reactive system.

1.3 Organization

Chapter 2 reviews some required preliminaries and essential concepts in symbolic and temporal planning. Chapter 3 reviews some of the current work in this domain and examines areas where they are deficient. Chapter 4 introduces `T-HTN` that forms the primary and core contribution of this thesis. This chapter details each component of the framework and explains the novel contributions made at each step. Further, chapter 5 presents the empirical results where we demonstrate the effectiveness of `T-HTN` on specific multi-robot scenarios. Finally, Chapter 6 concludes this thesis and provides the limitations of `T-HTN`, which forms the basis of future directions of research.

Chapter 2

Background

2.1 Classical Planning

A classical planning problem aims at synthesizing a sequence of actions that, when executed, transforms the initial state of the world to a prescribed goal state. This process requires a characterization of the initial state of the world along with the goal state and a comprehensive list of all possible actions. A simple classical planning problem deals with a finite state scope where each action is deterministic. Further, each action is considered instantaneous with a simplifying assumption that the underlying world state is static and fully observable. Additionally, it does not incorporate any rich model of time, making it less robust to execution-time uncertainty particularly in cases when there are strict deadlines imposed on the goals. This makes classical planning more of a search problem over a set of states that are often represented in a common language such as Planning Domain Description Language (PDDL) [12]. PDDL is a family of languages that allows one to define a planning problem. A planning problem specification consists of a *domain* file that encapsulates all the action enumerations and parameter definitions and a *problem* file that specifies the initial and the prescribed goal states. There have been multiple evolutions of PDDL over the years. The initial versions were limited to predicate logic modeling problems, whereas the more recent versions support time and numerics, making the language more expressive. Some extensions include support for *durative actions* that are just regular actions which have corresponding durations assigned to them. Efficient algorithms exist that build planning graphs and use heuristics to help guide the search procedure (e.g., [5]) for faster plan synthesis.

2.2 Hierarchical Task Networks

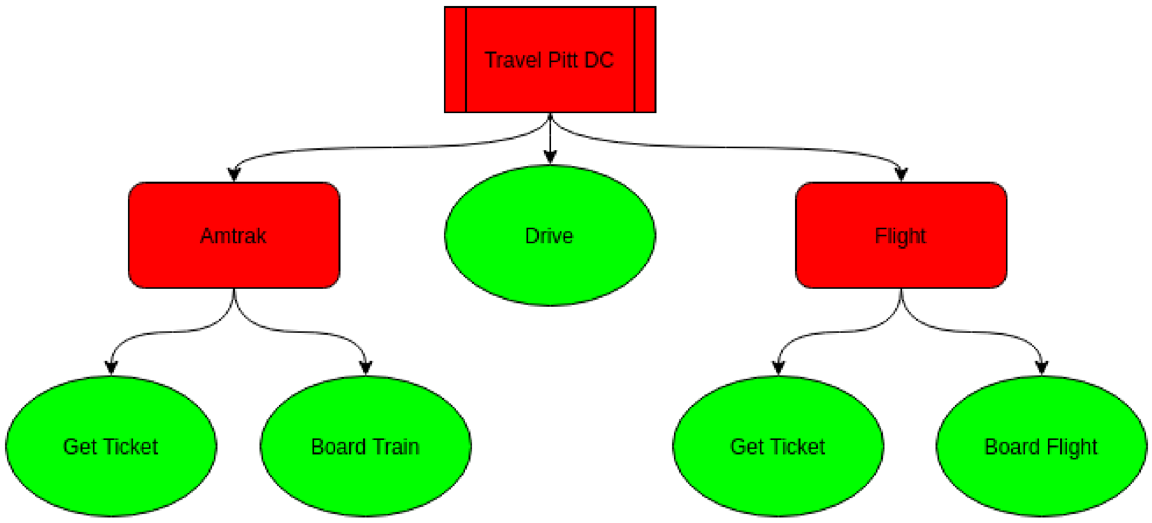


Figure 2.1: HTN representation of traveling from Pittsburgh to Washington DC. The node in green represents the task that should be accomplished. Nodes in red are high-level methods that can be further decomposed. Nodes in blue are the primitive tasks that correspond to concrete actions.

A common drawback of the simple classical planning approach is that it has a hard time leveraging the domain structure of a given problem. For most practical purposes, it is found that humans have a general idea of how to solve a particular problem based on their experiences. They understand that the given task can be broken down into smaller subtasks until they are nondecomposable at which point they must have corresponding robotic behaviors defined for them. Such primitive tasks are the schedulable units on the timelines. Hierarchical Task Networks (HTN) were designed to enable planning systems to use such structure. Within the HTN planning paradigm, the problem transforms into accomplishing a set of tasks rather than a pre-specified goal. The framework builds on top of *methods* that allows for the recursive decomposition of a high-level task or *nonprimitives* into smaller subtasks, terminating with primitive tasks referred to as *actions*. Such decomposition is generally utilized to identify all the solution options based on their preconditions and effects, thereby building a task network over which a planning search procedure uses guided heuristics similar to the classical planning approach to synthesize a composed plan comprising primitive tasks. One can find a detailed overview of HTN planning in [13].

Example

Suppose that a person wants to travel from Pittsburgh to Washington DC. There are multiple ways that they can achieve this task. They can either travel via Amtrak or take a flight or drive to the destination. This simple task can be represented by an HTN as shown in Figure 2.1.

2.3 Simple Temporal Networks

Simple Temporal Networks (STN) are often used when tackling planning and scheduling problems that involve temporal constraints. It is able to model a temporal plan in which it is possible to constrain the distance between pairs of *time points* whose occurrence is under the control of the executing agent (i.e., a real-time planner). STNs were first proposed by [11] in their seminal 1991 paper. A network of temporal constraints is defined by a set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ of continuous temporal variables or *time points* along with a set $\mathcal{C} = \{c_{12}, \ldots, c_{ij}\}$ of binary temporal constraints. A *time point* represents an event occurring at some point in time, often, a pair of them together representing the start and end of some task. Each temporal constraint limits the temporal distance between exactly two time points. In an STN, such temporal constraints $c_{ij} \in \mathcal{C}$ are represented in the form of an interval $[\mathtt{lb}, \mathtt{ub}] \in \mathbb{R}$ which essentially restricts the time that may elapse between the time points x_i and x_j by requiring that $x_j - x_i \in c_{ij}$. Such intervals may extend to (negative) infinity, implying that one or both of the bounds are unconstrained. To build the distance graph of the constraint network one uses the set $\mathcal{X}$ as nodes and represents the set $\mathcal{C}$ as edge weights between two nodes. The `lb` bound is often represented by an directed edge with weight $w_{ij} = -\mathtt{lb}$ going from x_j to x_i while the `ub` bound is represented by another directed edge with weight $w_{ij} = \mathtt{ub}$ going from x_i to x_j.

Further, one can always assume that an STN is fully connected. Such an assumption can be made without loss of generality as even if it is not valid, each connected component then acts as an individual STN. With this setup, one also needs to refer to the absolute time instead of just the temporal distance between two time points. To solve this, one can identify a temporal reference point that is often represented as *calendar zero* (CZ), which can be interpreted as the origin of time and is explicitly constrained to occur before any other time point. With this basic setup in place, a valid schedule for an STN is defined as an assignment of values from $\mathbb{R}$ to each time point in $\mathcal{X}$ such that all the constraints in $\mathcal{C}$ are satisfied. Additionally, an STN is called *consistent* if it admits at least one valid schedule.

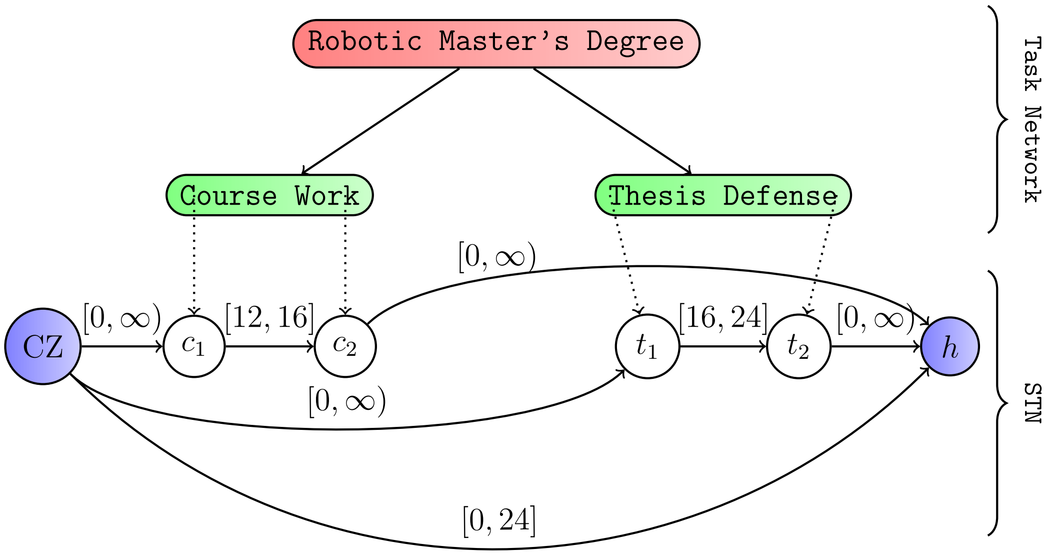


Figure 2.2: Example STN $\mathcal{S}$: getting Robotics master's degree at CMU. The red node corresponds to the high-level task of completing the Robotics master's at CMU which is decomposed into two primitive actions represented with green nodes. Each primitive action is associated with a pair of time points which correspond to their start and end times. The blue nodes represent the temporal reference point `CZ` and the horizon time h.

Example

Let's say we want to schedule the tasks required to complete a Robotics master's at CMU. The requirements of the degree are to complete the coursework and to defend a research thesis. These activities can be represented by their corresponding start and end time points: c_1 and c_2 for the course work which requires 12 to 16 months, and t_1 and t_2 for the thesis defense which requires 16 to 24 months. Let's say one starts their journey at a universal reference point CZ and has a long horizon time point referenced by h. We want to guarantee that the degree is conferred before 24 month period expires. Based on this example problem, the corresponding STN $\mathcal{S}$ is showcased in Figure 2.2.

2.4 Timeline-Based planning

All the approaches that we discussed till now have been action-based. Their fundamental concepts include state representation and action enumerations. These actions are executed to affect the state of the world, and a sequence of such actions is performed based on a plan to reach a satisfying goal state. Timeline-based planning is an entirely different approach and was first introduced in the context of planning, and scheduling for space-based operations [3, 17, 18, 26] and more recently in the context of human-robot collaboration [32]. The core motivation behind adopting this approach is the rich temporal reasoning that the framework provides. There is no clear separation between states and actions, and the planning problem is generally modeled as systems composed of multiple components. Each such component is called a *state variable* whose evolution over time is described by a *timeline*. This evolution is governed by a set of temporal constraints called the *synchronization rules*. Each state variable has an associated transition function that describes the states that this component can be over time. This framework allows one to model and reason about systems made of multiple components rather than the behavior of a single agent. Timeline-based planning has typically been done using fixed-time schedules, i.e., without STNs; however, few existing frameworks leverage STNs in conjunction with timelines. Each timeline is essentially just a sequence of *tokens* where a *token* describes one particular state of the state variable. Given a set of state variables $\mathcal{V}$

and a set of rules $\mathcal{R}$ over those state variables, the timeline-based planning problem involves finding a set of timelines $\mathcal{T}$ for all $v \in \mathcal{V}$ such that no rule $r \in \mathcal{R}$ is violated. Using STNs as a backbone, the timeline-based planning framework naturally exploits its advantages and produces temporally flexible plans by default.

Chapter 3

Related Work

Recent work by [32] showcased a timeline-based planning framework called PLATINUm that provided a new algorithm for temporal planning with uncertainty by adding *controllability* tags. They introduced some heuristic search capabilities that were grounded in hierarchical modeling. In addition to this, they also presented a robust plan execution module that completed the feedback loop for the planner to deal with temporal uncertainty. This work was modeled with human-robot collaboration in mind, where they modeled humans as uncontrollable agents and leveraged the flexible temporal nature of timelines. They proposed a three-pronged hierarchical approach where the first step involves supervision which models the process and high-level tasks that need to be accomplished. The second step includes coordination, which models the decomposition of high-level tasks into low-level tasks. The final step concludes with an implementation pipeline that models operations that allows a robot to complete a particular task. They perform these steps first to identify a partial plan, after which a solver refines that established plan until a complete and valid plan is found. The refinement process involves detecting flaws that violate the validity of the plan and performing corrective steps to fix those flaws. Further, the developed planner utilizes the controllability tags at the time of execution monitoring to establish the validity of the states of the controllable actors in the environment. Although the proposed methodology is robust to temporal uncertainty, their planner has difficulty working out the resource dependencies within the environment as that aspect is not explicitly modeled. Moreover, their flaw-resolving strategy can potentially lead to an infinite loop due to cyclic dependencies. Additionally, their reliance on Domain Description Language (DDL) further limits their widespread adoption in the planning community as domains written in this language lack the action-based planning perspective that PDDL provides. DDL was developed to emphasize the timeline-based planning perspective of an evolving system of components over time but does not explicitly support plan operators that PDDL supplies.

To address the lack of resource reasoning in PLATINUm, another work [33] presented a more theoretical formalization for adding explicit resource reasoning to a timeline-based planning framework. They proposed the addition of *resource profiles* to the original framework and redefined the validity of a feasible plan as the one that satisfies not only all the synchronization rules but also all the resource specifications. So once a plan is generated, their plan validator checks the modified resources within the plan. It ensures that these resource profiles were following the initial specifications mentioned in the domain. If they find potential violations, they extend the prior flaw-resolving strategies to account for such violations. A proposed method includes solving a Minimal Critical Set detection problem, where resource expressions that violate the capacity constraints of a resource are represented as Critical Sets. Although the proposed formalization is comprehensive, the underlying framework still inherits the same drawbacks as PLATINUm.

On the other hand, some other recent work [22] has looked at extending Simple Hierarchical Ordered Planner 2 (SHOP2) [15, 20, 21] which is a well known HTN planner to handle multi-capacity discrete resources along with complex temporal constraints. This work proposed integrating three carefully designed and interrelated submodules, including a resource model followed by a Check Consistency and Backtracking (CCB) module, which then feeds into a Guided Search module that tries to improve the resource utilization and minimizes the generated plan's makespan. Although this work does attempt to combine the efficiency of resource utilization and flexibility of STNs, they don't retroactively try to satisfy failing preconditions when looking for possible primitive actions in response to a prescribed goal. Further, they suffer partially in terms of optimality since they do not rely on existing specialized planners to generate plans for solved problems. The idea of combining HTNs with STNs is not new since some past work [34] has tried to leverage the causal relationships encoded within HTNs to reduce the time complexity of temporal propagation on the associated STN. One such proposed method includes a sibling-restricted propagation algorithm that exploits the HTN structure to limit the temporal constraints between only a parent and their children.

In complete contrast, some other recent work working in the action-based planning frameworks [2, 8] proposed the idea of *Affordance Templates* to combine the high-level task planning and behavior-based techniques. Affordance templates are essentially graphical representations in 3D worlds, such as RViz, that provide robot task goals and parameters in an object-centric coordinate frame. In addition, they added capabilities to prioritize and cluster goals so that the planner can adjust generated plans based on the incoming goals' priority levels. For example, they proposed a Goal Management module within ROSPlan [7] that modifies the set of current input goals to the ROSPlan pipeline following a simple protocol where the Goal Manager keeps on removing low priority goals from the pool one by one until the ROSPlan pipeline can successfully satisfy all the remaining goals. Further, they extended the ROSPlan architecture to support plan viability where they keep monitoring the state of the world as actions are executed to ensure that if they find any inconsistencies, then at that point, they can spawn a replanning request to fix it. Although the proposed end-to-end framework is shown to be robust in terms of execution, however, they fail to address the temporal uncertainty introduced into the system by, for example, the motion planning primitives. Suppose a plan required a robot to do a particular task, but while executing the said task, the robot failed to complete it due to motion planner failure. In that case, the plan viability check fails as it does not match the expected world state, thereby triggering a very compute-intensive replanning request. If the system had some amount of temporal flexibility incorporated by maintaining all four time bounds on the task start and completion rather than having a fixed-time plan which just maintains a start and an end, there would have been no need for another replanning request.

Finally, there has been work within the scheduling community that has coupled the use of hierarchical task templates with constraint-based procedures for allocating resources to requested tasks over time. One early example was the AMC Allocator [28], a system for generating airlift and tanker mission schedules world-wide for the USAF Air Mobility Command (AMC). To achieve scalability, an incremental-constraint-based search procedure was used, which together with air mission template descriptions, enabled efficient generation of good initial solutions. Then as additional decision-making time permitted, an extended iterative repair search was executed to further improve this solution. More recent research [24] has generalized this constraint-based search paradigm to operate with an STN-based, flexible times solution model, and demonstrated its effectiveness and scalability in solving dynamic, dial-a-ride pickup and delivery problems. Combining the findings of such work with an HTN-based perspective allows one to leverage the inherent causal relations that the structural representation of HTN provides and generate optimal and efficient plans that can be incrementally extended to include newly received requests.

Chapter 4

Methodology

Very little prior work exists that tries to combine the action-based and timeline-based planning paradigms, while both approaches have their advantages. The main goal of this thesis is to leverage them to develop a hybrid planning approach that can reason for tasks assignments for overall coordination and management of activities on board a space habitat while being scalable towards multiple agents and robust to temporal uncertainty. To realize this, we developed a multi-agent planning/scheduling framework - `T-HTN`, which starts with a two-layer representational structure that is typical of timeline-based planners: (1) A domain-level representation of tokens designating particular values that different state variables can take over time, along with constraints that define the legal transitions that can occur between values for each state variable, and (2) an underlying graph of time points and distances that constitutes a Simple Temporal Network (STN), which is used to compute the start and end time bounds on timeline tokens and to enforce consistency as they are inserted into the state variable timelines to achieve specific goal states. Using this domain-level representation, our designed search procedure for generating an executable plan in response to a given task request proceeds as an interleaved process of instantiating a task network and then searching for slots on timelines where the instantiated tasks can be feasibly inserted. These timelines are only associated with required resources and must be allocated to perform tasks defined by the generated task network. When we find feasible set of slots for all the necessary tasks, our developed framework goes through them in an ordered fashion to execute

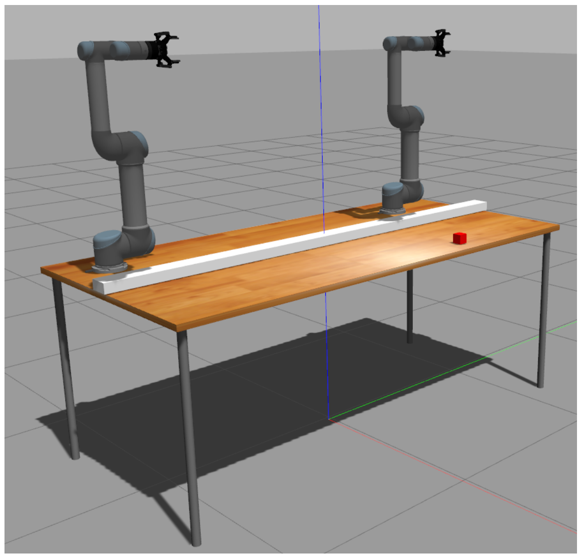


Figure 4.1: Consider two UR5 robotic arm manipulators which are mounted on a shared railway network. Let's indicate the robot on the left as UR5A and the one on the right as UR5B. Within this scenario, an incoming request mandates the movement of the red cube from its initial location to the near edge of the table. The rail network here is a simple rail segment that is internally represented as five different adjacent rail_blocks.

each action primitive on the corresponding robot utilizing that robot's internal motion planners. In the guided effort to support easy adoption in the planning community, we built `T-HTN` to be flexible enough to translate the generated plans into the standard PDDL for communication to individual robotic agents, with the interface to support specialized planners for optimal plan generation while also encompassing comprehensive information to aid in building a reactive system. Having presented a high-level overview of the approach, we will use a simple running example as shown in Figure 4.1 involving two UR5 robotic arms mounted on a shared railway network to explain each segment of the framework in detail.

4.1 Hierarchical Task Modeling

Current PDDL plan representations are designed principally for planners that reason from first principles, i.e., planners that reason either backward from the goal state to the initial state or forward from the initial state to the goal state by applying plan operators to achieve outstanding preconditions or effects. One exception is a recent work of [14], which has looked at the problem from an HTN perspective and hence proposed a hierarchical task planning extension to PDDL called Hierarchical Domain Definition Language (HDDL). However, like most contemporary HTN planning frameworks (e.g., [21]), plans tend to be expanded forward in time and do not support durative actions and, more generally, reasoning over timelines from a "God's eye" point of view of the future. Another recently developed planning framework [4] has similarly recognized the need and potential of such reasoning capabilities but utilizes its non-PDDL representation. Given our interest in being able to translate generated plans into standard PDDL for communication to individual robotic agents and using [14] and [4] as starting points, we developed a hierarchical extension to PDDL 2.1 that supports timeline-based reasoning.

```
(:action move_to_home_state
    :parameters (?r - robot)
    :precondition (and
        (unsafe state ?r.st)
    )
    :effect (and
        (= ?r.st home)
    )
)
```

(a)

```
(:action move_to_home_state
    :parameters (?r - robot)
     :duration (= ?duration 10)
    :precondition (and
        (at start (unsafe state ?r.st))
    )
    :effect (and
        (at end (= ?r.st home))
    )
)
```

(b)

Figure 4.2: Domain representation of a primitive action `move_to_home_state` without (left) and with (right) durative action and temporal qualifiers support

1. Specifically, we expanded the Extended BNF of HDDL to support durative actions similar to what PDDL 2.1 supports. Durative actions, however, are only supported at the level of primitive actions. In contrast, temporal bounds for high-level *tasks* can be estimated via the individual durations of their ultimately decomposed constituent actions and their partial/total ordering. These tasks can be decomposed into different subtasks via different *method* specifications. With this support, one can specify each action's duration constraint, which helps guide the search process later to maintain temporal consistency. Figure 4.2 shows a comparison as to how the domain representation changes with this

support. Further, support for temporal qualifiers such as `at start`, `at end` and `over all` was also added, where each such qualifier enforces the temporal bounds when a literal takes effect.

```
(:types
    state - object
    location - object
    rail_block - discrete_reusable_resource
    item {?loc - location} - discrete_reusable_resource
    map {?vertices - [rail_block] ?edges - [[rail_block]]} - object
    request {?release_time - object ?due_date - object
        ?demand - object} - object
    robot {?block - rail_block ?st - state
        ?grasp - item} - discrete_reusable_resource
)
```

(a) Domain representation of typed objects.

```
(:object-instances
    box {loc = box_pick_loc}
    ur5A {block = blockA, st = home, grasp = empty}
    ur5B {block = blockD, st = home, grasp = empty}
    network {vertices = [blockA blockB blockC blockD blockE],
        edges = [(blockA blockB) (blockB blockA) (blockB blockC)
            (blockC blockB) (blockC blockD) (blockD blockC)
            (blockD blockE) (blockE blockD)]}
    requestA {release_time = 0, due_date = 300,
        demand = [move_item box box_drop_loc]}
)
```

(b) Domain representation of typed objects instantiations.

Figure 4.3: Domain representation of typed objects where `state`, `location` and `rail_block` is a normal typed object whereas `robot` etc. are complex typed objects which support individual attributes.

2. In addition, we also added support for complex data types. We took some inspiration from the Action Notation Modeling Language (ANML) [27] specification, which supports resources and complex objects as well. With this support, one can define the attributes of each typed object in the domain representation, which helps in providing a more accessible and modular way to express the spatial constraints of that typed object. A planner can further utilize such constraints to make informed decisions regarding the object's state,

mainly if that object is of a resource type. Since timelines are only associated with resource types, their corresponding attributes are also maintained on the same timeline. This approach is slightly tangential to the perspective taken by traditional timeline-based planning frameworks since they tend to keep separate timelines for each state variable. The proposed approach provides an efficient way to maintain resource states on a single timeline with an implicit trade-off for computing the required states.

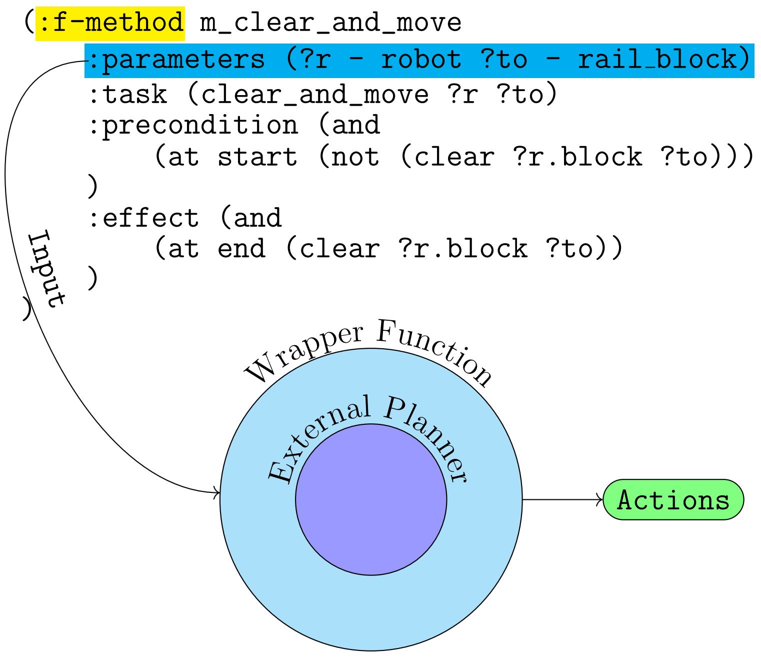
Additionally, to support the assignment of the attributes of these complex objects, we also added an assignment operator to the domain specification. When this assignment operator is used in conjunction with precondition predicates, it acts as an equality check. Still, when utilized in conjunction with effect predicates, then it serves as the regular assignment operator. Figure 4.3 shows an example of a complex data type and its corresponding instantiation in the domain representation. Further, we added support for *request* type objects, which provides release time, due date, and demand attributes to aid in providing a streamlined way to specify system goals to a planner. The release time and due date attributes enforce a global constraint on the goal specification. On the other hand, the demand attribute currently only supports task specification, which is internally tied to method specifications.

As shown in Figure 4.3 we also added initial support for discrete, reusable resources. These types of resources are consumed at the beginning of action but are given back at the end of the said action. For this initial implementation, we modeled these resources to be unit-capacity resources, but they could be multi-capacity as well.

3. One challenge within the planning process for multi-agent systems is the embedded combinatorics of enforcing global resource constraints, such as access to a shared rail in the problem shown in Figure 4.1. Consider a larger scenario where several robots are mounted on a common rail network with multiple paths, or equivalently where any set of heterogeneous robotic systems (walkers, freeflyers, etc.) are operating in a common confined space. As any given robotic agent attempts a movement in pursuit of carrying out its currently assigned task, there may be collaborative moves that several other robots must take to open

```
(:f-predicates
    (clear ?from ?to - rail_block)
)
```

(a) Domain representation of functional predicates.



```
(:f-method m_clear_and_move
    :parameters (?r - robot ?to - rail_block)
    :task (clear_and_move ?r ?to)
    :precondition (and
        (at start (not (clear ?r.block ?to)))
    )
    :effect (and
        (at end (clear ?r.block ?to))
    )
)
```

(b) Domain representation of functional methods.

Figure 4.4: Domain representation of functional predicates and methods. Using the working example as described in Figure 4.1 `clear` predicate returns true when the path between `from` and `to` parameters is empty while the `clear_and_move` method defines the task network for clearing and moving a blocking robot out of the way of the calling robot.

up a path and enable this movement. As the number of alternative paths that might be considered as options increases, the complexity of finding good ones via simple backward reasoning will quickly become overwhelming. One alternative approach is to rely on more specialized procedures for coping with enforcement of such global constraints to overcome this challenge. For example, the path clearing problem that was just described can be seen as an instance of the MAPF problem mentioned before, and there are now several algorithms for solving this problem in a globally optimal (or near-optimal) way. To accommodate use of such specialized procedures, we added two different functional constructs: (1) The functional predicate definitions with `f-predicate` and (2) The functional methods definitions with `f-method` keywords. Using these constructs, one can tie specialized external planners to the domain representation based on their unique name identifiers, which gives an added benefit for the planning and scheduling algorithm to focus more on allocating resources and tasks rather than trying to satisfy some specific problem whose optimal solution can be found by leveraging existing literature. Functional methods are slightly different in their representation than standard methods in terms of their defined task network. Since they are connected to a specialized external planner, we provide a simple wrapper function whose name is restricted to be the same as the corresponding functional method. Its purpose is to first convert the input from the parsed format to a setup that the specialized planner's API supports and then convert the output of the specialized planner to a sequence of actions along with their temporal constraints so that it can be tied to the task network. Figure 4.4 showcases an example where such constructs can be found useful.

4. Finally, we also added support for synchronization rules to the domain representation via a new construct called `sync-constraints`. This construct can be added as part of any standard method definition and currently supports two different types of Allen relations, namely, *meets* and *before*. Task A is said to *meet* another task B when task B is supposed to be executed immediately after the execution of task A has been completed. Similarly, task A is said to be *before* another task B when task B can be performed any time after task A as long as it is within the prescribed bounds of the synchronization constraint. Figure 4.5 provides a small example of how these relations can be used within

```
(:method m_pick_item
    :parameters (?r - robot ?i - item)
    :task (pick_item ?r ?i)
    :precondition (and
        (at start (safe_state ?r.st))
        (at start (not (= ?r.grasp ?i)))
        (at start (reachable ?r.block ?i.loc))
    )
    :effect (and
        (at end (= ?r.grasp ?i))
        (at end (= ?i.loc grasped))
    )
    :ordered-subtasks (and
        (task0 (grasp ?r ?i))
        (task1 (move_to_home_state ?r))
    )
    :sync-constraints (and
        (task0 meets task1)
    )
)
```

(a) Example of `meets` constraint

```
(:method m_move_item
    :parameters (?r - robot ?i - item ?drop_loc - location)
    :task (move_item ?i ?drop_loc)
    :precondition (and
        (at start (safe_state ?r.st))
        (at start (not (= ?r.grasp ?i)))
    )
    :effect (and
        (at end (= ?i.loc ?drop_loc))
    )
    :ordered-subtasks (and
        (task0 (pick_item ?r ?i))
        (task1 (drop_item ?r ?i ?drop_loc))
    )
    :sync-constraints (and
        (task0 before {0 +inf} task1)
    )
)
```

(b) Example of `before` constraint

Figure 4.5: Domain representation of `meets` and `before` synchronization constraints.

the domain representation.

With these modifications in place, we built a custom parser which understands these new constructs on top of an existing parser that supports HDDL using Bison and Flex libraries to construct internal planning data structures which aid the search process. Based on this specification, we made certain assumptions. First, we assumed the existence of hierarchical task models expressed at HTNs for achieving various goals (task requests) that might be received. Hence, instead of reasoning from the finite state machines specified for individual state variables to determine what sequences of tokens need to be inserted on their timelines to achieve a given goal state, as is typically done in timeline-based planning, relevant task models are instead retrieved and used to determine the sequence(s) of tokens (i.e., primitive tasks) and the constraints between them (i.e., causal relations, synchronization rules) that need to be feasibly situated in time to produce an executable plan. A second departure from typical domain modeling assumptions is taking a streamlined approach to timeline (or state variable) definition. Specifically, timelines are only associated with required resources and must be allocated to perform a given task (or subtask) in the task model that is being applied. HTN task models can designate resource requirements at different hierarchy levels, indicating the scope over which they must be allocated.

Working on the running example as shown in Figure 4.1, we modeled the domain representation for this case using 4 different primitive actions namely,

- `rail_move` which deals with moving a particular robot along the mounted rail network one rail_block at a time.
- `move_to_home_state` which changes the state configuration of the robot such that its collision box is minimally intrusive.
- `grasp` primitive models the grasping action of the robotic manipulator for any object.
- `release` primitive models the ability of the robotic manipulator to drop an already grasped object to a specified location.

In addition to the above-mentioned action primitives, we also modeled some high-level tasks to showcase how HTNs can be leveraged within timeline-based planning frameworks for efficient plan generation. To this effect, 3 different high-level tasks were added, namely,

- `pick_item` task involves grasping a specified object followed by making sure that the robotic manipulator changes its joint configuration to a “safe” state.
- `drop_item` tasks models the high-level action of releasing a grasped object by the robotic manipulator and then again going back to the “safe” state.
- `move_item` task models the overall action of a robotic manipulator that intends to move a specified object from one place to another.

4.2 Plan generation search procedure

Given the domain representation, we designed and developed a novel algorithm for generating and feasibly inserting a new task plan into a global plan/schedule of habitat activities. Central to this algorithm (different from prior timeline-based planning frameworks) is the forward scanning procedure for scheduling instantiated task networks on resource timelines and determining feasible intervals for executing the task with the resources that have been assigned. This ability to take advantage of the structure provided by pre-specified task models is key to achieving overall planning/scheduling efficiency in multi-agent domains.

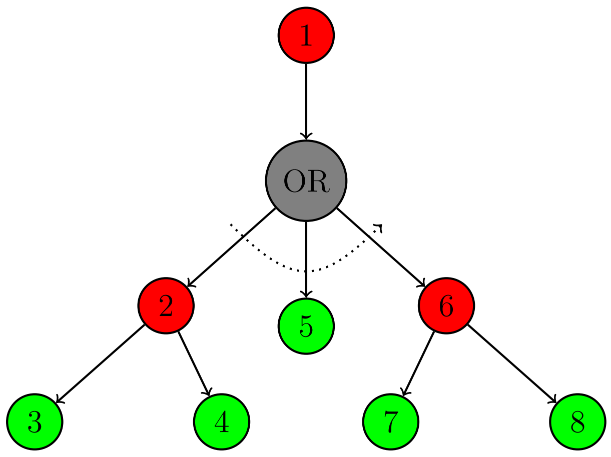


Figure 4.6: Figure 2.1 represented with the variant of AND/OR tree where green nodes are the leaves while red nodes are the internal high-level tasks and the OR node is showcased in gray. Since traveling from Pittsburgh to Washington D.C can be done in three separate ways, the first node introduces an OR node which further branches into their corresponding alternative decompositions as defined by the HTN.

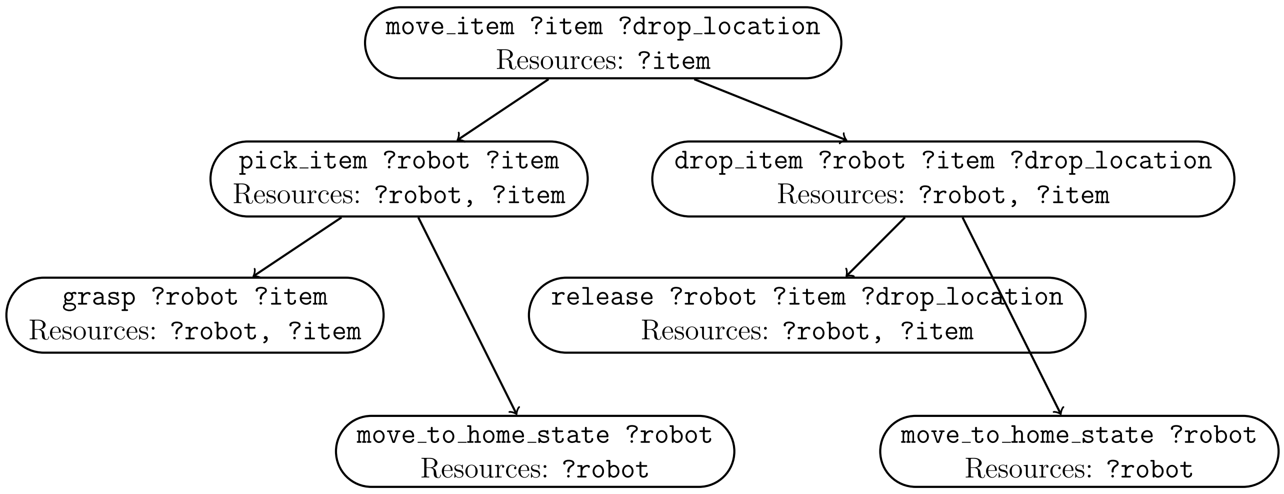


Figure 4.7: Generated path decomposition tree for the working example as described in Figure 4.1

The first step towards generating a feasible plan is to process the incoming request and map it to a known task model to create a *path decomposition tree* of all the potential possible alternatives. Since the demand in an incoming request maps to a particular high-level task, that task can be further implemented via different methods that provide alternative decompositions of the input task. To generate the resulting path decomposition tree, we build a variant of the AND/OR tree where each internal node within the tree represents either an OR node for the higher-level parent task while the leaves represent the action primitives. The OR nodes have a special meaning where their child nodes represent all the possible alternative decompositions of its parent node. Hence, whenever a high-level task has multiple associated method decompositions defined in the domain model represented as an HTN, then a corresponding OR node is introduced into the path decomposition tree as shown in Figure 4.6. As defined, it must be noticed that an OR node can never be a leaf of such a tree just based on the defined construction property. The creation of this tree follows from the simple recursive expansion of the method definitions till the action primitives are hit, at which point the growth terminates. Figure 4.7 showcases the generated path decomposition tree for the working example where the incoming request's demand pertains to moving the red cube from its original location to the near edge of the table.

Algorithm 1 Algorithm to return all the possible instantiations of a given path decomposition tree.

```
procedure ENUMERATE DECOMPOSITIONS(vertex)
    Initialize empty leafs vector
    if vertex is a leaf then
        Add vertex to leafs
        return leafs
    else if vertex is OR then
        for all children $c$ of vertex do
            leafs += ENUMERATE DECOMPOSITIONS(c)
        end for
    else
        Initialize empty op vector
        for all children $c$ of vertex do
            op += ENUMERATE DECOMPOSITIONS(c)
        end for
        leafs = cartesian_product(op)
    end if
    return leafs
end procedure
```

Once the path decomposition tree is generated, the next step is to enumerate all the possible decompositions based on which we execute the grounding process. This process allows one to list all the decompositions by combinatorically expanding on the existing OR nodes. Algorithm 1 provides a high-level recursive algorithm that enables one to do so in an efficient manner while guaranteeing that none of the possible alternatives are skipped. This algorithm follows from a simple depth-first search procedure where at each level, we keep track of the choices that were made till there and then recursively maintain a cartesian product of all such decisions. The worst-case time complexity of Algorithm 1 is dominated by the size of the cartesian product, which is being computed in Line 15.

After this, for each possible alternative decomposition, we instantiate a task network, tied with an underlying STN, and composed of the tokens that correspond to the high-level tasks or action primitives. Each token constitutes a start and end time point for a particular task. It consists of information regarding the potential parameter assignments as well. If the token corresponds to an action primitive,

Algorithm 2 Algorithm to return an instantiated task network connected to an underlying STN that enforces the pre-specified temporal constraints.

```
 1: procedure EXPAND(vertex, tree, STN)
 2:     Create a token for vertex
 3:     Connect token to tree
 4:     if vertex is not leaf then
 5:         for all children $c$ of vertex do
 6:             EXPAND(c, tree)
 7:         end for
 8:         Add synchronization constraints to the STN
 9:         Add contains constraints to the STN
10:     end if
11:     return tree
12: end procedure
13: procedure INSTANTIATE TASK NETWORK(search_tree, STN)
14:     Instantiate an empty tree
15:     tree $\leftarrow$ EXPAND(search_tree.root, tree, STN)
16:     Add the release time constraint to the search_tree.root
17:     Add the due date constraint to the search_tree.root
18:     return tree
19: end procedure
```

a corresponding duration constraint is enforced between the start and end time points. This task network forms the fundamental structure with which our developed framework, `T-HTN` first grounds the unassigned variables and then utilizes it to search for “slots” on the required resource timelines where the instantiated tokens can then be feasibly inserted. As described previously, timelines are composed of tokens and a *sequencing* constraint between a pair of tokens on the same timeline are enforced to maintain their ordering within the underlying STN. This construction of the task network follows a similar depth-first search procedure where each node represents a token. A token expansion involves creating corresponding child tokens and establishing the pre-specified temporal constraints. Additionally, all the child tokens inherit the precondition and effect literals of their parent tokens. The temporal constraints include the synchronization rules such as `meets` and `before` constraints that have been introduced earlier.

Further, we enforce a *contains* temporal constraint between each of the high-

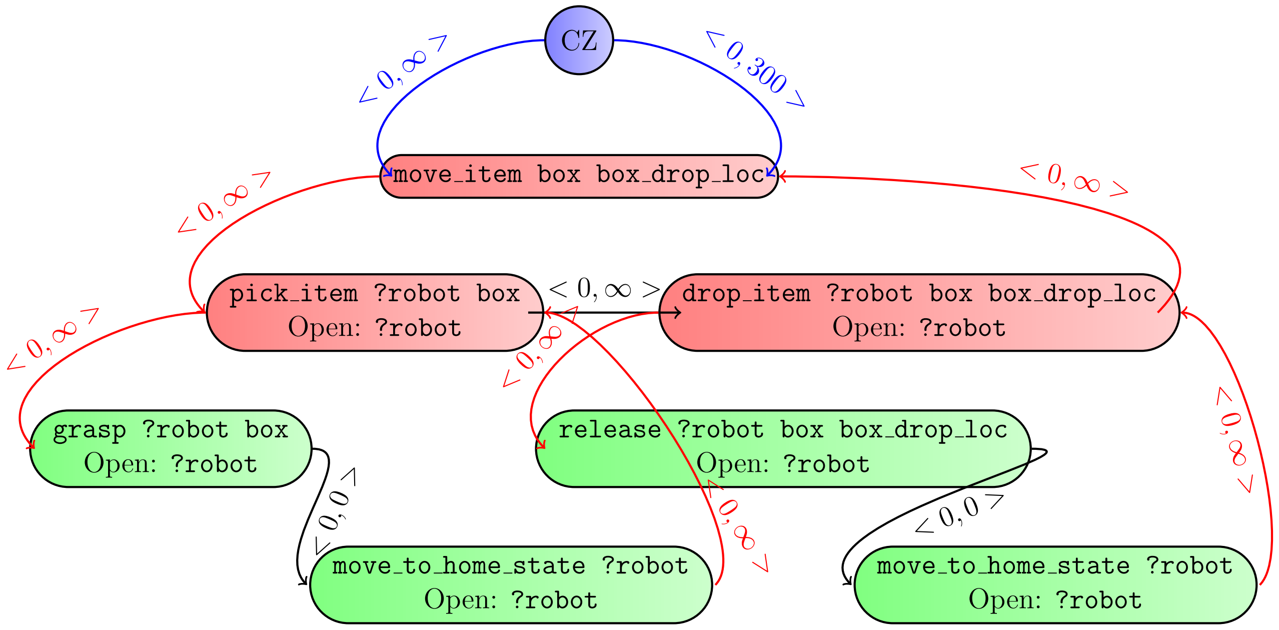


Figure 4.8: Instantiated task network for the working example as described in Figure 4.1. Nodes in red represent the high-level tasks, and the nodes in green represent the action primitives. The blue edges correspond to the release time and due date constraints, while the red edges correspond to the contains constraint. The black edges correspond to the pre-specified synchronization rules.

level tokens, and their corresponding decompositions so that the two levels are tied together. Such constraints ensure that the rules imposed on the aggregate tasks are also executed on the constituent tasks, propagating constraints further down the network. If the constituent tasks of an aggregate task are known to be *ordered*, then we only need to enforce the *contains* constraint between the start time point of the parent task to the start time point of the first task in ordered decomposition and the end time point of the last child in the ordered decomposition to the end time point of the parent task. Once all the tokens have been generated, the whole network is connected to the calendar zero temporal reference point based on the release time and due date constraints specified by the incoming request. To efficiently compute all the resource and parameter assignments, we segregate them based on whether they have been already assigned or not. The ones that were assigned move to a *closed* set while the rest move to the *open* set. A cartesian product is then computed on all the possible assignments of the *open* set parameters and is used to ground the generated task network. Algorithm 2 provides a high-level overview of how task network generation proceeds. Assuming that there can be V nodes in the generated tree, the worst-case time complexity of the algorithm is $O(V)$ since each vertex needs to be visited at least once so that it can be part of the task network. Figure 4.8 shows the generated task network for the working example as described in Figure 4.1.

Once the instantiated task network is built, it acts as a template for all the possible resource and parameter assignments. For each possible combination of resource assignments, `T-HTN` determines a set of feasible slots by a forward timeline scanning process that repeatedly attempts to forward schedule the leaf tasks in the instantiated task network at each possible start point. The timeline scanning process iterates through all the required resource timelines in order. It identifies slots on those timelines characterized by the previous and next tokens that are already placed on the timeline. The search of slots is pivoted around the resource of type *robot* while considering all the other resources as dependent resources. The leaf tasks of the instantiated task network are only deemed to be added on the *robot* timelines.

In contrast, their corresponding tokens for the other required resources are added on the dependent resource timelines. For each combination of slots identified on the *robot* and dependent resource timelines, we query the underlying STN to check whether the previous token's earliest finish time is smaller than the leaf task's earliest start time and the leaf task's earliest finish time is smaller than the next task's earliest start time. In addition, it is also checked whether the leaf task's duration constraint can fit in a slot based on the next token's latest start time and the previous token's earliest finish time. This process helps prune many infeasible combinations of slots early on, thereby speeding up the process significantly. Following these checks, we construct a coherent world state of the environment by iteratively modifying the initial world state with the effect literals encapsulated by the currently considered slots. This updated world state is utilized to check the preconditions of the leaf task, and assuming that all the precondition literals pass, the algorithm goes ahead and enforces the temporal constraints imposed by the leaf task's token on the underlying STN. These temporal constraints include first removing a sequencing constraint between the previous token's end time point and the next token's start time point, followed by adding two new sequencing constraints between the previous token's end time point with leaf task token's start time point and leaf task token's end time point with next token's start time point. Once all the leaf tasks of the instantiated task network have been scheduled, `T-HTN` logs the slots' information and continues the same procedure for other possible resource and parameter assignments. Finally, once all options have been generated, they are evaluated according to some set of objective criteria (e.g., minimize overall task makespan, complete task as early as possible, reduce disruption to plans of other robots), and `T-HTN` commits to the best option.

Note that sometimes it is hard to encapsulate the entire precondition check within single or multiple literals. By utilizing the functional predicate constructs introduced earlier, we use specialized algorithms to compute such prerequisite checks efficiently and optimally. Hence in the process of checking preconditions against an updated world state, if any precondition literal fails either via a functional predicate call or via a violation of a constant literal, we attempt to satisfy such failing preconditions in two different ways. First, we iterate through the list of action primitives and identify potential actions whose effect literals match the failing precondition literal. If we finds such an alternative, the newly instantiated tokens are then added to the same task network, and the procedure continues normally. Second, we iterate through the list of functional methods and identify potential solutions whose effect literals match the failing precondition literal. If we find such a method, we call the specialized planner associated with that functional method. We update the task network by adding all

Algorithm 3 Algorithm to find a set of feasible slots for a given instantiated task network while attempting to satisfy any failing precondition literals.

```
procedure SATISFY PRECONDITION(literal, task_network, STN)
    Find the satisfying task tk
    Identify tk's required resources and parameter assignments
    if tk is an action primitive then
        Create token corresponding to tk
        Add token to task_network
        Add temporal constraints of token to STN
    else
        Call the specialized external planner which returns a set of tokens
        for all token in tokens do
            Add token to task_network
            Add temporal constraints of token to STN
        end for
    end if
end procedure
procedure FIND SLOTS(task_network, STN)
    Instantiate an empty slots data structure
    for all Leaf tasks tk in task_network.leafs do
        Collect tk's required resource $\mathcal{R}$ assignments
        for all Slots s over $\mathcal{R}$ do
            Check temporal bounds of s using STN
            Compute the world state ws using s
            for all Precondition literal p over tk do
                if p fails against ws then
                    SATISFY PRECONDITION(p, task_network, STN)
                end if
            end for
            Enforce temporal constraints of tk onto the STN
            slots += s
        end for
    end for
    return slots
end procedure
```

the newly generated tokens, after which the procedure continues normally. Whenever T-HTN finds an alternative action primitive or method, it validates that alternative by comparing it's set of precondition literals with the updated world state. If there is any violation, T-HTN continues to look for other alternatives until they are exhausted. Since this process can potentially lead to infinite cycles, we employ a conservative approach where the operation of satisfying preconditions is terminated after the first recursive level. Algorithm 3 provides a high-level overview of the outlined search procedure. Assuming that there are T leaf tasks to be scheduled and at most N potential slots for each such task, then the worst-case complexity of the algorithm is $O(TN)$ which is going to be heavily dominated by the size of slots since as each leaf gets scheduled $N \gg T$.

To summarize, our developed framework, T-HTN combines Algorithms 1, 2 and 3 to satisfy any incoming request given a set of timelines tied to an underlying STN. It first parses the incoming request and generates a corresponding path decomposition tree which gets processed by Algorithm 1 to generate all the possible alternative decompositions. Each decomposition is then passed to Algorithm 2 to form a corresponding task network that enforces all the relevant temporal constraints on the underlying STN. The generated task network is then passed to Algorithm 3 which finds a set of feasible slots on the required resource timelines while attempting to satisfy any failing precondition literals. Since Algorithms 2 and 3 are repeated for all possible decompositions, the overall complexity of T-HTN also depends on the total number of such possible decompositions as defined by the input HTN. Assuming that there are at most D such decompositions, the worst-case complexity of our approach is $O(DTN)$ which is heavily dominated by Algorithm 3.

Looking back at the working example, assuming the robot parameter assignment in the corresponding task network shown in Figure 4.8 was UR5A, it is clear to observe that UR5B must move out of the way for the actions to take place successfully. This means that when our framework tries to schedule the instantiated task network with UR5A as the robot parameter assignment, it triggers a satisfying precondition procedure which calls m_clear_and_move functional method. This method is internally tied to a specialized external planner that computes the path for the offending robot to move out of the way of a given path for the calling robot. Moreover, to pick the box, UR5A is first expected to be close to the object before attempting the grasp.

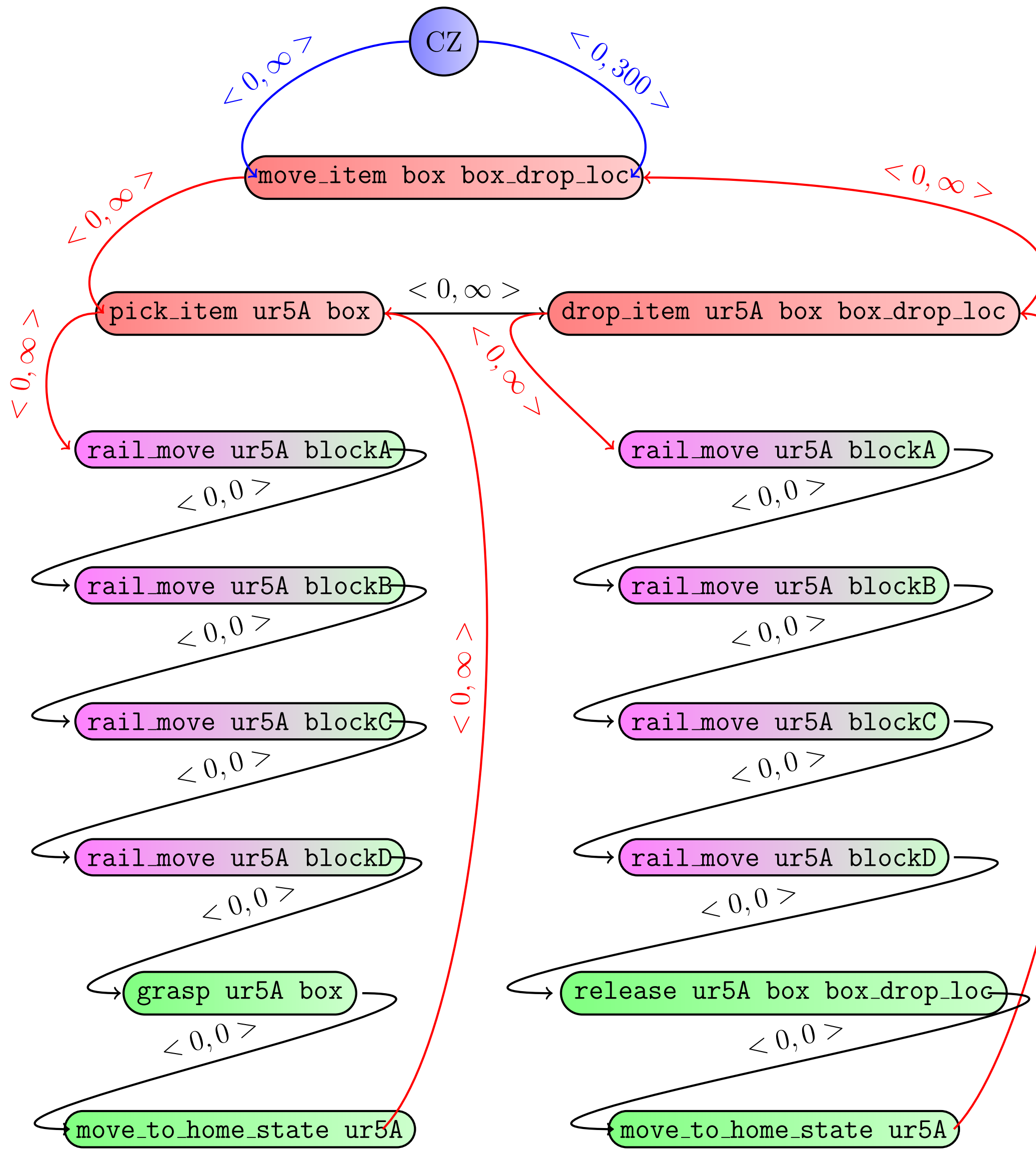


Figure 4.9: Updated instantiated task network for the working example as described in Figure 4.1. Nodes in red represent the high-level tasks, and the nodes in green represent the action primitives. The nodes in magenta were added to the original task network by a specialized planner who was triggered by `T-HTN` to satisfy a failing precondition. The blue edges correspond to the release time and due date constraints, while the red edges correspond to the contains constraint. The black edges correspond to the pre-specified synchronization rules.

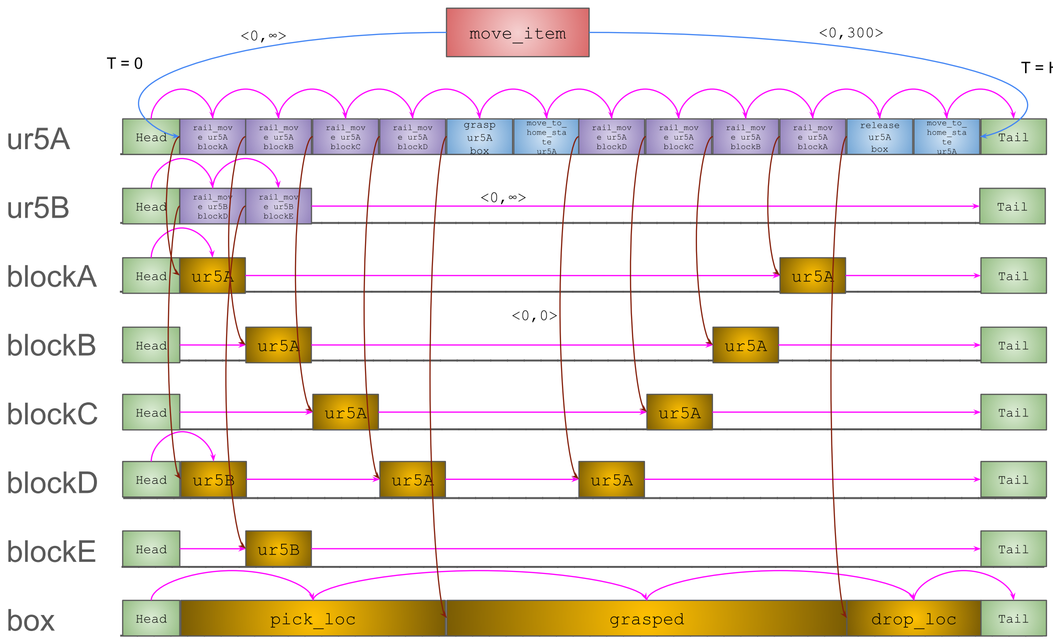


Figure 4.10: Final snapshot of the timelines generated by `T-HTN` in response to scheduling the working example outlined in Figure 4.1. The red node encapsulates the task network, which was shown in Figure 4.9 and the blue arrows signify the release time and due date constraints. The pink arrows relate to the $\langle 0, \infty \rangle$ sequencing constraint. In contrast, the brown arrows mark the $\langle 0, 0 \rangle$ contains constraint that joins the tokens on the *robot* and corresponding dependent timelines. The head and tail tokens on all the timelines act as auxiliary tokens, which do not have any significance apart from helping in a coherent token insertion procedure.

This prerequisite condition fails, triggering another `reachable` functional method that is also tied to another specialized external planner that computes the path of the calling robot to the required destination. This results in the generation of an updated task network, which is shown in Figure 4.9.

Finally, the overall plan/schedule of activities is generated, extended, and managed incrementally over time as new pending requests and unexpected execution results that require re-planning are received. New tasks are allocated to specific resources and added to the overall plan as new requests are received. In some cases, the remaining temporal flexibility in the current plan/schedule (or equivalently the continuing availability of the resources that need to be allocated to accomplish the task) will seamlessly accommodate additional requests. In other, more resource-constrained situations, the addition of new tasks may cause the removal or delay of some less important, previously scheduled tasks. In re-planning settings, it may also be the case that some previously planned/scheduled tasks may no longer be relevant and can be retracted to create resource availability for performing corrective tasks. This inherently incremental search approach to overall planning and scheduling of habitat activities provides both the temporal flexibility to allow individual robotic systems to locally respond to execution anomalies if possible, while also providing the ability to rapidly re-plan from a more global perspective with attention to minimizing the level of disruption to initially planned activities. Figure 4.10 showcases the final snapshot of the timelines generated by our developed framework, `T-HTN` as a result of scheduling the use-case scenario outlined in the working example.

Chapter 5

Demonstrations

In this chapter, we will take a closer look at how we execute the working example as described in Figure 4.1 within a simulation environment. Further, we will present the output specification for that particular use-case scenario that is fed to a simple execution engine that can carry out the generated plan by scanning the timelines. Although the working example only deals with a single incoming request, to establish the efficiency and correct implementation of the developed framework, we also present a multi-request variant of the same working example. For these experiments, `T-HTN` utilizes an objective metric to prioritize plans that complete a given request as early as possible. The entire framework, including the domain representation parser and the developed planner, was built in C++ for efficient code execution. Moreover, to showcase the superiority of our developed framework, we compare its performance with another temporal planner called POPF [10] that belongs to the action-based planning paradigm. POPF is a forwards-chaining temporal planner built on the foundations of grounded forward search in combination with linear programming to handle continuous linear numeric change. With default specifications, POPF tries to generate plans that are optimized with respect to its overall makespan.

5.1 One request case

Going back to the working example as described in Figure 4.1, once `T-HTN` schedules the corresponding task network as shown in Figure 4.9, our execution engine iterates through the resource timelines and maintains a priority queue of tokens that are ordered based on their earliest start times as queried by the underlying STN. Using that priority queue, the execution engine carries out all the required tokens in order. This process involves connecting the action primitives with the motion primitives implemented via Moveit! Library [9] working internally on the robot in a simulated environment. RViz was chosen to visualize robotic activities as it acts as an excellent 3D visualizer for the Robot Operating System (ROS). Figure 5.1 outlines the output specification of `T-HTN` for this particular use-case scenario. This output specification includes the resource perspective of the generated plan where tokens on each resource timeline are presented in order. Each token is further characterized by its corresponding start and end time points, which are mapped to an underlying STN. Figure A.1 in Appendix A showcases the resultant STN generated by `T-HTN` while planning for this particular use-case scenario. The earliest start time and latest finish time of each time point are also reported within this output plan specification.

Listing 5.1: The output plan specification generated by `T-HTN` for the working example described in 4.1

```
"timelines": {
    "blockA": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start58: [0.00, 60.00]",
                "rail_block_ur5A_occupied_end59: [20.00, 80.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start90: [180.00, 240.00]",
                "rail_block_ur5A_occupied_end91: [200.00, 260.00]"
            ]
        }
    ],
    "blockB": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start60: [20.00, 80.00]",
                "rail_block_ur5A_occupied_end61: [40.00, 100.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start88: [160.00, 220.00]",
                "rail_block_ur5A_occupied_end89: [180.00, 240.00]"
            ]
        }
    ],
    "blockC": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start62: [40.00, 100.00]",
                "rail_block_ur5A_occupied_end63: [60.00, 120.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start86: [140.00, 200.00]",
                "rail_block_ur5A_occupied_end87: [160.00, 220.00]"
            ]
        }
    ],
    "blockD": [
        {
            "ur5B_occupied": [
                "rail_block_ur5B_occupied_start70: [0.00, 100.00]",
                "rail_block_ur5B_occupied_end71: [20.00, 120.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start64: [60.00, 120.00]",
                "rail_block_ur5A_occupied_end65: [80.00, 140.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start84: [120.00, 180.00]",
                "rail_block_ur5A_occupied_end85: [140.00, 200.00]"
            ]
        }
    ],
    "blockE": [
        {
            "ur5B_occupied": [
                "rail_block_ur5B_occupied_start72: [20.00, inf]",
                "rail_block_ur5B_occupied_end73: [40.00, inf]"
            ]
        }
    ],
    "box": [
        {
            "ur5A_grasped": [
                "item_ur5A_grasped_start48: [80.00, 140.00]",
                "item_ur5A_grasped_end49: [110.00, 170.00]"
            ]
        },
        {
            "box_drop_loc_released": [
                "item_box_drop_loc_released_start74: [200.00, 260.00]",
                "item_box_drop_loc_released_end75: [230.00, 290.00]"
            ]
        }
    ],
    "ur5A": [
        {
            "rail_move ur5A blockA": [
                "robot_rail_move_start50: [0.00, 60.00]",
                "robot_rail_move_end51: [20.00, 80.00]"
            ]
        },
        {
            "rail_move ur5A blockB": [
                "robot_rail_move_start52: [20.00, 80.00]",
                "robot_rail_move_end53: [40.00, 100.00]"
            ]
        },
        {
            "rail_move ur5A blockC": [
                "robot_rail_move_start54: [40.00, 100.00]",
                "robot_rail_move_end55: [60.00, 120.00]"
            ]
        },
        {
            "rail_move ur5A blockD": [
                "robot_rail_move_start56: [60.00, 120.00]",
                "robot_rail_move_end57: [80.00, 140.00]"
            ]
        },
        {
            "grasp ur5A box": [
                "robot_grasp_start38: [80.00, 140.00]",
                "robot_grasp_end39: [110.00, 170.00]"
            ]
        },
        {
            "move_to_home_state ur5A": [
                "robot_move_to_home_state_start40: [110.00, 170.00]",
                "robot_move_to_home_state_end41: [120.00, 180.00]"
            ]
        },
        {
            "rail_move ur5A blockD": [
                "robot_rail_move_start76: [120.00, 180.00]",
                "robot_rail_move_end77: [140.00, 200.00]"
            ]
        },
        {
            "rail_move ur5A blockC": [
                "robot_rail_move_start78: [140.00, 200.00]",
                "robot_rail_move_end79: [160.00, 220.00]"
            ]
        },
        {
            "rail_move ur5A blockB": [
                "robot_rail_move_start80: [160.00, 220.00]",
                "robot_rail_move_end81: [180.00, 240.00]"
            ]
        },
        {
            "rail_move ur5A blockA": [
                "robot_rail_move_start82: [180.00, 240.00]",
                "robot_rail_move_end83: [200.00, 260.00]"
            ]
        },
        {
            "release ur5A box box_drop_loc": [
                "robot_release_start44: [200.00, 260.00]",
                "robot_release_end45: [230.00, 290.00]"
            ]
        },
        {
            "move_to_home_state ur5A": [
                "robot_move_to_home_state_start46: [230.00, 290.00]",
                "robot_move_to_home_state_end47: [240.00, 300.00]"
            ]
        }
    ],
    "ur5B": [
        {
            "rail_move ur5B blockD": [
                "robot_rail_move_start66: [0.00, 100.00]",
                "robot_rail_move_end67: [20.00, 120.00]"
            ]
        },
        {
            "rail_move ur5B blockE": [
                "robot_rail_move_start68: [20.00, inf]",
                "robot_rail_move_end69: [40.00, inf]"
            ]
        }
    ]
}
```

5.2 Multi-request case

Since the one request case starts with empty timelines, the developed framework is not put to the test since there is only one candidate slot to work with. Hence, to test the performance of `T-HTN` upon introduction of multiple requests, we designed a variant of the first use-case scenario where prior to moving the box, another request to transfer a can from the near edge of the table to the far edge of the table is imposed. Figure 5.1 shows a visual representation of the initial state of the world in a simulation environment. Following the methodology described in Chapter 4, `T-HTN` tries to satisfy the first request similar to the one request case, following which it tries to schedule the second request. In this case, due to a simplifying assumption that restricts interleaving task plan executions, there are two potential slots where the resultant task network can be placed based on the release time and due date constraints on the request: (1) Before the first request is executed. (2) After the first request is executed. `T-HTN` evaluates the first potential slot, finds that the generated plan violates certain spatial constraints imposed by the first request, and hence deems that slot to be infeasible. The second slot is then evaluated and found to be able to schedule that request successfully. After committing to the feasible slots, the same execution engine is invoked, and RViz is used to visualize the robotic activity as described previously. Figure 5.2 outlines the output specification of `T-HTN` for the multi-request variant. In addition, the corresponding STN that was generated while planning for this use-case scenario is included in Figure A.2 in the Appendix A.

Listing 5.2: The output plan specification generated by `T-HTN` for the multi-request variant example as described in Figure 5.1

```
"timelines": {
    "blockA": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start64: [40.00, 180.00]",
                "rail_block_ur5A_occupied_end65: [60.00, 200.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start112: [400.00, 540.00]",
                "rail_block_ur5A_occupied_end113: [420.00, 560.00]"
            ]
```

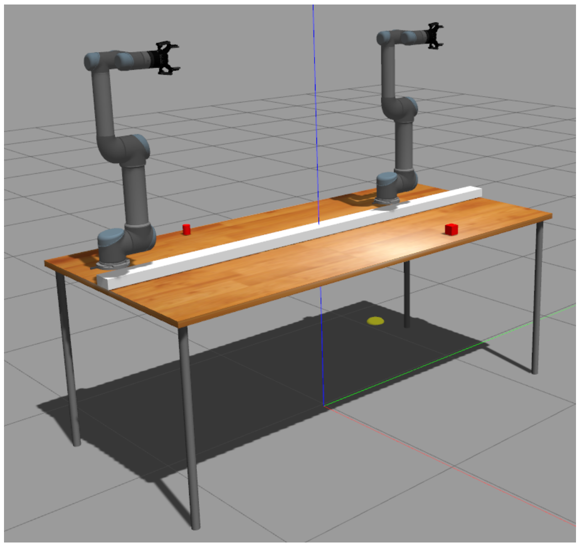


Figure 5.1: Two request variant of the use-case scenario that has been introduced before.

```
        }
    ],
    "blockB": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start66: [60.00, 200.00]",
                "rail_block_ur5A_occupied_end67: [80.00, 220.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start110: [380.00, 520.00]",
                "rail_block_ur5A_occupied_end111: [400.00, 540.00]"
            ]
        }
    ],
    "blockC": [
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start68: [80.00, 220.00]",
                "rail_block_ur5A_occupied_end69: [100.00, 240.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start108: [360.00, 500.00]",
                "rail_block_ur5A_occupied_end109: [380.00, 520.00]"
            ]
        }
    ],
    "blockD": [
        {
            "ur5B_occupied": [
                "rail_block_ur5B_occupied_start76: [0.00, 220.00]",
                "rail_block_ur5B_occupied_end77: [20.00, 240.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start70: [100.00, 240.00]",
                "rail_block_ur5A_occupied_end71: [120.00, 260.00]"
            ]
        },
        {
            "ur5A_occupied": [
                "rail_block_ur5A_occupied_start106: [340.00, 480.00]",
                "rail_block_ur5A_occupied_end107: [360.00, 500.00]"
            ]
        }
    ],
    "blockE": [
        {
            "ur5B_occupied": [
                "rail_block_ur5B_occupied_start78: [20.00, inf]",
                "rail_block_ur5B_occupied_end79: [40.00, inf]"
            ]
        }
    ],
    "box": [
        {
            "ur5A_grasped": [
                "item_ur5A_grasped_start94: [300.00, 440.00]",
                "item_ur5A_grasped_end95: [330.00, 470.00]"
            ]
        },
        {
            "box_drop_loc_released": [
                "item_box_drop_loc_released_start96: [420.00, 560.00]",
                "item_box_drop_loc_released_end97: [450.00, 590.00]"
            ]
        }
    ],
    "can": [
        {
            "ur5A_grasped": [
                "item_ur5A_grasped_start52: [0.00, 140.00]",
                "item_ur5A_grasped_end53: [30.00, 170.00]"
            ]
        },
        {
            "can_drop_loc_released": [
                "item_can_drop_loc_released_start54: [120.00, 260.00]",
                "item_can_drop_loc_released_end55: [150.00, 290.00]"
            ]
        }
    ],
    "ur5A": [
        {
            "grasp ur5A can": [
                "robot_grasp_start42: [0.00, 140.00]",
                "robot_grasp_end43: [30.00, 170.00]"
            ]
        },
        {
            "move_to_home_state ur5A": [
                "robot_move_to_home_state_start44: [30.00, 170.00]",
                "robot_move_to_home_state_end45: [40.00, 180.00]"
            ]
        },
        {
            "rail_move ur5A blockA": [
                "robot_rail_move_start56: [40.00, 180.00]",
                "robot_rail_move_end57: [60.00, 200.00]"
            ]
        },
        {
            "rail_move ur5A blockB": [
                "robot_rail_move_start58: [60.00, 200.00]",
                "robot_rail_move_end59: [80.00, 220.00]"
            ]
        },
        {
            "rail_move ur5A blockC": [
                "robot_rail_move_start60: [80.00, 220.00]",
                "robot_rail_move_end61: [100.00, 240.00]"
            ]
        },
        {
            "rail_move ur5A blockD": [
                "robot_rail_move_start62: [100.00, 240.00]",
                "robot_rail_move_end63: [120.00, 260.00]"
            ]
        },
        {
            "release ur5A can can_drop_loc": [
                "robot_release_start48: [120.00, 260.00]",
                "robot_release_end49: [150.00, 290.00]"
            ]
```

```
},
{
    "move_to_home_state ur5A": [
        "robot_move_to_home_state_start50: [150.00, 290.00]",
        "robot_move_to_home_state_end51: [160.00, 300.00]"
    ]
},
{
    "grasp ur5A box": [
        "robot_grasp_start84: [300.00, 440.00]",
        "robot_grasp_end85: [330.00, 470.00]"
    ]
},
{
    "move_to_home_state ur5A": [
        "robot_move_to_home_state_start86: [330.00, 470.00]",
        "robot_move_to_home_state_end87: [340.00, 480.00]"
    ]
},
{
    "rail_move ur5A blockD": [
        "robot_rail_move_start98: [340.00, 480.00]",
        "robot_rail_move_end99: [360.00, 500.00]"
    ]
},
{
    "rail_move ur5A blockC": [
        "robot_rail_move_start100: [360.00, 500.00]",
        "robot_rail_move_end101: [380.00, 520.00]"
    ]
},
{
    "rail_move ur5A blockB": [
        "robot_rail_move_start102: [380.00, 520.00]",
        "robot_rail_move_end103: [400.00, 540.00]"
    ]
},
{
    "rail_move ur5A blockA": [
        "robot_rail_move_start104: [400.00, 540.00]",
        "robot_rail_move_end105: [420.00, 560.00]"
    ]
},
{
    "release ur5A box box_drop_loc": [
        "robot_release_start90: [420.00, 560.00]",
        "robot_release_end91: [450.00, 590.00]"
    ]
},
{
```

```
            "move_to_home_state ur5A": [
                "robot_move_to_home_state_start92: [450.00, 590.00]",
                "robot_move_to_home_state_end93: [460.00, 600.00]"
            ]
        }
    ],
    "ur5B": [
        {
            "rail_move ur5B blockD": [
                "robot_rail_move_start72: [0.00, 220.00]",
                "robot_rail_move_end73: [20.00, 240.00]"
            ]
        },
        {
            "rail_move ur5B blockE": [
                "robot_rail_move_start74: [20.00, inf]",
                "robot_rail_move_end75: [40.00, inf]"
            ]
        }
    ]
}
```

5.3 Benchmarking

To benchmark `T-HTN`'s performance, we designed another multi-request variant of the original scenario, which involves moving random objects from one location to another in the presence of a rail network that acts as a shared global resource constraint between the two robotic manipulators. All requests are associated with moving a distinct and new object from its original spawn location to another prespecified location. Each request within the `T-HTN`'s domain specification is associated with a separate release-time and due-date constraints. However, within the POPF's domain specification, there is one global deadline enforced on all the requests as there is no support to specify such constraints for each goal separately. We compare the two planners based on three metrics that include measuring how long it takes to generate the plan, the number of actions used to satisfy all the incoming requests, and finally, the makespan of the final developed plan. Since our framework handles requests one at a time, we employed a MAPF based simple heuristic function over all the requests to order them such that the corresponding movement operations are minimized. On the other hand, POPF does not allow one to order its goal specifications, thereby limiting us from extending similar heuristics to its planner. We define the computation time of a particular planner for satisfying incoming requests as the elapsed time from the point of invoking the framework to the point of plan generation. Table 5.1 presents the results of our experiments and showcases the superiority of `T-HTN` when compared to POPF.

# Requests	Planner	Computation Time(s)	# Actions	Makespan
1	POPF	000.10	15	230
	T-HTN	**000.03**	**13**	**180**
2	POPF	001.17	33	590
	T-HTN	**000.02**	**22**	**480**
3	POPF	000.68	34	**430**
	T-HTN	**000.05**	**31**	780
4	POPF	002.28	42	**570**
	T-HTN	**000.04**	**40**	1080
5	POPF	163.90	53	**890**
	T-HTN	**000.08**	**49**	1340
10	POPF	NA	NA	NA
	T-HTN	**000.18**	**80**	**2780**
20	POPF	NA	NA	NA
	T-HTN	**000.52**	**159**	**5780**

Table 5.1: Comparison of `T-HTN` with POPF planner with respect to computation time, number of actions and the makespan of the generated plan.

Based on these results, we can see that for a smaller number of requests, `T-HTN` can completely outperform POPF in all the mentioned metrics, but as the number of requests increases, POPF can generate plans with better makespan. However, it is essential to note that the POPF planner cannot develop any viable plan for any problem involving more than five requests since it faces memory issues. On the other hand, `T-HTN` can handle a large number of requests very quickly with minimal performance drop in terms of computation time, thereby showing that `T-HTN` is very scalable. It further reinstates that our developed framework can handle large problems involving multiple robots under complex resource constraints. The only metric where `T-HTN` underperforms against the POPF planner is the makespan of the generated plans. We believe that such performance drop is due to our search procedure which returns as soon as a viable slot is found on the required resource timelines. This search strategy causes `T-HTN` to only ever use the first robotic manipulator for all the incoming requests since it can find a feasible set of slots for all requests with that first robot itself. Further, it is essential to note that we strictly enforced that no two requests are interleaved during execution within `T-HTN`. Still, POPF does not adhere to such constraints, which could also explain the blow-up in terms of the makespan of generated plans by `T-HTN`.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

In this thesis, we presented a comprehensive hybrid HTN-based planner/scheduler `T-HTN` that combines the structural power of HTN representation with the expressivity and flexibility of timeline-based planning frameworks from a goal regression and resource allocation perspective. Further, this thesis showcases `T-HTN`'s ability to enable global coordination of multiple robotic and autonomous systems (as well as human agents) in carrying out inspection, maintenance, and repair tasks on board a deep space habitat.

In this system, the HDDL domain representation language was extended to support multiple novel constructs to bridge the gap between action-based and timeline-based planning paradigms. Using the updated domain representation, we designed a planner/scheduler to effectively account for complex temporal and spatial constraints on task execution and extract significant inter-agent coordination constraints while providing individual agents with the flexibility to locally cope with some execution time uncertainty. Additionally, we proposed a novel extension to the HTN paradigm where one reasons about failing preconditions by constructing task networks that satisfy them, followed by a robust plan search procedure that incorporates specialized external planners to generate highly optimized plans.

Finally, we conducted initial testing in multi-robot scenarios involving two UR5 robot arms mounted on a shared rail network to evaluate the effectiveness of the proposed framework in managing resources that must be shared across multiple agents. We also demonstrated the superiority of our framework in comparison to another widely used temporal planner in terms of computation time, the number of actions utilized to generate the plan, and scalability.

6.2 Future Work

Several simplifying assumptions were made in the initial implementation of `T-HTN`, providing focal points for future work. First, the resource timelines were realized exclusively as single capacity resources (i.e., resources capable of doing just one task at a time). Although it seems pretty straightforward, one potential future direction is to extend resource timeline representations to accommodate multi-capacity robotic systems that can simultaneously accomplish multiple tasks (e.g., perform a visual inspection while carrying out a move-object request). Such support will require slight modification of the domain representation and handling of the corresponding resource profiles with the timelines. A second simplification that was made was to simulate the execution of a MAPF algorithm for clearing the path for a planned move (essentially hard-coding the expected path clearing movements it would produce to test and debug the procedure for handling preconditions within instantiated task networks). One obvious extension is to incorporate a state-of-the-art MAPF algorithm into `T-HTN` to explore complex scenarios where using such specialized planners will be advantageous. A third simplification that was made was to restrict `T-HTN` from interleaving execution of task plans. Accommodating multiple requests that allow for overlapping plan executions, although it could potentially lead to much more efficient task execution behavior, also runs the risk of search space explosion, which brings us to another critical extension that involves including domain-based heuristics to the search procedure to help prune infeasible alternatives early on to better tackle the combinatoric explosion of the search space.

Hence, we aim to continue testing with variants of dual UR5 robot arm problems, first to assess performance characteristics and subsequently to analyze and quantify the benefits of the current HTN-based planning approach. We will open-source the design and implementation of an API for communicating (1) assigned tasks/goals and (2) coordination constraints with other agents to individual robotic and autonomous systems, and for receiving execution status updates back from individual agents to enable a multi-level, reactive planning framework. Finally, in addition to the proposed research directions, we intend to benchmark `T-HTN` on a typical use-case scenario widely used within the planning community to generate more quantitative results. However, this process has two issues: (1) It requires mapping the existing problems into the new syntax required for `T-HTN`. (2) One also needs to extend these problems to reason over more complicated temporal constraints to highlight the capabilities of `T-HTN`. Lack of similar work within the planning community makes these extensions to the problems more challenging.

Appendix A

Appendix

Figure A.1 presents the final STN representation that was generated while planning for the one request use-case scenario. Figure A.2 offers the final STN representation that was developed while planning for the multi-request use-case scenario as presented in Chapter 5. The nodes in Figures A.1 and A.2 represent all the time points that were generated by `T-HTN` while the edges between pairs of nodes correspond to their temporal constraints. Each temporal constraint is marked with $\langle$ `lb`, `ub` $\rangle$ values which are enforced in accordance with the given domain representation.

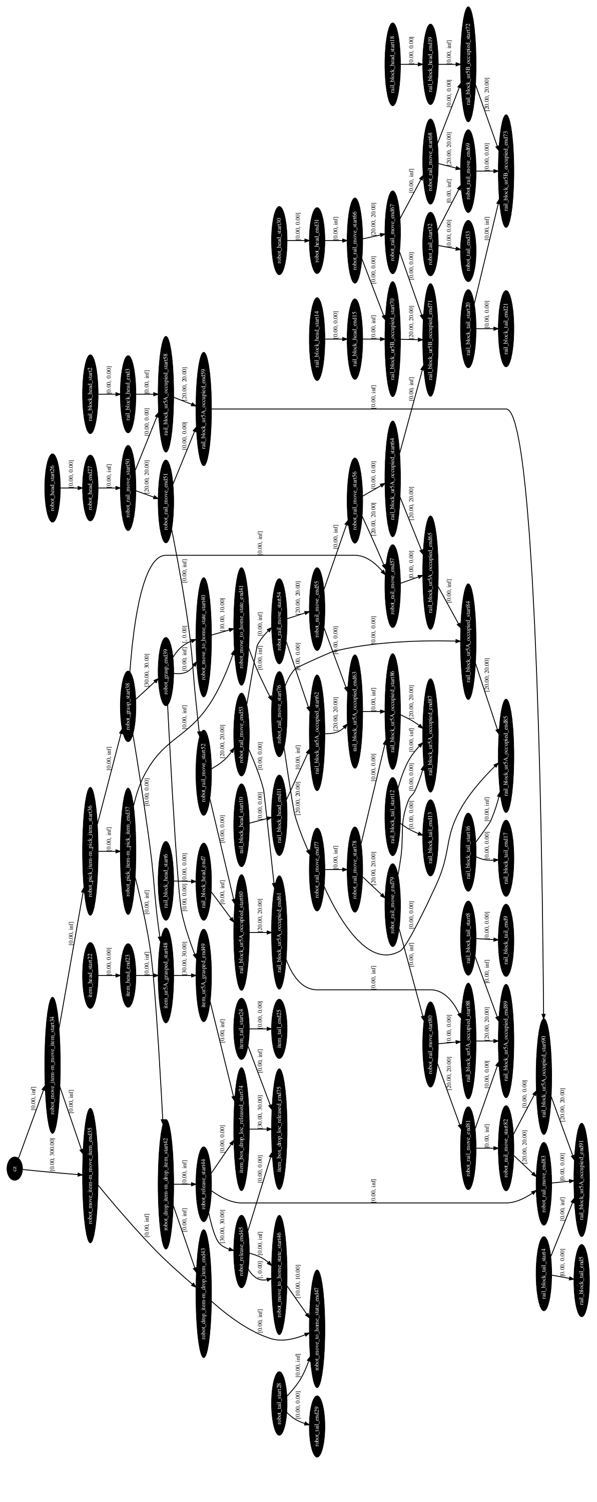


Figure A.1: STN for one request use-case scenario.

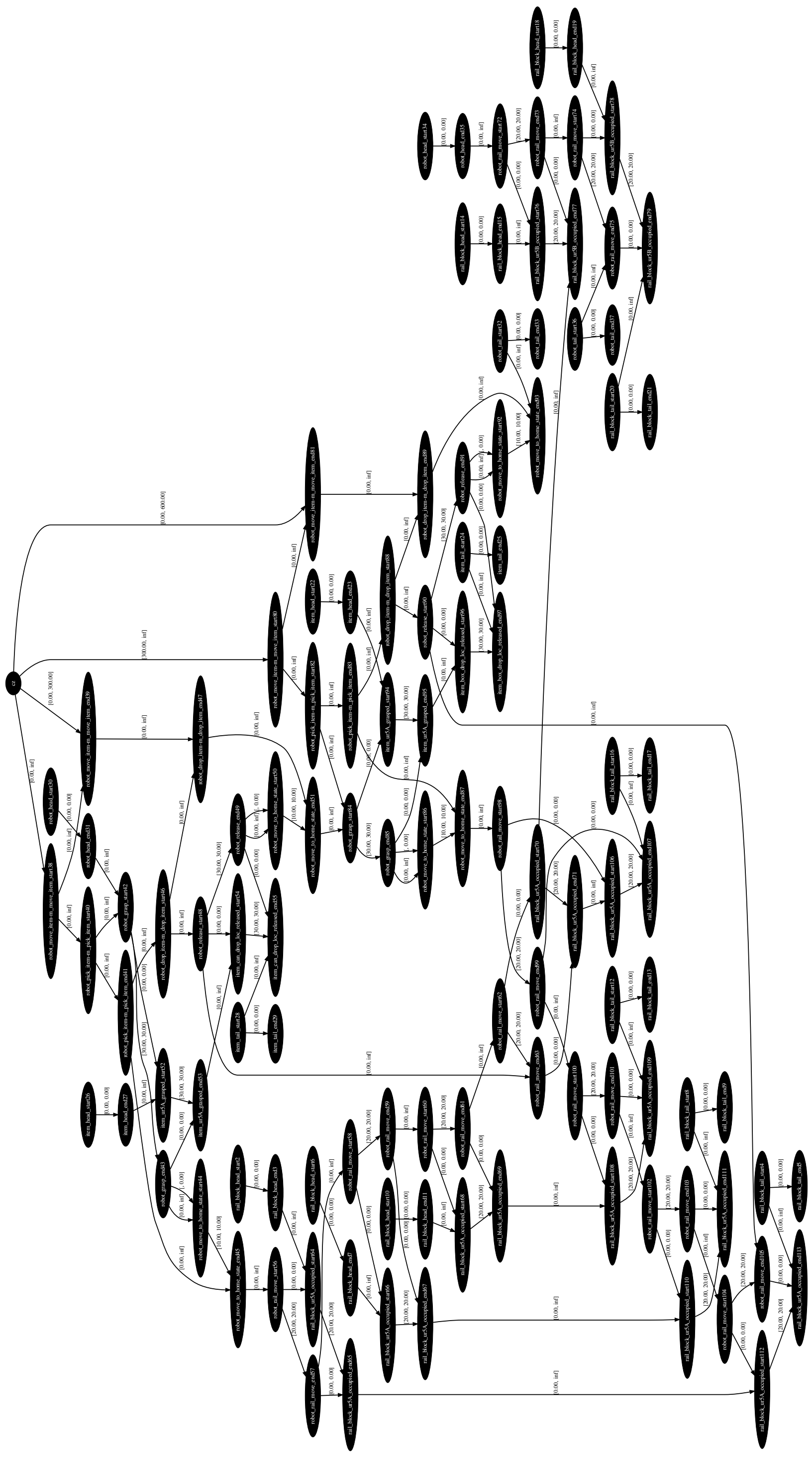


Figure A.2: STN for multi-request use-case scenario.

Bibliography

[1] David Aha. Goal reasoning: Foundations, emerging applications, and prospects. *AI Magazine*, 39(2):3–24, 2018. 1.1

[2] Shaun Azimi, Emma Zemler, and R Morris. Autonomous robotics manipulation for in-space intra-vehicle activity. In *Proceedings of the ICAPS Workshop on Planning and Robotics*, 2019. 3

[3] J. Barreiro, Matthew E. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, James C. Ong, E. Remolina, Tristan B. Smith, and D. Smith. Europa: A platform for ai planning, scheduling, constraint programming, and optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012. 2.4

[4] A. Bit-Monnot, Malik Ghallab, Felix Ingrand, and David E. Smith. Fape: A constraint-based planner for generative and hierarchical temporal planning. *ARXiv Preprint*, 2020. 4.1

[5] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):281–300, 1997. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(96)00047-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370296000471`. 2.1

[6] John L. Bresina and Paul H. Morris. Mixed-initiative planning in space mission operations. *AI Magazine*, 28(2), 2007. 1.1

[7] Michael Cashmore, M. Fox, D. Long, D. Magazzeni, Bram Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, 2015. 3

[8] Emma Zemler Shaun Azimi Kevin Chang and Robert A Morris Jeremy Frank. Integrating task planning with robust execution for autonomous robotic manipulation in space. In *Proceedings of the ICAPS Workshop on Planning and Robotics*, 2020. 3

[9] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study, 2014. 5.1

[10] Amanda Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, ICAPS'10, page 42–49. AAAI Press, 2010. 5

[11] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1):61–95, 1991. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(91)90006-6. URL `https://www.sciencedirect.com/science/article/pii/0004370291900066`. 2.3

[12] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *ArXiv*, abs/1106.4561, 2003. 2.1

[13] Ilce Georgievski and Marco Aiello. An overview of hierarchical task network planning. *CoRR*, abs/1403.7426, 2014. URL `http://arxiv.org/abs/1403.7426`. 2.2

[14] D. Holler, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford. Hddl: An extension to pddl for expressing hierarchical planning problems. In *Proceedings AAAI 2020*, pages 9883–9891, New York, NY, 2 2020. 4.1

[15] Okhtay Ilghami. Documentation for jshop2. *Department of Computer Science, University of Maryland, Tech. Rep*, pages 41–42, 2006. 3

[16] Jayanth K. Mogali, Willem van Hoeve, and Stephen F. Smith. Template matching and decision diagrams for multi-agent path finding. In *Proceedings 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence and Operations Research (CPAIOR)*, 9 2020. 1.1

[17] N. Muscettola. Hsts: Integrating planning and scheduling, 1993. 2.4

[18] Nicola Muscettola, Stephen Smith, Amedeo Cesta, and Daniela D'Aloisi. Coordinating space telescope operations in an integrated planning and scheduling architecture. *IEEE control systems*, 12, 03 1992. doi: 10.1109/37.120450. 2.4

[19] Nicola Muscettola, P. Pandurang Nayak, Barney. Pell, and Brian C. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1/2):5–48, 1998. 1.1

[20] Dana Nau, Héctor Munoz-Avila, Yue Cao, Amnon Lotem, and Steven Mitchell. Total-order planning with partially ordered subtasks. In *IJCAI*, volume 1, pages 425–430, 2001. 3

[21] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugar Kuter, J. William Murdock, Dan Wu, and Fusan Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003. 3, 4.1

[22] Chao Qi, Dan Wang, Héctor Muñoz-Avila, Peng Zhao, and Hongwei Wang. Hierarchical task network planning with resources and temporal constraints.

Knowledge-Based Systems, 133:17–32, 2017. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2017.06.036. URL https://www.sciencedirect.com/science/article/pii/S0950705117303167. 3

[23] Greg Rabideau, Dan Tran, Steve Chien, B. Cichy, Robert Sherwood, D. Mandl, S. Frye, S. Shulman, J. Szwaczkowski, D. Boyer, and J. van Gaasbeck. Mission operations of earth observing-1 with onboard autonomy. In *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*, pages 367–373, Pasadena, CA, 7 2006. 1.1

[24] Zack Rubinstein, Stephen Smith, and Laura Barbulescu. Incremental management of oversubscribed vehicle schedules in dynamic dial-a-ride problems. In *Proceedings 26th AAAI Conference on Artificial Intelligence (AAAI '12)*, pages 1809 – 1815, July 2012. 3

[25] G. Sharona, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015. 1.1

[26] R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, and A. Fukunaga. Using aspen to automate eo-1 activity planning. In *1998 IEEE Aerospace Conference Proceedings (Cat. No.98TH8339)*, volume 3, pages 145–152 vol.3, 1998. doi: 10.1109/AERO.1998.685789. 2.4

[27] David E. Smith, Jeremy Frank, and W. Cushing. The anml language, 2007. 2

[28] S.F Smith, M.A Becker, and L.A Kramer. Continuous management of airlift and tanker resources: A constraint-based approach. *Mathematical and Computer Modelling*, 39(6):581–598, 2004. ISSN 0895-7177. doi: https://doi.org/10.1016/S0895-7177(04)90542-0. URL https://www.sciencedirect.com/science/article/pii/S0895717704905420. Defense transportation: Algorithms, models, and applications for the 21st century. 3

[29] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T.K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Twelfth International Symposium on Combinatorial Search (SoCS 2019)*, 6 2019. 1.1

[30] Miland Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997. 1.1

[31] A. Torreno, E. Onaindia, and O. Sapena. Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626, 2014. 1.1

[32] A. Umbrico, A. Cesta, M. C. Mayer, and Andrea Orlandini. Platinum: A new framework for planning and acting. In *AI*IA*, 2017. 2.4, 3

[33] Alessandro Umbrico, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini. Integrating resource management and timeline-based planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28(1): 264–272, Jun. 2018. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13905. 3

[34] Neil Yorke-Smith. Exploiting the structure of hierarchical plans in temporal constraint propagation. In *AAAI*, pages 1223–1228, 2005. 3