Practical Resolution Methods for MDPs in Robotics Exemplified with Disassembly Planning

Alejandro Suárez-Hernández[1], Carme Torras[1] and Guillem Alenyà[1]

***Abstract*—In this paper we focus on finding practical resolution methods for Markov Decision Processes (MDPs) in robotics. Some of the main difficulties of applying MDPs to real-world robotics problems are: (1) having to deal with huge state spaces; and (2) designing a method that is robust enough to dead ends. These complications restrict or make more difficult the application of methods such as Value Iteration, Policy Iteration or Labeled Real Time Dynamic Programming (LRTDP). We see in determinization and heuristic search a way to successfully work around these problems. In addition, we believe that many practical use cases offer the opportunity to identify hierarchies of subtasks and solve smaller, simplified problems. We propose a decision-making unit that operates in a probabilistic planning setting through Stochastic Shortest Path Problems (SSPPs), which generalize the most common types of MDPs. Our decision-making unit combines: (1) automatic hierarchical organization of subtasks; and (2) on-line resolution via determinization. We argue that several applications of planning benefit from these two strategies. We exemplify our approach with a robotized disassembly application. The disassembly problem is modeled in Probabilistic Planning Definition Language (PPDDL), and serves to define our experiments. Our results show many advantages of our method over LRTDP, like a better capability to handle problems with large state spaces and state definitions that change when new fluents are discovered.**

***Index Terms*—Planning, Scheduling and Coordination; Hybrid Logical/Dynamical Planning and Verification; Task Planning**

I. Introduction

NEW versatile industrial robot need to operate in less and less structured environments. We would like to enable robots to tackle problems in a more flexible manner, using declarative models with little to no control knowledge.

In this paper we are concerned with planning in environments that exhibit uncertainty in the form of probabilistic effects. The dynamics of these environments can be captured thanks to the *Markov Decision Process* (MDP) formalism, which is of particular interest in robotics. However, many techniques for solving MDPs, like *Value Iteration* and *Policy Iteration*, struggle against the large size of the state spaces present in practical applications. More sophisticated techniques, like *Labeled Real Time Dynamic Programming* (LRTDP), address the large number of states up to a certain extend, but can be compromised in the presence of *dead ends*. We argue that: (1) on-line resolution of MDPs via determinization has still many practical advantages nowadays, and can be used to effectively circumvent these issues through heuristic search; and (2) many real-life problems offer the opportunity to bound the complexity of the planning task by identifying smaller subtasks and reducing the size of the state.

We will resort to a specific application to exemplify these points: the automatic disassembly of electromechanical devices. We pose the problem as follows: a robotic arm has to retrieve the most valuable and/or hazardous components from a device presented in front of it. MDPs offer a principled way of handling outcome uncertainty and are a suitable and (until now) overlooked framework for this application. We are targeting a large range of devices, so we consider generic actions and a not completely controlled environment. Actions may have undesirable outcomes that deviate from their intended effect. Under these requirements, it is not practical to develop a fixed disassembly script, and modeling and solving the problem as an MDP becomes more appealing. This domain offers a clear hierarchy of subtasks: components have precedence relations among them. In addition, the application has beneficial implications for sustainability. We believe that these reasons make the problem well-suited to be solved through probabilistic planning methods and interesting from an environmental and social perspective. We have developed a decision-making unit that combines determinization with subtask selection thinking of this particular domain.

The rest of this paper is structured as follows: Section II reviews previous literature related to determinization, task hierarchization and planning disassembly sequences; Section III describes our decision making unit, including the hierarchization and the determinization strategies; Section IV discusses the experimentation process and provides some quantitative and qualitative results; and Section V wraps up and gives some future work ideas.

II. Related Work

Kaelbling and Lozano-Pérez's contribution [1] combines Hierarchical Planning in the Now with both world and state uncertainty. Similarly to us, they pose an *Stochastic Shortest Path Problem* (SSPP) that is later determinized and solved classically. From them we have borrowed the α-*Cost-Transition-Likelihood* (ACTL) determinization.

Manuscript received: September 19, 2018; Revised November 19, 2018; Accepted February 6, 2019.

This paper was recommended for publication by Editor Jingshan Li upon evaluation of the Associate Editor and Reviewers' comments. This work is partially funded by the European project IMAGINE (Robots Understanding their Actions by Imagining their Effects), H2020-ICT-2016-1-731761, by the Spanish Ministry of Science and Innovation HuMoUR TIN2017-90086-R and the Spanish State Research Agency through the Maria de Maeztu Seal of Excellence to IRI (MDM-2016-0656).

[1] Authors are with Institut de Robòtica i Informàtica Industrial (CSIC-UPC). Llorens i Artigas 4-6, 08028 Barcelona, Spain. E-mails: {asuarez,torras,galenya}@iri.upc.edu

Digital Object Identifier (DOI): see top of this page.

Perhaps one of the most well-known successes in determinization techniques is *FF-Replan* [2], unofficial winner of the *International Probabilistic Planning Competition* (IPPC) in 2004 and 2006, and is often used as a baseline for new and improved techniques. Later, *FF-Hindsight* [3], [4] and *Robust-FF* [5] appeared, and were able to cope well with Littman *et al.*'s *probabilistic interesting problems* [6], specifically designed to beat replanners. We can see that stochastic domain determinization receives a lot of attention nowadays. For instance: hindsight optimization is an active research topic [7], [8], and the work of Kolobov *et al.* [9] is heavily influenced by determinization. This suggests that determinization-based probabilistic planning is still worth exploring nowadays.

LRTDP and UCT are two of the most successful algorithms employed to deal with MDPs in an on-line fashion, and are implemented respectively by G-Pack [10], [11] and PROST [12], the top performing planners of the latest IPPCs. As a baseline for this paper, we will use an earlier implementation of LRTDP by Bonet and Geffner [13].

In our previous work [14] we have also studied the benefits of hierarchical organization of subtasks. Doing this we seek to bound the complexity of planning and avoid expensive replanning in the event of undesired outcomes.

A relatively recent advance in automatic disassembly is *Liam* [15], a robot by *Apple Inc.* aimed at disassembling *iPhone 6* devices. We can find more automatic or semi-automatic disassembly systems, like: a robot for assisting in recycling batteries [16]; Bdiwi *et al.*'s workstation for cars [17]; and, less recently, Scholz-Reiter *et al.*'s disassembly cell [18]. These contributions are meant to excel in a very specific task, at the cost of limiting their application to controlled environments. Others have tackled the problem of automatic disassembly from a lower level perspective, like Zhang *et al.* [19], who seek to find disassembly motions for retrieving components that are closely coupled.

A more general disassembly approach is given by Puente *et al.* [20], [21]. Their device representation and precedence management are similar to ours. Hui *et al*'s contribution [22] is also very interesting in this regard. They apply genetic algorithms to extract good disassembly sequences from feasibility graphs. However, these researchers do not take into account uncertainty in a explicit way, nor do they consider preparatory actions like switching tools or rotating the device. On the other hand, the work of Alshibli *et al.* [23] is interesting because it seeks to address state uncertainty. Even so, they do not consider neither actions with multiple outcomes nor the possibility of reaching dead ends.

To the best of our knowledge, MDPs have not been applied to automatic device disassembly.

III. Proposed Methodology for Planning Disassembly Sequences

We model the disassembly domain as an SSPP, a class of MDP that is more general than Infinite Horizon Discounted Reward MDPs and Finite Horizon MDPs [24]. In SSPPs, rewards are reformulated as costs, there is no discount factor ($\lambda = 1$), and there is at least one absorbing (goal) state. For the purpose of this paper, we focus entirely on the planning problem. That is: we assume that the information about the current state of the device and the robot is extracted from the sensors and processed into a convenient form.

The SSPP is specified in PPDDL (*Probabilistic PDDL* [25]), a probabilistic extension of PDDL (*Planning Domain Definition Language* [26]). The devices are described in terms of their components and the relationships among them. We have aimed at crafting a domain that is general enough to fit a wide range of devices. In order to do this, we use object types and predicates that are not specific to a particular type of product. More details on these are given in Section III-A.

We only have access to the information that can be inferred from sensors. That is, total occlusions prevent the robot from knowing the whole configuration of the device. Partial occlusions are also interesting, because they give an idea of the precedence relationships between componennts. For the purpose of this paper, we focus on high-level planning and assume that the information about the environment comes from perception routines that are capable of computing facts such as the object types and partial occlusions.

We propose a decision-making unit that combines determinization with subtask selection. First, we will give more details on the disassembly domain that we have designed. Then, we will describe our decision unit's architecture. Next, we explain how subtasks are identified and isolated in a hierarchical fashion. Finally, we focus on the determinization method.

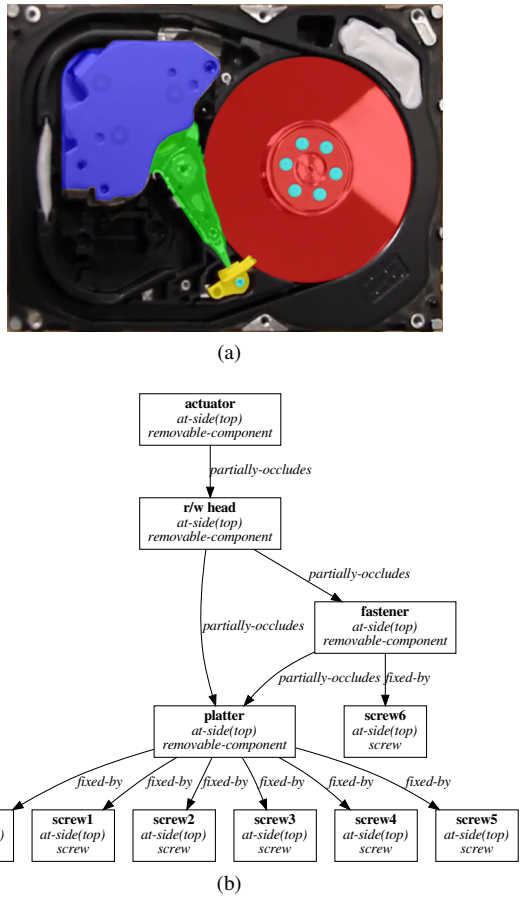
A. In-Depth View of the Disassembly Domain

The proposed problem consists in disassembling small electromechanical devices like hard drives. Fig. 1 illustrates a hard drive and a graph representing its topology.

We have hand-crafted a PPDDL domain that covers a large number of scenarios. The device is described in terms of generic PDDL object types like *screw*, *lever-point*, and *removable-component*. Relations and other circumstantial nuances are provided with predicates like *(fixed-by ?c - removable-component ?s - screw)*, *(at-side ?c - component ?s - side)* and *(partially-occludes ?c1 ?c2 - component)*. We believe that the proposed domain is interesting and rich from a planning perspective, and a good representative of the class of real-world problems that we want to target with our methodology.

The robot capabilities to cope with this task consist both of preparatory (e.g. picking, flipping, and placing the device and switching tools) and retrieval actions (e.g. unscrewing, levering, sucking, and bashing). The robot can perform both in-hand operations thanks to a SCARA finger, or operate on the device while it is resting on the table. Retrieval actions are not directly applied to components. Instead, from each component we identify a set of candidate points called *affordances*. Each affordance has a different degree of confidence (e.g. a levering point that is close to a corner of a surface is generally worse than one that is in the middle of a segment).

In our model, *dead ends* may arise if a tool or a component breaks, meaning that we cannot continue the task. All our



(a) (b)

Fig. 1. (a) Top view of a hard drive without its cover lid. The main components are highlighted. (b) Representation of the visible components as a graph.

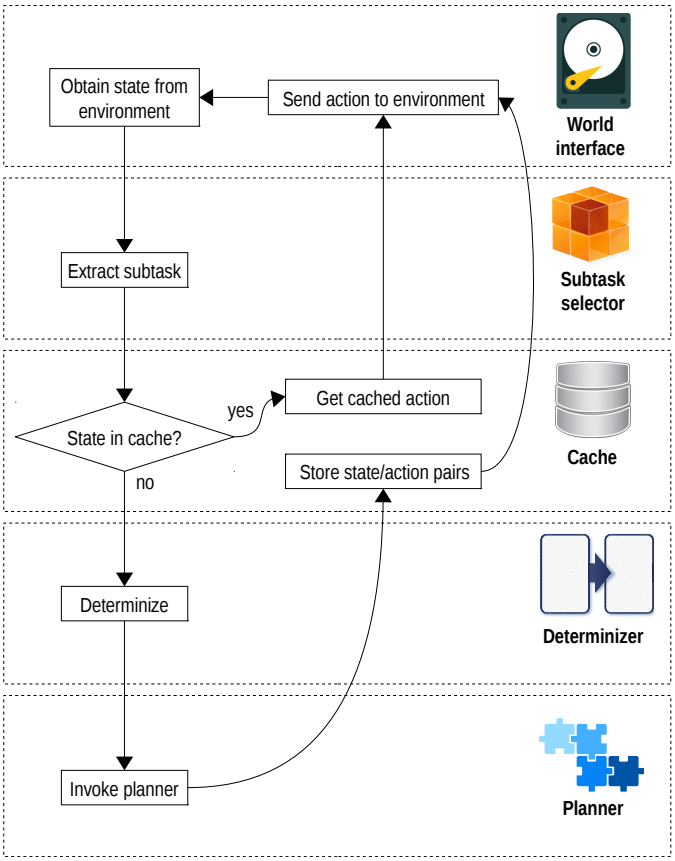


Fig. 2. Overview of the system's architecture and flow.

actions have a uniform cost to penalize plan length. Occlusions may prevent the robot from knowing the whole configuration of the device (e.g. the components below a closed lid). Therefore, the robot may learn of the existence of new fluents on-the-go. All these characteristics add up to the complexity of the domain. The decision unit has to evaluate diverse disassembly approaches, pondering which additional preparatory actions are worth executing and when it is convenient to switch tools.

B. Architecture Outline

Our system is composed of 5 modules: (1) a *world interface* that provides the current state as a set of PPDDL predicates; (2) a *subtask selector* that is in charge of deciding the next component to be retrieved, based on the component hierarchy induced by the topological constraints among components; (3) a *cache* for storing state/action pairs to avoid replaning if the previously computed plan performs as expected; (4) a *determinizer* that transforms the PPDDL specification into PDDL via ACTL [1]; and (5) a deterministic *planner* for processing the deterministic version of the problem and suggesting a plan. Fig. 2 depicts these blocks and their interaction more clearly.

The predicates provided by the *world interface* consist not only of the device's perceived configuration, but also of the robot status (e.g. which tool it is currently holding) and its interaction with the device (e.g. whether it is currently holding the device). They would normally be inferred from the perception routines and the robot's encoders. However, for our experiments we obtain them from a simulator, described later in Section IV.

On the other hand, the *cache* is trivially implemented, as it consists solely of a map from states to actions. The map is filled with the expected state trajectory derived from a deterministic plan and the actions suggested by this plan.

In this work we employ an out-of-the-box installation of *Fast Downward* [27] as the *planner*. While conceived several years ago, this is a planner that receives a lot of attention nowadays thanks to its modularity, which allows to extend it with state-of-the-art heuristics. Fast Downward is an essential tool in recent research on dead ends and heuristic search.

Therefore, in the rest of the section we will focus on the remaining two modules: the *subtask selector* and the *determinizer* .

C. Hierarchical Task Organization

The *subtask selector* module operates converting the set of predicates that describes the device into a graph representation. This graph is useful because it allows to reason more easily about the precedence relationships among components. Namely, it allows us to identify a set of components that are candidate for retrieval.

We perform topological sorting, removing from the state all the components and associated affordances that are partially

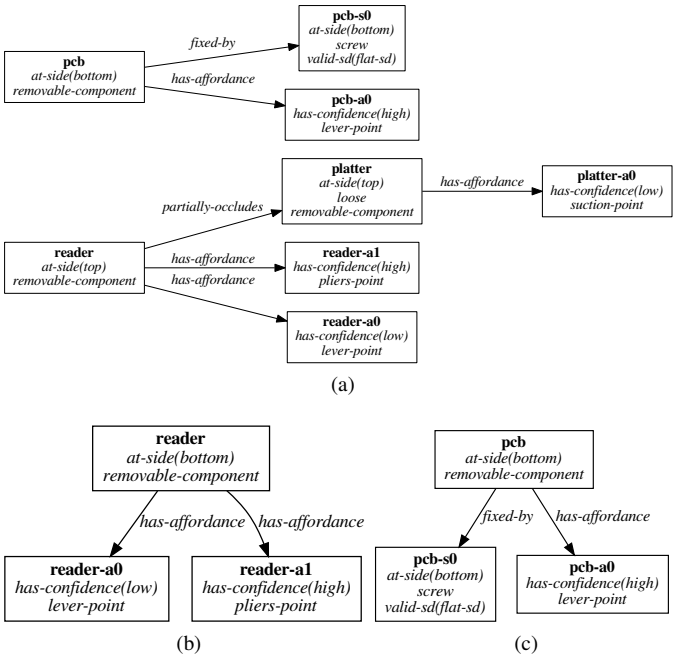


Fig. 3. (a) Device with partial occlusions. The only components eligible for retrieval are *reader* and *pcb*. (b) and (c) show the simplified state that results from focusing on each of these, respectively.

occluded by some other component. The rationale is that partially occluded components are often kept in place by the occluding component. See for instance Fig. 1a. The platter is partially occluded by the plastic fastener. During the manual disassembly of this hard drive, we found that the fastener was effectively restraining the platter, and its retrieval was a requirement.

It is not always the case that partial occlusion imposes hard precedence between two components. An example of this is the R/W head, which is occluded by the actuator, and that nonetheless can be extracted. However, in general it is still advantageous to take care of occluding components first, as they may be hiding unseen affordances (e.g. lever or suction points that are better than the currently visible ones). Another possibility is that removing the occluding object can result in more progression than expected (e.g. removing the actuator will also take away the R/W head, which is an unexpected beneficial consequence).

More complex assemblies require special treatment. We could think of two mutually occluding components. This special case can be handled grouping both of them in a single macro-component. On the limitations side, we could also think of a contrived assembly in which an occluded component keeps in place its occluding component. Since this is the opposite of what we are assuming, our subtask selection procedure would not work. While this could be addressed with a more clever subtask selection procedure, we have not found any assembly like this in practice, and we believe that our approach is illustrative enough to show how to break a large problem into smaller subtasks.

Fig. 3 illustrates the subtask selection procedure. The device's graph (Fig. 3a) induces a component hierarchy in which *pcb* and *reader* are the only components with no precedents.

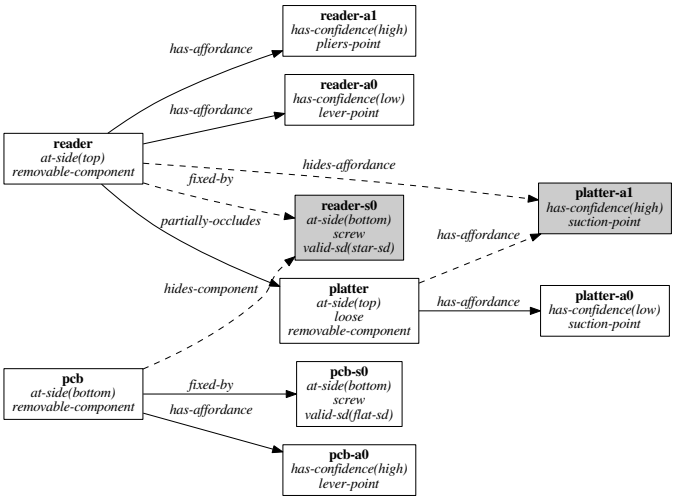


Fig. 4. Device with unseen components and relations (total occlusions). Namely, *pcb* is hiding some screws that are fixing the *reader* component at the opposite side of the hard drive. This corresponds to a real-life model.

Function `UpdateComponentStack` ($\text{stack}_{n-1}, \text{state}_n$)
  $\text{notPreceded} \leftarrow$ `TopologicalSort` (State_n)
  $\text{new} \leftarrow \text{NotPreceded} \setminus \text{Stack}_{n-1}$
  $\text{retrieved} \leftarrow \text{Stack}_{n-1} \setminus \text{NotPreceded}$
  $\text{stack}_n \leftarrow \text{stack}_{n-1} \setminus \text{retrieved}$
  `AddToBottom` $(\text{stack}_n, \text{new})$
  return stack_n
Function `SelectSubtask` $(\text{state}_{n-k+1..n}, \text{stack}_{n-1})$
  $\text{stack}_n \leftarrow$ `UpdateComponentStack` $(\text{stack}_{n-1}, \text{state}_n)$
  if `Stagnated` $(\text{state}_{n-k+1..n})$ **then**
    `AddToBottom` $(\text{stack}_n,$ `PopTop` $(\text{stack}_n))$
  $\text{C} \leftarrow$ `GetTop` (stack_n)
  $\text{state}'_n \leftarrow$ `IsolateComponent` $(\text{state}_n, \text{C})$
  return $\text{state}'_n, \text{stack}_n$

Fig. 5. Subtask selection algorithm

Without a knowledge base nor learning capabilities in the decision-making unit, there is no obvious way of choosing one over the other. Therefore, we allow the module to select randomly one of the candidates. It may be the case that the current view of the device is not complete. Consider Fig. 4, which shows the ground truth of the device depicted in Fig. 3a. In this figure it is easy to appreciate that *pcb* should be retrieved first, instead of *reader*. Therefore, if the *subtask selector* chooses to retrieve *reader* first, the robot would get stuck, risking to break one of the device's components. We address this issue by monitoring the state. If there is no progression in several successive steps, the selection algorithm assumes that there is some hidden precedence relationship and chooses a different component to retrieve.

This whole process is illustrated more clearly in the algorithm shown in Fig. 5. Here, each *stack* variable references a stack of ready-to-retrieve components. Each time no progression is observed in k successive steps, the top component is pushed to the bottom of the stack and the next component is retrieved. The *UpdateComponentStack* routine takes care of removing from the stack the components that have already been retrieved, and adds the components that were not visible in former states. *TopologicalSort* extracts all the components

with no precedences. *IsolateComponent* performs a graph reachability analysis and removes all the predicates and objects irrelevant for the retrieval of a certain component (e.g. think of *pcb* and *reader* in Fig. 3).

D. On-line Solving Via Determinization

Each of the subtasks retrieved via the `SelectSubtask` routine shown in Fig. 5 is still a probabilistic planning problem, described in the PPDDL language. We seek to solve each of them on-line (that is, avoiding the computation of full policies).

We transform our PPDDL problem into a PDDL one to take advantage of heuristic search and obtain plans in a timely fashion. However, to perform this transformation and obtain plans that make sense from their probabilistic counterpart point of view, it is necessary to compile the outcome probabilities into deterministic features. The idea is to force the planner to prefer more likely-to-succeed plans. The method of choice for this paper is ACTL.

ACTL is similar to *All-Outcome* (AO) determinization in that it creates a new deterministic action for each probabilistic action-outcome pair of the original domain. If there are N actions, each with M outcomes, ACTL will produce $N \cdot M$ deterministic actions or less (it can produce less actions if there are empty outcomes, since these are superfluous from the point of view of the deterministic planner).

Then ACTL assigns a *transformed cost* (C') to each action a in a way that penalizes less likely effects. To do so, it combines the probability of generating a certain state s' from the current state s (expressed as $\mathbb{P}(s'|s,a)$) with the cost of the original action (C) weighted by some parameter α:

$$C'(s,a,s') = \alpha C(s,a,s') - \log \mathbb{P}(s'|s,a) \qquad (1)$$

The original cost of the action $C(s,a,s')$ can be linked to some metric that we would like to optimize, such as execution time or energetic consumption. For the purpose of this paper, we are interested in optimizing plan length, so we set $C(s,a,s') = 1 \ \forall a,s,s'$. The goal is the retrieval of the most important components (e.g. *reader*, *pcb* and *platters*).

The α factor is a parameter that establishes a compromise between the original cost and the probability-based term of the *transformed cost*. We think of it as a way to trade-off eagerness to safety and vice versa: $\alpha = 0$ implies that the cost depends exclusively on the outcome probabilities, so the planner will try to find safe (likely to succeed) plans; $\alpha \to \infty$, on the other hand, gives more relevance to the previous cost of the action, diminishing the importance of probabilities and optimizing the original accumulated cost (the planner becomes more eager).

Fig. 6 illustrates the eagerness-safety trade-off. This plot corresponds to a domain with uniform-cost actions, so α is a proxy to control the number of plan steps at the cost of success likelihood. It is worth highlighting that the plot has been obtained with an admissible heuristic in order to get an optimal plan. The use of admissible heuristics is not always feasible in practice.

To apply ACTL, we flatten the probabilistic effects in all the actions. This makes much easier identifying the outcomes and their associated probabilities. Then, we transform each action-outcome pair into a deterministic action using Equation 1. Let us exemplify this transformation with the action shown in Fig. 7. Notice how the *bash* action expresses compactly 8 different outcomes (of which one is empty). These outcomes are translated into a deterministic action that shares the precondition and parameters of the original action. Fig. 8 shows one of these deterministic actions.

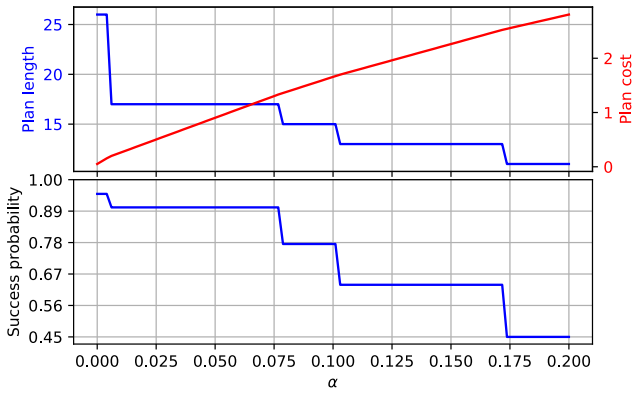


Fig. 6. Plot depicting the safety-eagerness compromise. Increasing α sacrifices plan success likelihood to potential speed gains, in terms of plan steps.

```
(:action bash
 :parameters (?comp - removable-component
  ?side - side)
 :precondition (and
  (not (broken-component ?comp))
  (not (broken-tool hammer))
  (at-side ?comp ?side)
  (current-side ?side) (current-tool hammer)
  (forall (?comp_ - component)
    (not (partially-occludes ?comp_ ?comp))))
 :effect (and
  (probabilistic
    0.25 (and
         (forall (?screw - screw)
           (not (fixed-by ?comp ?screw)))
         (forall (?screw - screw ?side_ - side)
           (not (at-side ?screw ?side_)))
         (probabilistic 0.5 (loose ?comp)))
    0.10 (broken-component ?comp))
  (probabilistic 0.05 (broken-tool hammer))
  (decrease (reward) 1)))
```

Fig. 7. Action with nested probabilistic effects.

```
(:action bash_o0
 ...
 :effect (and
  (forall (?screw - screw)
    (not (fixed-by ?comp ?screw)))
  (forall (?screw - screw ?side_ - side)
    (not (at-side ?screw ?side_)))
  (loose ?comp) (broken-tool hammer)
  (increase (total-cost) 6.075173815233827)))
```

Fig. 8. One of the deterministic actions derived from *bash* using ACTL with $\alpha = 1$. Parameters and preconditions are the same as in the original action, so they are omitted.

TABLE I
SUCCESS RATIO (S), AVG. ELAPSED TIME (E) AND COST (C) OF EPISODES WITH SIMPLE DEVICES.

Method	S	E (s)	C (steps)
LRTDP (FF)	27/150 (18%)	1.53	**13.63**
ao	98/150 (65.33%)	1.13	19.22
ml	84/150 (56%)	**0.45**	17.57
alph-0.00	98/150 (65.33%)	1.14	20.70
alph-0.05	**100/150 (66.67%)**	1.18	18.70
alph-0.10	92/150 (61.33%)	1.23	20.58
alph-0.20	98/150 (65.33%)	1.20	19.73
alph-0.40	97/150 (64.67%)	0.83	16.39
alph-0.80	**100/150 (66.67%)**	0.94	18.35
alph-1.00	92/150 (61.33%)	0.98	18.34

TABLE II
EXPERIMENT RESULTS WITH COMPLEX DEVICES.

Method	S	E (s)	C (steps)
ao	75/150 (50%)	1.49	21.27
ml	82/150 (54.67%)	**0.56**	20.27
alph-0.00	**102/150 (68%)**	1.31	22.53
alph-0.05	**102/150 (68%)**	1.42	21.83
alph-0.10	96/150 (64%)	1.43	22.43
alph-0.20	91/150 (60.67%)	1.27	22.07
alph-0.40	95/150 (63.33%)	1.26	20.67
alph-0.80	80/150 (53.33%)	1.10	19.98
alph-1.00	83/150 (55.33%)	1.13	**19.78**

IV. EXPERIMENTAL EVALUATION

We have conducted a series of experiments to test our proposal. We have crafted 10 PPDDL disassembly problems[1], each of which related to a different device. 5 of these represent simple devices with no occlusions, so the state should change in predictable ways from step to step. The remaining 5 are more complex and have occlusions. This means that new components may appear in the state after performing a retrieval action.

The world state is obtained from a simulator that is similar in nature to *MDPsim*[2]. It maintains a virtual state of the world in the form of predicates that are modified with each executed action, via outcome sampling. The main difference from *MDPsim* is that our simulator hides the non-observable objects and relations (those that are occluded).

We compare multiple approaches for solving these problems. As a baseline, we use an LRTDP algorithm initialized with the FF heuristic. We chose Bonet's *mini-gpt* implementation because it accepts PPDDL problems. On the other hand, we use several configurations of our decision unit. These configurations differ from each other in the determinization technique. Our code is freely available[3].

We have tested several values of α for ACTL. We have included in the comparison AO and *Most-Likely-Outcome* (MLO) determinization to see if ACTL shows any advantage over them. Notice that MLO and AO represent the same approach adopted by FF-Replan [2].

We have executed 30 episodes for each problem and configuration to estimate the percentage of solved problems (**S**), the average time per successful round (**E**) and the average episode's accumulated cost (**C**), which is equivalent to the plan length. The time limit for each episode is constrained to 5 minutes.

Results are shown in Table I for simple devices and Table II for complex ones. Each row depicts the results for a different solver. There are different flavors of our decision unit: *ml* and *ao* stand for MLO and AO determinization, respectively. ACTL determinization is represented as *alph-a*, where a is the value of the α parameter. For each method, we have reported *success ratio*, *elapsed time* and *cost* (disassembly sequence). While we think that the success ratio is the most important one in the disassembly domain, these metrics can be weighted differently depending on the particular application.

Notice that the success ratio is, at most, 66.67% on the simplest devices and 68% on the most complex ones. The reason behind this moderate success ratio is that the probabilities of the undesirable outcomes in the hand-crafted domain (events such as breaking a tool or certain components) are exaggerated so there is a non-negligible chance of arriving to a dead end. Therefore, the planner has to consider the trade-offs between different action routes that accomplish the same objective. Otherwise, the presence of alternative sequences of actions is irrelevant, and the disassembly domain loses a lot of its richness. In practice, the presence of dead ends also results in much harder problems. Therefore, the objective of our testbed is to compare determinization methods among them in problems that are difficult, in the sense that it is hard to avoid the dead ends.

We want to highlight that we applied LRTDP exclusively to simple devices. The rationale is that out-of-the-box LRTDP cannot handle dynamic state specifications (i.e. varying number of fluents). However, the results for the set of simple devices make evident a surprisingly low ratio of solved episodes. LRTDP managed to produce a successful disassembly sequence for the first device most of the times (27 out of 30), but the next devices caused LRTDP to take too much time, raising time-outs for the remaining episodes (180s per episode). Since the first device is also the easiest one, LRTDP takes less steps than the other methods when averaged through successful episodes. None of the other methods triggered a time-out.

There is no much difference between the determinization methods applied to simple devices, except perhaps in the case of MLO, which takes considerably less time than the other approaches. This is because MLO produces less actions than the other determinization methods, but sacrifices many relevant outcomes. This is evidenced by its lower success ratio with both simple and complex devices.

ACTL-based methods do not take substantially more time than AO and provide similar results for simple devices and a larger success ratio. However, we think that Table II provides the most interesting and promising results, since it illustrates perfectly the eagerness-safety trade-off. On the one hand, ACTL with $\alpha = 0$ achieves the highest success ratios. On the

[1]Shown in the supplementary material: www.iri.upc.edu/people/asuarez/documents/supplement-suarez2018.pdf

[2]https://github.com/hlsyounes/mdpsim

[3]https://github.com/sprkrd/planning_tools/

other hand, we can see how the success ratio can be sacrificed to potential cost gains with higher α values.

V. Conclusions and Future Work

We have developed a decision-making unit that decomposes large MDPs into smaller subtasks. These are then solved through a re-planning cycle that employs determinization. We have proposed the problem of disassembling electromechanical devices to exemplify our approach.

We think that the results depicted here show that determinization-based resolution methods for MDPs are suitable for practical problems. First, since they are based on heuristic search, and are fed with the most recent state specification, they are not so vulnerable to large state specifications nor to new variables introduced on-the-go (like newly discovered components, in our disassembly example). Secondly, *dead ends* are not so problematic from a theoretical standpoint, because they do not produce convergence issues. We can simply rely on the planner to find a candidate plan, and conclude that there is a *dead end* if the planner is able to demonstrate so or if it fails to find a plan in a certain amount of time. Moreover, this scheme fits very well in the goal-oriented nature of SSPPs. We also believe that the subtask selection strategy is useful and can work very well in problems that exhibit some type of hierarchical structure, since it bounds the complexity of planning for real-time applications.

The proposed domain is general enough to fit a large set of devices, since it is defined in terms of types and predicates that are not specific to a particular type of product. In this paper we have focused on the use case of dismantling hard drives, so we think that an interesting idea for future work is to try to describe other types of products using the concepts defined in our domain.

However, notice that we have not addressed explicitly partial observability. That is, we work with MDPs that grow over time when new variables are discovered, but we do not maintain a probability distribution over several alternative states that fit the current observation. We would like to explore in this direction in the future. In this same line of work, we think that it would be worth to incorporate *a priori* knowledge about the product's structure. This knowledge can be derived from images, CAD models or the robot's experiences, and can help initializing the probability distribution over the state variables, or identifying precedence relationships between components that cannot be detected from the perceived state.

Also, we have assumed static environments, that is, environments in which the only modifications are provoked by the robotic agent. This is the most natural way to model a problem with an action-centered language such as PPDDL. However, we would like to extend the present work to transition-based languages such as RDDL. One of the main appeals of RDDL is that it allows exogenous effects and concurrency. Moreover, it would allow us to take advantage of more modern state-of-the-art MDP solvers like PROST.

References

[1] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[2] S. Yoon, A. Fern, and R. Givan, "FF-Replan: A baseline for probabilistic planning," in *ICAPS*, no. 7, 2007, pp. 352–359.

[3] S. Yoon, A. Fern, R. Givan, and S. Kambhampati, "Probabilistic Planning via Determinization in Hindsight," in *AAAI*, 2008, pp. 1010–1016.

[4] S. Yoon, W. Ruml, J. Benton, and M. B. Do, "Improving determinization in hindsight for online probabilistic planning," in *ICAPS*, 2010, pp. 209–216.

[5] F. Teichteil-Köenigsbuch, G. Infantes, and U. Kuter, "RFF: A robust, FF-based MDP planning algorithm for generating policies with low probability of failure," in *ICAPS*, 2008.

[6] I. Little and S. Thiebaux, "Probabilistic planning vs replanning," in *ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[7] A. Raghavan, S. Sanner, R. Khardon, P. Tadepalli, and A. Fern, "Hindsight Optimization for Hybrid State and Action MDPs," in *AAAI*, 2017, pp. 3790–3796.

[8] M. Issakkimuthua, A. Fern, R. Khardon, P. Tadepalli, and S. Xue, "Hindsight Optimization for Probabilistic Planning with Factored Actions," in *ICAPS*, 2015, pp. 120–128.

[9] A. Kolobov, "Towards scalable MDP algorithms," in *IJCAI*, 2011, pp. 2818–2819.

[10] A. Kolobov, Mausam, and D. S. Weld, "LRTDP vs. UCT for Online Probabilistic Planning," in *AAAI*, 2012, pp. 1786–1792.

[11] A. Kolobov, P. Dai, Mausam, and D. S. Weld, "Reverse iterative deepening for finite-horizon MDPs with large branching factors," in *ICAPS*, no. 1, 2012, pp. 146–154.

[12] T. Keller and P. Eyerich, "PROST: Probabilistic Planning Based on UCT," in *ICAPS*, 2012.

[13] B. Bonet and H. Geffner, "MGPT: A probabilistic planner based on heuristic search," *JAIR*, vol. 24, pp. 933–944, 2005.

[14] A. Suárez-Hernández, G. Alenyà, and C. Torras, "Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation," in *IEEE/RSJ IROS*, 2018, pp. 4061–4066.

[15] C. Rujanavech, J. Lessard, S. Chandler, S. Shannon, J. Dahmus, and R. Guzzo, "Liam - An Innovation Story," Apple, Tech. Rep., 2016.

[16] K. Wegener, W. H. Chen, F. Dietrich, K. Dröder, and S. Kara, "Robot assisted disassembly for the recycling of electric vehicle batteries," *Procedia CIRP*, vol. 29, pp. 716–721, 2015.

[17] M. Bdiwi, A. Rashid, and M. Putz, "Autonomous disassembly of electric vehicle motors based on robot cognition," in *IEEE ICRA*, 2016, pp. 2500–2505.

[18] B. Scholz-Reiter, H. Scharke, and A. Hucht, "Flexible robot-based disassembly cell for obsolete TV-sets and monitors," *Robotics and Computer-Integrated Manufacturing*, vol. 15, no. 3, pp. 247–255, 1999.

[19] L. Zhang, X. Huang, Y. J. Kim, and D. Manocha, "D-plan: Efficient collision-free path computation for part removal and disassembly," *Computer-Aided Design and Applications*, vol. 5, no. 6, pp. 774–786, 2008.

[20] F. Torres, S. T. Puente, and R. Aracil, "Disassembly Planning Based on Precedence Relations among Assemblies," *The International Journal of Advanced Manufacturing Technology*, vol. 21, no. 5, pp. 317–327, 2003.

[21] S. T. Puente, F. Torres, and R. Aracil, "Non-Destructive Disassembly Robot Cell for Demanufacturing Automation," in *IFAC Proceedings Volumes*, vol. 36, no. 23, 2003, pp. 97–102.

[22] W. Hui, X. Dong, and D. Guanghong, "A genetic algorithm for product disassembly sequence planning," *Neurocomputing*, vol. 71, no. 13-15, pp. 2720–2726, 2008.

[23] M. Alshibli, A. El Sayed, E. Kongar, T. M. Sobh, and S. M. Gupta, "Disassembly Sequencing Using Tabu Search," *Journal of Intelligent and Robotic Systems*, vol. 82, no. 1, pp. 69–79, 2016.

[24] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[25] H. L. Younes and M. L. Littman, "PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects," CMU-CS-04-162, Tech. Rep., 2004.

[26] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - The Planning Domain Definition Language," 1998. [Online]. Available: www.cs.yale.edu/homes/dvm

[27] M. Helmert, "The Fast Downward Planning System," *JAIR*, vol. 26, pp. 191–246, 2006.