Partial Order Temporal Plan Merging for Mobile Robot Tasks

Lenka Mudrova[1] and **Bruno Lacerda**[1] and **Nick Hawes**[1]

Abstract. For many mobile service robot applications, planning problems are based on deciding *how* and *when* to navigate to certain locations and execute certain tasks. Typically, many of these tasks are independent from one another, and the main objective is to obtain plans that efficiently take into account where these tasks can be executed and when execution is allowed. In this paper, we present an approach, based on merging of partial order plans with durative actions, that can quickly and effectively generate a plan for a set of independent goals. This plan exploits some of the synergies of the plans for each single task, such as common locations where certain actions should be executed. We evaluate our approach in benchmarking domains, comparing it with state-of-the-art planners and showing how it provides a good trade-off between a fast but imprecise approach of sequencing the plans for each task, and a unified approach where a planner is called for the large problem of achieving all goals.

1 INTRODUCTION

Consider a mobile service robot operating in an office building for a long period of time, where it autonomously performs tasks to assist the occupants in their everyday activities. One can imagine a wide array of tasks for such a robot to execute, for example:

- *"Bring me a cup of coffee."*
- *"Check if there are people in office 123."*
- *"Check if the emergency exits are clear."*

Note that this class of task have a common set of properties that one can take advantage of:

1. They require the robot to navigate to certain locations to execute certain actions, i.e., they include *spatial constraints*;
2. The actions associated with them can be executed concurrently. For example, a sensing action is often fixed to a location, but processing and reasoning about the sensed data can be done in parallel with the robot's movement;
3. Their goals are independent, in the sense that executing a certain task ω_i is not a precondition to successfully execute task ω_j. This means that they can be straightforwardly split into a set of different planning problems.

Furthermore, even though the goals are independent, the existence of spatial constraints means that there might be "synergies" between the independent plans, e.g. the locations visited while executing task ω_i might also be of use for executing task ω_j. Along with these spatial constraints, there are also timing constraints related to navigating between locations.

[1] School of Computer Science, University of Birmingham, UK, email: {lxm210, b.lacerda, n.a.hawes}@cs.bham.ac.uk

In this paper, we build on these insights to present an algorithm that, given a set of plans for each task, efficiently *merges* them into a global plan for all tasks that interleaves actions working towards different goals.

Our merging algorithm is based on partial-order planning (POP), a least-commitment search in the space of (partial order) plans. POP presents clear benefits for our robot-oriented merging approach, including:

1. The least-commitment approach of partial order planning yields more "merging points" between plans, when compared to a totally ordered plan;
2. POP presents a flexible approach to temporal planning with durative concurrent actions, allowing parallel action execution straightforwardly;
3. POP produces plans with more flexibility in execution as commitments can be determined at execution time when temporal information is more certain.

In summary, the main contributions of this paper are (i) the definition of a class of planning problems that are well-suited for the specification of *execution routines* for mobile service robots; and (ii) a partial order plan merging algorithm that is able to generate plans for a large amount of tasks, while taking advantage of possible synergies between such tasks, thus improving the overall robot's behaviour. For the class of problems we tackle in this paper, our approach is competitive with state-of-the-art forward chaining planners in benchmarking domains. Furthermore, the use of POP allows us to easily tackle concurrent actions, which allows us to outperform the state-of-the-art forward chaining planners in domains where reasoning about concurrent actions is required.

The structure of the paper is as follows. In Section 2, we provide an in-depth overview of state-of-the-art planners, and their limitations for our domain. In sections 3 and 4 we formalise the problem and background we rely on. Finally sections 5 and 6 present our novel algorithm and its evaluation.

2 RELATED WORK

2.1 Temporal planners

In order to compare our proposed algorithm with state-of-the-art temporal planners, we focus on those that successfully participated in the temporal track of the latest International Planning Competition (IPC) in 2014[1]. The 6 following planners participated in the competition: YAHSP3 [30] pre-computes relaxed plans for estimated future states which are then exploit to speed-up the forward

[1] https://helios.hud.ac.uk/scommv/IPC-14/index.html

state-space search. YAHSP3-MT [31] is a multi-thread extension of YAHSP. The YASHP3 planner is also exploited by another contestant, DAE$_{YAHSP}$ [3]. DAE uses a *Divide-and-Evolve* strategy in order to split the initial problem into a set of sub-problems, and to evolve the split based on the solutions found by its wrapped planner. In general, DAE$_X$ can be used with any planner, with the version we evaluate wrapping YAHSP. Two other participants extend well-known approaches to the temporal domain. First, the temporal-FD planner [9] expands the FD algorithm [12]. The search state is extended with a discrete timestamp component, and state expansion can be performed either by inserting an action or by incrementing the timestamp. Second, ITSAT [21] expands a satisfiability checking (SAT) approach to the temporal domain. ITSAT is the only planner from the aforementioned that is able to handle concurrent actions properly.

2.2 Temporal partial order planners

Another important class of temporal planners are those that provide a temporal partial order plan as a solution. Versatile Heuristic Partial Order Planner (VHPOP) [24] is one of the pioneers in this field. It builds on the classical backward plan-space search used by partial order planners, adding to it a set of different heuristics that allow for a more informed choice of which *flaw* to solve, or which plan to explore. The use of these heuristics yields big improvements in terms of speed, when comparing to previous partial order planners. In contrast, the more recent OPTIC [1] planer combines advantages of partial order planning with fast forward chaining techniques, which are very popular in current planners, due to their speed and scalability. Broadly speaking, OPTIC is a complete forward search algorithm with backtracking.

2.3 Planning & Execution for Mobile Service Robots

The CoBot service robots [29] operate in an office building performing several predefined tasks. A server based architecture [8] manages incoming tasks from a web-based user interface, schedules tasks across several robots [7], and keeps track of task execution. A similar centralised system architecture is used by the mobile service robot Tangy [16] which performs a sequence of predefined tasks when schedules of users are taken into account. This problem is modelled by mixed-integer programming and constraint programming techniques [4]. Mixed-integer programming is also used for scheduling in the integrated control framework presented in [18]. In this work, a stochastic high-level navigation planner provides expectations on travel times between different locations to the scheduling algorithm. In contrast to the previous architectures, robots Rin and Rout use a constraint network [22]. This network is continuously modified by an executor, a monitor and a planner in order to create configuration plans which specify causal, temporal, resources and information dependencies between individual actions. All the above works are based on scheduling approaches, which rely on a less fine grained model of the environment, where tasks are seen as black-boxes, being pre-specified instead of planned for, and with the scheduler trying to order them such that a set of timing constraints is satisfied. This does not allow for direct reasoning over the possible synergies between different tasks, and the possibility to interleave actions from different tasks in order to minimise execution time.

In recent years, there has also been work aiming at closing the loop between task planning and real world execution on a robot platform. The ROSPlan framework [5] is a general framework that allows for a task planner to be embedded into the Robot Operating System (ROS), a middleware widely used in the robotics community. As a proof of concept, ROSPlan has integrated the POPF planner [6], an ancestor of OPTIC. This integration with execution is also part of our future work, and we plan to explore how our techniques can be integrated in such an execution framework.

Additionally, some modelling languages have been developed with the goal of having a closer integration between planning and execution. Of particular interest are the NDDL [2] and the ANML [25] modelling languages. These are based on the notion of *timelines*, i.e., sequences of *tokens*. A token is a timed predicate that holds within a start and end time. The timeline representation was developed by NASA and used in open-source project EUROPA [13] in order to model and plan for real world problems and to allow a close integration with the execution of such plans. This representation was also exploited in T-REX [17], a model-based architecture for a robot control which used a tight integration of planning and execution. Another system closely integrating planning and execution is FAPE [20], built on the ANML language. Unfortunately, these system based on timelines do not have scalability of the state-of-the-art planning approaches we presented above.

To summarise, none of the reviewed systems is capable of executing a great variety of tasks and of interleaving execution of these tasks in order to minimise execution time.

2.4 Merging algorithms

There has also been a fair amount of research on the problem of plan merging.

The first planning system using merging was probably NOAH [23], as described in [10]. In NOAH, three criteria were introduced to handle possible interactions between plans: eliminate redundant preconditions, use existing operators, and optimise disjuncts. NONLIN [26] is also able to recognise if any goal is achievable by an operator already in a plan. If such operator is detected, then ordering constraints and variable bindings are used to change the such that the found operator is used to fulfil the goal.

Temporal and conditional plan merging is done in [27] which extends work of Yang [34]. For two input plans, the algorithm provides a new order of actions while detecting and removing useless actions, by checking if their effects are already fulfilled by some preceding action.

Related techniques to plan merging are *plan repair* and *plan refinement*. Refinement planning focuses on how to introduce a new action to an existing plan, and was introduced in [15]. Work in [14] uses a *partial plan* to save current refinements. The opposite case, i.e., removing an action from the plan is handled in *unrefinement planning* [28]. This addresses the *plan repair* problem of changing a plan when it cannot be executed. Despite the fact that it was proved that modifying an existing plan is no more efficient than a complete replanning in the general case [19], plan repair might still be efficient in certain domains. An example of recent plan refinement is planning for highly uncertain domains, such as underwater robotics [11]. In this case, one plan achieving a subset of tasks is produced. While it is executed, the current state is observed in order to limit uncertainties. If the robot has unexpected available resources, allowing it to perform more tasks, a pre-computed plan achieving another task is inserted into the global plan. Our proposed algorithm combines ideas from aforementioned merging approaches in order to allow flexible execution on a mobile robot.

3 PARTIAL ORDER PLANNING

We start by introducing the fundamental definitions and notation on POP that will be used throughout the paper. For a thorough overview of POP, we refer the reader to [33]. Furthermore, our merging algorithm assumes that plans have already been generated, so all actions we deal with are grounded. Thus, we omit details about lifted actions and bindings when describing the planning problems.

We start by defining a task in our framework. A task domain is a set $D = \{f_1, .., f_n\}$ of atomic formulas (atoms) that will describe a given state of the world. A state in this domain is represented by $F \subseteq D$, where f is true in state F if $f \in F$. For a durative action $a \in A$, we define its set of preconditions as the set of literals (atoms or their negation) $pre(a)$, its set of effects as the set of literals $\mathit{eff}(a)$ and its duration as $d(a) \in \mathbb{R}$. Furthermore, we split a into its start point $a_\vdash$ and end point $a_\dashv$, and define the sets $A_\vdash$ and $A_\dashv$ of action start points and end points, respectively. Finally, a planning problem is defined as $P = (I, G)$ such that $I \subseteq D$ is the initial state, and $G \subseteq D$ is the goal, where we say that a state F achieves the goal if $G \subseteq F$. A task is then defined as $\omega = \langle D, A, P \rangle$.

A partial order plan (POP) is a tuple $\pi = \langle \mathcal{A}, \mathcal{L}, \mathcal{O} \rangle$, where:

- $\mathcal{A}$ is a set of actions;
- $\mathcal{L}$ is a set of causal links. A causal link $a_j \xrightarrow{l} a_k$ represents that literal $l \in pre(a_k)$ is fulfilled as an effect of action a_j;
- $\mathcal{O}$ is a set of ordering constraints defining a partial order on the set $\mathcal{A}$. An ordering constraint $a_j \prec a_k$ represents that action a_j must finish before action a_k can start.

Given a POP π, an open condition $\xrightarrow{l} a_j$ means that literal $l \in pre(a_j)$ has not yet been linked to the effect of any action. An unsafe link (or a threat) is a causal link $a_j \xrightarrow{l} a_k$ such that there is an action $a_m \in \mathcal{A}$ that could possibly be ordered between a_j and a_k and threatens $a_j \xrightarrow{l} a_k$ by having $\neg l \in \mathit{eff}(a_m)$. The set of flaws of a POP π is given by the union of its open conditions and unsafe links. A POP planner searches through the space of POPs trying to resolve all flaws of π. To do that, the planner tries to close open conditions by adding new actions to $\mathcal{A}$ and new causal links to $\mathcal{L}$, and solve threats by adding new orderings to $\mathcal{O}$ to make sure that the threatening action a_m does not occur between the unsafe link $a_j \xrightarrow{l} a_k$. This can be done by, for example, *demotion*, i.e., adding the constraint $a_k \prec a_m$ to $\mathcal{O}$, or *demotion*, i.e., adding $a_m \prec a_j$ to $\mathcal{O}$.

In this work, we use the VHPOP planner as described in Section 2.2.

4 PROBLEM DEFINITION AND SOLUTION APPROACHES

In this paper, we are interested in solving the problem of finding a POP for a given set of input tasks. More specifically, given a set of tasks $\Omega = \{\omega_1, \ldots, \omega_n\}$, where $\omega_i = \langle D_i, A_i, (I_i, G_i) \rangle$ and the initial states of each problem are mutually consistent (I_j is consistent with I_k if for all $f \in D_j \cap D_k$, $f \in I_j$ if and only if $f \in I_k$), we want to find a plan that achieves all goals $G_1, ..., G_n$.

There are several ways of solving such a problem, in the remainder of this section, we present three different approaches – *unifying*, *sequencing* and *plan merging*. We argue that the merging approach provides a good trade-off between the plan quality of the unifying approach and the efficiency of the sequencing approach, hence, in the next section we will present a plan merging algorithm to solve our problem.

4.1 Unifying planning algorithm

This approach relies of *unifying* the set of tasks into a single task, i.e., $\omega = \left\langle \bigcup_{i \in \{1...n\}} D_i, \bigcup_{i \in \{1...n\}} A_i, (\bigcup_{i \in \{1...n\}} I_i, \bigcup_{i \in \{1...n\}} G_i) \right\rangle$ Then, one can use an appropriate planning algorithm to find a solution for the single unified task. While this approach can more easily take advantage of relations between goals in different tasks (e.g., two tasks that should be executed in the same location), it can suffer from scalability issues as finding a plan for the unified task can be much harder than finding plans for each individual task by itself. This approach is used by most planners, such as VHPOP, OPTIC and all presented planners from IPC 2014. Generally, the unified tasks are modelled a priori and passed directly as an input to such systems.

4.2 Sequencing planning algorithm

This approach generates a set of independent plans $\Pi_\omega = \{\pi_1, \ldots, \pi_n\}$, one for each task ω_i, and then *sequences* them to create a single final plan. For the resulting plan to be valid, one needs to decide on an ordering of the tasks and then modify the initial states of each task according to the final state of the plan for the preceding task. The ordering of the tasks can be done using a *scheduling algorithm* that can take into account extra timing constraints on the execution of tasks. This approach is common for mobile service robots, e.g., [29, 18], due to its simplicity and efficiency. This is because the planning problems to be solved when planning for the tasks independently will in general be much smaller than the single unified one problem. However, simple sequencing comes at the price of plan quality: this approach does not allow for the interleaving of actions from plans for different tasks, taking advantage of synergies between them.

4.3 Merging planning algorithm

This approach combines both aforementioned methods. It also plans for tasks separately, obtaining $\Pi_\omega = \{\pi_1, \ldots, \pi_n\}$ but then it reasons over each plan, *merging* them together into a better plan than the one obtained by simple sequencing. The final plan π_f consists of parts of the task plans in Π_ω and newly created plans Π_{join} which are used in order to connect these parts such that the final plan is still free of flaws. Furthermore, while the merging procedure adds an overhead at plan generation time when compared to the sequencing approach, it allows us to inspect each individual plan, and find synergies between plans for different tasks that allows us to interleave execution for different goals. A typical benefit of this approach in the mobile robot domain is providing the possibility to execute actions from different tasks when these actions share a common location. The algorithm we present in the next section follows this approach.

5 PROPOSED ALGORITHM

In this section, we present our merging algorithm. Before we describe it, we need to address an issue that can hinder the performance of the merging algorithm, and present a solution for it.

5.1 Dependency Caused by External Constraints

5.1.1 Problem Illustration

As stated in the introduction, we assume independent tasks, i.e., tasks where the goals can be partitioned and for which the execution of a plan for task ω_i is not a precondition to successfully execute task

ω_j. However, these tasks can become dependent due to external constraints. In mobile robots domains, these are typically spatial constraints. We will illustrate this problem on the following example.

DeliveryBot A mobile robot delivers packages around a building. A robot can *move between locations* (with duration 10), *load packages* (with duration 2), and *unload packages* (with duration 2). The robot receives two tasks:

1. ω_1: *"Deliver package1 to location s2."*
2. ω_2: *"Deliver package2 to location s0."*

Initial states

I_1	I_2	I_{fa}
(at s0) (pack1 s1)	(at s0) (pack2 s2)	(at s0) (pack1 s1) (pack2 s2)

Planning states

π_{1a}	π_{2a}	π_{fa}
(move s0 s1)[10] (load pack1 s1)[2] (move s1 s2)[10] (unload pack1 s2)[2]	(move s0 s2)[10] (load pack2 s2)[2] (move s2 s0)[10] (unload pack2 s0)[2]	(move s0 s1)[10] (load pack1 s1)[2] (move s1 s2)[10] (unload pack1 s2)[2] (load pack2 s2)[2] (move s2 s0)[10] (unload pack2 s0)[2]

Goal states

G_1	G_2	G_{fa}
(at s2) (pack1 s2)	(at s0) (pack2 s0)	(pack1 s2) (pack2 s0) (at s0)

Makespan

π_{1a}	π_{2a}	π_{fa}
24	24	38

Table 1. Plans π_{1a} and π_{2a} are merged into a single plan π_{fa}

Assume the initial state is I = {*(at s0), (pack1 s1), (pack2 s2)*}, and a partial order planner produces optimal plans π_{1a} and π_{2a}, as depicted in Tab. 1. An example of a final merged plan, π_{fb}, with makespan 38 is also given in the rightmost column of Tab. 1. Notice that action *(move s0 s2)* from plan π_2 is not used as its effects are satisfied by action *(move s1 s2)* from plan π_1. However, if the initial state is I = {*(at s0), (pack1 s1), (pack2 s3)*}, a partial order planner produces the plan π_{2b} which is again optimal, see Tab. 2. In this case, action *move(s0 s3)* will need to merged as well as its effects are not satisfied. Hence, the output plan has makespan 58. However, an optimal plan will not contain *(move s2 s0), (move s0 s3)* and will instead directly use action *(move s2 s3)*. Thus, the optimal plan for the two goals has makespan 48.

As the biggest contributor to cost of the plan is travel time in mobile robot domains, we see external constraints, such as spatial in this case, as a limiting factor for plan merging. Hence we argue that the dependency caused by external constraints must be addressed.

5.1.2 *Preprocessing External Constraints*

We currently handle the dependency caused by the external constraints, such as task location in our domain, manually by adding atoms related to the external constraints into the initial state. We will

Initial states

I_1	I_2	I_{fb}
(at s0) (pack1 s1)	(at s0) (pack2 s3)	(at s0) (pack1 s1) (pack2 s3)

Planning states

π_{1b}	π_{2b}	π_{fb}
(move s0 s1)[10] (load pack1 s1)[2] (move s1 s2)[10] (unload pack1 s2)[2]	(move s0 s3)[10] (load pack2 s3)[2] (move s3 s0)[10] (unload pack2 s0)[2]	(move s0 s1)[10] (load pack1 s1)[2] (move s1 s2)[10] (unload pack1 s2)[2] (move s2 s0)[10] (move s0 s3)[10] (load pack2 s2)[2] (move s2 s0)[10] (unload pack2 s0)[2]

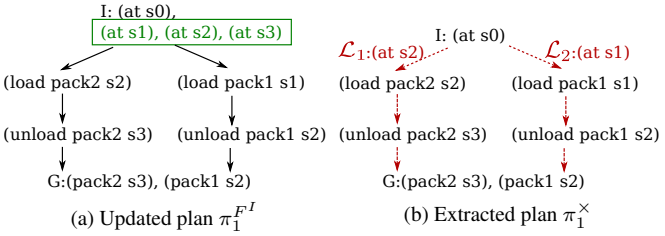
Goal states

G_1	G_2	G_{fb}
(at s2) (pack1 s2)	(at s0) (pack2 s0)	(pack1 s2) (pack2 s0) (at s0)

Makespan

π_{1a}	π_{2a}	π_{fa}
24	24	58

Table 2. Plans π_{1b} and π_{2b} are merged into a single plan π_{fb}



(a) Updated plan $\pi_1^{F^I}$ (b) Extracted plan $\pi_1^{\times}$

Figure 1. Preprocessing stages for task ω_1 in the DeliveryBot domain.

research an automated approach for tackling external constraints in our future work.

To illustrate the preprocessing in the DeliveryBot domain, the initial state for the task is updated with F^I = {*(at s1), (at s2), (at s3)*} and the new plans $\Pi_\omega^{I^I}$ are obtained, see Fig. 1a. Finally, F^I is extracted away from the plans and plans $\Pi_\omega^{\times}$ are obtained, see Fig. 1b when $\times$ signals that these plans are not any more valid. The red arrows highlights which causal links are not satisfied now.

5.2 POMer$_X$

After the above preprocessing, we tackle the following problem

$$\pi_f = POMer_X(\Pi^{\times}_{ext-\omega}, \mathbb{I}, \mathbb{G}\}, \quad (1)$$

where $\mathbb{I}, \mathbb{G}$ are sets of the initial state and the goals for all the input tasks, respectively and X stands for a POP algorithm which is used to provide plans. $\Pi^{\times}_{ext-\omega}$ is a set of extended partial plans $\pi^{\times}_{ext-\omega}$. An extend partial plan $\pi_{ext} = \langle \mathcal{A}, \mathcal{L}, \mathcal{O}, \mathcal{A}_{sat}, \mathcal{L}_{sat}, \mathcal{O}_{sat} \rangle$ is a POP plan π extended by subsets of actions $\mathcal{A}_{sat} \subseteq \mathcal{A}$, links $\mathcal{L}_{sat} \subseteq \mathcal{L}$ and orderings $\mathcal{O}_{sat} \subseteq \mathcal{O}$ which are satisfied in a merging state. The

merging state is expressed as a tuple

$$S = \langle t, F, F^+, F^-, Q_\vdash, Q_\dashv, \mathcal{L}, \Pi_{exp-join}, \Pi^\times_{exp-\omega}, \pi_s \rangle \quad (2)$$

where

- t is duration of the current plan π_s;
- F is a set of atomic propositions that holds in the state;
- F^+ is a set of achievers i.e.,
- F^- is a set of deleters i.e.,
- $Q_\vdash$ is a queue of starts of actions which can be merged into plan π_s;
- $Q_\dashv$ is a queue of ends of actions. Each end (a, t_s) of an action a must be merged into the plan when $t = t_s + d(a)$ where t_s is the time of the state S when the start of the action was added to the plan;
- $\mathcal{L}$ is a set of causal links that still need to be satisfied;
- $\Pi_{ext-join}$ is a set of temporary extended POP plans satisfying preconditions of $Q_\vdash$ or conditions of $\mathcal{L}$
- $\Pi^\times_{ext-\omega}$ is the set of extended POP plans for each task;
- $\pi_s = \langle \mathcal{A}_{\pi_s}, \mathcal{L}_{\pi_s}, \mathcal{O}_{\pi_s} \rangle$ is the POP plan that reaches the current state.

POMer$_X$ algorithm is a greedy complete search algorithm which searches over these states. The following operators are used to form the algorithm. The complete algorithm is shown in Algorithm 1.

arePlansMerged: This boolean operator tests, for each input plan $\pi^\times_{ext-i} \in \Pi^\times_{ext-\omega}$, if all its actions, links and orderings are satisfied, i.e., $\mathcal{A} = \mathcal{A}_{sat}$ and $\mathcal{L} = \mathcal{L}_{sat}$ and $\mathcal{O} = \mathcal{O}_{sat}$. Moreover, if all goals G_i of plan $\pi^\times_{exp-i}$ are such that $G_i \in F$. If this holds, the places are successfully merged;

updateActionsStarts(Π_{exp}): For each plan $\pi_{exp} \in \Pi_{exp}$ the set of the active actions is updated as $Q_\vdash = Q_\vdash \cup \mathcal{A}_{unsat}$ where $\mathcal{A}_{unsat}$ contains only those unsatisfied actions which do not have to be placed after start of any other unsatisfied action, i.e., $\mathcal{A}_{unsat} \cap \mathcal{A}_{sat} = \emptyset$ and $\mathcal{A}_{unsat} \subseteq \mathcal{A} \backslash \mathcal{A}_{sat}$. The placement requirement is requested by some unsatisfied ordering or link. Of course, if an action a is already presented in $Q_\vdash$ it is not added again.

updateLinks(Π_{exp}) : For each plan $\pi_{exp} \in \Pi_{exp}$ the set of the active links is updated as $\mathcal{L} = \mathcal{L} \cup \mathcal{L}^\times_{unsat}$ when $\mathcal{L}^\times_{unsat}$ refers to a set of *unsatisfied broken causal links* in plan π_{exp}. An unsatisfied causal link $a_j \xrightarrow{l} a_k$ occurs when l has not yet been achieved in the merged plan. This can happen in two different situations. First, the achiever (or deleter, in the case of a negative literal) action for l has not yet been merged to the final plan. This link will become satisfied after the achiever/deleter action is merged, thus it is not considered broken. In contrast, due to our preprocessing, the input plans contains *broken* causal links $\mathcal{L}^\times$, for which the achiever/deleter action is not present in the relaxed plan. Hence, unsatisfied broken causal links are those which are not satisfied even though their achievers in the relaxed plan are merged.

updateTemporaryPlans(f) : $\Pi_{exp-join} = \Pi_{exp-join} \cup \pi_{exp-join-l}$ when $\pi_{exp-join-l}$ is a plan to achieve unsatisfied literal l of an unsatisfied broken causal link. This plan is obtained by creating a new temporary task $\omega_l = (D_l, A_l(F, F_l))$ where D_l is the domain of literal l, the description of the current search state F is the initial state, and the unsatisfied literal l is the goal. The POP planner is then called to solve this simple sub-problem. If a plan cannot be found, an empty plan with infinite duration is added to set $\Pi_{exp-join}$. If l is achieved in the description of the current search state F, an empty plan with zero duration is added for integrity of operation choosePlan.

choosePlan : This operator returns the plan $\pi_{exp-min-dur} \in \Pi_{exp-join}$ with minimal duration. If more than one plan have minimal duration, the plans created in order to achieve a precondition of an action are chosen. If there is still more than one such plan, one of them is randomly chosen

get : get an action a_{sat} such that F satisfies $pre(a_{sat})$;

canBeMerged(a) : This boolean operation checks if the action a is not threatening any current causal link or violating any ordering in the partially ordered plan π_s.

backtrack($\Pi_{exp-join-old}$) : As none of the actions from $Q_\dashv$ can be merged, the algorithm backtracks to the previous state and $\Pi_{exp-join}$ in this recovered state is replaced by $\Pi_{exp-join-old}$ in order to propagate the plans. This safes not only runtime but also it is how the merging algorithm will not try to add the same failing plan again. If $\Pi_{exp-join-old} = \emptyset$ then all options in the recovered state were tried but none of them expand the state. Hence, the algorithm backtracks one more time.

Algorithm 1 POMer$_X(\Pi^\times_{exp-\omega}), \mathbb{I}, \mathbb{G}$

```
S = ⟨0, 𝕀, F_0^+, ∅, ∅, ∅, ∅, ∅, Π̃^×_ω, ∅⟩
while !arePlansMerged() do
    updateActionsStarts(Π^×_{exp−ω})
    updateLinks(Π^×_{exp−ω})
    if Π_{exp−join} = ∅ then
        for a ∈ Q_⊢ do
            updateTemporaryPlans(pre(a))
        end for
        for l ∈ ℒ do
            updateTemporaryPlans(f_l)
        end for
    end if
    π_{exp−min−dur} = choosePlan()
    updateActionsStarts(π_{exp−min−dur})
    updateLinks(π_{exp−min−dur})
    repeat
        a_sat = get()
    until canBeMerged(a_sat ) or a_sat = ∅
    if a_sat = ∅ then
        Π_{exp−join} = Π_{exp−join} \ π_{exp−min−dur}
        backtrack(Π_{exp−join})
    else
        mergeAction(a_sat)
    end if
end while
return π_s
```

mergeAction(a) : This operation merges a to the final plan, and updates the current state. It consists of:

createActions() : Action a is added to set of actions of plan π_s, i.e., $\mathcal{A}_{\pi_s} = \mathcal{A}_{\pi_s} \cup a$.

createLinks() : The preconditions of a are linked with their achievers F^+ creating new causal links in plan π_s for each $l \in pre(a)$ add to $\mathcal{L}_{\pi_s}$ $F^+(l) \xrightarrow{l} a$ if l is a positive literal, or $F^-(l) \xrightarrow{l} a$ if l is a negative literal.

createOrderings() : The orderings in plan π_s first assume that action a is concurrent to others already in the plan. If effects of a are threatening any existing link in π_s, the orderings of plan

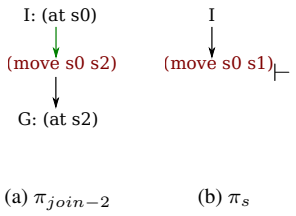


Figure 2. Processing plans from POMer$_X$ algorithm during first iteration.

π_s are updated in a way that a is *promoted* after the threaten link. We only use promotion as the demotion will be against the greedy decision of choosing actions based on their minimal duration. This same idea is applied by OPTIC planner.

updateAtoms() : The set of atoms F and the set of achievers F^+ and deleters F^- is updated by the effects of the a.

updatePlans() : In the input plans $\Pi^{\times}_{exp-\omega}$, the newly satisfied actions, links and orderings are added to the subsets $\mathcal{A}_{sat}, \mathcal{L}_{sat}, \mathcal{O}_{sat}$, respectively.

deleteActionsStarts : From set $Q_{\vdash}$ are deleted such actions whose effects are in the new F as they were satisfied by merged action a. Hence, not all actions from the input plans have to be merged into the final plan.

deleteLinks: From set $\mathcal{L}$ are deleted such links whose conditions are satisfied in the new F.

updateActionsEnds : $Q_{\dashv} = Q_{\dashv} \cup (a, t_s)$ when $t_s = t + d$ when d is action's duration and t the time valid in the current state.

Algorithm 2 mergeOption(activeOption, Θ)

```
createActions()
createLinks()
createOrderings()
updateAtoms()
updatePlans()
deleteActionsStarts()
deleteLinks()
if a ∈ Q⊢ then
    Q⊣ = Q⊣ ∪ (a, t)
end if
if a ∈ Q⊣ then
    t = Q⊣[a]
end if
```

The proposed algorithm using aforementioned operations is in Alg. 1 and method *mergeAction(a)* is in Alg. 2. The functionality of the algorithm using the DeliveryBot example is illustrated in Tab. 3. To summarise the algorithm, the new state is generated by merging an action a from an input plan $\pi^{\times}_{exp-i}$ or from a joining plan $\pi_{exp-join-f}$. The minimal duration of plans is used as a criterion how to choose only one action to merge. A joining plan is generated using the wrapped planner X in order to satisfied preconditions of actions in $Q_{\vdash}$ or conditions of links in $\mathcal{L}$. The algorithm is complete as it backtracks in its decisions.

6 EVALUATION

We have developed a version of our algorithm POMer$_{VHPOP}$ which embeds the VHPOP planner [24] for plan generation for individual tasks[2]. In this section, we evaluate the POMer$_{VHPOP}$ algorithm and compare it with other temporal planners based on plan quality, measured using the makespan of the found solution, and scalability. For each found plan, we run VAL, the validator of PDDL plans [3] in order to ensure that the plan is valid for domain and problem. The evaluated planners maximum memory usage was limited to 8 GB. All evaluation was run on Lenovo ThinkPad E-540 with Intel i74702MQ Processor (6MB Cache, 800 MHz).

6.1 Domains and problems

We evaluate using domains taken from IPC 2014. However, we generate our own planning problems in these domains, as our algorithm is based on the assumptions that tasks, i.e., goals in problems, are independent. Moreover, we assume only a single agent performing the actions. For the merging planning algorithm, we generate n problems P corresponding to n tasks, each problem has only a single goal. For the sequencing planning solution, we generate problems with single goals as with merging approach, but we set the initial state of problem p_{i+1} to the final (goal) state of problem p_i. This is done in order to be able to sequence the output plans to a single plan. The ordering of the problems is decided by a simple FIFO strategy. For the unifying planning solution, we generate a single unified problem P_n with n goals.

6.1.1 *Drivelog domain*

The Drivelog domain is the IPC 2014 domain closest to our main focus of mobile service robots as it has spatial constraints. In order to generate problems, we take the hardest problem from IPC 2014, i.e., problem P_{21}, and modify it so that it has a single agent, i. e., one truck with the driver already boarded. Then, we split the 23 goals for package placing into 23 single problems. The initial state of these single problems is extended by adding all atoms related to the spatial constraints, i.e., all *(at ?loc)* where *?loc* stands for any location in the problem.

6.1.2 *TMS domain*

This domain is another benchmarking domain for IPC 2014 which requires *concurrent* actions, a type of problem for which POP problems are especially suited. Even though this domain is about producing ceramic pieces, we choose it in order to demonstrate the capability of POMer$_{VHPOP}$ to handle concurrent actions. In this domain, a kiln represents the agent. Hence, our problems contain initial state that a kiln is always ready. We again use the hardest problem from IPC 2014 and we create 17 problems. The smallest problem contains 2 goals and the largest 50 goals.

6.2 POMer$_{VHPOP}$ in comparison to VHPOP

First, we analyse how our proposed algorithm improves over its wrapped planner, in this case VHPOP. Thus we compare three algorithms: POMer$_{VHPOP}$, VHPOP used to solve the unified problem, and VHPOP used to solve the sequencing problem. All algorithms were run on problems for Drivelog domain for 30 min and

[2] Available at https://github.com/mudrole1/POMer
[3] http://www.inf.kcl.ac.uk/research/groups/PLANNING/

Table 3. Changes to state S as the algorithm proceeds, where F^+ and F^- are omitted. The line refers to the number of line in Alg. 1

Line	t	F	$Q_{\vdash}$	$Q_{\dashv}$	$\mathcal{L}$	$\Pi_{ext-join}$	$\Pi^{\times}_{ext-\omega}$	π_s
1	0	(at s0) (pack1 s1) (pack2 s2)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	Fig. 1a	$\emptyset$
3	0	(at s0) (pack1 s1) (pack2 s2)	$\emptyset$	$\emptyset$	**(at s1)[10]** (at s2)[10]	Fig. 2a	Fig. 1a	$\emptyset$
7	0	(at s0) (pack1 s1) (pack2 s2)	(move so s1)$\vdash$[10]	$\emptyset$	(at s1)[✓] (at s2)[?]	Fig. 2a	Fig. 1a	$\emptyset$
12	0	(pack1 s1) (pack2 s2) (at s1)	$\emptyset$	((go so s1),0)$\dashv$	(at s1)[✓] (at s2)[?]	$\emptyset$	Fig. 1a	Fig. 2b

could use 8 GB of memory. The makespans are depicted in Fig. 3a. VHPOP-unifying is able to find a solution for only five problems before it reaches the memory limit. We also report on time and memory consumed, see Fig. 3b and Fig. 3c, respectively. As expected, VHPOP-unifying consumes the most memory for the most of the case and in problem 4, and problems 6-23 it does not find a solution before the limit of 8 GB is found. In contrast, the sequencing approach will be always the fastest and the most memory efficient however it always finds the worst makespan. For the largest problem, the makespan found by the sequencing approach is double the one found by POMer$_{VHPOP}$. This means that if the makespan is expressed in duration POMer$_{VHPOP}$ saves about 460 min in the biggest problem comparing to the fast sequencing approach even though it takes up to 7 min to provide a solution. To summarise, we can state that our merging algorithm wrapped around VHPOP significantly improved scalability of standalone VHPOP. As POMer$_{VHPOP}$ is not yet optimised, it is the slowest approach.

6.3 POMer$_{VHPOP}$ in comparison to IPC planners

This evaluation is focused on comparing properties of our proposed algorithm POMer$_{VHPOP}$ with the state of the art planners from IPC 2014, such as DAE$_{YAHSP}$, YAHSP3-MT, TFD and Itsat, as described in Section 2. Additionally, we also compare to the POP-based OPTIC planner [1] and to VHPOP using the sequencing solution.

6.4 Drivelog domain

For each problem from the Drivelog domain, the aforementioned planners were called with limited runtime to the time how long POMer$_{VHPOP}$ has run, see Fig. 3b. In order to express a quality of the found makespan, we introduce estimates of the best and the worst makespan. As the worst estimate, we use makespan found by VHPOP in sequencing approach and as the best estimate, we run DAE$_{YAHSP}$ for 30 minutes. Fig. 4 shows the recorded makespan for each problem. Even though the makespan is not a continuous function, we visualise the worst and the best estimates as a line in order to highlight these limits.

Note that DAE$_{YAHSP}$ struggles in problems 7 and 11 to provide a good solution. In both cases, the found solution is even slightly worse than the worst estimate. Hence, we exclude these problems from our following analysis. Our algorithm is better than the best estimate in seven problems by total difference 53.75. This means that an average difference per plan is 7.67 units in which makespan is recorded. As all packages must be loaded and unloaded in both plans, the POMer$_{VHPOP}$ has only two options how to find better plan - place loading and unloading in concurrency and find better path between locations. In 14 cases, POMer$_{VHPOP}$ found worse solution than the best estimate by total difference 442.75 or 31.63 average difference. The most of these cases is for problems with more goals which is expected as the greedy heuristics is driven to a local optima which occurs more often in bigger problems.

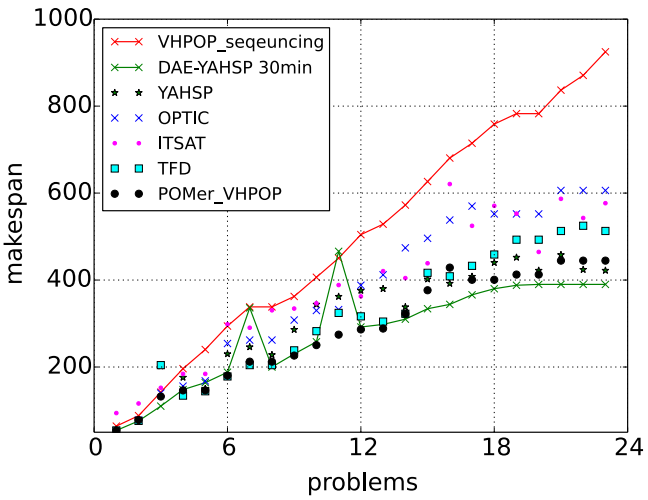
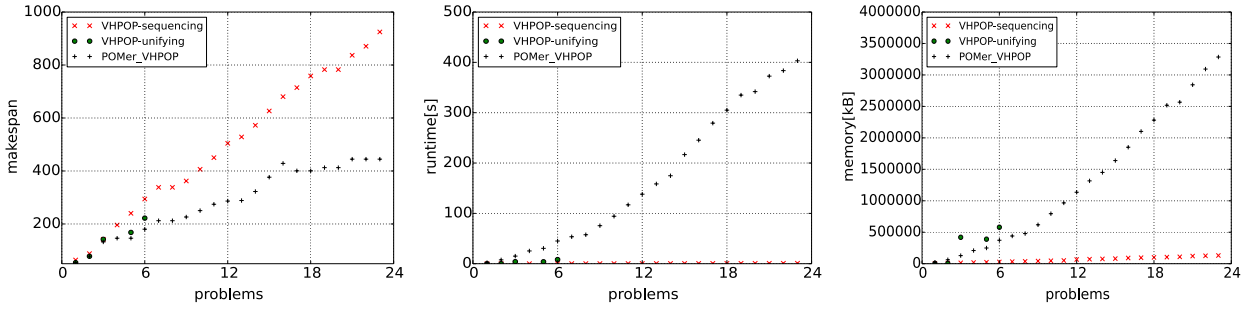


Figure 4. Makespans for problems in Drivelog domain.

6.5 TMS domain

Even though, planners DAE$_{YAHSP}$, YAHSP and TFD perform very well in Drivelog domain, they are unable to find a valid solution in TMS domain as they do not handle concurrent actions correctly. As result, we are comparing POMer$_{VHPOP}$ with only OPTIC, ITSAT and we add original VHPOP as well, as it is known that POP planners are able to handle concurrency naturally. However an interesting phenomena occurs for our problems – all planners find almost the



(a) Makespan (b) Time (c) Memory

Figure 3. Comparision of POMer$_{VHPOP}$, VHPOP-unifying and VHPOP-sequencing.

same makespans. The negligible difference in OPTIC is caused by a fact that starts of two actions can never occur at the same time due to forward chaining so one start is postponed by $\Delta t = 0.001$ after previous. This phenomena occurs due to a fact that a kiln used for baking ceramic has no resource limits. Thus all the ceramic pieces can be baked in parallel.

7 CONCLUSION

We presented an approach for merging of partial order plans especially suited for mobile service robots that need to execute tasks at different locations in an environment. The approach is based on first solving relaxed problems for each individual task, and then perform search over the solutions for these relaxed problems, stitching them together in a way that takes advantage of the synergies between the different tasks. We provided an evaluation of our approach on two benchmarking domains, showing that, for the class of problems we are interested in, it is competitive with state-of-the-art temporal planners. Furthermore, it illustrated our approaches flexibility, as it can perform well in the two domains we analysed, while the other approaches have issues in at least one of the domains.

Future work includes developing an automatic relaxation of the individual problems, and tackling issues related to the execution of the plans we are generating in a mobile robot. This includes closing the loop between plan generation and execution, for which we feel partial order plans are better suited than totally ordered ones, and tackle other common issues for service robotics, such as timing constraints on task execution, the uncertainty inherent to execution in the real world, or merging of plans for new tasks arriving during execution.

REFERENCES

[1] J. Benton, Amanda Jane Coles, and Andrew Coles, 'Temporal planning with preferences and time-dependent continuous costs', in *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, (2012).

[2] Sara Bernardini and David E. Smith, 'Developing domain-independent search control for europa2', in *ICAPS-07 Workshop on Heuristics for Domain-independent Planning*, (2007).

[3] Jacques Bibaï, Pierre Savéant, Marc Schoenauer, and Vincent Vidal, 'An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning', in *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-2010)*, pp. 18–25, Toronto, ON, Canada, (May 2010). AAAI Press.

[4] K. E. C. Booth, T. T. Tran, G. Nejat, and J. C. Beck, 'Mixed-integer and constraint programming techniques for mobile robot task planning', *IEEE Robotics and Automation Letters*, **1**(1), 500–507, (Jan 2016).

[5] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcís Palomeras, Natàlia Hurtós, and Marc Carreras, 'Rosplan: Planning in the robot operating system', in *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pp. 333–341, (2015).

[6] A. J. Coles, A. I. Coles, M. Fox, and D. Long, 'Forward-chaining partial-order planning', in *The International Conference on Automated Planning and Scheduling (ICAPS-10)*, (May 2010).

[7] Brian Coltin and Manuela M. Veloso, 'Online pickup and delivery planning with transfers for mobile robots', in *Proc. of 2014 IEEE Int. Conf. on Robotics and Automation (ICRA)*, (2014).

[8] Brian Coltin, Manuela M. Veloso, and Rodrigo Ventura, 'Dynamic user task scheduling for mobile robots.', in *Proc. of 2011 AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, (2011).

[9] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger, 'Using the context-enhanced additive heuristic for temporal and numeric planning', in *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, (2009).

[10] David E. Foulser, Ming Li, and Qiang Yang, 'Theory and algorithms for plan merging', *Artificial Intelligence*, **57**(23), 143 – 181, (1992).

[11] C. Harris and R. Dearden, 'Contingency planning for long-duration auv missions', in *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pp. 1–6, (Sept 2012).

[12] Malte Helmert, 'The fast downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[13] M. Do J. Frank M. Iatauro T. Kichkaylo P. Morris J. Ong E. Remolina T. Smith D. Smith J. Barreiro, M. Boyce, 'Europa: A platform for ai planning, scheduling, constraint programming, and optimization', in *Proc. of 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*.

[14] Subbarao Kambhampati, 'Refinement planning as a unifying framework for plan synthesis', *Artificial Intelligence Magazine*, **18**(2), 67–97, (1997).

[15] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang, 'Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning', *Artificial Intelligence*, **76**, 167–238, (1995).

[16] Wing-Yue Geoffrey Louie, Tiago Stegun Vaquero, Goldie Nejat, and J. Christopher Beck, 'An autonomous assistive robot for planning, scheduling and facilitating multi-user activities', in *Proc. of 2014 IEEE Int. Conf. on Robotics and Automation (ICRA)*, (2014).

[17] Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen, 'T-rex: A model-based architecture for auv control', *3rd Workshop on Planning and Plan Execution for Real-World Systems*, **2007**, (2007).

[18] Lenka Mudrova, Bruno Lacerda, and Nick Hawes, 'An integrated control framework for long-term autonomy in mobile service robots', in *2015 European Conference on Mobile Robots (ECMR)*. IEEE, (2015).

[19] Bernhard Nebel and Jana Koehler, 'Plan reuse versus plan generation: A theoretical and empirical analysis', *Artificial Intelligence*, **76**, 427–454, (1995).

[20] Filip Dvořák, Arthur Bit-Monnot, Félix Ingrand, and Malik Ghallab, 'A

flexible anml actor and planner in robotics', in *ICAPS PlanRob WorkShop*, Portsmouth, USA, (June 2014).
[21] Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani, 'New encoding methods for sat-based temporal planning', 110–117, (2013).
[22] Maurizio Di Rocco, Federico Pecora, and Alessandro Saffiotti, 'When robots are late: Configuration planning for multiple robots with dynamic goals.', in *Proc. of 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (2013).
[23] Earl D. Sacerdoti, 'The nonlinear nature of plans', in *Proceedings of the 4th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'75, pp. 206–214, (1975).
[24] Reid G. Simmons and Håkan L. S. Younes, 'VHPOP: versatile heuristic partial order planner', *CoRR*, **abs/1106.4868**, (2011).
[25] D. E. Smith, J. Frank, and W. Cushing, 'The anml language', in *ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, (2008).
[26] A. Tate, 'Generating project networks', in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI'77, pp. 888–893, (1977).
[27] Ioannis Tsamardinos, Martha E. Pollack, and John F. Horty, 'Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pp. 264–272, (2000).
[28] Roman Van Der Krogt and Mathijs De Weerdt, 'Plan repair as an extension of planning', in *In Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05*, pp. 161–170, (2005).
[29] Manuela M. Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Thomas Kollar, Cetin Mericli, Mehdi Samadi, Susana Brandao, and Rodrigo Ventura, 'Cobots: Collaborative robots servicing multi-floor buildings.', in *Proc. of 2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, (2012).
[30] Vincent Vidal, 'YAHSP2: Keep it simple, stupid', in *Proceedings of the 7th International Planning Competition (IPC-2011)*, pp. 83–90, Freiburg, Germany, (June 2011).
[31] Vincent Vidal, Lucas Bordeaux, and Youssef Hamadi, 'Adaptive K-parallel best-first search: A simple but efficient algorithm for multi-core domain-independent planning', in *Proceedings of the 3rd Symposium on Combinatorial Search (SOCS-2010)*, pp. 100–107, Stone Mountain, GA, USA, (July 2010). AAAI Press.
[32] David Wang and Brian C. Williams, 'tburton: A divide and conquer temporal planner', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 3409–3417, (2015).
[33] Daniel S. Weld, 'An introduction to least commitment planning', *AI magazine*, **15**(4), 27, (1994).
[34] Qiang Yang, *Intelligent Planning: A Decomposition and Abstraction Based Approach*, Springer-Verlag, London, UK, UK, 1997.