Mobile Robot Planning Using Action Language $\mathcal{BC}$ with an Abstraction Hierarchy

Shiqi Zhang[1](✉), Fangkai Yang[2], Piyush Khandelwal[1], and Peter Stone[1]

[1] Department of Computer Science, The University of Texas at Austin,
2317 Speedway, Stop D9500, Austin, TX 78712, USA
{szhang,piyushk,pstone}@cs.utexas.edu

[2] Schlumberger Software Technology, Schlumberger Ltd,
5599 San Felipe Rd, Houston, TX 77056, USA
fkyang@cs.utexas.edu

Abstract. Planning in real-world environments can be challenging for intelligent robots due to incomplete domain knowledge that results from unpredictable domain dynamism, and due to lack of global observability. Action language $\mathcal{BC}$ can be used for planning by formalizing the preconditions and (direct and indirect) effects of actions, and is especially suited for planning in robotic domains by incorporating defaults with the incomplete domain knowledge. However, planning with $\mathcal{BC}$ is very computationally expensive, especially when action costs are considered. We introduce algorithm *PlanHG* for formalizing $\mathcal{BC}$ domains at different abstraction levels in order to trade optimality for significant efficiency improvement when aiming to minimize overall plan cost. We observe orders of magnitude improvement in efficiency compared to a standard "flat" planning approach.

1 Introduction

To operate in real-world environments, intelligent robots need to represent and reason with a large amount of domain knowledge about robot actions and environments. However, domain knowledge given to the robot is usually incomplete (due to unpredictable domain dynamism) and defeasible (i.e., usually true but not always). From STRIPS [4] to PDDL [17], many action languages (and their extensions) have been developed to support automated plan generation by formalizing action preconditions and effects. While some action languages support reasoning about the knowledge not directly related to actions, e.g., PDDL has semantics to reason with axioms [25], most action languages lack a strong capability of reasoning with incomplete knowledge in dynamic domains, making it difficult to embrace rich domain knowledge into planning scenarios. Action language $\mathcal{BC}$ can be used for planning with guaranteed soundness by formalizing the preconditions and (direct and indirect) effects of actions [14]. $\mathcal{BC}$ inherits the knowledge representation and reasoning (KRR) advantages from action languages $\mathcal{B}$ [9] and $\mathcal{C}+$ [10], and is especially suited for planning in robotic domains.

Unfortunately, in robotic domains where action costs need to be considered, planning with action language $\mathcal{BC}$ is very computationally expensive.

F. Calimeri et al. (Eds.): LPNMR 2015, LNAI 9345, pp. 502–516, 2015.
DOI: 10.1007/978-3-319-23264-5_42

For instance, in the office domain presented in [13], generating the optimal plan to visit three people in different rooms takes more than 5 min on a powerful desktop machine (details in Sect. 5), where the optimal plan has about 30 actions. Such long planning time prevents the robot from being deemed useful in real-world environments.

Hierarchical planning has been studied for years and people have developed many algorithms including Hierarchical Task Network (HTN) [3] and Hierarchical Planning in the Now (HPN) [12]. Different from existing work on hierarchical planning that aims to reduce the amount of search with guaranteed optimality (e.g., [16]), we trade optimality for significant improvements in efficiency (similar to HPN). We adapt the idea of describing task domains at different abstraction levels [6] and propose an algorithm to enable hierarchical planning with action language $\mathcal{BC}$ in real-world robotic domains.

This algorithm has been fully implemented in simulation and on a physical robot. Experiments on a mail collection problem show 2 orders of magnitude improvements of efficiency with a 11.25 % loss in optimality, compared to a baseline algorithm that plans with a non-hierarchical domain description in $\mathcal{BC}$ [13]. To the best of our knowledge, this is the first work that combines the KRR advantages of a modern action language and the efficiency of hierarchical planning to enable mobile robots to compute provably sound plans in real-world environments.

2 Related Work

This work is closely related to research areas including action languages and hierarchical planning. We select representative research on these topics.

Action Languages: The planning domain definition language (PDDL) has been widely applied to planning problems [17]. One of the most appealing advantages of (the official versions of) PDDL is its syntax, which despite being simple supports important features of STRIPS [4], ADL [20], and other features such as conditional action effects (PDDL1.2) and numeric fluents (PDDL2.1). Furthermore, advanced planning algorithms such as Fast-Foward [19] and Fast-Downward [11] have been implemented in existing planning systems supporting PDDL.

While PDDL is strong in efficient plan generation, the official versions of PDDL do not focus on reasoning with default knowledge, which is important for robots to plan with incomplete knowledge in dynamic environments. Action language $\mathcal{C}+$ supports the representation and reasoning with defaults [10], but does not allow recursively defined fluents that are frequently needed in robotic domains (action language $\mathcal{B}$ does), as will be shown in Sect. 3. $\mathcal{BC}$, an action language recently developed based on answer set semantics [8], can be used to compute provably sound plans while supporting representation of and reasoning with defaults with exceptions at different levels [14].

Recently, a two-level architecture has been developed for KRR in robotics [26], where the high level uses action language AL for symbolic planning and the low level uses probabilistic algorithms for modeling uncertainties. In that work, each default is associated with a consistency-restoring rule for restoring consistency in history. In contrast, we intentionally make our robots memoryless to avoid reasoning about history, i.e., whenever robot observations have conflicts with defaults, our robot starts over by replanning with defaults and the observed "facts".

Hierarchical Planning: In existing research on hierarchical planning, the hierarchy is frequently constructed through setting up connections either between actions or between states. For instance, macro-actions (also called *complex* or *composite* actions) are described as a sequence of primitive actions and possibly some imperative constructs, e.g., hierarchical task network [3], planning with composite actions [1], planning with complex actions [18], ordered task decomposition [2], and hierarchical planning in the now [12]. These macro-actions are either directly expanded after a plan is generated, or expanded in the reasoning process using a predefined structure. These macro-actions limit the flexibility of reducing plan costs at a finer abstraction level.

Another way of constructing the hierarchy is to describe the domain at different abstraction levels through setting up connections between states, where a state at a coarser (higher) level includes a set of states at a finer (lower) level [6,23,24]. Planning in such systems happens in a top-down manner and constraints extracted from coarser levels help improve the efficiency in computing plans at finer levels. This mechanism allows more flexibility in planning at finer levels, compared to macro-based hierarchical planning algorithms. In this paper, we introduce action costs to such abstraction-based hierarchical planning algorithms and implement the algorithm using action language $\mathcal{BC}$ on a real robot system.

3 Abstraction Hierarchy Formalization

A $\mathcal{BC}$ action description D denotes a transition system $T(D)$, which is a digraph whose vertices are *states*, which is a set of atoms, and whose edges are *actions*. A transition in $T(D)$ is of the form $\langle s, a, s' \rangle$, where a is an action constant, and s and s' are states before and after executing a. A path $P(n)$ of length n in the transition system is of the form:

$$\langle s_0, a_0, \dots, s_{n-1}, a_{n-1}, s_n \rangle$$

where s_i $(0 \leq i \leq n)$ are states and a_i $(0 \leq i \leq n-1)$ are actions. We use $Len(P)$ to denote the length of a path. $P^s(i)$ denotes state s_i and $P^a(i)$ denotes action a_i. We use $f(D)$ to represent the set of fluents occurring in D, and $a(D)$ to represent the set of actions occurring in D. To define the notion of abstraction hierarchy, we first define the cost function C that maps a tuple (s, a) to an integer $C(s, a)$ that denotes the cost of executing action a at state s. Furthermore, $cost\big(P(n)\big)$ is the cost of path $P(n)$:

$$cost\big(P(n)\big) = \Sigma_{0 \leq i < n} C(s_i, a_i) \tag{1}$$

Given an action description D and a cost function C, its *abstraction hierarchy* $\mathcal{H}$ is a tuple $(\mathcal{D}, \mathcal{L})$: $\mathcal{D}$ is a list of action descriptions $D_1, D_2, \ldots, D_d$ such that $f(D_i) \subseteq f(D_j)$ for $1 \leq i < j \leq d$, where $D_d = D$ and d is the depth of $\mathcal{H}$; and $\mathcal{L}$ is the *step bound estimation function.*

$$\mathcal{L}(a) = \max_{\langle s, a, s' \rangle \in T(D_i)} \Big(Len\big(\hat{P}(s, s')\big)\Big) \tag{2}$$

Given an action constant $a \in a(D_i)$, $\mathcal{L}$ maps a to an integer $\mathcal{L}(a)$ representing the minimum number of steps needed to ensure that the effect of a can be optimally achieved using actions in $a(D_{i+1})$ as shown in Eq. 2, where $\mathcal{L}$ is independent of s and s', and is precomputed to reduce the planning time[1]. $\hat{P}$ represents the path of the plan that leads the transition from s to s' with minimum plan cost. If we use $A(s)$ to represent the set of literals that specify state s, $\hat{P}(s, s')$ can be computed by:

$$\hat{P}(s, s') = \mathop{\arg\min}_{\substack{P(n) \in T(D_{i+1}),\, n \in \mathbb{N}, \\ A(P^s(0)) \subseteq A(s),\, A(P^s(n)) \subseteq A(s')}} \Big(cost\big(P(n)\big)\Big) \tag{3}$$

Note that states s and s' and action a are at level i while path P is at level $i+1$. Intuitively, the abstraction hierarchy $\mathcal{H}$ contains a set of action descriptions where each description formalizes the same dynamic domain at a different granularity. The hierarchy is organized from the most coarse description D_1 to the most concrete description D_d. Different from existing work on hierarchical planning using macro actions, we use function $\mathcal{L}$ to provide step bounds in the search for plans at lower levels. This is an important criterion of our approach as it provides flexibility in reducing overall plan costs in lower levels. As an example, we next apply this hierarchy to a real-world robot planning problem in $\mathcal{BC}$.

Mail Collection Problem: A mobile robot drops by offices at 2 pm every day to collect outgoing mail from the residents. However, some people may not be in their offices at that time, so they can pass their outgoing mail to colleagues in other offices, and send this information to the robot. When the robot collects the mail, it should obtain it while only visiting people as necessary. An example floor plan is shown in Fig. 1. We will use meta-variables E, E_1, $E_2, \ldots$ to denote people (*alice*, *bob*, *carol*, *daniel* and *erin*), R, R_1, $R_2, \ldots$ to denote rooms, and K, K_1, $K_2, \ldots$ to denote doors. Specifically, o_1, o_2, o_3, o_4 are offices, lab_1 is a lab and *cor* is a room, where offices and labs are sub-sorts of room.

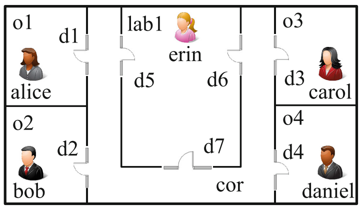


Fig. 1. Example floor plan.

[1] As a preprocessing step, computing $\mathcal{L}$ does not affect the runtime efficiency, so we leave the discussion of its complexity to future work.

This example domain has been formulated at three levels of abstraction. The fluents at the most abstract level primarily describe how mail is passed from one person to another and if mail has been collected from each person. At the middle level, we add fluents to describe the connections of rooms through doors, but still disregard the details about the robot's more refined position in a room and if the doors are open or not. Finally, all domain details are added into the bottom level. An action in the bottom level must be primitive (i.e., can be physically executed by the robot) and currently this hierarchy is manually constructed.

Action Description D_1: In D_1, we use $passto(E_1, E_2)$ to describe E_1's mail has been passed to E_2. The current locations of the robot and a person E are described by $loc = R$ and $inside(E, R)$ respectively. Whether the robot has collected mail from person E is represented by $mailcollected(E)$. For instance, the static law below states: if E_1's mail has been passed to E_2 and that the robot has collected mail from E_2, then E_1's mail has been collected as well using a recursive definition of fluent *mailcollected*:

$$mailcollected(E_1) \textbf{ if } mailcollected(E_2),\ passto(E_1, E_2).$$

The two laws below state that person E cannot be in two different rooms at the same time and that by default E's location does not change over time (a commonsense law of inertia), where *inside* is an *inertial fluent.*

$$\begin{array}{l} \sim inside(E, R_2) \;\; \textbf{if} \;\; inside(E, R_1) \;\; (R_1 \neq R_2). \\ \textbf{inertial} \;\; inside(E, R). \end{array}$$

Action *serve* in D_1 states that serving person E in room R causes $mailcollected(E)$ to be true and the robot to be in R.

$$\begin{array}{l} serve(E) \;\; \textbf{causes} \;\; mailcollected(E). \\ serve(E) \;\; \textbf{causes} \;\; loc = R \;\; \textbf{if} \;\; inside(E, R). \end{array}$$

Action Description D_2: D_2 inherits all fluents and corresponding non-action rules from D_1 (actions of D_1 are discarded) and further adds fluents to describe whether a room has a door using $hasdoor(R, K)$ and whether two adjacent rooms are connected through a door using $acc(R_1, K, R_2)$. The static laws below state that if two rooms share the same door, then they are accessible to each other through the door and that *acc* is symmetric.

$$\begin{array}{l} acc(R_1, K, R_2) \;\; \textbf{if} \;\; hasdoor(R_1, K),\ hasdoor(R_2, K). \\ acc(R_1, K, R_2) \;\; \textbf{if} \;\; acc(R_2, K, R_1). \end{array}$$

We add defaults to reason with incomplete knowledge. For instance, rooms R_1 and R_2 are not accessible through door K by default. This default value can be reverted if there is evidence supporting the opposite.

$$\textbf{default} \;\; \sim acc(R_1, K, R_2).$$

Using the fluents in D_2, we can formalize action $collectmail(E)$ that is similar to action *serve* in D_1, and action $cross(K)$ that allows the robot to cross door K

to move from room R_1 to room R_2, if R_2 is accessible from R_1 through door K. There is a restriction on the executability of $cross(K)$: the robot cannot cross a door if that door is not accessible from the robot's current location.

$$\begin{array}{l} cross(K) \textbf{ causes } loc = R_2 \textbf{ if } loc = R_1,\ acc(R_1, K, R_2). \\ \textbf{nonexecutable } cross(K) \textbf{ if } loc = R, \sim hasdoor(R, K). \end{array}$$

Action Description D_3*:* D_3 inherits all fluents and corresponding non-action rules from D_2 (actions of D_2 are discarded) and further introduces fluents $beside(K)$ to describe whether the robot is beside door K, $facing(K)$ to describe whether the robot is beside and facing door K, and $open(K)$ to describe if door K is open. We use Monte Carlo Localization [5] to estimate the robot's exact position (including orientation) in physical environments. Using an occupancy-grid map with manually added semantic labels, this exact position specifies the values of loc, $beside$ and $facing$ and is also used for path planning. Using the fluents in D_3, we can formalize the primitive actions $approach(K)$, $opendoor(K)$, $gothrough(K)$, and $collectmail(E)$. D_3 corresponds to the "flat" action description presented in previous work [13].

Action descriptions D_1, D_2 and D_3 together determine $\mathcal{D}$, the first element of the abstraction hierarchy $\mathcal{H}$. The other element is the step bound estimation function $\mathcal{L}$, which is partially decided by the cost function C, as presented Eq. 1. The value of $C(s, a)$ is assigned empirically based on robot experiments using existing approach [13]. As an illustrative example, let us consider the calculation of $\mathcal{L}(serve)$ using Eqs. 2 and 3. Since $serve$ is an action in D_1, we first collect all possible $\langle s, serve, s'\rangle \in T(D_1)$. The longest path in D_2 that can be used to achieve the same effect as a $serve$ action in D_1 occurs when $loc = o_3 \in s$ and $mailcollected(alice) \in s'$. The corresponding path $P_2(5)$ includes the following actions in the order of execution:

$$cross(d_3), cross(d_6), cross(d_5), cross(d_1), collectmail(alice).$$

Consequently, $\mathcal{L}(serve) = 5$. Similarly, $\mathcal{L}(cross) = 3$.

4 Planning Using an Abstraction Hierarchy

In this section, we formally define two planning problems that aim to minimize the plan length (Type-I) and plan cost (Type-II) respectively, where the first is a special case of the second. Then we propose two algorithms to solve Type-II problems using an abstraction hierarchy.

Type-I Problem: A Type-I planning problem aims at minimizing the plan length (i.e., the number of actions), and is defined as a tuple (D, S, G). D is an action description; S is a *state constraint set* including state constraints of the form $i : A_i$, where i is an integer denoting the *timestamp* at which A_i (a set of fluent atoms) needs to be met; and G is a list of fluent atoms G_i, which are *goals*. The initial system state is specified as a part of the state constraint set as $0 : A_0 \in S$.

Given (D, S, G), a *satisfactory path* is a path $P(n)$ (defined in Sect. 3) of the transition system $T(D)$ such that $A_i \subset s_i$ for every $i : A_i \subset S$, and $G_i \in s_n$ for every $G_i \in G$. The *satisfactory plan* is the list $(a_0, \cdots, a_{n-1})$ obtained from $P(n)$. A satisfactory plan is a *shortest plan* if the length of the satisfactory path is minimal among all satisfactory paths. Algorithms that solve this problem do not consider the overall cost of plans. To find the shortest-length plan in a Type-I problem, we incrementally increase the plan length in solvers until a satisfactory plan is found.

Type-II Problem: A Type-II problem aims at minimizing overall plan cost, and can be defined as a tuple (D, S, G, C), where C is the cost function of actions. Given an optimizing planning problem, an *optimal path* $P(n)$ is a satisfactory path of the satisfactory planning problem (D, S, G) such that the overall cost of the path, $cost\big(P(n)\big)$, is minimal among all satisfactory paths of (D, S, G). Incrementally lengthening the plan length will not necessarily lead to the optimal plan because a very long plan could have the lowest cost. Algorithms that solve this problem compute plans toward minimizing the overall cost of the plan.

Without concurrent actions, a Type-I problem can be reduced to a Type-II problem by using unit cost for any (s, a) in function C. Therefore, we will focus on applying the abstraction hierarchy to Type-II problems.

4.1 PlanHG: The Proposed Planning Algorithm

Given a Type-II problem (D, S, G, C) and hierarchy $\mathcal{H} = (\mathcal{D}, \mathcal{L})$, for a state $P^s(i)$ in a path $P(n)$, we define its *shifted timestamp* in Eq. 4. The shifted timestamp for state $P^s(i)$ is the timestamp when this state constraint needs to be achieved when $P(n)$ is further elaborated at the next level $i + 1$. State constraint $sh(i) : P^s(i)$ is functionally a "bottleneck" that guides the solution path in the next level of hierarchy by reducing the search space.

$$sh(i) = \sum_{a_j \in P(n),\ j<i} \mathcal{L}\big(P^a(j)\big) - 1 \tag{4}$$

Furthermore, we impose the restriction that the only constraint contained in S is the initial state that can be sensed by the robot. This restriction allows us to easily project S and G on to each level of the hierarchy as S_i and G_i, respectively. As a result, we obtain an optimizing planning problem at each abstraction level: (D_1, S_1, G_1, C), $(D_2, S_2, G_2, C), \ldots$, (D_d, S_d, G_d, C). For a Type-II problem (D_i, S_i, G_i, C) at the ith level, let the path obtained from level $i - 1$ be $P_{i-1}(n)$. We define the *extended state constraint set* at the ith level, S'_i, in Eq. 5. Therefore, a *guided* Type-II problem (D_i, S'_i, G_i, C) is formed at the ith level using (D_i, S_i, G_i, C) and state constraints extracted from level $i - 1$. We call it a "guided"problem because the state constraints reduce the search space in planning at the ith level.

$$S'_i = S_i \cup \bigcup_{1 \le j \le n-1} sh(j) : P^s_{i-1}(j). \tag{5}$$

Algorithm 1. PlanHG: Planning using $\mathcal{H}$ while applying $\mathcal{L}$ globally

```
Input: Type-II problem $(D, S, G, C)$, and abstraction hierarchy $\mathcal{H} = (\mathcal{D}, \mathcal{L})$, where
    $\mathcal{D} = (D_1, \ldots, D_d)$, and $D_d = D$
1: create a list of problems $(D_i, S_i, G_i, C), 1 \leq i \leq d$ using $(D, S, G, C)$ and $\mathcal{D}$
2: generate path $P_1$ for $(D_1, S_1, G_1, C)$
3: for level $i \in \{2, \ldots, d\}$ do
4:     compute $S'_i$ based on $S_i$ and $P_{i-1}$, using Eq. (5)
5:     generate path $P_i$ for $(D_i, S'_i, G_i, C)$
6: end for
7: return the plan obtained from $P_d$
```

Solving Type-II problems directly using the optimization function of answer set solvers may require prohibitively long time. Using an abstraction hierarchy, we can obtain a list of guided Type-II problems. In practice, each level has action *noop*(I) of zero cost representing no operation at timestamp I. The optimal path generated at a higher level is passed down as "bottlenecks" such that the Type-II problem at a lower level becomes a guided Type-II problem, until the bottom level is reached. This approach guarantees the soundness of generated plans but may lead to sub-optimal results. We present Algorithm 1 that solves Type-II problems using an abstraction hierarchy $\mathcal{H} = (\mathcal{D}, \mathcal{L})$. We call this algorithm *PlanHG* to identify the use of the hierarchy and *global* minimization of plan costs at each level.

In the mail collection domain, the robot can obtain the initial state constraint from its internal knowledge base and sensor readings. For instance, initially the robot can perceive that it is located in *cor* and beside d_4. Such information is used to automatically create a state constraint set S:

$$\{0\!: loc = cor,\ 0: ibeside(d_4),\ 0\!:\sim\!facing(D)\}.$$

Given a goal G of *mailcollected*$(erin)$, at level 1 the projection S_1 becomes $\{0\!: loc = cor\}$ and the goal $G_1 = G$. The solver returns the optimal path:

$$\langle s_0 = \{loc = cor, \sim\!mailcollected(erin)\}, a_0 = \{serve(erin)\},$$
$$s_1 = \{loc = lab_1, mailcollected(erin)\}\rangle$$

Now, we can compute the shifted timestamps for s_0 and s_1 given $\mathcal{L}(serve) = 5$ (Sect. 3) and we obtain the guided state constraint set S'_2:

$$0\!: loc = cor, 0\!:\sim\!mailcollected(erin),$$
$$5\!: loc = lab_1, 5: mailcollected(erin).$$

The guided Type-II problem (D_2, S'_2, G_2, C) aims to find an optimal plan such that at time 0 the robot is in *cor*, at time 5 the robot is in lab_1 and Erin's mail is collected, and the goal of Erin's mail being collected is achieved. Indeed, the optimal plan generated at this level consists of two actions: $cross(d_7), collectmail(erin)$. Note that $cross(d_7)$ is selected because it has a lower cost than $cross(d_5)$ and $cross(d_6)$. Using this plan, we can generate the

Algorithm 2. PlanHL: Planning using $\mathcal{H}$ while applying $\mathcal{L}$ locally

```
Input: Type-II problem $(D, S, G, C)$, and abstraction hierarchy $\mathcal{H} = (\mathcal{D}, \mathcal{L})$, where $\mathcal{D} = (D_1, \ldots, D_d)$, and $D_d = D$
1: generate path $P'$ for $(D_i, S, G, C)$, where, in a top-down manner, $i = 1$ at the first call.
2: if $P'$ includes only primitive actions then
3:     return $P'$
4: end if
5: generate a list of optimizing planning problems using $P'$ and $(\mathcal{D}, \mathcal{L})$: $(D_i, j_k : s_k, s_{k+1}, C)$, where $k \in \{1, \ldots, l-1\}$, and $l$ is the length of $P'$
6: for $k \in \{1, \ldots, l-1\}$ do
7:     call Algorithm 2 to solve $(D_{i+1}, j_k : s_k, s_{k+1}, C)$, and compute $P'_k$
8: end for
9: return $P = (P'_1, \ldots, P'_{l-1})$
```

next level of state constraints that require the robot to be in lab_1. At level 3, the robot will execute $approach(d_6)$, $open(d_6)$, $gothrough(d_6)$ instead of going through d_7 because this plan meets the state constraint requirements, but is cheaper due to the robot's current position (beside d_4). This flexibility is attributed to the strategy that instead of expanding macro-actions, we generate plans for the same problem described at different abstraction levels and meet the requirement of state constraints.

We will use *PlanFG* to represent a special form of algorithm PlanHG that does not pass state constraints to lower levels but simply plans at the bottom level. PlanFG is a "flat" planning algorithm as presented in [13].

4.2 PlanHL: A Baseline Planning Algorithm

Alternatively, instead of satisfying all state constraints simultaneously, we can treat each pair of consecutive state constraints as a specification of a subproblem. In this case, the step bound estimation function $\mathcal{L}$ is used for finding local optimal plans. Following this idea, a guided Type-II problem at the ith level, (D_i, S'_i, G_i, C), can be split into a sequence of Type-II subproblems $(D_i, j_k : s_k, s_{k+1}, C)$ for $0 \leq k \leq l-1$, where S'_i is of the form: $\{j_0 : s_0, \ldots, j_l : s_l\}$, where $j_1 < j_2 < \ldots < j_l$.

The optimal paths of these problems are then joined to obtain the solution to the original Type-II problem. This algorithm is presented in Algorithm 2, where the implementation uses depth-first search to recursively call itself until reaching the bottom level. We name this algorithm *PlanHL* to identify the use of $\mathcal{H}$ and *local* minimization of plan costs using function $\mathcal{L}$ at each level. In comparison to PlanHL, algorithm PlanHG (proposed) does not decompose the original problem to subproblems at each level. Instead, it generates paths for the original problem to simultaneously satisfy all state constraints (by applying step bound estimation function $\mathcal{L}$ globally) at each level, toward minimizing the overall plan cost. Since both the algorithms are sacrificing plan quality for efficiency in solving Type-II problem, neither of the algorithms can guarantee the optimality (i.e. minimal cost) of generated plans, but the provably sound semantics of action language $\mathcal{BC}$ ensures the soundness of PlanHG (and PlanHL).

5 Experiments

The abstraction hierarchy $\mathcal{H}$ (Sect. 3) and the planning algorithms (Sect. 4) have been fully implemented in simulation and on a real robot using the mail collection problem domain. This section describes the results of experiments evaluating the efficiency, solution quality, and scalability of the proposed algorithm. Generally, a planning algorithm's quality can be measured by optimality and efficiency. Since we trade optimality for significant efficiency improvement in this work, our hypotheses are: 1) PlanHG can solve planning problems that existing "flat" algorithms cannot solve in reasonable time (PlanHG vs. PlanFG); and 2) PlanHG can generate better-quality plans than the ones generated by existing hierarchical algorithms (PlanHG vs. PlanHL).

5.1 Experiments in Simulation

The simulated domain used in experiments consists of 10 people, 20 rooms and 25 doors in an office environment, where mail needs to be collected from everyone inside the building. No two people are in the same room. We vary how mail is passed between people such that the number of people that need to be visited to collect all the mail varies from 1 to 10. Initially, the robot is placed in the corridor beside a randomly-selected door. Each data point is an average of 1000 trials. If the trials take more than 5 h, we terminate the trials and take the average over the available data. Action descriptions in $\mathcal{BC}$ are translated into logic programming, and the algorithms are implemented natively in CLINGO 4.3 [7]. Unless otherwise stated, experiments were conducted on a 32-bit laptop machine with 4 G memory and 2.0 GHz Dual Core processor.

PlanHG vs. PlanFG on Type-I Problems: We first compared PlanHG against PlanFG on the efficiency of solving Type-I problems that aim at minimizing the length of plans. The approach of applying PlanFG on Type-I problems was presented in previous research [15]. The planning time is plotted in Fig. 2a. Not surprisingly, PlanHG leads to significantly reduced planning time over PlanFG

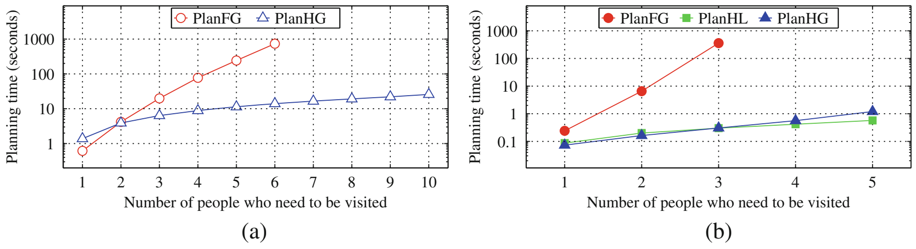


Fig. 2. (a) PlanHG vs. PlanFG in efficiency on the Type-I problems (i.e., minimizing plan length); and (b) PlanHG vs. PlanFG in efficiency on Type-II problems (i.e., minimizing plan cost).

by inserting state constraints (as "bottlenecks") at lower levels. For instance, creating a plan to visit six people (in six different rooms) takes PlanHG less than 20 seconds, but requires more than 13 min for PlanFG. Therefore, PlanHG can significantly reduce the planning time in solving Type-I planning problems. While PlanHG is not guaranteed to find the shortest length plan, both PlanHG and PlanFG find the shortest length plan in our testing domain.

PlanHG vs. PlanFG on Type-II Problems: As shown in Fig. 1, multi-entrance rooms make the shortest plan not necessarily the lowest-cost plan. We compare PlanHG against PlanFG on Type-II problems that aim at minimizing overall plan costs. Previous research has studied applying the PlanFG algorithm on Type-II problems [13], where the lowest-cost plan is found by searching among all plans of length less than a user specified upper-bound. Instead, we use Eqs. 2 and 3 of PlanHG to estimate this upper bound. The efficiency has been significantly improved, because instead of directly solving the Type-II problem, PlanHG solves a set of low-weight *guided* Type-II problems generated using the abstraction hierarchy.

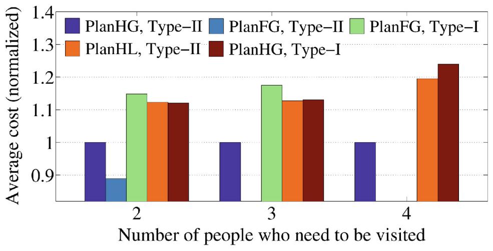
Figure 2b shows the significant improvement in efficiency against PlanFG. To run larger numbers of trials, the experiments were conducted on a powerful desktop machine with 15 G memory and Intel Core i7 CPU at 3.40 GHz. For instance, to create a plan visiting three people, PlanFG needs 5.98 min, while PlanHG requires less than one second. When a small number of people need to be visited, PlanHL took more time than PlanHG, because PlanHL calls the ASP solver more frequently. Although PlanHG (the proposed approach) becomes slower than PlanHL while planning for visiting more than three people, both require significantly less time than PlanFG. PlanHL's significant loss in plan quality will be discussed.

Scalability of PlanHG on Type-II Problems: We next evaluate the scalability of PlanHG to learn how the planning time changes given different problem domains. We keep the number of people who need to be visited fixed at three, and then vary the total number of people in the building from 5 to 15, rooms from 10 to 25, and doors from 13 to 27. Table 1 presents the planning time as the size of the domain increases.

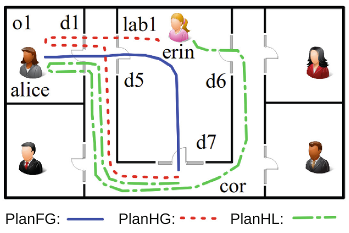
Table 1. Scalability of PlanHG on Type-II problems with three people need to be visited (in seconds).

# of ppl.	Number of rooms			
	10	15	20	25
5	1.41	2.86	5.65	10.28
10	1.83	4.18	7.69	11.56
15	-	6.20	9.93	14.58

Plan Quality: Figure 3a compares all approaches in plan quality (i.e. cost) in the domain shown in Fig. 1. In this set of experiments, the trials (totally 1000) are paired for different algorithms: the robot is initially placed in the corridor beside a randomly-selected door; and n people (n varies from 2 to 4) are randomly selected to need the robot's visit. Realistic action costs are learned and associated with the actions using algorithms presented in [13]. We observe that algorithms solving a Type-I problem do not perform as well as those solving the



(a)



(b)

Fig. 3. (a) Evaluation of plan quality in minimizing plan cost using different planning algorithms (normalized, paired)—results of trials that would require longer then five hours to complete were not included; and (b) Visual illustration of computed plans (worst case) in a test case.

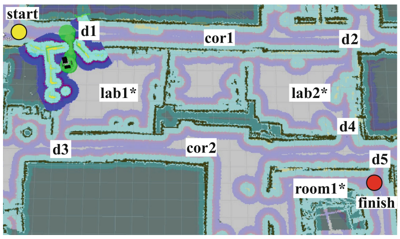
corresponding Type-II problem as they do not attempt to minimize overall plan cost, but in turn have much faster execution times.

Figure 3a shows that PlanFG, the "flat" planning approach that computes optimal plans, produces plans of the best quality in overall plan cost, but cannot solve Type-II problems with more than two people in reasonable time (as shown in Fig. 2b). Comparing with PlanFG on Type-II problem with two people, we find PlanHG has only a 11.25 % loss in optimality. Compared to PlanHL, the baseline hierarchical planning algorithm, PlanHG significantly improved the quality of generated plans—when compared over 1000 trials using a student's t-test with $p\text{-}value < 10^{-50}$.

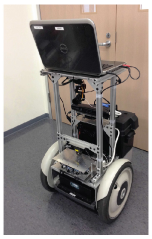
Figure 3b presents a test case of planning to visit two people, to demonstrate why PlanHG can produce lower cost plans than PlanHL, where the robot is initially beside d_7 at the corridor, and the goal is to collect mail from *Alice* and *Erin*. We present the plans in the worst case. As expected, algorithm PlanFG generated the optimal plan with the minimum cost (195). While planning with PlanHG, the robot decided to visit room o_1 first (suboptimal) because the robot's finer position (e.g., $beside(d_7)$) could not be represented at level 1—D_1 only "knows" the robot is in the corridor. While planning with PlanHL, the robot decided to go through d_6 because the subproblem is to find the optimal plan going into lab_1. As a result, PlanHG and PlanHL produce plans with costs of 205 and 345 respectively. Without minimizing plan cost globally, the robot could not know going through d_5 could reduce the overall cost.

5.2 Illustrative Trials of PlanHG on a Robot

Algorithm PlanHG has been implemented on an autonomous Segway-based robot—see Fig. 4b. The robot uses a Hokuyo URG-04LX LIDAR and a Kinect RGB-D camera for sensing and navigation. The robot moves in indoor environments at a maximum speed of 0.7m/s. Figure 4a shows part of the real world map generated using a simultaneous localization and mapping (SLAM) algorithm. Since manipulation tasks are not the focus of this paper, similar to [22],



(a)



(b)

Fig. 4. (a) Part of the inflated occupancy-grid map with a path (green bubbles) planned for going through a door; and (b) The robot platform used in experiments.

the robot simply asks help from humans to open doors. The system architecture has been implemented using Robot Operating System (ROS) [21].

As a trial, we initially placed the robot at a position labeled by the yellow dot in Fig. 4a and asked the robot to collect mail from three people in lab_1, lab_2 and $room_1$ respectively. Using PlanHG, the robot found the plan within 2 seconds. In contrast, the robot needed more than 5 min to find the plan when PlanFG was used (see Fig. 2b). The plan suggests following this path: $start \xrightarrow{d_1} lab_1{*} \xrightarrow{d_1} cor_1 \xrightarrow{d_2} lab_2{*} \xrightarrow{d_4} cor_2 \xrightarrow{d_5} room_1{*}$, where mail was collected at the rooms labeled with the star sign. The red dot in Fig. 4a shows the position where the robot finished the task. It should be noted that there are multiple plans of similar lengths leading to the goal. For instance, the robot can cross d_3 after serving the first person in lab_1. This plan is not preferred, because d_3 is a narrow door and has a high cost of navigating through it. A video of the robot's performance can be viewed online.[2]

6 Conclusions

In this paper, we present algorithm PlanHG for robotic task planning using an abstraction hierarchy represented in action language $\mathcal{BC}$. The hierarchy is obtained by composing additional domain descriptions at coarser granularities and plans computed at coarser levels are used to generate "bottlenecks" in the form of search depth bounds at lower levels. This work combines the KRR advantages of $\mathcal{BC}$ and the efficiency of hierarchical planning to enable mobile robots to compute provably sound plans in real-world environments. The hierarchy and algorithm have been fully implemented in simulation and on real robots. We observed orders of magnitude improvements in efficiency with only a 11.25 % loss in optimality compared to a "flat" planning approach.

Acknowledgments. This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at

[2] https://youtu.be/-QpFj7BbiRU.

Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-1330072, CNS-1305287), ONR (21C184-01), AFOSR (FA8750-14-1-0070, FA9550-14-1-0087), and Yujin Robot.

References

1. Chen, X., Jin, G., Yang, F.: Extending C+ with composite actions for robotic task planning. In: ICLP (Technical Communications), pp. 404–414 (2012)
2. Dix, J., Kuter, U., Nau, D.S.: Planning in answer set programming using ordered task decomposition. In: Günter, A., Kruse, R., Neumann, B. (eds.) KI 2003. LNCS (LNAI), vol. 2821, pp. 490–504. Springer, Heidelberg (2003)
3. Erol, K., Hendler, J.A., Nau, D.S.: HTN planning: complexity and expressivity. In: National Conference on Artificial Intelligence (AAAI) (1994)
4. Fikes, R.E., Nilsson, N.J.: Strips: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3), 189–208 (1972)
5. Fox, D., Burgard, W., Dellaert, F., Thrun, S.: Monte carlo localization: efficient position estimation for mobile robots. In: National Conference on Artificial Intelligence (AAAI) (1999)
6. Galindo, C., Fernandez-Madrigal, J.A., Gonzalez, J.: Improving efficiency in mobile robot task planning through world abstraction. IEEE Trans. Robot. **20**(4), 677–690 (2004)
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo= asp+ control: preliminary report. In: arXiv preprint (2014) arXiv:1405.3694
8. Cabalar, P.: Answer set; programming? In: Balduccini, M., Son, T.C. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565, pp. 334–343. Springer, Heidelberg (2011)
9. Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. Artif. Intell. (ETAI) **3**, 193–210 (1998)
10. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artif. Intell. (AIJ) **153**(1–2), 49–104 (2004)
11. Helmert, M.: The fast downward planning system. J. Artif. Intell. Res. **26**, 191–246 (2006)
12. Kaelbling, L.P., Lozano-Pérez, T.: Hierarchical task and motion planning in the now. In: IEEE International Conference on Robotics and Automation (ICRA). IEEE (2011)
13. Khandelwal, P., Yang, F., Leonetti, M., Lifschitz, V., Stone, P.: Planning in action language $\mathcal{BC}$ while learning action costs for mobile robots. In: International Conference on Automated Planning and Scheduling (ICAPS) (2014)
14. Lee, J., Lifschitz, V., Yang, F.: Action Language $\mathcal{BC}$: a preliminary report. In: International Joint Conference on Artificial Intelligence (IJCAI) (2013)
15. Lifschitz, V.: Answer set programming and plan generation. Artif. Intell. **138**, 39–54 (2002)
16. Marthi, B., Russell, S., Wolfe, J.: Angelic hierarchical planning: optimal and online algorithms (revised). Technical Report, UCB/EECS-2009-122, EECS Department, University of California, Berkeley (2009)
17. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL-the planning domain definition language (1998)
18. McIlraith, S.A., Fadel, R.: Planning with complex actions. In: International Workshop on Non-Monotonic Reasoning (NMR) (2002)

19. Nebel, B.: The FF planning system: fast plan generation through heuristic search. J. Artif. Intell. Res. **14**, 253–302 (2001)
20. Pednault, E.: ADL: exploring the middle ground between STRIPS and the situation calculus. In: International Conference on Principles of Knowledge Representation and Reasoning (KR) (1989)
21. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Open Source Software in Robotics Workshop (2009)
22. Rosenthal, S., Veloso, M.M.: Mobile robot planning to seek help with spatially-situated tasks. In: AAAI (2012)
23. Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artif. Intell. **5**(2), 115–135 (1974)
24. Tenenberg, J.D.: Abstraction in planning. Ph.D. thesis, University of Rochester (1988)
25. Thiébaux, S., Hoffmann, J., Nebel, B.: In defense of PDDL axioms. In: International Joint Conference on Artificial Intelligence (IJCAI) (2003)
26. Zhang, S., Sridharan, M., Gelfond, M., Wyatt, J.: Towards an architecture for knowledge representation and reasoning in robotics. In: Beetz, M., Johnston, B., Williams, M.-A. (eds.) ICSR 2014. LNCS, vol. 8755, pp. 400–410. Springer, Heidelberg (2014)