A Task Modelling Formalism for Industrial Mobile Robot Applications

Anders Lager[1,2], Alessandro V. Papadopoulos[2], Giacomo Spampinato[1], Thomas Nolte[2]

Abstract— **Industrial mobile robots are increasingly introduced in factories and warehouses. These environments are becoming more dynamic with human co-workers and other uncertainties that may interfere with the robot's actions. To uphold efficient operation, the robots should be able to autonomously plan and replan the order of their tasks. On the other hand, the robot's actions should be predictable in an industrial process. We believe the deployment and operation of robots become more robust if the experts of the industrial processes are able to understand and modify the robot's behaviour. To this end, we present an intuitive novel task modelling formalism, Robot Task Scheduling Graph (RTSG). RTSG provides building blocks for the explicit definition of alternative task sequences in a compact graph format. We present how such a graph is automatically converted to a task planning problem in two different forms, i.e., a Mixed Integer Linear Program (MILP) and a Planning Domain Definition Language specification (PDDL). Converted RTSG models of a mobile kitting application are used to experimentally compare the performance of one MILP planner and two PDDL planners. Besides providing this comparison, the experiments confirm the equivalence of the converted MILP and PDDL problem formulations. Finally, a simulation experiment verifies the assumed correlation between a cost model, based on path lengths, and the makespan.**

I. INTRODUCTION

To support human labour with repetitive, non-ergonomic and simple tasks, the need is ever increasing for having mobile robots able to perform versatile industrial robot tasks like kitting and machine tending.

For efficient operation in a dynamic, collaborative working space where unexpected situations are expected to occur, the ability to plan and replan tasks autonomously in a robust way becomes a key success factor. The maturity of AI Planning has seen tremendous progress over the last decades and several modelling formalisms with high expressiveness have been demonstrated successfully in industrial robot applications and other domains [1], [2]. However, these modelling formalisms are often complex and do not take advantage of the domain expert's intuition and skills in understanding what is a valid task sequence. We consider a domain expert as someone who has expert knowledge in the tasks that the robot shall perform in a certain industrial context—not an expert in robot programming. We strongly believe that enabling the competence of domain experts is crucial to reduce the threshold for the successful commissioning of competitive industrial robot applications on a larger scale.

This work was funded by The Knowledge Foundation (KKS), project ARRAY, and by The Swedish Foundation for Strategic Research (SSF).

[1]A. Lager and G. Spampinato are with ABB AB, Västerås, Sweden `name.surname@se.abb.com`

[2]A. Lager, A. V. Papadopoulos, and T. Nolte are with Mälardalen University, Västerås, Sweden `name.surname@mdh.se`

In industrial robot applications, it is important to avoid unexpected action sequences that reach a defined goal state, at the price of potentially unexpected and undesired side effects. Finding a *feasible* plan is often easy since working procedures typically are well organized. The challenge often lies in finding an efficient plan.

In this paper, we present a novel modelling formalism, Robot Task Scheduling Graph (RTSG), that addresses these problems while leveraging AI planning. One goal with RTSG is to combine the knowledge and experience of domain experts with the efficiency of automated planning/scheduling. The modelling formalism provides building blocks for describing variable sequences of robot actions to reach high-level goals. As RTSG is graph-based, it enables an intuitive visual overview.

We do not claim RTSG to be the most expressive modelling approach but we argue it is sufficiently expressive for a semi-structured industrial mobile robot application. We present how an RTSG model can be *automatically* converted to a task scheduling problem in two different forms: Mixed Integer Linear Programming (MILP) and Planning Domain Definition Language (PDDL) [3]. This enables RTSG to be used with MILP solvers as well as PDDL-based planners. Improving the efficiency of these planners is a relevant problem, but it is beyond the scope of this paper that focuses on the capabilities of the modelling formalism.

An experimental comparison of the performance for two PDDL-based planners and one MILP solver is presented. These experiments are applied to modelled use cases for a mobile kitting application, measuring the planning time and the efficiency of generated task sequences. Additionally, the results indicate an equivalence of the MILP and the PDDL representations of the RTSG scheduling problem. Finally, in a simulation study, we show that a transition cost model based on path lengths is a valid approach to minimize the resulting makespan.

The rest of the paper is organized as follows. Section II presents the related work. Section III gives an intuitive description of the RTSG modelling formalism. Section IV describes a conversion from RTSG to MILP. Section V describes a conversion from RTSG to PDDL. Section VI presents the experimental results, while Section VII concludes the paper.

II. RELATED WORK

RTSG fills a gap between existing modelling formalisms of robot action sequences by combining a desirable set of properties:

- Intuitive modelling approach for a domain expert.

TABLE I: Properties of modelling formalisms for industrial robot applications

Modelling formalism	Intuitive for domain expert	Task sequence variability guided by domain expert	Automated planning/ scheduling
PDDL	–	–	✓
HTN	–	✓	✓
CRAM	–	✓	–
RTSG	✓	✓	✓
Robot Skills	✓	–	✓
State Machine	(✓)	✓	–
Petri Net	(✓)	✓	–
Behavior Trees	(✓)	✓	–
Block-based Programming	✓	–	–

- Intended for use with an automated planner/scheduler to generate efficient task sequences.
- Leverages domain experts intuition, skills and knowledge on a suitable variability of task sequences.

A recent literature review investigated different approaches for representations of action sequences used for robot task planning and execution in a dynamic environment [4]. A selection of these representation approaches is given in Table I. The selection covers all representations available in ROS that have been used with industrial robot applications. Additionally, robot skills, block-based programming and Behaviour Trees have been added. The first column, indicating the intuitiveness of the formalism, is subjective and not backed with empirical evidence. The indicated existence of properties for the second and the third column is indicated by the referenced works.

PDDL [5] is an expressive modelling formalism to set up general planning problems (see Section V-A). Modelling a planning problem is focused on creating objects in the world and give facts about them and their mutual relations while providing operators that may change these facts to reach a goal condition. However, the representation is not intuitive for a domain expert, and there is no explicit way to indicate preferences on action sequences.

Hierarchical Task Networks (HTN) is another general modelling formalism [6]. In HTN, modelling is about specifying partial orders of tasks. It supports compound tasks that can be decomposed by alternative methods into smaller subtasks in a desired partial order. This provides a way to indicate alternative task sequences. Primitive subtasks correspond to operators in PDDL. The specification of preconditions and variables for methods and operators, the specification of operator effects and the flexible binding of variables give great expressiveness. However, managing these general concepts can be challenging for a domain expert.

Similar to HTN, Cognitive Robotic Abstract Machine (CRAM) [7] that is based on Reactive Plan Language [8] has an action-centric modelling formalism. It is an expressive programming language supporting a hierarchical task structure. The general purpose of CRAM is to be a tool to write robust robot control programs. However, it does not provide support for automated planning and a programming language is not intuitive for a domain expert.

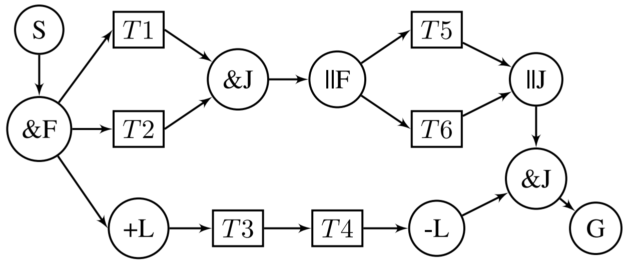

Fig. 1: Robot Task Scheduling Graph.

Robot skills build on the idea that the knowledge of an expert of robot programming can be encoded into the implementation of tasks in the form of reusable skills that can be used as simple building blocks when modelling a robot application. These skills can include preconditions and effects to support runtime execution but also automated planning [9]. However, this concept does not explicitly support using a domain expert's intuition of the task sequence variability.

Block-based programming is primarily intended to simplify programming by providing configurable building blocks, e.g., CoBlox [10].

Other general modelling formalism's can be used for modelling complex robot behaviour and guiding the execution e.g., State Machines [11], PetriNets [12], [13] and Behaviour Trees [14]. These powerful modelling techniques are not primarily intended for automated planning and the modelling complexity can be challenging.

In some approaches, no apparent high-level modelling formalism is used and the problem representation is purely mathematical, typically in the form of an optimization problem, e.g., [15]. This can give good results but is less intuitive and the problem formulation is harder to adapt to meet new requirements.

In assembly applications, directed graphs, AND/OR-graphs [16] (based on the assembly parts) or precedence graphs [17] (based on assembly operations) can be used to model the variability of assembly sequences that will fulfil a specified assembly. ASML is a later approach [18]. These techniques are used to search for a good design for manufacturability but also for finding an efficient assembly sequence. They are not intended for runtime task planning/replanning. RTSG presents an approach akin to assembly modelling formalisms, to guide automated scheduling of tasks.

III. RTSG Modelling formalism

With the building blocks informally described below, RTSG provides a modelling formalism for a domain expert that guides the selection of a task sequence to be decided by an automatic planner with respect to some optimization objective.

An RTSG model, as exemplified in Figure 1, is a directed acyclic graph having one start node (S) with a single outgoing edge representing an initial state. At the other end, there is one goal node (G) with a single incoming edge representing a desired goal state. In between, a set of robot tasks leading towards the goal are represented by rectangular nodes having a single incoming edge and a single outgoing edge. In addition, a collection of *logical* nodes impose different scheduling dependencies between tasks. Edges represent precedence constraints: If there is a directed path between two tasks, e.g., $T1$ and $T5$, the first task must precede the latter task in a plan where both tasks are scheduled. AND-Pairs split the graph with an AND-Fork node (&F) into parallel branches and rejoin them with an AND-Join node (&J). Tasks in different AND-Fork branches, e.g., $T1$, $T2$ and $T3$ may be scheduled in any mutual order since there is no directed path between them. OR-Pairs split and rejoin the graph in a similar way with an OR-Fork (||F) and an OR-Join (||J). However, the resulting parallel branches represent alternatives, i.e., only tasks in one of the branches, e.g., $T5$ or $T6$, will be scheduled. Lock-Pairs encapsulate a part of a single branch between an AddLock node (+L) and a RemoveLock node (-L). The set of tasks between a Lock-pair, e.g., $\{T3, T4\}$ will be scheduled as a coherent sub-array of the full task sequence, i.e., uninterrupted by other tasks. Pairs may encapsulate other pairs in a hierarchy, e.g., an OR-Pair may contain other OR-pairs that split alternative branches into sub-alternatives.

Complementing the graph, a task cost estimation should be provided. Apart from specifying the cost of performing different tasks, it should include transition costs between any pair of tasks allowed by the RTSG model to be scheduled in consecutive order.

IV. Conversion from RTSG to MILP

Section IV-A presents how an RTSG model is converted to a MILP scheduling problem with decision variables, optimization objective and constraints. Section IV-B specifies how the scheduling problem is modified in a replanning scenario.

A. Conversion from RTSG to MILP

1) Notation: A is the set of all task nodes, S is the start node and G is the goal node. The following notation is used when combining them: $A^S = A \cup S$, $A^G = A \cup G$ and $\tilde{A} = A \cup S \cup G$. $O \subseteq A$ denotes all tasks encapsulated by OR-Pairs. $j \prec k$ indicates that j precedes k, where $j, k \in \tilde{A}$.

2) Variables and objective: Decision variables are given by $X_{j,k} \in \{0, 1\} \quad j, k \in \tilde{A}$.

$$X_{j,k} = \begin{cases} 1, & \text{if task } j \text{ is followed by task } k. \\ 0, & \text{otherwise.} \end{cases}$$

$K_{j,k} \in \mathbb{R}_{\geq 0}$ represents the cost for performing task k after task j:

$$K_{j,k} = \tau_{j,k} + \alpha_k \tag{1}$$

where $\tau_{j,k}$ is the transition cost and α_k is the action cost.

The objective is to minimize the cost function J:

$$J = \sum_{j \in A^S} \sum_{k \in A^G} X_{j,k} K_{j,k}. \tag{2}$$

3) General constraints:

$$X_{j,j} = 0 \qquad \forall j \in \tilde{A} \tag{3}$$

$$\sum_{k \in A^G} X_{j,k} = 1 \qquad \forall j \in A^S \setminus O \tag{4}$$

$$\sum_{k \in A^G} X_{j,k} \leq 1 \qquad \forall j \in O \tag{5}$$

$$\sum_{j \in A^S} X_{j,k} = 1 \qquad \forall k \in A^G \setminus O \tag{6}$$

$$\sum_{j \in A^S} X_{j,k} \leq 1 \qquad \forall k \in O \tag{7}$$

$$\sum_{k \in A^G} X_{j,k} = \sum_{k \in A^S} X_{k,j} \qquad \forall j \in O \tag{8}$$

$$\sum_{j \in A^G} X_{j,S} = 0 \tag{9}$$

$$\sum_{k \in A^S} X_{G,k} = 0 \tag{10}$$

$$\sum_{j \in V} \sum_{k \in V} X_{j,k} \leq |V| - 1 \qquad \forall V \subseteq A, V \neq \emptyset \tag{11}$$

There can be no transition between the same task (3). Tasks outside OR-Pairs will occur once (4) and (6). Tasks inside OR-Pairs will occur at most once (5), (7), and (8). There is no transition to the start state (9) and there is no transition from the goal state (10). There can be no cyclic sub-tours between tasks (11).

4) Precedence constraints: Assuming $D \subseteq \tilde{A}$ is any ordered subset with elements D_i. Precedence constraints must hold for these subsets in general and especially if they become a sub-array of the task sequence:

$$\sum_{j=1}^{|D|-1} X_{D_j, D_{j+1}} \leq |D| - 2 \quad \forall D \subseteq \tilde{A},\ |D| \geq 2,\ D_{|D|} \prec D_1. \tag{12}$$

5) Lock constraints: $L \subseteq A$ are the set of all tasks encapsulated by a Lock-Pair.

The first tasks in L are defined as $L^F = \{a \in L \mid b \nprec a \quad \forall b \in L\}$. The last tasks in L are defined as $L^L = \{a \in L \mid a \nprec b \quad \forall b \in L\}$.

$$X_{j,k} = 0 \qquad \forall L, \forall j \in A^S \setminus L, \forall k \in L \setminus L^F \tag{13}$$

$$X_{j,k} = 0 \qquad \forall L, \forall j \in L \setminus L^L, \forall k \in A^G \setminus L \tag{14}$$

$$\sum_{j \in A^S \setminus L} \sum_{k \in L^F} X_{j,k} \leq 1 \quad \forall L \tag{15}$$

$$\sum_{j \in L^L} \sum_{k \in A^G \setminus L} X_{j,k} \leq 1 \quad \forall L \tag{16}$$

There can only be transitions to the first tasks from external tasks (13) and there can at most be one such transition (15). Similarly, there can only be transitions from the last tasks to external tasks (14) and there can at most be one such transition (16).

6) OR-Pair constraints: The constraints presented here can handle a nested structure of OR-Pairs. However, a potential simplification of such a structure, e.g., with algebraic rules, is out of the scope for this work. If an OR-pair is contained by an outer OR-Pair, it is denoted an *internal* OR-Pair and at most one of its alternative branches will be scheduled. The outermost OR-Pairs are denoted *external* OR-Pairs and exactly one of their branches will be scheduled. OR-Pairs contain *OP nodes* that can be of two types: tasks and internal OR-Pairs.

$O_1, \ldots, O_v$ are sets of OP nodes contained by OR-Pair $1, \ldots, v$. $O_{p1}, \ldots, O_{pm}$ are the set of OP nodes contained by branches $1, \ldots, m$ of OR-Pair p. $O_{pq}^T = \{a \in O_{pq} \mid a \text{ is a task}\}$. $O_{pq}^{OP} = \{a \in O_{pq} \mid a \text{ is an internal OR-Pair}\}$

One *primary* OP node, $P_{pq} \in O_{pq}$, is arbitrary selected for each OR-Pair branch.

Three help operators (17), (18), and (19) are defined to support the definition of OR-Pair constraints. Note that operators F and H return a set while R returns a *set of sets*.

$$F(O_{pq}) = \begin{cases} \{a\} & \text{if } P_{pq} \text{ is task } a. \\ F(O_{r1}) \cup \ldots \cup F(O_{rm}) & \text{if } P_{pq} \text{ is OR-pair } O_r \end{cases} \tag{17}$$

$$H(O_p) = F(O_{p1}) \cup F(O_{p2}) \cup \ldots \cup F(O_{pm}) \tag{18}$$

$$R(O_{pq}) = O_{pq}^T \setminus P_{pq} \cup \bigcup_{i \in O_{pq}^{OP} \setminus P_{pq}} \{H(i)\} \tag{19}$$

Given these definitions, the OR-Pair constraints can be summarized:

$$\sum_{j \in A^G} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{s,j} = 1 \qquad \forall \text{ external } O_p \tag{20}$$

$$\sum_{j \in A^S} \sum_{q=1}^{m} \sum_{s \in F(O_{pq})} X_{j,s} = 1 \qquad \forall \text{ external } O_p \tag{21}$$

$$\sum_{j \in A^G} \sum_{k \in r} X_{k,j} = \sum_{j \in A^G} \sum_{s \in F(O_{pq})} X_{s,j} \quad \forall O_{pq}, \forall r \in R(O_{pq}) \tag{22}$$

$$\sum_{j \in A^S} \sum_{k \in r} X_{j,k} = \sum_{j \in A^S} \sum_{s \in F(O_{pq})} X_{j,s} \quad \forall O_{pq}, \forall r \in R(O_{pq}) \tag{23}$$

One of the branches of external OR-Pairs will be scheduled (20), (21). If the primary OP node in an OR-Pair branch is scheduled, so will the remaining OP Nodes in the same branch (22), (23).

B. Replanning

At a point of replanning, the ordered set $C = \{C_1, \ldots, C_l\}$ represents completed tasks. Additional constraints for completed transitions (24) (25):

$$X_{S,C_1} = 1 \tag{24}$$

$$X_{C_i,C_{i+1}} = 1 \quad \forall i = 1, \ldots, l-1 \tag{25}$$

Furthermore, the cost matrix, $K_{j,k} \quad j \in A^S \setminus C, k \in A^G \setminus C$, is updated to consider a new starting location and an updated world state. Finally, the cost matrix is updated to consider completed tasks:

$$\begin{aligned} &K_{S,C_1} = 0 \\ &K_{C_i,C_{i+1}} = 0 \quad \forall i = 1, \ldots, l-1 \\ &K_{C_l,j} = K_{S,j} \quad \forall j \in A^G \setminus C \end{aligned}$$

V. Conversion from RTSG to PDDL

A. PDDL

PDDL is a modelling formalism originating from STRIPS [19] that has evolved from the planning competitions held by The International Conference on Autonomous Planning and Scheduling since 1998. In PDDL, a planning problem is described in terms of objects in the world (e.g., robot, gripper, box and location), an initial state and the desired goal state. The initial state and the goal state are specified as a list of facts. A fact is related to a set of objects and it is defined with a binary predicate (e.g., gripper is holding box). Actions (e.g., place box on location) can be applied to change facts. Actions have parameters (e.g., robot, gripper, box, location). They also have preconditions as a binary function of predicates (e.g., gripper is holding box and robot is at location). If the preconditions hold for some set of parameters, an action can be performed that will change the facts according to the action's list of effect predicates applied to the parameters (e.g., gripper is not holding box, the box is on location). Finding a plan is about finding a sequence of actions applied to the objects that step-by-step will change the facts from the initial state to the goal state. A simple objective for finding an *optimal* plan is to minimize the number of actions. The PDDL2.1 specification [5] introduced syntax for temporal and numerical planning. It also includes *metrics* that allows for specifying an objective. With these language extensions, it is possible to convert an RTSG model into a PDDL scheduling problem.

B. Conversion from RTSG to PDDL

With respect to RTSG, we identify the natural PDDL objects as the RTSG nodes. The reason for converting nodes to PDDL objects is that nodes have several relations and properties that can be defined as predicates or numerical functions, e.g., the edges that connect them or the transition cost between them. Two types of actions are needed. The first type is to run a task and the second type is to fire a transition for a logical node. Running a task has a duration while firing a transition is instant. The purpose of transition actions is to guide the scheduling of tasks according to the constraints imposed by the RTSG. The occurrences of transition actions in a planned action sequence do not correspond to real robot actions. However, they can be used to improve the visualization of a runtime execution state and progress:

In Figure 3, the active task is orange, completed tasks are green while non-started tasks are grey. Completed logical nodes are light green while the remaining are white. This illustrates the execution progress as a gradual green propagation of the graph that will follow the outgoing edges of completed tasks/transitions.

Listing 1: PDDL domain

```
(define (domain RTSG)
    (:types
      node - object
      task logical andjoin2 - node
      startcond goalcond robtask - task
      andfork orfork orjoin - logical
      nofork - orfork)
    (:predicates
      (edge ?n1 ?n2 - node)
      (fired ?n - node)
      (latest-completed ?t - task)
      (andjoin2-inputs ?n1 ?n2 - node)
      (orfork-branch ?orf - orfork ?to - node)
      (branch-not-selected ?orf - orfork)
      (not-locked ?from ?to - task))
    (:functions
      (cost ?from ?to - task))
    (:durative-action RUN-TASK
      :parameters (?this ?prev - task ?input - node ?orf -
            orfork)
      :duration (= ?duration (cost ?prev ?this))
      :condition (and
          (at start (latest-completed ?prev))
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf))
          (at start (not-locked ?prev ?this)))
      :effect (and
          (at start (not(latest-completed ?prev)))
          (at start (not(branch-not-selected ?orf)))
          (at end (latest-completed ?this))
          (at end (fired ?this))))
    (:durative-action FIRE-LOGICAL
      :parameters (?this - logical ?input - node ?orf -
           orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input ?this))
          (at start (fired ?input))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this))))
    (:durative-action FIRE-ANDJOIN2
      :parameters (?this - andjoin2 ?input1 ?input2 - node
            ?orf - orfork)
      :duration (= ?duration 0)
      :condition (and
          (at start (edge ?input1 ?this))
          (at start (edge ?input2 ?this))
          (at start (fired ?input1))
          (at start (fired ?input2))
          (at start (andjoin2-inputs ?input1 ?input2))
          (at start (orfork-branch ?orf ?this))
          (at start (branch-not-selected ?orf)))
      :effect (and
          (at start (not(branch-not-selected ?orf)))
          (at end (fired ?this)))))
```

Listing 2: PDDL problem

```
(define (problem RTSG-config)
(:domain RTSG)
    (:objects
        s - startcond
        g - goalcond
        af1 - andfork
        aj1 aj2 - andjoin2
        of1 - orfork
        oj1 - orjoin
        t1 t2 t3 t4 t5 t6 - robtask
        nfs nfg ... nft4 - nofork    ; Dummy objects
    )
    (:init
        (fired s)
        (latest-completed s)
        (edge s af1)
        (edge af1 t1) ... (edge aj2 g)
        (not-locked s t1) ... (not-locked t6 g)
        (andjoin2-inputs t1 t2)
        (andjoin2-inputs oj1 t4)
        (orfork-branch of1 t5)
        (orfork-branch of1 t6)
        (branch-not-selected of1)
        (orfork-branch nfs s)        ; Dummy fact
        ...                          ; ...
        (orfork-branch nfg g)        ; Dummy fact
        (branch-not-selected nfs)    ; Dummy fact
        ...                          ; ...
        (branch-not-selected nft4)    ; Dummy fact
        (= (cost s t1) 100) ... (= (cost t6 g) 100))
    (:goal (fired g))
    (:metric minimize total-time)
)
```

Converted domain and problem files for the RTSG model in Figure 1 are shown in Listings 1 and 2. The syntax used from PDDL2.1 has been reduced to enable the POPF2 planner [20] that is supported by ROSPlan [21] making it an attractive choice for robotics research. POPF2 does not support some of the PDDL2.1 requirements, among them negative preconditions, disjunctive preconditions and conditional effects. This adds some complexity to the conversion by a need to use antonym predicates (e.g., "not-locked" instead of "locked"), dummy objects, redundant facts and redundant actions.

In the following, a walkthrough is made through the different sections of the converted PDDL domain and problem files in Listings 1 and 2. The contents of the domain sections are mostly fixed and independent of the RTSG model with only a few stated exceptions for AND-Join nodes. The problem sections are populated from the RTSG model as specified in the walkthrough. With this specification, the conversion from a general RTSG model to PDDL2.1 can be fully automated.

Domain sections:

1) Types: RTSG node types are arranged as different types in a hierarchy: A *node* is a PDDL *object*. A *task* is a *node* that affects the cost of the plan. There are three types of *tasks*: *robottask*, *startcond* and *goalcond*. The remaining types are used to define logical nodes of different types. The *nofork* type is used to create dummy objects that support the handling of alternative task sequences. The *andjoin2* type is used to create AND-Joins having two incoming edges. If the RTSG model has AND-Joins with more incoming edges, additional types are needed to cover them as well, e.g., *andjoin3*, *andjoin4* etc.

2) Predicates: The edges between two RTSG nodes are indicated with an *edge* predicate. A completed RTSG node is indicated with a *fired* predicate. The *latest-completed* predicate indicates if a task is the latest completed task. A group of all X nodes having an outgoing edge to the very same AND-Join are indicated with an *andjoinX-inputs* predicate. Nodes having an incoming edge from a specific OR-Fork are indicated with an *orfork-branch* predicate. The same predicate is also used to indicate other nodes, but these are created with a *nofork* in the problem sections. The *branch-not-selected* predicate indicates that there has been no selection of an alternative branch for an OR-Fork. Finally, the *not-locked* predicate indicates that a transition is possible between two tasks.

3) Functions: A *cost* function indicates the cost, as a numerical value, required to perform a task after finishing a previous task.

4) Durative actions for running tasks: There is one action that runs tasks. The parameters indicate which task to run, the previous task, the node that is connected to the incoming edge and an associated orfork (a dummy or a real). The action's duration time is set to the cost to run the task after the previous task. The preconditions require that an action already has been run for the node connected to the incoming edge. It also requires that a transition from the previous task is allowed (*not-locked*). The primary effect of the action is to indicate that the action for the task has run (*fired*). The combination of preconditions and effects avoids concurrent tasks and prevents the scheduling of tasks in more than one alternative OR-Pair branch. Note that *goalcond* also is a `task` and running it will reach the goal state, e.g., by moving to a certain location.

5) Durative actions for firing transitions: The remaining actions are used to fire transitions for logical nodes. The parameters indicate for which logical node the transition will occur, a node that is connected to an incoming edge and an associated orfork. The action's duration time is always zero. The preconditions require that an action already has been run for the node connected to the incoming edge. The primary effect of the action is to indicate that the action for the logical node has run. The combination of preconditions and effects prevents the scheduling of tasks in more than one alternative OR-Pair branch.

Similar but separate actions are used to fire transitions for AND-Join nodes. One such action is needed for every used number of incoming edges on AND-Join nodes in the RTSG model. There is only one difference between these actions: The preconditions require that actions have been run for *all* nodes connected to the incoming edges.

Problem sections:

6) Objects: One *node* (of corresponding *type*) is created for each node in the RTSG model except for AddLocks and RemoveLocks. One dummy *nofork* object is created for each of these *nodes* that do not have an incoming edge from an OR-Fork node.

7) Init: The start node is indicated as *fired* and it is also indicated to be the *latest-completed task*. *Edge* facts are created between the *nodes* according to the RTSG model, but where AddLock and RemoveLock nodes are bypassed. *No-lock* facts are defined for all possible transitions between *tasks* with respect to the precedence constraints and Lock-Pair constraints imposed by the RTSG model. For all AND-Joins in the RTSG model, an *andjoinX-inputs* fact is created indicating all *nodes* that are connected to the incoming *edges*. For all *nodes* having an incoming *edge* from an *orfork*, an *orfork-branch* fact is created indicating this OR-Fork, and for all remaining *nodes*, an *orfork-branch* fact is created indicating the corresponding *nofork*. A *branch-not-selected* fact is created for all OR-Forks, indicating if an outgoing branch has not yet been selected. Finally, numerical *cost* facts are created to specify the cost of all possible transitions between *tasks*.

8) Goal and metric: The goal is to reach the condition that the goal node has been *fired*. The metric indicates that an optimal plan should minimize the total duration of the plan (*total-time*).

C. Replanning

In a replanning scenario, some modifications are required for the init section of the problem Listing 2. A *fired* predicate is added for all completed tasks and for all logical nodes that precede completed tasks. The *latest-completed* predicate is removed for the start node and added for the latest completed task. Transition costs from the latest completed task are updated to account for a new start location of the robot. Potential obstacles may affect some transition costs, e.g., if the robot needs to move another way between two tasks. If there are completed tasks in one OR-Fork branch, scheduling of tasks in alternative branches are avoided by *not* creating a *branch-not-selected* predicate for the corresponding *orfork*.

VI. Results

A. Experimental setup

The targeted application is a mobile robot operating in a warehouse for picking customer orders in the form of kits, i.e., boxes filled with specified objects. The robot moves around the warehouse shelves and performs robot tasks as specified by an RTSG model. In the graph, a robot task represents an action where a specific object is handled at a specific location. The RTSG models for three different use cases (A, B and C) are shown in Figures 2, 3 and 4. All use cases start with the fetching of 2 empty kit boxes and allow them to be filled in parallel. In use cases A and B, the kit boxes are filled in two layers separated by an interlayer. Use case B has more precedence constraints than A while use case C has quite few precedence constraints.

Gazebo [22] was used to set up a simulated mobile robot in a simple warehouse world having shelves of different shapes, see Figure 5. Dijkstra's algorithm was used to generate two-dimensional collision-free paths between handling locations at the different shelves. The path lengths were used to define transition costs.

The experiments were run with Ubuntu 18.04.5 on an Intel i5-4570 quad-core processor with 7.6 GB RAM.

B. Experimental results

The use cases were tested with three different planners in a comparative study where all problem formulations were automatically generated from the RTSG models. Gurobi Optimizer [23], hereby referred to as MILP, was used with the MILP problem formulation. POPF2 and Temporal Fast Downward (TFD) [24] were used with the PDDL problem formulation. Each use case was run 100 times. For each run, the location of the objects to be handled by the robot tasks was randomized among 52 fixed locations at the different

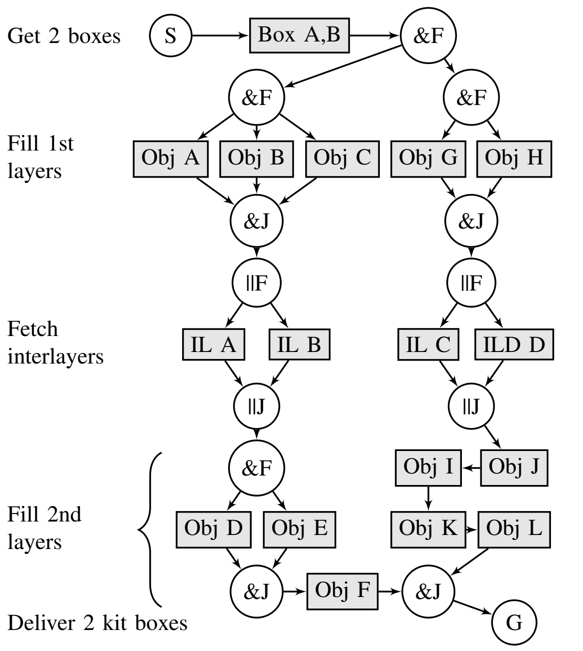

Fig. 2: RTSG model for use case A.

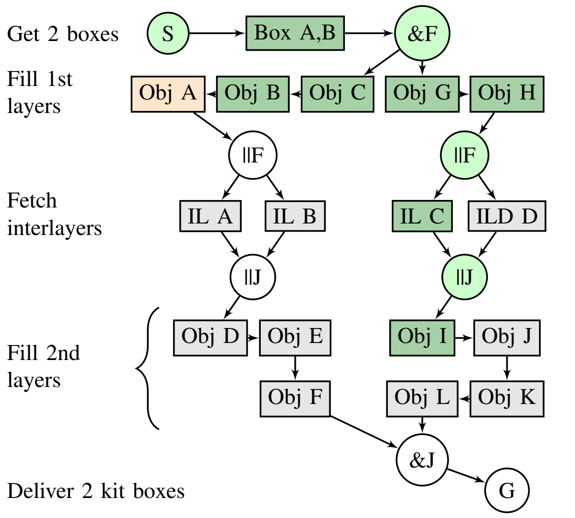

Fig. 3: RTSG model for use case B. The colouring of the graph nodes is discussed in subsection V-B

shelves. The common start- and goal location, i.e., the delivery station, was fixed. Comparisons of the planners' achievements of objective values and planning times are displayed in Figure 6. MILP and TFD indicated optimal solutions and reached the same objective value for all runs and the very same sequence for 89% of the runs. This suggests an equivalence of the PDDL conversion and the corresponding MILP problem formulation. MILP solved C within a few seconds while needing several minutes for A and B. The reason for this performance difference is presumably a fast-growing amount of precedence and non-cyclic constraints needed for A and B compared to the less constrained C. On the other hand, TFD needed a minute to solve use case C while solving A and B within a few seconds.

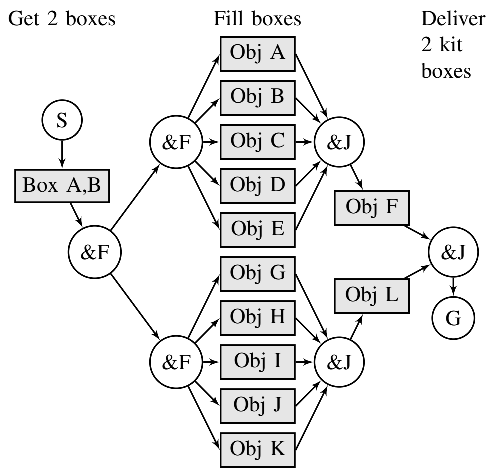

Fig. 4: RTSG model for use case C.

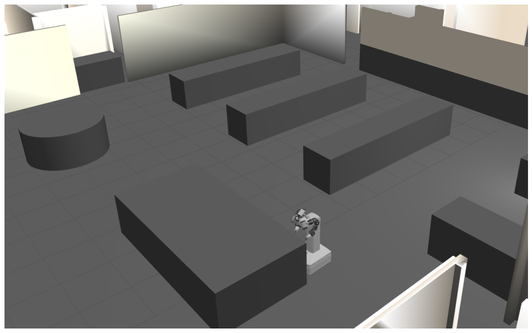

Fig. 5: Simple warehouse world.

POPF2 found valid but non-optimal solutions within 200 ms for A and B and within 2 seconds for C. The average cost increase for POPF2 was 15%, 14% and 11% for A, B and C respectively. The fast planning time for POPF2 can be attractive if a less optimized plan is acceptable.

In these experiments, the transition costs are based on path lengths but the optimization goal is often to minimize the makespan. To verify the correlation of path lengths and makespan an additional experiment was performed: To this end, a selection of ten MILP generated task sequences for use case B was made. The selection included the lowest and the highest costs and intermediate cost values with a fairly uniform distribution within this interval. These cycles were simulated 20 times and the cost vs makespan is given in Figure 7. ROS Navigation Stack [25] was used to navigate the robot with Adaptive Monte Carlo Localization for localization and Timed Elastic Bands for trajectory generation. The experiment indicates a linear correlation between cost and makespan. However, the non-monotonic part of the curve occurring between the two highest costs also confirms that there is an uncertainty imposed by a simplified cost model when applied to a more realistic setting.

VII. Conclusion and future work

We have presented Robot Task Scheduling Graph, a novel task modelling formalism. Descriptions are provided on how

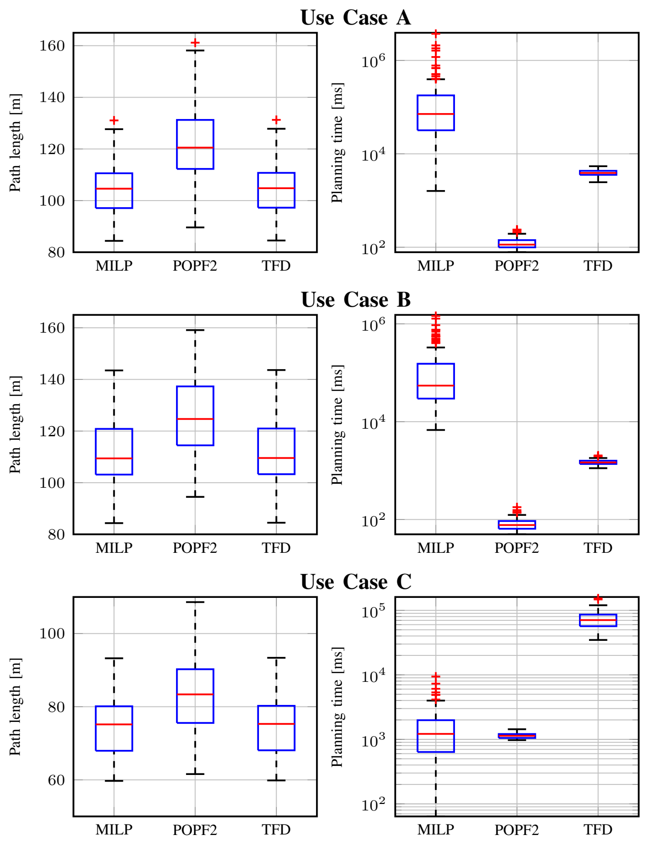

Fig. 6: Use cases results.

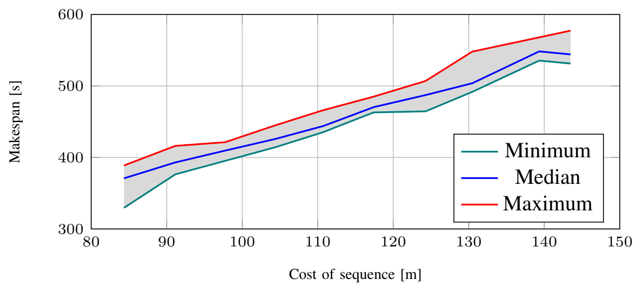

Fig. 7: Sequence cost vs simulated makespan for use case B. The grey area highlights the distribution of the obtained makespans in simulation.

to automatically convert an RTSG model to a scheduling problem for MILP solvers as well as for PDDL planners. An experimental comparison of two PDDL planners and one MILP solver suggests the converted problem formulations for MILP and PDDL are equivalent. These experiments also indicate that MILP solvers are more efficient than PDDL planners for an RTSG model with fewer precedence constraints while the opposite hold for an RTSG model with more precedence constraints. Mobile robot simulations of scheduled task sequences confirm the validity of estimating transition costs from 2-dimensional path lengths.

Future work may cover an extension to modelling and scheduling in the context of multi-robot applications and continuous applications. Another line of work may cover efficient modelling and handling of disturbance behaviours.

References

[1] E. Karpas and D. Magazzeni, "Automated planning for robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 417–439, 2020.

[2] B. Miloradović, B. Çürüklü, M. Ekström, and A. V. Papadopoulos, "GMP: A genetic mission planner for heterogeneous multi-robot system applications," *IEEE Trans. on Cybernetics*, 2021.

[3] D. McDermott, M. Ghallab, A. Howe, C. A. Knoblock, A. Ram, M. Veloso, D. S. Weld, and D. Wilkins, "PDDL – the planning domain definition language," 1998.

[4] H. Nakawala, P. J. S. Goncalves, P. Fiorini, G. Ferringo, and E. D. Momi, "Approaches for action sequence representation in robotics: A review," in *IROS*, 2018, pp. 5666–5671.

[5] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *ArXiv*, vol. abs/1106.4561, 2003.

[6] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: An htn planning system," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 379–404, 2003.

[7] T. Rittweiler, "Cram design & implementation of a reactive plan language," 2010.

[8] D. Mcdermott, "A reactive plan language," 1993.

[9] M. Crosby, F. Rovida, M. Pedersen, R. Petrick, and V. Krüger, "Planning for robots with skills," in *Workshop on Planning and Robotics (PlanRob)*, 2016, pp. 49–57.

[10] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. Shepherd, and D. Franklin, "Evaluating CoBlox: A comparative study of robotics programming environments for adult novices," in *Conf. on Human Factors in Computing Systems (CHI)*, 2018, pp. 1–1.

[11] J. Bohren and S. Cousins, "The smach high-level executive," *Robotics & Automation Magazine, IEEE*, vol. 17, pp. 18–20, 2011.

[12] V. Ziparo, L. Iocchi, P. Lima, D. Nardi, and P. F. Palamara, "Petri net plans: A framework for collaboration and coordination in multi-robot systems," *Autonomous Agents and Multi-Agent Systems*, vol. 23, pp. 344–383, 2011.

[13] H. Costelha and P. Lima, "Robot task plan representation by Petri nets: Modelling, identification, analysis and execution," *Autonomous Robots*, vol. 33, 2012.

[14] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 5420–5427.

[15] I. Nielsen, Q. V. Dang, G. Bocewicz, and Z. Banaszak, "A methodology for implementation of mobile robot in adaptive manufacturing environments," *Journal of Intelligent Manufacturing*, vol. 28, pp. 1171–1188, 2015.

[16] L. Mello and A. Sanderson, "Representations of mechanical assembly sequences," *Robotics and Automation, IEEE Transactions on*, vol. 7(2), pp. 211–227, 1991.

[17] X. Niu, H. Ding, and Y. Xiong, "A hierarchical approach to generating precedence graphs for assembly planning," *Int. Journal of Machine Tools and Manufacture*, vol. 43, no. 14, pp. 1473–1486, 2003.

[18] A. Salmi, P. David, J. Summers, and B. Eric, "A modelling language for assembly sequences representation, scheduling and analyses," *Int. Journal of Production Research*, vol. 52, pp. 3986–4006, 2014.

[19] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189 – 208, 1971.

[20] A. Coles, A. Coles, A. Clark, and S. Gilmore, "Cost-sensitive concurrent planning under duration uncertainty for service-level agreements." 2011.

[21] G. Canal and M. Cashmore, "ROSPlan: AI planning and robotics," 2019.

[22] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[23] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2021. [Online]. Available: http://www.gurobi.com

[24] P. Eyerich, R. Mattmüller, and G. Röger, "Using the context-enhanced additive heuristic for temporal and numeric planning," in *ICAPS*, 2009.

[25] R. L. Guimarães, A. S. de Oliveira, J. A. Fabro, T. Becker, and V. A. Brenner, *ROS Navigation: Concepts and Tutorial*, 2016, pp. 121–160.