TLP **17** (5–6): 1027–1047, 2017. © Cambridge University Press 2017
doi:10.1017/S1471068417000321 First published online 22 August 2017

Hybrid conditional planning using answer set programming

IBRAHIM FARUK YALCINER, AHMED NOUMAN,
VOLKAN PATOGLU and ESRA ERDEM

Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey
(*e-mail:* fyalciner@sabanciuniv.edu, ahmednouman@sabanciuniv.edu,
vpatoglu@sabanciuniv.edu, esraerdem@sabanciuniv.edu)

submitted 2 May 2017; revised 20 June 2017; accepted 4 July 2017

Abstract

We introduce a parallel offline algorithm for computing hybrid conditional plans, called HCP-ASP, oriented towards robotics applications. HCP-ASP relies on modeling actuation actions and sensing actions in an expressive nonmonotonic language of answer set programming (ASP), and computation of the branches of a conditional plan in parallel using an ASP solver. In particular, thanks to external atoms, continuous feasibility checks (like collision checks) are embedded into formal representations of actuation actions and sensing actions in ASP; and thus each branch of a hybrid conditional plan describes a feasible execution of actions to reach their goals. Utilizing nonmonotonic constructs and nondeterministic choices, partial knowledge about states and nondeterministic effects of sensing actions can be explicitly formalized in ASP; and thus each branch of a conditional plan can be computed by an ASP solver without necessitating a conformant planner and an ordering of sensing actions in advance. We apply our method in a service robotics domain and report experimental evaluations. Furthermore, we present performance comparisons with other compilation based conditional planners on standardized benchmark domains.

KEYWORDS: Conditional planning, hybrid planning, answer set programming, cognitive robotics

1 Introduction

Conditional planning is concerned with reaching goals from an initial state, in the presence of incomplete knowledge and sensing actions (Warren 1976; Peot and Smith 1992; Pryor and Collins 1996). Since all contingencies are considered in advance, a conditional plan is essentially a tree of actions where the root represents the initial state, leaves represent goal states, and each branch of the tree from the root to a leaf represents a possible execution of actuation actions and sensing actions to reach a goal state. The existence of a conditional plan is an intractable problem: for polynomially bounded plans with partial observability, it is PSPACE-complete (Baral *et al.* 1999). Recall that classical planning considers complete knowledge and full observability, and it is NP-complete for polynomially bounded plans (Erol *et al.* 1995).

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

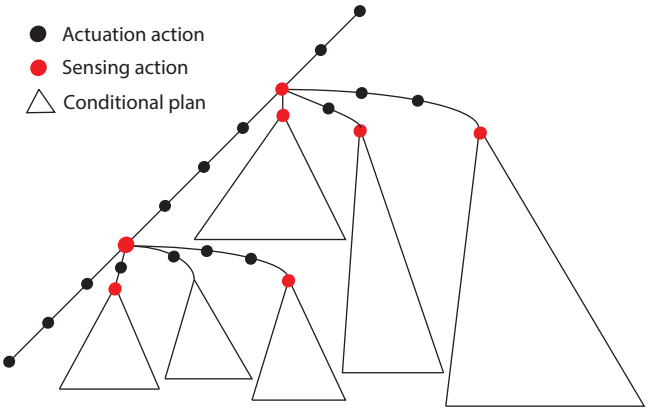


Fig. 1. A sample conditional plan.

We are concerned about computing conditional plans that are executable by robots. In real-life robotic applications, applicability of high-level planning techniques (like navigation of a robot) necessitate low-level feasibility checks (like collision checks). That a valid plan exists does not mean that the plan is executable by a robot. Therefore, some sort of hybrid planning is needed that combines planning with feasibility checks. This is a challenging problem, considering that high-level planning is done over discrete representations of the world whereas feasibility checks are performed over continuous spaces of robotic configurations.

Along these lines, several hybrid planning approaches have been proposed in the literature, where classical planning is integrated with low-level feasibility checks: by modifying the search algorithms of the planners by embedding feasibility checks to compute successor states, or by modifying the representations of the domain descriptions by embedding feasibility checks into preconditions (and other constraints) via semantic attachments.

In this paper, we consider both sorts of challenges, and introduce a novel method for hybrid robotic planning under incomplete knowledge that relies on the following contributions.

- In conditional planning, in addition to familiar *actuation* actions, *sensing* actions are also considered as part of the planning problem, and different plans are computed according to all possible outcomes of such sensing actions. As a result, a conditional plan is a tree of actions (Figure 1), where the branching occurs at vertices that characterize sensing actions, while other vertices characterize actuation actions. We introduce an offline compilation-based method to incrementally construct a conditional plan that utilizes two sorts of parallelization.

  (i) Parts of branches (i.e., sequences of actuation and sensing actions that reach a goal state from a state where a sensing action occurs) are computed in parallel using classical planners and combined into subtrees as they are computed.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

(ii) Subtrees of a conditional plan are computed in parallel according to (i) and then combined into larger subtrees. During this incremental construction, previously computed subtrees are re-used to improve computational efficiency.

- Since the existing classical planning approaches consider full observability and actuation actions only, they cannot be used in computation of branches in (i) which also involve partial observability and sensing actions. To handle this challenge, we introduce a novel method to represent partial states (defined over functional fluents) and sensing actions using answer set programming (ASP) so that hybrid sequential plans of actuation actions and sensing actions can be computed using the existing ASP solvers.
  - Cardinality expressions are used to represent "unknowns" (e.g., when the value of a functional fluent is not known) in a partially observed state.
  - Defaults are used to further describe assumptions about partially observed worlds (e.g., unless the location of an object is known, the object is assumed to be not on the table since the table is fully observable).
  - Defaults are used to describe exogenous occurrences (and nonoccurrences) of sensing actions. In this way, there is no need to decide on the order of sensing actions in advance (before conditional planning).
  - Nondeterministic outcomes of sensing actions are represented using nondeterministic choices and cardinality constraints.

  Since our method plans also for the sensing actions required to reach the goal in every branch of a conditional plan, unlike the existing compilation-based conditional planning methods, there is no need to decide on the order of sensing actions in advance (before conditional planning) or separate their computation from that of the actuation actions. Also, since our method allows for modeling sensing actions as nondeterministic actions, unlike the existing compilation-based conditional planning methods, there is no need to use conformant planners to compute parts of branches that consist of actuation actions only and that occur between two sensing actions.
- To incrementally compute hybrid conditional plans for robotic applications as described above, we adopt a representation-based approach to embed low-level feasibility checks into action descriptions so that the computed sequences of actions in (i), and thus conditional plans, are more likely to be physically implementable. In particular, we extend our representation-based approach to sensing actions as well. Note that, since the existing representation-based approaches consider deterministic actuation actions only, they cannot be adopted for the computation of branches in (i) which also involve sensing actions.

We evaluate our method experimentally over a real-world service robotics domain where a bimanual mobile manipulator sets up a kitchen table in the presence of incomplete knowledge about kitchenware. Furthermore, we present performance comparisons with other compilation based conditional planners on standardized benchmark domains.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Table 1. *A comparison of offline compilation-based conditional planners*

Name	Hybrid	Parallel	Mode	Language (variant)
CLG	No	No	Offline/online	PDDL
PO-PRP	No	No	Offline	PDDL/STRIPS
DNFct	No	No	Offline	PDDL
HCplan	Yes	Yes	Offline	C+
HCP-ASP	Yes	Yes	Offline	ASP

2 Related work

There is a variety of conditional planners, despite the hardness of computing conditional plans: even for polynomially bounded plans with limited number of sensing actions, its complexity is PSPACE-complete (Baral *et al.* 1999). Some of these planners are online, such as CLG (Albore *et al.* 2009), K-Planner (Bonet and Geffner 2011), SDR (Brafman and Shani 2012), HCP (Maliah *et al.* 2014), and CPOR (Komarnitsky and Shani 2016). Since sensing is done online, these solvers do not end up handling many contingencies and thus sometimes do not reach a goal state. Some of the conditional planners are offline, such as Contingent-FF (Hoffmann and Brafman 2005), POND (Bryce *et al.* 2006), PKS (Petrick and Bacchus 2002), ASCP (Tu *et al.* 2007), DNFct (To *et al.* 2011), PO-PRP (Muise *et al.* 2014), HCplan (Nouman *et al.* 2016). These planners construct whole conditional plans with decision points for sensing outcomes, where each execution via a branch of the plan leads to a goal state. Our planner is an offline planner.

Offline conditional planners can be further classified into two groups: search-based approaches and compilation-based approaches. The former group views conditional planning as a nondeterministic search problem in belief space (e.g., Contingent-FF, PKS, POND), whereas most of the planners in the latter group compile a conditional planning problem into many conformant planning problems by deciding for the order of sensing actions and then computing the branches of the conditional plan by means of conformant planning (e.g., CLG, DNFct, PO-PRP). Recall that conformant planning considers incomplete initial state and no observability, and its aim is to find an action sequence that reaches the goal for every possible initial state (Goldman and Boddy 1996). Our planner can be characterized as a compilation-based planner since we compile a conditional planning problem into many sequential planning problems.

Essentially, we propose a generic hybrid conditional planning algorithm, called HCP-ASP, with an offline compilation-based approach. Except for HCplan, it is different from other offline compilation-based approaches, since it is hybrid (so low-level feasibility checks are integrated into conditional planning), it models multi-valued sensing actions as nondeterministic actions (so ordering of sensing actions and solving conformant planning problems are not needed as in the related work), and it allows for concurrency of actuation actions. See Table 1 for a comparison.

HCP-ASP has similarities with HCplan: they both compute hybrid conditional plans, utilize parallel computation of branches, use nonmonotonic *default* constructs

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

to represent nonoccurrences of sensing actions, and outcomes of sensing actions are determined nondeterministically. They do not have to decide for the order of sensing actions in advance. On the other hand, the input language for HCplan is $\mathscr{C}+$ (Giunchiglia *et al.* 2004), while it is ASP (Brewka *et al.* 2016) for HCP-ASP. In HCplan, outcomes of sensing actions are nondeterministically decided by external computations, whereas in HCP-ASP they are nondeterministically decided by ASP. Moreover, HCP-ASP keeps track of visited belief states and avoids recomputing subtrees of a hybrid conditional plan.

Although not compilation-based, the offline conditional planner ASCP is also related to HCP-ASP: It is not hybrid, not parallel, and its input language is the action language $\mathscr{A}_K^c$; but it uses ASP to compute conditional plans. Based on 0-approximations (Son and Baral 2001), the idea is to approximate a belief state by a consistent set of fluent literals that are known to be true, and to define transitions considering what definitely holds and what may change at the next state. With such an approximation, the computational complexity of conditional planning becomes NP-complete. Given upper bounds for the height of the tree and the number of its branches, ASCP generates a conditional plan with one call of an ASP solver based on an intelligent formulation of possible transitions. HCP-ASP does not consider 0-approximations of belief states, but requires that the initial values of fluents that cannot be identified by any sensing action are known in advance, decided nondeterministically (e.g., by a disjunction over the possible values), or specified with respect to some assumptions (e.g., by defaults). Since HCP-ASP is compilation-based, it calls an ASP solver multiple times to build a conditional plan.

There is a variety of recent work on hybrid planners that combine motion planning with classical planners. Some of them are based on modifications/introductions of search algorithms for motion/task planning (Gravot *et al.* 2005; Hauser and Latombe 2009; Gaschler *et al.* 2013; Kaelbling and Lozano-Pérez 2013; Lagriffoul *et al.* 2014; Srivastava *et al.* 2014), some of them are based on formal methods (Plaku 2012; Dantam *et al.* 2016), and some of them are based on modification of representation of domains (Caldiran *et al.* 2009; Erdem *et al.* 2011; Hertle *et al.* 2012; Erdem *et al.* 2016). HCP-ASP is similar to the latter group, since feasibility checks are embedded in action descriptions via external atoms (in the spirit of semantic attachments in theorem proving (Weyhrauch 1978)) without having to modify the classical planners or motion planners. We refer the reader to the recent studies (Lagriffoul *et al.* 2014; Erdem *et al.* 2016) surveying and empirically analyzing some of these approaches.

3 Answer set programming

ASP (Brewka *et al.* 2016) is a form of knowledge representation and reasoning paradigm oriented towards solving combinatorial search problems as well as knowledge-intensive problems. The idea of ASP is to represent a problem as a "program" whose models (called "answer sets" (Gelfond and Lifschitz 1991)) correspond to the solutions. The answer sets for the given program can be computed

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

by special implemented systems called ASP solvers, such as CLINGO (Gebser *et al.* 2014).

We consider ASP programs (i.e., nondisjunctive HEX programs (Eiter *et al.* 2005)) that consist of rules of the form

$$Head \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$$

where $n \geqslant m \geqslant 0$, *Head* is an atom or $\bot$, and each A_i is an atom or an external atom. A rule is called a *fact* if $m = n = 0$ and a *constraint* if *Head* is $\bot$. Please see Appendix A of the supplementary material accompanying the paper at the TPLP archive, for more details.

We model robotic action domains as ASP programs as described by Erdem *et al.* (2016a). We use the ASP solver CLINGO with relevant feasibility checkers to compute hybrid plans (i.e., discrete task plans integrated with continuous feasibility checks).

There are several important aspects of ASP that are worth emphasizing here as they are closely relevant to our contributions in this paper. First, it is possible to embed the results of external computations into ASP programs using special constructs, called "external atoms," in the spirit of semantic attachments in theorem proving (Weyhrauch 1978). For instance, the ASP rule

$$\bot \leftarrow at(r, x_1, y_1, t), goto(r, x_2, y_2, t), not\ \ \&path_exists[x_1, y_1, x_2, y_2]()$$

expresses that, at any step t of the plan, a robot r cannot move from (x_1, y_1) to (x_2, y_2) if it is not known that a collision-free trajectory between them exists. Here, collision check is done by the external atom $\&path_exists[x_1, y_1, x_2, y_2]()$ implemented in C++, utilizing the bidirectional RRT algorithm (Kuffner Jr and LaValle 2000). External atoms allow us to compute hybrid plans.

Second, it is possible to express defaults in ASP thanks to "default negation" *not*. The defaults allow us to model our assumptions about a dynamic robotic world with partial observability. For instance, consider a kitchen where a service robot sets up the table for a meal. Some locations are partially observable, but the table is fully observable (i.e., the robot knows what is on the table). The following ASP rule

$$\neg at(o, Table, t) \leftarrow not\ \ at(o, Table, t)$$

expresses that by default the objects o in a kitchen are not assumed to be on the table unless they are known to be on the table.

Third, it is possible to express nondeterministic choice in ASP using "choice expressions" with "cardinality constraints." Choice expressions help us to model occurrences and nonoccurrences of actions. For instance, the following ASP rule

$$\{sense(at(o), t)\} \tag{1}$$

expresses that the action of sensing the location of an object can occur any time. Choice expressions with cardinality constraints help us to model nondeterministic effects of sensing actions. For instance, the following ASP rule

$$1\{at(o, l, t+1) : loc(l)\}1 \leftarrow sense(at(o), t) \tag{2}$$

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

describes that if sensing is applied to check the location of an object o (i.e., $sense(atObj(o), t)$), then we know that the object o is at one of the possible locations l; here, the location l is nondeterministically chosen by the ASP solver.

Fourth, it is possible to express "unknowns" using "cardinality expressions"; e.g., the rule

$$\neg at(o, m, t) \leftarrow \{at(o, l, t) : loc(l)\}0$$

expresses that if object's location is not known (i.e., $\{at(o, l, t) : loc(l)\}0$) then it definitely can not be at a robot's hand m.

Fifth, we can express "weak constraints" to minimize, e.g., the number of sensing actions:

$$\tilde{\leftarrow} senseAct(t)\ [2@2, t].$$

Finally, the incremental computation of an answer set by an ASP solver, like CLINGO, allows for minimization of makespans (i.e., lengths) of plans.

4 Hybrid conditional plans

A hybrid conditional plan can be characterized as a labeled directed tree (V, E). The set $V = V_a \cup V_s$ of vertices consists of two types of vertices. The vertices in V_a (called actuation vertices) characterize hybrid actuation actions (e.g., the robot's navigation and manipulation actions integrated with feasibility checks). The vertices in V_s (called sensing vertices) characterize sensing actions or information gathering actions in general (e.g., finding out or checking the location of an object). The leaves of the tree are in V_a.

The set E of edges between vertices in V characterizes the order of actions: an edge (x, y) expresses that the action denoted by the vertex x is to-be executed before the action denoted by y. Each vertex in V_a has at most one outgoing edge based on the assumption that the actuation actions are deterministic. Each vertex in V_s has at least two outgoing edges. Each sensing action may lead to different outcomes/observations.

Let us denote by E_s the set of outgoing edges from vertices in V_s. Then a labeling function maps every edge (x, y) in E_s by a possible outcome of the sensing action characterized by x.

5 A parallel algorithm for computing a hybrid conditional plan

HCP-ASP computes a hybrid conditional plan utilizing the expressive formalism and efficient solvers of ASP. HCP-ASP first computes a hybrid sequential plan of sensing actions and actuation actions using ASP; this sequential plan characterizes a single branch of the hybrid conditional plan being constructed. After identifying the sensing actions in this plan, for each sensing action and for each outcome of the sensing action, HCP-ASP computes a hybrid conditional plan in parallel. Finally, it combines these hybrid conditional plans as children of the sensing nodes. The overall algorithm is depicted in Algorithm 1. It is important to emphasize the novelties of our algorithm by explaining how it handles some of the challenges.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Algorithm 1 HCP-ASP($\mathscr{D}$,$\mathscr{P}$,S,nT)

Input: Robotic domain description $\mathscr{D}$, planning problem $\mathscr{P}$ with an initial belief state (i.e., the set of fluents with their known values) and goals, the set S of all possible sensing actions where each sensing action s is tupled with the set O_s of all its possible outcomes, and the maximum number nT of threads to run in parallel

Output: A hybrid conditional plan, specified by its root *root*, if one exists; otherwise, failure.

// *hashT* (initially empty) denotes a hash table: for each visited belief state e with a key $e.id$, contains a hybrid conditional plan tree

// *taskQ* (initially empty) denotes a queue of planning tasks (i.e., hybrid sequential planning problems) to be computed in parallel, where each task $\langle e, s, o_s \rangle$ is characterized by an initial belief state e, sensing action s in S to be executed at e with an outcome o_s

// create the first planning task, and push it to the task queue

$e_0 \leftarrow$ initial belief state according to $\mathscr{P}$

$taskQ.Push(\langle e_0, null, null \rangle)$

// while there are tasks in *taskQ*, solve them in parallel

while $taskQ.Size() > 0$ **do**

  do in parallel for $\langle e, s, o_s \rangle \in taskQ$ **with** nT **threads**

// extract the next planning task from *taskQ*

    $\langle e, s, o_s \rangle \leftarrow taskQ.Pull()$

    $\mathscr{P} \leftarrow$ copy and modify $\mathscr{P}$ with $\langle e, s, o_s \rangle$ so that the initial belief state is e and, vif $s \neq null$, the first action of the computed plan is the sensing action s in S with outcome o_s in O_s

    $existsP, P, H \leftarrow$ compute a hybrid sequential plan P for $\mathscr{P}$, with its history H, over $\mathscr{D}$

    if *existsP* **then**

// if $b_{\langle e,s,o_s \rangle}$ is the first branch computed

      if $e == e_0$ **then**

        $b_{\langle e,s,o_s \rangle} \leftarrow$ create a branch of the tree according to H, starting with the node $n_{\langle e,s \rangle}$ that characterizes the first action s of P, and labeling the outgoing edges of sensing nodes in the branch accordingly

        $root := b_{\langle e,s,o_s \rangle}$

      else

        $b_{\langle e,s,o_s \rangle} \leftarrow$ create a branch of the tree according to H, starting with the node that characterizes the second action of P, and labeling the outgoing edges of sensing nodes in the branch accordingly

        $root \leftarrow$ link the branch $b_{\langle e,s,o_s \rangle}$ to the existing hybrid conditional plan tree *root* as a child of the sensing node $n_{\langle e,s \rangle}$ that characterizes s and whose incoming edge describes e

        $o_e \leftarrow$ extract the observed outcome of s executed at e, from H

        $root \leftarrow$ label the edge from $n_{\langle e,s \rangle}$ to $b_{\langle e,s,o_s \rangle}$ with o_e in the tree *root*

//explore the branch $b_{\langle e,s,o_s \rangle}$ and expand it at sensing nodes considering other possible outcomes

      for $i = 1 \rightarrow |b_{\langle e,s,o_s \rangle}| - 1$ **do**

        $e_i, a_i \leftarrow$ identify from H, the belief state e_i and the action a_i executed at e_i

//check whether a subtree is already computed for e_i

        $vs := hashT[e_i.id]$

        if vs is not null **then**

          $n_{\langle e_i,a_i \rangle} \leftarrow$ add vs as a child of $n_{\langle e_i,a_i \rangle}$

// create a new planning task for other possible outcomes of a_i and push it to the task queue

        else if a_i is a sensing action in S **then**

          $n_{\langle e_i,a_i \rangle}, o_{a_i} \leftarrow$ identify the sensing node $n_{\langle e,a_i \rangle}$ in branch $b_{\langle e,s,o_s \rangle}$ executed at e_i, and the label o_{a_i} of its outgoing edge

          for $o_s \in O_s$ different from o_{a_i} **do**

            $taskQ.Push(\langle e_i, a_i, o_s \rangle)$

        $hashT[e_i.id] \leftarrow$ the hybrid conditional plan tree rooted at $n_{\langle e_i,a_i \rangle}$

return *root*

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

5.1 Computation of a single branch of the hybrid conditional plan

Considering the incomplete knowledge about the initial state, partial observability of the domain, deciding for (non)occurrences of sensing actions, nondeterministic effects of sensing actions, and necessity of feasibility checks for actuation and sensing actions for robotic applications, make it challenging to compute even a single branch of a hybrid conditional plan. We deal with these challenges by describing the robotic domain and the planning problem in ASP.

Representing actuation actions. We formalize preconditions and effects of actuation actions in robotic domains as described by Erdem *et al.* (2016a). For some examples, please see Appendix B of the supplementary material accompanying the paper at the TPLP archive.

Integrating feasibility checks. We integrate low-level feasibility checks into computation of a hybrid sequential plan, utilizing external atoms, as described in Section 3, with respect to different methods of integration, as surveyed by Erdem *et al.* (2016).

Representing belief states. We assume that values v of functional fluents $f(\bar{x})$ at step t are represented by atoms of the form $f(\bar{x},\bar{v},t)$ in ASP.

We assume that some functional fluents are fully observable (i.e., known by the robot), while some are partially observable (i.e., can be identified by sensing actions). If a functional fluent is fully observable then a uniqueness constraint

$$\leftarrow 2\{f(\bar{x},\bar{v},t) : dom_{\bar{x}}(\bar{v})\} \tag{3}$$

and an existence constraint

$$\leftarrow \{f(\bar{x},\bar{v},t) : dom_{\bar{x}}(\bar{v})\}0$$

are included in the domain description. If the fluent is partially observable then a uniqueness constraint (3) is added only; the value of the fluent is identified when the relevant sensing action is performed.

For every functional fluent $f(\bar{x})$, if there exist some value $\bar{v}$ such that $f(\bar{x},\bar{v},t)$ belongs to an answer set then we believe that $f(\bar{x})$ is known at step t. Otherwise, if there is no $\bar{v}$ such that $f(\bar{x}) = \bar{v}$ belongs to an answer set for time step t, we believe that the value of $f(\bar{x})$ is not known. In that sense, partial assignments of values to fluents represent belief states in our framework.

Representing initial values of fluents. If the initial values of fully observable fluents are known, then they are specified by a set of facts. Otherwise, they are decided nondeterministically (e.g., by a disjunction over the possible values or by a choice rule):

$$\{f(\bar{x},\bar{v},0)\} \leftarrow dom_{\bar{x}}(\bar{v})$$

or specified with respect to some assumptions (e.g., by defaults):

$$\neg f(\bar{x},\bar{v},0) \leftarrow not\ \ f(\bar{x},\bar{v},0), dom_{\bar{x}}(\bar{v}).$$

If the initial values of partially observable fluents are known, then they are specified by a set of facts.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Representing nondeterministic hybrid sensing actions. We characterize sensing actions by atoms of the form $sense(f(\bar{x}),t)$ which describes an occurrence of a sensing action that determines the value of a unique partially observable fluent $f(\bar{x})$ at time step t.

We suppose that sensing actions may occur exogenously at any time, and we represent this assumption by the following choice rule, like in rule (1):

$$\{sense(f(\bar{x}),t)\}.$$

We suppose that sensing actions $sense(f(\bar{x}),t)$ are not executed if the value of the relevant fluent $f(\bar{x})$ is known. We express this assumption by the following constraint:

$$\leftarrow sense(f(\bar{x}),t), 1\{f(\bar{x},\bar{v},t) : dom_{\bar{x}}(\bar{v})\}.$$

After that, we describe that whenever a sensing action $sense(f(\bar{x}),t)$ is executed, the observed outcome $\bar{v}$ is nondeterministically chosen among possible values of $f(\bar{x})$. We express this nondeterministic choice of an outcome using choice expressions and cardinality constraints in ASP, like in rule (2):

$$1\{f(\bar{x},\bar{v},t+1) : dom_{\bar{x}}(\bar{v})\}1 \leftarrow sense(f(\bar{x}),t).$$

Sometimes, executability of a sensing action necessitates further conditions to hold. For instance, to check whether a plate is clean or not, the robot has to be holding the plate. The preconditions of a sensing action may involve feasibility checks as well. For instance, checking what is on the table is possible if the camera is "reachable" to a position over the table. This can be expressed by a constraint as follows:

$$\leftarrow sense(at(o),t), not\ \ \&reachable[Table]().$$

Incremental computation of a hybrid sequential plan of sensing actions and actuation actions. Given a domain description and a planning problem as described above, the ASP solver CLINGO can compute a plan to reach a goal state from a possibly incomplete initial state. CLINGO first grounds this whole program and then computes an answer set for the ground program; the answer set characterizes the plan.

We improve this computation (grounding and answer set finding), by ensuring that CLINGO incrementally performs this computation by increasing the upper bound on the maximum time step one by one, but without having to ground the whole theory every time. This is possible thanks to the "incremental mode" of CLINGO.

5.2 Parallel computation of subtrees

Our algorithm maintains a queue *taskQ* of tasks $\langle e,s,o_s\rangle$ where e is an initial belief state e, s is a sensing action, o_s is an outcome of s. Initially, *taskQ* contains the task $\langle e_0,null,null\rangle$ only (Algorithm 1, Line 2). Each task $\langle e,s,o_s\rangle$ in *taskQ* describes a hybrid sequential planning problem with the initial state e and the goal as same as in the hybrid conditional planning problem $\mathscr{P}$ (Lines 5&6). This planning problem has

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

further constraints, if s and o_s are not *null*: s is a sensing action executed at belief state e and the outcome of this execution is o_s (Line 7). Note that these constraints can be easily expressed in ASP.

For every task $\langle e, s, o_s \rangle$, an ASP solver computes a hybrid sequential plan $P = \langle a_0, a_1, \ldots, a_k \rangle$ and its history $H = \langle b_0, a_0, b_1, a_1, \ldots, b_k, a_k, b_{k+1} \rangle$, where $s{=}a_0$, $e = b_0$, each a_i is an actuation action or a sensing action executed at a belief state b_i and reaching a belief state b_{i+1} $(0 \leqslant i < k)$, the last action a_k is an actuation action, and the last belief state b_{k+1} is a goal state according to $\mathscr{P}$. Such a plan P describes a branch $b_{\langle e,s,o_s \rangle} = \langle n_{\langle b_0,a_0 \rangle}, n_{\langle b_1,a_1 \rangle}, \ldots, n_{\langle b_k,a_k \rangle} \rangle$ of the tree that starts at the node $n_{\langle e,s \rangle} = n_{\langle b_0,a_0 \rangle}$ and ends at the leaf $n_{\langle a_k,b_k \rangle}$. For every sensing action a_i in P, the edge from $n_{\langle b_i,a_i \rangle}$ to $n_{\langle b_{i+1},a_{i+1} \rangle}$ is labeled by its outcome o_{a_i} observed in b_{i+1}. Once we compute all the branches that start at a sensing node, we can combine them at that node (Lines 9–16).

The tasks are added to the queue *taskQ* at different times: for every other outcome of every sensing node in a branch $b_{\langle e,s,o_s \rangle}$, a task is constructed (Lines 17–26). Since the tasks in *taskQ* are not dependent on each other, they are solved in parallel utilizing at most nT threads (Line 4).

Discussion about correctness. Every branch of the tree from the root to a leaf constructed incrementally as described above characterizes a hybrid sequential plan, under the assumptions that a plan can be constructed by merging two plans: $\mathscr{D}$ respects the Markov principle (so actions do not have delayed effects), and $\mathscr{D}$ does not contain any temporal constraints that ensure/prevent occurrences of actions or observations of fluents in some order, or any constraints to guarantee some conditions over the whole hybrid sequential plan (e.g., a constraint on the total cost of actions). Indeed, let $P = \langle a_0, a_1, \ldots, a_k \rangle$ be a hybrid sequential plan computed for a planning problem $\mathscr{P}$ with respect to a domain description $\mathscr{D}$, where each a_i is an actuation action or a sensing action $(0 \leqslant i < k)$ and the last action a_k is an actuation action. Let $H = \langle b_0, a_0, b_1, a_1, \ldots, b_k, a_k, b_{k+1} \rangle$ be a history of this plan, where every action a_i is executed at a belief state b_i and reaching a belief state b_{i+1} $(0 \leqslant i < k)$, and the last belief state b_{k+1} is a goal state according to $\mathscr{P}$. For every sensing action a_j in P $(0 < j < k)$ with an outcome o_{a_j} observed at b_{j+1}, a task $\langle b_j, a_j, o_{a_j} \rangle$ is constructed and pushed into *taskQ*. Consider one of these tasks: $\langle b_j, a_j, o_{a_j} \rangle$. Let $P' = \langle a_j = a'_0, a'_1, \ldots, a'_{k'} \rangle$ be a hybrid sequential plan, with its history $H' = \langle b_j = b'_0, a'_0, b'_1, a'_1, \ldots, b'_{k'}, a'_{k'}, b'_{k'+1} \rangle$, computed for the planning problem characterized by this task. Then, due to the assumptions mentioned above, the sequence $\langle a_0, a_1, \ldots, a_j, a'_1, \ldots, a'_{k'} \rangle$ of actions is also a hybrid sequential plan computed for a planning problem $\mathscr{P}$, with a history $\langle b_0, a_0, b_1, a_1, \ldots, b_j, a_j, b'_1, a'_1, \ldots, b'_{k'}, a'_{k'}, b'_{k'+1} \rangle$: the first part of the plan, $\langle a_0, a_1, \ldots, a_j \rangle$, does not prevent the last part of the plan, $\langle a'_1, \ldots, a'_{k'} \rangle$, and vice versa. Note that these assumptions can be waived if we modify our algorithm slightly by computing every branch of the tree from the root, instead of constructing branches by combining sub-branches: every task $\langle b_j, a_j, o_{a_j} \rangle$ is understood as a planning problem obtained from $\mathscr{P}$ by adding constraints to ensure that $\langle b_0, a_0, b_1, a_1, \ldots, b_j, a_j \rangle$ is a prefix of the history of the plan and that o_{a_j} is observed right after execution of a_j. On the other hand, note that the hybrid sequential plans computed from the root have

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

larger makespans (and thus they lead to computations over larger ASP programs) compared to the plans computed from intermediate states.

5.3 Avoiding recomputation of subtrees

Our algorithm keeps a hash table *hashT* that maps a belief state to a hybrid conditional plan computed from that state (Lines 18–21,26). In this way, identical parts of a hybrid conditional plan are not computed repeatedly. This then allows the representation of a hybrid conditional plan more compactly as a directed acyclic graph (DAG).

For some domains, when the desired goal relies on a subset of fluents only, we define "equivalence classes" of belief states according to these fluents. This allows us to utilize the hash table with respect to these equivalence classes. If belief states e and e' belong to the same equivalence class, and if a hybrid conditional plan is computed for e, then we utilize this tree also for e' despite their differences.

For instance, consider the *colorball* planning domain (Albore *et al.* 2009) used as a benchmark for conditional planners. In this domain, the goal is to dispose a given set of balls with unknown locations and colors into thrash boxes according to their colors. As long as a ball is disposed into the trash, its color or location does not affect the rest of the plan anymore. So we can safely say that two different belief states are equivalent if they only differ in color and/or location of any ball that is trashed. Fortunately, we can specify this information in ASP (in a separate file) by identifying what parts of belief states could be ignored as follows:

$$redundant(color(b,c,t)) \leftarrow trashed(b,t)$$
$$redundant(location(b,l,t)) \leftarrow trashed(b,t).$$

When a ball b is thrashed, its color c and location l are irrelevant to belief state comparisons. Here *redundant* is declared specially for our planner to identify partial belief states for comparisons.

5.4 Advantages of hybridity of conditional plans

Since each branch of a conditional plan depicts a possible execution of actuation/sensing actions to reach a goal, it is essential that these actions are checked against relevant feasibility constraints in real-world applications. In robotics applications, these constraints are required for collision-free navigation and reachable/graspable manipulation, as depicted in Figure 2 with two conditional plans computed for a robotics scenario, where two bimanual mobile manipulators are responsible for setting up a kitchen table: none of the branches of the nonhybrid plan (Figure 2(a)) is executable in real world, since the actuation actions (denoted red) are not feasible; whereas every branch of the hybrid plan (Figure 2(b)) is feasible in real world. See Appendix C of the supplementary material accompanying the paper at the TPLP archive, for more details about the domain.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

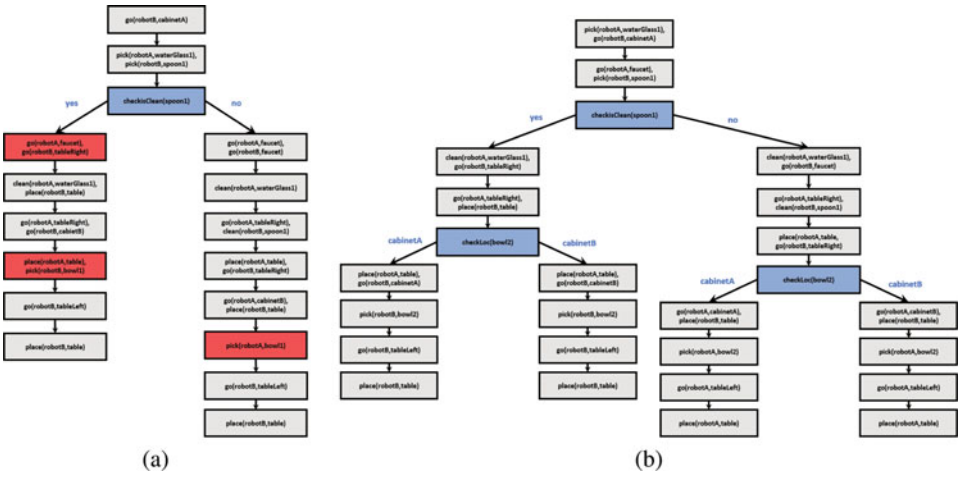


Fig. 2. Two hybrid conditional plans generated (a) without and (b) with feasibility checks.

6 Experimental evaluations

All experiments are conducted on a PC workstation running Ubuntu 16.04 on 16 2.4 GHz Intel E5-2665 CPU cores (32 threads) with 64 GB memory. For parallel computations, the maximum number of available threads is limited to 20; walltimes for the threads with the longest computations are reported. Computations that exceed 5 h are marked as TO.

Comparisons with the hybrid compilation-based conditional planner HCPlan. We have compared HCP-ASP (with CLINGO 4.5.4) with HCPlan (with CCALC using MINISAT 2.2.0) over the robotics kitchen *table setting* domain in the paper by Nouman *et al.* (2016). This domain models a dynamic service robotics scenario, where a bimanual mobile manipulator is responsible for setting up a kitchen table; it can navigate around the kitchen and manipulate kitchenware. The robot can perform sensing actions to identify the locations/cleanliness of objects, and the type of food requested by the user. The goal is for the robot to setup the table for lunch, with proper kitchenware for the requested food type. This domain includes several low-level feasibility checks, such as kinematic reachability, bimanual motion planning for robot arms, and collision-free navigation planning. For our experiments, we have constructed 12 planning problem instances in this domain, with solutions (i.e., conditional plans) whose heights range over 16–33 and branching factor ranges over 2–4. As suggested by the PRE integration method of Erdem *et al.* (2016), the feasibility checks are performed in advance as in the paper by Nouman *et al.* (2016) and cached in a hash table, for both planners.

Table 2 summarizes the results of our experiments. Since HCP-ASP generates a more compact representation of a tree as a DAG due to re-use of subtrees whereas HCplan generates a tree, we report the tree sizes for plans. For each instance and the computed hybrid conditional plan, we present the maximum branch length (i.e., makespan of the longest hybrid sequential plan), the number of sensing nodes in

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Table 2. *Comparison of HCP-ASP with HCPlan on the table setting domain*

	Max branch length		# of sensing nodes		# of Leaves		Tree size		Time	
Scen.	HCplan	HCP-ASP	HCplan	HCP-ASP	HCplan	HCP-ASP	HCplan	HCP-ASP	HCplan	HCP-ASP
1	17	17	4	3	12	11	147	114	19.66	18.85
2	21	21	11	9	19	17	168	186	25.91	47.70
3	21	20	15	8	35	25	306	243	31.38	26.79
4	23	23	20	24	31	32	294	283	35.24	106.77
5	23	22	31	24	45	32	398	329	41.66	180.80
6	21	25	31	26	72	40	566	377	33.8	53.82
7	26	26	49	54	102	68	771	615	45.73	164.48
8	26	26	53	54	91	68	676	615	38.96	164.11
9	29	24	103	58	222	96	1,535	698	53.88	239.26
10	29	31	231	224	347	280	2,296	1,844	67.99	202.25
11	29	29	227	156	349	218	2,524	1,591	76.28	280.52
12	33	31	1,483	576	2,178	773	16,119	5,164	390.29	973.28

the tree (i.e., total number of sensing actions in the plan), the number of leaves of the tree (i.e., number of different hybrid sequential plans represented in the tree), the number of nodes in the tree (i.e., total number of actuation and sensing actions in the plan), and the computation time in seconds. The instances are arranged in ascending order of tree sizes.

HCplan uses CCalc with incremental grounding. Also, HCP-ASP uses Clingo with incremental grounding. Therefore, both planners try to minimize the makespans of hybrid sequential plans. Although the planners start with a shortest hybrid sequential plan of the same makespan, due to the occurrences of different actions in the plans in different orders, the trees may have different heights in the end. This can be observed from the maximum branch lengths.

Using weak constraints, HCP-ASP further tries to minimize the number of sensing actions, once it minimizes actuation actions. Due to this, the trees computed by HCP-ASP have less number of sensing actions, and often with less number of leaves and nodes. Depending on how the tree changes, the computation time may increase due to such further optimizations.

Due to parallel computations of hybrid sequential plans and subtrees, the computation times reflect in general the computation time of the longest hybrid sequential plan. Indeed, for the longest branches of the trees of Instance 12, Clingo takes much longer time than CCalc.

Comparisons with nonhybrid compilation-based conditional planners We have also compared HCP-ASP with nonhybrid compilation-based planners CLG 2.0, DNFct 1.0 and PO-PRP 0.3, over two standard benchmarks (Albore *et al.* 2007; Albore *et al.* 2009). In the *colorball* domain, the goal is to dispose a given set of x balls with unknown locations and colors into thrash boxes according to their colors; the environment is represented as a $n \times n$ grid. The location and color of a ball can be observed when the agent and the ball are in the same cell. In the *doors* domain, the goal is to reach a goal location by moving in a $n \times n$ grid with hidden doors; open doors can be detected by sensing actions. We have formalized these domains in ASP, by allowing multi-valued sensing outcomes and concurrency of actuation actions. Since CLG generates a tree, while DNFct and PO-PRP generate DAGs, both DAG sizes and tree sizes are reported for HCP-ASP.

Table 3 presents the results of these experiments. CB $x - n$ (resp. Doors n) denotes instances of the *colorball* (resp. *doors*) domain. For each instance, the size of DAG and/or tree (the number of nodes), the maximum branch length, and total computation time in seconds are reported in the table. MO indicates the planner did not have enough memory to complete the problem. Note that the available version of PO-PRP has a 5GB memory limit.

The results indicate that, for most of the scenarios, HCP-ASP generates shorter trees and smaller DAGs. HCP-ASP scales in time better than CLG and PO-PRP, with a performance very close to that of DNFct. While PO-PRP shows remarkable time performance in available instances, the time and memory scaling of this planner seem poorer than the others. HCP-ASP outperforms other planners in generated DAG size, which has high correlation with time and memory efficiency. The DAGs

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Table 3. *Comparison of offline compilation-based conditional planners*

	CLG			DNFct			PO-PRP			HCP-ASP			
	Tree Size	Max Depth	Time [sec]	DAG Size	Max Depth	Time [sec]	DAG Size	Max Depth	Time [sec]	Tree Size	DAG Size	Max Depth	Time [sec]
CB 3-1	95	21	0.04	35	19	0.03	89	23	0.02	81	51	17	0.7
CB 3-2	2,641	34	0.63	609	40	0.47	1,655	48	0.16	1,800	415	28	2
CB 3-3	60,924	48	17.86	8,021	57	8.45	33,247	73	3.96	38,977	3,395	41	29.8
CB 3-4	1,329,235	61	542	93,379	72	268	MO	MO	MO	805,768	32,593	48	928
CB 4-1	295	52	0.2	67	36	0.04	271	46	0.06	242	100	33	1.4
CB 4-2	20,050	74	11.75	1,928	70	2.85	12,432	90	1.46	14,067	1,362	58	9.9
CB 4-3	1,136,920	100	966	39,678	100	227	MO	MO	MO	708,580	27,942	84	681
CB 5-1	586	65	0.59	117	68	0.95	556	78	0.1	524	172	63	4
CB 5-2	72,817	107	100	5,153	113	18	49,767	169	15.22	53,576	3,479	98	38.5
CB 5-3	TO	TO	TO	134,008	157	2,338	MO	MO	MO	4,718,549	13,879	149	13,173
CB 9-1	3385	197	44.45	351	184	47.28	2607	279	1.1	3210	621	211	77.5
CB 9-2	1,700,499	312	4,918	43,554	360	1,564	MO	MO	MO	1,238,013	42,213	378	6,779
Doors 5	144	24	0.18	146	26	0.05	105	28	0.06	156	68	27	1.2
Doors 7	2,153	51	3.38	2,193	53	0.79	1,282	60	0.34	2,381	179	59	6.5
Doors 9	46,024	95	188	44,998	89	29.76	23,897	104	7.92	52,065	381	109	47.4
Doors 11	1,213,759	135	19,228	1,161,651	124	1,239	MO	MO	MO	1,242,179	776	167	265

computed by HCP-ASP are much smaller than the trees they represent. This shows the usefulness of identifying equivalence classes to detect similar states.

Comparisons with ASCP As discussed in Section 2, although not compilation-based, the offline nonhybrid conditional planner ASCP also uses ASP to compute conditional plans. Therefore, we have compared these two ASP-based conditional planners, over one of the benchmarks of ASCP: Bomb in the Toilet with Sensing Actions (BTS) (Weld *et al.* 1998). In our experiments with ASCP, we have directly used the ASP encoding of BTS transformed from $\mathscr{A}_K^c$ with the ASP solver CLINGO (Version 4.5.4). According to the results (presented in Appendix D of the supplementary material accompanying the paper at the TPLP archive), finding a tree with one call of CLINGO (using ASCP) takes more time, compared to computing and combining the branches of the tree in parallel (using HCP-ASP). For instance, for $m = 17$, it takes more than an hour to compute a tree with ASCP whereas it takes about a second for HCP-ASP.

Evaluations of different versions of HCP-ASP Using the *colorball* and *doors* domains, we also experimentally evaluate the following three versions of HCP-ASP: v_0 (base planner with parallelism, revisited states, and incremental planning), v_1 (v_0 improved with equivalence classes), v_2 (v_1 improved with more efficient representation of the domain to capture multi-valued sensing outcomes and concurrency of actuation actions). The version v_2 is used in Tables 2 and 3.

Table 4 presents the experimental results. For each instance, the size of DAG, the maximum branch length, parallel efficiency (i.e., ratio of sequential computation time to (20 $\times$ parallel computation time)), and the total computation time in seconds are reported in the table.

The *colorball* domain demonstrates the effects of improvements in different versions of HCP-ASP, while the *doors* domain is useful to evaluate the potential of partial state comparisons, where a problem can be divided into sub-problems. Since the *doors* domain is very simple, it does not allow for v_2, as there is no sensing outcome that can converted to multi-valued format and there is only a single actuation action not allowing for concurrency.

Comparison of computation times of v_0 and efficiency values indicate that parallelism reaches its full potential with up to 99.9% efficiency as the problem size gets larger. This is expected as the number of tasks that can be computed at any time depends on the number of sensing actions and the outcomes at each branch. From the results of v_1, one can observe that utilization of equivalence classes results in highly reduced DAG size and number of branches explored, significantly improving the computation time.

From the results of v_2, one can observe that the use of multi-valued functional fluents and thus multi-valued sensing outcomes (e.g., identifying the color of a ball) decrease the length of branches compared to the case with boolean fluents and thus sensing outcomes (e.g., identifying that a ball has a specified color), since outcome of a single multi-valued sensing action can be obtained by an exponential number of boolean sensing actions. Similarly, the use of concurrent actions reduce the length of plans, since multiple actions can be executed at the same time.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Table 4. *Comparison of different versions of HCP-ASP*

	v_0				v_1				v_2			
	DAG Size	Max Depth	Eff. %	Time sec	DAG Size	Max Depth	Eff. %	Time sec	DAG Size	Max Depth	Eff. %	Time sec
CB 3-1	63	20	10	0.8	63	20	10	1.1	51	17	11	0.7
CB 3-2	728	39	60	3.3	635	39	58	3.6	415	28	78	2
CB 3-3	6,960	52	40	93	4,780	52	34	90	3,395	41	97	29.8
CB 3-4	113,466	67	89	6,738	55,904	69	26	5,835	32,593	48	100	928
CB 4-1	117	37	11	1.6	121	40	12	1.8	100	33	12	1.4
CB 4-2	1,934	66	64	12.7	1,503	69	57	12.8	1,362	58	93	9.9
CB 4-3	34,074	101	98	788	24,500	102	84	635	27,942	84	99.8	681
CB 5-1	181	68	8	4.6	181	68	8	5.5	172	63	10	4
CB 5-2	4,939	121	65	65.5	3,872	118	49	76.6	3,479	98	96	38.5
CB 5-3	123,623	166	99	10,023	109,157	162	98	8,168	138,793	149	100	13,173
CB 9-1	634	220	8	102	634	220	8	106	621	211	10	77.5
CB 9-2	45,584	397	96	8,916	43,235	397	96	9,504	42,213	378	98	6,779
Doors 5	158	27	13	1	68	27	10	1.2				
Doors 7	2,410	65	54	7.6	179	59	13	6.5				
Doors 9	49,381	109	99	523	381	109	19	47.4				
Doors 11	TO	TO	TO	TO	776	167	30	265				

7 Conclusion

We have introduced an offline parallel compilation-based hybrid conditional planner, called HCP-ASP, utilizing various features and utilities of ASP solver CLINGO. From the representation point of view, compared to many existing offline compilation-based conditional planners, HCP-ASP allows multi-valued functional fluents, multi-valued sensing actions, explicit representation of occurrences, preconditions and nondeterministic effects of sensing actions, weak constraints to minimize sensing actions or actuation actions, concurrency of actuation actions, and integration of feasibility checks into action descriptions. From the computation point of view, HCP-ASP re-uses earlier computations due to partial state comparisons via equivalence classes specified in ASP, generates trees in a more compact form of DAGs, performs incremental computations of branches (i.e., hybrid sequential plans), and utilizes parallel computation of branches in a subtree as well as parallel computation of subtrees. Due to these novelties, HCP-ASP not only provides a new conditional planner with a more expressive representation language and promising computational performance, but also provides a useful tool for hybrid robotic applications with incomplete knowledge and partial observability. Our planner HCP-ASP and the benchmarks are available at `http://cogrobo.sabanciuniv.edu/?p=1073`.

Acknowledgements

This work is partially supported by the Science Academy (BAGEP). We thank T. C. Son and H. Geffner for useful discussions about HCP-ASP, and A. Albore, T. C. Son, and C. Muise for their helps with CLG, DNFct, and PO-PRP, respectively.

Supplementary materials

For supplementary material for this article, please visit `https://doi.org/10.1017/S1471068417000321`

References

ALBORE, A., PALACIOS, H. AND GEFFNER, H. 2007. Fast and informed action selection for planning with sensing. In *Proc. of CAEPIA*, 1–10.

ALBORE, A., PALACIOS, H. AND GEFFNER, H. 2009. A translation-based approach to contingent planning. In *Proc. of IJCAI*, 1623–1628.

BARAL, C., KREINOVICH, V. AND TREJO, R. 1999. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proc. of IJCAI*, 948–955.

BONET, B. AND GEFFNER, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. of IJCAI*, 1936–1941.

BRAFMAN, R. I. AND SHANI, G. 2012. Replanning in domains with partial information and sensing actions. *JAIR 45*, 565–600.

BREWKA, G., EITER, T. AND TRUSZCZYNSKI, M. 2016. Answer set programming: An introduction to the special issue. *AI Magazine 37,* 3, 5–6.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Bryce, D., Kambhampati, S. and Smith, D. E. 2006. Planning graph heuristics for belief space search. *JAIR 26*, 35–99.

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E. and Patoglu, V. 2009. Bridging the gap between high-level reasoning and low-level control. In *Proc. of LPNMR*.

Dantam, N. T., Kingston, Z. K., Chaudhuri, S. and Kavraki, L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Proc. of RSS*.

Eiter, T., Ianni, G., Schindlauer, R. and Tompits, H. 2005. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proc. of IJCAI*, 90–96.

Erdem, E., Gelfond, M. and Leone, N. 2016a. Applications of answer set programming. *AI Magazine 37,* 3, 53–68.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V. and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*.

Erdem, E., Patoglu, V. and Schüller, P. 2016. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Communications 29,* 2, 319–349.

Erol, K., Nau, D. S. and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence 76,* 1–2, 75-88.

Gaschler, A., Petrick, R. P., Giuliani, M., Rickert, M. and Knoll, A. 2013. KVP: A knowledge of volumes approach to robot task planning. In *Proc. of IROS*.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2014. *Clingo* = ASP + control: Preliminary report. In *Technical Communications of ICLP*, vol. 14(4–5). TPLP, Online supplement.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N. and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153*, 49–104.

Goldman, R. P. and Boddy, M. S. 1996. Expressive planning and explicit knowledge. In *Proc. of AIPS*, 110–117.

Gravot, F., Cambon, S. and Alami, R. 2005. aSyMov:A Planner that deals with intricate symbolic and geometric problems. In *Proc. of ISRR*, 100–110.

Hauser, K. and Latombe, J.-C. 2009. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *BTAMP at ICAPS*.

Hertle, A., Dornhege, C., Keller, T. and Nebel, B. 2012. Planning with semantic attachments: An object-oriented view. In *Proc. of ECAI*, 402–407.

Hoffmann, J. and Brafman, R. I. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. of ICAPS*, 71–80.

Kaelbling, L. P. and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *IJRR 32,* 9–10, 1194–1227.

Komarnitsky, R. and Shani, G. 2016. Computing contingent plans using online replanning. In *Proc. of AAAI*, 3159–3165.

Kuffner Jr, J. and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proc. of ICRA*, 995–1001.

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A. and Karlsson, L. 2014. Efficiently combining task and motion planning using geometric constraints. *IJRR 33,* 14, 1726–1747.

Maliah, S., Brafman, R. I., Karpas, E. and Shani, G. 2014. Partially observable online contingent planning using landmark heuristics. In *Proc. of ICAPS*.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press

Muise, C. J., Belle, V. and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proc. of AAAI*, 2322–2329.

Nouman, A., Yalciner, I. F., Erdem, E. and Patoglu, V. 2016. Experimental evaluation of hybrid conditional planning for service robotics. In *Proc. of ISER*.

Peot, M. A. and Smith, D. E. 1992. Conditional nonlinear planning. In *Proc. of AIPS*, 189–197.

Petrick, R. P. A. and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS*, 212–222.

Plaku, E. 2012. Planning in discrete and continuous spaces: From LTL tasks to robot motions. In *Proc. of TAROS*, 331–342.

Pryor, L. and Collins, G. 1996. Planning for contingencies: A decision-based approach. *JAIR 4*, 287–339.

Son, T. C. and Baral, C. 2001. Formalizing sensing actions: A transition function based approach. *Artificial Intelligence 125,* 1–2, 19–91.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S. and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of ICRA*.

To, S. T., Son, T. C. and Pontelli, E. 2011. Contingent planning as AND/OR forward search with disjunctive representation. In *Proc. of ICAPS*.

Tu, P. H., Son, T. C. and Baral, C. 2007. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *TPLP 7,* 4, 377–450.

Warren, D. H. D. 1976. Generating conditional plans and programs. In *Proc. of AISB*, 344–354.

Weld, D. S., Anderson, C. R. and Smith, D. E. 1998. Extending graphplan to handle uncertainty & sensing actions. In *Proc. of AAAI*, 897–904.

Weyhrauch, R. W. 1978. *Prolegomena to a Theory of Formal Reasoning*. Technical report, Stanford University.

https://doi.org/10.1017/S1471068417000321 Published online by Cambridge University Press