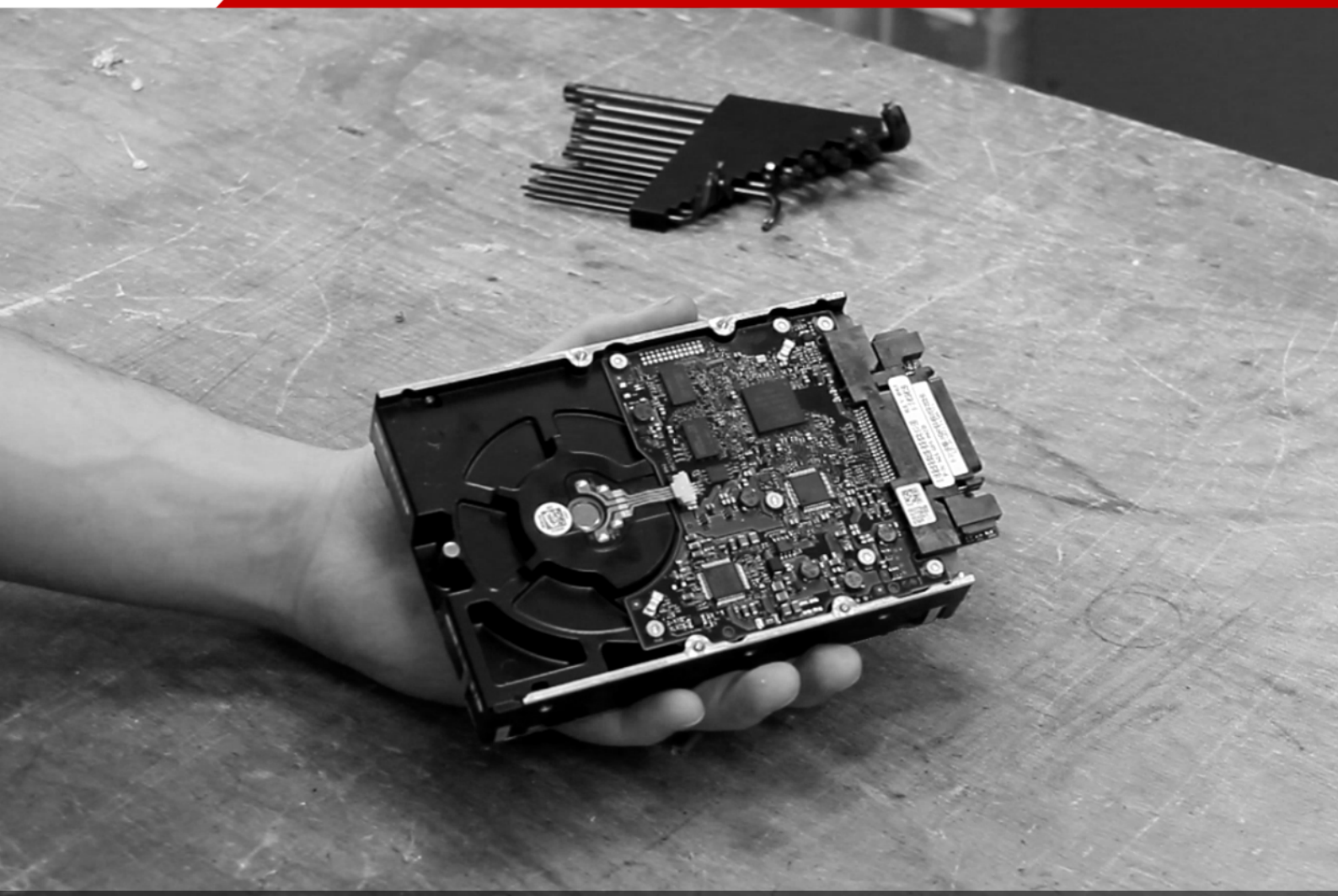
Institut de Robòtica i Informàtica Industrial



Towards effective planning strategies for robots in recycling

by Alejandro Suárez Hernández

to obtain the M.Sc. in Artificial Intelligence

Co-directed by
Guillem Alenyà Ribas & Carme Torras Genís (IRI, CSIC-UPC)

Supervised by
Cecilio Angulo Bahón (ESAII, UPC)

To be defended in
27th of Juny 2018

FIB Universitat Politècnica de Catalunya BarcelonaTech
Facultat d'Informàtica de Barcelona

Universitat de Barcelona
Facultat de Matemàtiques i Informàtica

Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili

Towards Effective Planning Strategies for Robots in Recycling

Thesis for the M.Sc. in Artificial Intelligence

Alejandro Suárez Hernández

Abstract

In this work we present several ideas for planning under uncertainty. Our intention is to apply these ideas to recycle electromechanical devices with a robotic arm. This domain presents two challenges: (1) since is not possible to guarantee the desired action outcomes due to limited precision and exogenous factors, the transition model is probabilistic; and (2) it is impossible to know the whole configuration of the device that is being disassembled at once due to occlusions. We opt to formulate the problem as a goal-based MDP (*Markov Decision Process*) or, equivalently, a SSPP (Stochastic Shortest Path Problem).

In general, MDPs' solutions consist in *policies* (i.e. a mapping from states to actions) rather than action sequences. Some of the most well-known algorithms to deal with MDPs are *Value Iteration* and *Policy Iteration*. However, these algorithms compute full policies and scale badly when the state consists of a large number of variables. Due to the *curse of dimensionality*, the state space grows too large. In addition, the computational effort invested in obtaining full policies for solving the MDP is wasted when new objects and, consequently, new state variables, are discovered. This renders the previous policy invalid because the specification of the state and the goal changes.

Therefore, we explore the effectiveness of an on-line planning method based on determinization followed by classical planning. We take advantage of the well-performing and state-of-the-art systems such as Fast Downward, or the older but still widely used Fast Forward. We consider the following determinization methods, each presenting different strengths and drawbacks: *All-Outcome*, *Single-Outcome*, α*-Cost-Transition-Likelihood* and *Hindsight Optimization*. Another exploitable feature of our particular domain is that there are strong precedence relations between the components. This allows us to *plan hierarchically*, bounding the plan horizon and, thus, reducing the computational effort, specially if replanning is necessary.

We have hand-crafted a testbed of disassembly problems to test these approaches. In addition, the determinization techniques are tested against domains from past IPPCs (*International Probabilistic Planning Competitions*) to see how suitable they are under different circumstances.

Desenvolupament de Estratègies Efectives de Planificació per Reciclatge amb Robots

Tesi per al Màster en Intel·ligència Artificial

Alejandro Suárez Hernández

Resum

En aquest treball presentem diverses idees per planificar sota incertesa. Volem aplicar aquestes idees a reciclar dispositius electromecànics mitjançant un braç robot. Aquest domini presenta dos reptes: (1) no és possible garantir el resultat desitjat de les accions a causa de la precisió limitada del robot i de factors externs i, per tant, el model de transició és probabilístic; i (2) és impossible conèixer la configuració completa del dispositiu que està sent desacoblat degut a les oclusions entre components. Optem per formalitzar el problema com un MDP (*Markov Decision Process*) amb objectius o, equivalentment, un SSPP (*Stochastic Shortest Plan Problem*).

En general, les solucions dels MDPs consisteixen en *polítiques* (i.e. funcions que assignen a cada estat l'acció òptima que s'ha d'executar) en lloc de seqüències d'accions. Alguns dels algorismes més coneguts per resoldre MDPs són *Value Iteration* i *Policy Iteration*. No obstant això, aquests algorismes computen polítiques completes i la seva complexitat és elevada quan l'estat està format per un elevat nombre de variables. Això és perquè l'espai d'estats és excessivament gran (*curse of dimensionality*). A més a més, l'esforç computacional invertit en calcular polítiques completes per resoldre el MDP no s'aprofita quan es detecten nous objectes i, per tant, canvia la definició de l'estat i dels objectius, invalidant la política anterior.

Així doncs, explorem l'efectivitat d'un mètode en línia de planificació basat en determinització i planificació clàssica. Aprofitem l'eficiència dels planificadore més avançats, com ara Fast Forward. Considerem els següents mètodes de determinització: *All-Outcome*, *Single-Outcome*, α-*Cost-Transition-Likelihood* i *Hindsight Optimization*. Una altra característica explotada en el nostre domini és que hi ha fortes relacions de precedència entre components. Això ens permet aplicar *planificació jeràrquica*, acotant l'horitzó de planificació i, per tant, reduint l'esforç de còmput, sobretot si és necessari replanificar.

Hem elaborat un conjunt de problemes de desacoblament per evaluar aquestes técniques. A més a més, les tècniques de determinització són provades en dominis de les IPPCs (*International Probabilistic Planning Competitions*) de anys anteriors, amb l'objectiu de veure com s'adeqüen a altres aplicacions diferents de la nostra.

Desarrollo de Estrategias Efectivas de Planificación para Reciclaje con Robots

Tesis para el Máster en Inteligencia Artificial

Alejandro Suárez Hernández

Resumen

En este trabajo presentamos varias ideas para planificar bajo incertidumbre. Queremos aplicar estas ideas a reciclar dispositivos electromecánicos mediante un brazo robot. Este dominio presenta dos desafíos: (1) no es posible garantizar el resultado deseado de las acciones debido a la precisión limitada del robot y a factores externos y, por ende, el modelo de transición es probabilístico; y (2) es imposible conocer la configuración completa del dispositivo que está siendo desensamblado debido a las oclusiones entre componentes. Optamos por formular el problema como un MDP (*Markov Decision Process*) con objetivos o, equivalentemente, un SSPP (*Stochastic Shortest Plan Problem*).

En general, las soluciones de los MDPs consisten en *políticas* (i.e. funciones que asignan a cada estado la acción óptima que se ha de ejecutar) en lugar de secuencias de acciones. Algunos de los algoritmos más conocidos para resolver MDPs son *Value Iteration* y *Policy Iteration*. Sin embargo, estos algorítmos computan políticas completas y su complejidad es elevada cuando el estado está conformado por un elevado número de variables. Esto se debe a que el espacio de estados es excesivamente grande (*curse of dimensionality*). Además, el esfuerzo computacional invertido en calcular políticas completas para resolver el MDP no se aprovecha cuando se detectan nuevos objetos y, por consiguiente, cambia la definición del estado y de los objetivos, invalidando la política anterior.

Así pues, exploramos la efectividad de un método en línea de planificación basado en determinización y planificación clásica. Aprovechamos la eficiencia de los planificadore más avanzados, tales como Fast Forward. Consideramos los siguientes métodos de determinización: *All-Outcome*, *Single-Outcome*, α-*Cost-Transition-Likelihood* y *Hindsight Optimization*. Otra característica explotada en nuestro dominio es que hay fuertes relaciones de precedencia entre componentes. Esto nos permite aplicar planificación jerárquica, acotando el horizonte de planificación y, por tanto, reduciendo el esfuerzo de cómputo, sobre todo si es necesario replanificar.

Hemos elaborado un conjunto de problemas de desensamblado para evaluar estos métodos. Además, las técnicas de determinización son probadas en dominios de las IPPCs (*International Probabilistic Planning Competitions*) de años anteriores, con el objetivo de ver cómo se adecúan a otras aplicaciones diferentes de la nuestra.

Contents

CHAPTER 1

SCOPE AND CONTEXTUALIZATION

1.1 Introduction

Industrial *robots* are programmed to perform highly specialized and repetitive actions in controlled environments. Therefore, their autonomy is quite limited and they are restricted to a small set of problems. We would like that robots that are not in such controlled scenarios are able solve a larger variety of problems, and even to react in the presence of adverse events. In other words, we would like robots to reason about their environment and about the potential effects that their actions may exert on it. Such capabilities would allow for great flexibility in the following ways: (1) tasks can be switched without reprogramming; (2) multiple solutions to the same problem can be found and assessed in terms of risk and potential gains in execution speed or other criteria; and (3) contingency mechanisms can be applied, should adverse events hinder the execution of the task.

We can think of several applications that are potential targets of these desirable qualities. One possibility is to use robots to assist old and impaired people for household chores or treatment [1, 2]. Another application is to allow robots to perform maintainance in difficult-to-access environments, such as subaquatic facilities [3, 4].

In this document we are concerned with *automatically disassembly* of electronic devices with a robotic arm. Namely, we want to enable the robot to retrieve the most valuable and/or dangerous components. The recycling application is very promising from environmental and scientific points of view. Our intention is to provide strategies for performing intelligent action selection. That is, given a disassembly scenario, the robot should be able to perform the most suitable action taking into consideration risk and speed criteria. We tackle this objective using tools from the area of *Automatic Planning and Scheduling*, namely probabilistic planning with MDPs (*Markov Decision*

Processes). In this document we explore methods of on-line resolution via determinization and hierarchical organization of the identified tasks. We use PPDDL (*Probabilistic Planning Domain Definition Language*) to define the MDP.

1.2 Motivation

The research presented in this document has been conducted aiming at developing techniques that can be effectively employed to *recycle* contraptions such as hard drives, hair trimmers, remote controls or electronic toys. Fig. 1.1 shows some examples of the devices that we have in mind.

(a) (b) (c)

Figure 1.1: (a) Bottom view of a hard drive being disassembled. (b) GSM amplifier being disassembled. (c) Several handie-talkie models being displayed.

In our view, this application poses very attractive challenges from the scientific point of view. In real life deterministic environments (i.e. fully observable state variables and non-random action outcomes) are more an exception than a rule, and this becomes evident in disassembly scenarios. On the one hand, the whole structure of the device cannot be perceived at once because of occlusions and perception noise, as Fig. 1.1 illustrates. Therefore, progression in the disassembly task is required to discover hidden components and geometrical relations. On the other hand, actions might not produce always the same outcome, due to positioning errors, noise in the movement of the robots and exogenous factors. For instance, levering the PCB of a hard drive in order to extract it from the bay might not success entirely, leaving the PCB hovering over one of the edges of the casing. In such case, one may need to consider an additional action like levering from a different point or holding the case upside down to let the loosen components fall.

We cannot forget the social and environmental impact of the proposed application. The current recycling industry is dominated by the *crush and separate* paradigm. Trash is crushed into very small bits that are then filtered and classified using physical properties such as density or inductance. However, the uniformity of the separated debris cannot be guaranteed. Even more importantly, devices often contain components or substances that are hazardous for the environment. In such cases, it is required that a human

(a) (b)

Figure 1.2: (a) Top view of a hard drive with the lid. (b) View from the same perspective without the lid. The previously occluded inner components are now visible.

manually removes the source of danger. This has associated health risks for the operator. Moreover, it is inefficient and costly, so a great amount of trash is simply incinerated, posing a serious thread for the ecosystems. Another drawback of this method is that precious or reusable components (e.g. PCBs, capacitors, magnets) are destroyed, when they can be used right away or after some refurnishing in the manufacturing of new products.

In light of these arguments, a robotic recycling system is appealing from the sustainability perspective as well. To avoid the drawbacks of the *crush and separate* method, we would like the following features for the robot: (1) it should ideally adapt to different electromechanical contraptions (including different brands/models of the same device); (2) it should correctly identify the tasks that it has to complete to perform a successful disassembly; and (3) it should be able to work even with damaged devices.

1.3 Involvement in H2020 Imagine

This research has been conducted as part of the European H2020 project *Imagine: Robots Understanding Their Actions by Imagining Their Effects*, or just *Imagine*[1]. The scientific objective of *Imagine* is to make robots aware of their environment and, in general, to achieve the autonomy objectives discussed previously in Section 1.1. The project focuses on the recycling application introduced in the former section.

There are seven European research centers participating in *Imagine*, each working on a different subsystem. One of these subsystems is a planning and decision unit, which is developed by the IRI-CSIC (*Institut de Robòtica i Infor-*

[1] `https://imagine-h2020.eu/`

màtica Industrial[2]*-Consejo Superior de Investigaciones Científicas*[3]) partner. This thesis is hosted in this institution and is closely related to the development of the *Imagine*'s decision unit. This subsystem is a fundamental part of the overall system, since it evaluates the scene and selects the action that is more suitable for the perceived state. This means that it has to make sure that the action is beneficial in the long term and that the risk of falling into a dead end is controlled.

The project also features PHYS and ASC as novel ideas for the considered application. The first is a very realistic and powerful physics simulator that can give a very precise idea of the state that results from executing an action, but makes intensive use of computational resources. The second is an Association Engine that identifies the points of interest of the scene, and also learns a high level physics model ("folks physics") that is much quicker to invoke than PHYS, but also less accurate. They are very useful and have great potential, because the planner can interact with them and "imagine" plans before any action. Another key aspect of the project is the concept of ADES, or *Action Description*, for storing the symbolic description of the actions and the DMPs (*Dynamic Movement Primitive*) associated to each one. Other aspects of the project include the perception of the scene and the design and construction of a multifunctional gripper.

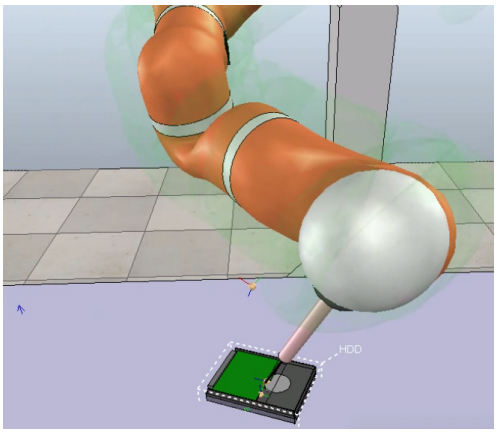


Figure 1.3: Execution of a levering action in the V-REP simulator.

As of today, we have been participating actively on *Imagine* for more than a year. In April of 2018 the project was evaluated by a Project Officer and three experts in different areas of robotics. It received very positive critics. Part of the presentation for the evaluation meeting consisted in a demonstration that displayed basic prototypes of the different subsystems interacting with each other, and was able to solve very simple disassembly simulated scenarios involving a hard drive (see Fig. 1.3). This demonstration already featured a planner prototype in the loop. Our objectives for this year is to extend this planner with new capabilities, including the ones presented here, but also with a rich and beneficial interaction with PHYS and ASC.

[2] `http://www.iri.upc.edu/`
[3] `http://www.csic.es/`

1.4 Planning concepts

We see in the area of *Automatic Planning and Scheduling* tools that potentially bring us toward the desirable qualities exposed in Section 1.1. Planning is a branch inside Artificial Intelligence that aims at solving problems that are defined *declaratively*, with as little control (procedural) knowledge as possible. That is, a planning algorithm operates with a model that encodes the dynamics of the domain and, for each problem, outputs an action sequence or a policy that solves it, has a high success probability or has a high expected reward.

Planning is not a discipline that is particular to robotics and, therefore, there exist multiple benchmarks and work in the field that is general enough to be of use in many areas. One of the most successful conferences that spreads knowledge in the field of planning is ICAPS (International Conference in Automatic Planning and Scheduling) [4]. The conference is held every year and hosts a competition that seeks to push the boundaries of the state of the art planning systems.

In this section we seek to give a brief overview of some of the most of the most important concepts in planning. Later, In Chapter chap:theory we give a more formal description of the planning language and paradigm that is relevant for our work.

1.4.1 Domain and problem

One of the common themes in planning is the separation between the *domain* description and the *problem instance* facts. This separation is not always present, since, given a planning paradigm, it is possible to provide an ad-hoc representation of the problem (e.g. a transitions and reward matrices for an MDP, or a graph for deterministic problems). However, the domain/instance concept gives a principled way for defining problems declaratively using specialized modeling languages (more on this later).

The domain models the logic of the environment. That is, the predicates and variables that define the state, and the actions that are available to the agent(s). This domain is common for all the problems that follow that share the same logic. For instance, all the problems from the recycling application would use the same domain, that specifies the levering and unscrew actions.

On the other hand, each particular instance has associated a set of facts that are linked to this very problem. That includes the initial state, the goal condition, the optimization criteria and so on. Let us provide an example for the recycling scenario: for disassembling a hard drive, the initial state consists in the specification of the current configuration of the device, as the robot sees it (the cover and some screws on the top and the PCB, the motor-PCB connector and some screws on the bottom); the goal consists in the removal

[4] `http://www.icaps-conference.org/`

of all the macro-components (the cover and the PCB). Notice that in this particular example, once the cover is removed, the robot will be able to see more macro-components and update the goal. The goal can also include the optimization of some metric (plan length, some measure of cost, reward...).

1.4.2 Planning paradigms

For our work we focus on goal-based MDPs for representing recycling problems. However, we think it is illustrative to present the planning that we have considered in order to justify our choice:

Deterministic planning: A deterministic planning problem is given by a graph of states connected by the allowed transitions due to actions, an initial state and a set of goal states. Solutions to deterministic problems consist of ground actions sequences. As the name suggests, the actions are deterministic (i.e. they always produce the same outcome), and the state is fully observable. PDDL (*Planning Domain Definition Language* [5]) is the language of choice for specifying domains and instances in the *Deterministic Track* of the IPC (*International Planning Competition*) organized by the ICAPS. This paradigm is often hard to apply in the real world because the environments are noisy. However, in certain applications it is acceptable to assume that the actions will always render the expected outcomes, that all the relevant state information is always available and that small deviations from the expected outcomes are tolerable and can be handled via replanning. Deterministic planning has a place within our work, since part of our approach consists in determinizing a probabilistic domain.

MDP: MDP stands for *Markov Decision Process*. They drop the deterministic state-transition assumption and allow for probabilistic effects. Moreover, MDPs are typically formulated as reward-maximization problems rather than graph search. However, we can also think of specific class of MDPs which are goal-based and that are also also known as SSPPs (*Stochastic Shortest Path Problem*). In these, there exists a subset of states called goals or absorbing states such that all actions lead to self-transitions with 0 reward. MDPs are specified by the *transition probability*, the reward function and a discount factor that determines how important are the long-term versus the short term gains. The solution for general MDPs are *policies*. That is, a function that tells the agent which action should it pick in each state to maximize the accumulated discounted reward (this is formalized in Chapter 2). The computation of full or off-line policies is costly in large-scale MDPs, although there exist online methods to overcome this issue. We apply this paradigm to our problem because it is adequate for expressing uncertainty about the action outcomes, but stills allows for efficient solving techniques, avoiding the issues of more sophisticated techniques. The language employed in the IPPCs (*International*

Probabilistic Planning Competition) for describing the domains' logic and the problems' instances are PPDDL (*Probabilistic PDDL* [6]) and RDDL (*Relational Dynamic Influence Diagram Language* [7]).

POMDP: POMDPs (*Partially Observable MDP*) introduce an additional level of uncertainty, dropping the assumption of full observability. The planning agent no longer has access to the ground truth about the current state variables. Instead, it receives *observations* that are drawn from a probability distribution that depends on the true state and the last executed action. Therefore, the agent operates in belief space (i.e. a continuous space of probability distributions over all the possible states). Due to this, the resolution methods of POMDPs become more sophisticated and intensive from a computational perspective. While, seemingly, this is the best approach to our recycling domain, the main source of state uncertainty comes from unknown objects that cannot be seen due to occlusions. Therefore, the state is *incomplete* (which is different than unobserved), and POMDPs would not greatly help at this issue. We assume that other sources of state uncertainty can be handled effectively via replanning. The RDDL language has enough expressivity for specifying POMDPs, too. Other languages for specifying POMDPs are Cassandra's `pomdp-sove` file format [8] and PomdpX [9].

Reinforcement Learning: Since planning deals with tuning an agent's behavior to operate in a certain environment, it is possible to see some similarities between planning and Reinforcement Learning. The main difference is that, in their most basic form, a Reinforcement Learning agent is model-free and dependent on its learning capabilities to decide the suitable action for a certain state, while a planner takes advantage of a model of the domain and employs informed search or other heuristic methods to find a plan (although definitively there exist works that allow partial domains and that incorporate learning capabilities to complete them [10, 11]). Reinforcement Learning is a "tabula rasa" framework [12], while planning takes advantage of an existing model of the domain to tune the agent's performance. In Reinforcement Learning it is often assumed (although it is, by no means, a necessary condition) that the underlying model of the environment follows the same principles as an MDP, although it is unknown to the agent.

1.4.3 Modeling languages

Above we mentioned a couple of options for modeling MDP problems in a principled way: PPDDL [6]) and RDDL [7]). These languages allow to easily encode the dynamics of the environment, and to interpret already encoded ones. The language of choice for our work is PPDDL. Here, we give the highlights of both in order to justify our choice. Later on, in Chapter 2, we describe PPDDL in more detail.

PPDDL: effects-based language. That means that all the transitions are caused by an action executed by the agent(s). All the possible changes in the state are organized as effects of the different actions that are available. In fact, PPDDL is just an extension over PDDL to add probabilistic outcomes to the actions and rewards. PPDDL does not provide any "iddle" action nor any language mechanism for concurrency and exogenous effects (i.e. effects that are independent of the selected action). In other words, PPDDL considers that all the changes that may occur in the state are the result of conscious choices made by the agent. Much like in PDDL, the state is represented as a collection of predicates. Numeric variables can be also encoded in the state, although with very limited support from the available planners. state variables. We have chosen PPDDL because an effects-based language makes it much easier to design and manipulate the disassembly domain. The easiness of manipulation becomes specially important for the determinization techniques.

RDDL: transition-based language. RDDL regards everything as a fluent or variable, including the actions and the rewards. The language allows to specify a transition function for all the state variables. The transition function may depend on the executed action and the previous state variables' values. It can be defined stochastically as well. Therefore, RDDL has the potential of expressing domain logics with exogenous effects much more naturally than PPDDL. However, since the changes are not grouped into action, some domains that can be written very easily in PPDDL are somewhat more tedious to reproduce in RDDL. The main reason we have chosen PPDDL over RDDL, however, is that RDDL makes it considerably harder to implement the determinization techniques that we explore in this document. Since we do not need the additional expressivity of RDDL, this is not a big concern. As a side note, RDDL is expressive enough to also represent partial observability.

1.5 Approach

As said before, we resort to the MDP formalism to express our problem, capturing the uncertainty over the state transitions. One of the main limitations of MDPs for real-life problems is that they assume full observability of the state variables. In practice, this is not true for our application. However, dropping this assumption results in POMDPs, whose resolution is computationally costly and scales badly with the size of the state space. Moreover, POMDPs do not provide any substantial expressivity gain when the agent does not know all the state variable, which happens when the agent is not aware of all the relevant objects of the scene.

We update and solve a new MDP each time the specification of the state changes (i.e. when new objects are discovered after performing an action). Even though solving MDPs is much less intensive than solving POMDPs, con-

ventional algorithms that obtain a full policy, such as *Value Iteration* and *Policy Iteration*, still suffer from the curse of dimensionality when there are many state variables due to the *curse of dimensionality*. Then, computing a full policy becomes infeasible. We explore two complementary strategies for dealing with this: (1) determinization algorithms together with classical planners; and (2) decompositon of the main goal into sub-goals, taking advantage of the topological precedences imposed by some geometrical and physical features (e.g. partial occlusion).

The first of these two strategies consists in constructing a deterministic version of the problem and solving the determinized problem with a planner such as Fast Forward[5] [13] or Fast Downward[6] [14]. The rationale behind this idea is that solving the determinized problem is typically much faster than computing an off-line policy. The first action suggested by the deterministic planner is executed. Then, the process is repeated for the next state. Optionally, we can cache plans to execute them in sequence if everything goes according to expectation (i.e. the agent does not replan until a deviation from the expected outcomes is detected). Good determinization strategies encode the stochasticity of the outcomes somehow (e.g. as action costs), so the classical planner comes up with plans that make sense from the probabilistic perspective or, said differently, have a good chance of succeeding.

The second strategy has more to do with limiting the planning complexity. The recycling domain is a good candidate for task decomposition, since there are clear precedence relations between components. For instance, it makes sense to disassemble the R/W arm of a hard drive before the platters, because the arm may move and block the platters. The robot can identify the macro-components that are worth retrieving as the main tasks, and sort them topologically given the precedence relations. Then, the planner solves one small task at a time, without caring about the rest and omitting from the state the facts and variables that are not relevant for the selected task.

1.6 Scope and goals

Let us remind the general objectives presented in Section 1.1: we seek to enable a robotic arm to perform automatic disassembly of an electromechanical device. We approach the problem from the planning perspective, using the MDP formalism. That is, we allow actions to have associated stochastic effects. We seek to solve MDPs via on-line methods based on determinization, and to limit the planning complexity with a hierarchical decomposition of the recycling goal.

The present work assumes that perception and low-level robot control routines are already provided, and that there is an existing interface to retrieve

[5] `https://fai.cs.uni-saarland.de/hoffmann/ff.html`

[6] `http://www.fast-downward.org/`

the state (as logic predicates) from the world and to execute actions on it. We present tools for modeling stochastic problems from the recycling domain, as well as techniques for dealing efficiently with these problems. Although all the examples and benchmarks have been conceived for the particular use case of the hard drive, the techniques described here can be extrapolated to other devices by extending the state specification with variables particular to these devices.

More specifically, we cover the following points:

- We propose a tentative domain for the recycling application, following the premises of the Imagine project. We provide a set of predicates that are meant to represent the different components of the device and the links among them. Moreover, the domain contains the schemata of several manipulation action for retrieving the device's components or flipping it. PPDDL defines an action-centric DBN (*Dynamic Bayes Network*) with full observability or, equivalently, a MDP.

- We explore and test several determinization techniques to achieve fast approximate on-line solutions to MDPs. The rationale is to avoid costly policy recalculations each time new objects are discovered (e.g. after removing the lid of a hard drive). The considered determinization algorithms are: (1) All-Outcome and Single-Outcome [15]; (2) α-Cost-Transition-Likelihood [16]; and (3) Hindsight optimization [17, 18].

- We complement the previous techniques with hierarchical planning, taking advantage of the strong precedences that arise naturally in the considered application. This occurs, for instance, when the reader of a hard drive is occluding the platters.

We assess the effectiveness of these strategies testing them against benchmark problems. On the one hand, we have all the problems from past IPPCs (*International Probabilistic Planning Competitions*, hosted each year by the ICAPS). These are useful to evaluate the suitability of the different determinization strategies in domains different from ours. On the other hand, we have hand-crafted a set of benchmark problems that consist of hard drive variations with different disposition of components. The determinization algorithms are evaluated on top of these, too. In addition, these serve the purpose of assessing the computational gain of the hierarchical planning method that is specific for the recycling domain.

1.7 Related work

In this section we review some contributions that are related somehow to our own work. Some of these contributions are purely theoretical, while others are practical recycling stations.

G-Pack [19] and PROST [20] are two state-of-the art probabilistic planners that operate with RDDL (Gourmand actually operates with a PPDDL dialect that results from translating RDDL models) and provide on-line solutions to the input MDPs. They interact with a server that sends the most recent state. They have outperformed the rest of the planners of the 2011 and the 2014 IPPC. Gourmand is based on the *Labeled Real-Time Dynamic Programming* algorithm [21], while PROST uses *Monte Carlo Tree-Search* with the UCT (*Upper Confidence bound applied to Trees*) variant [22]. Both algorithms perform well in an on-line setting, with no one being consistently better than the other in all the IPPC benchmarks, and are worth of consideration in future work.

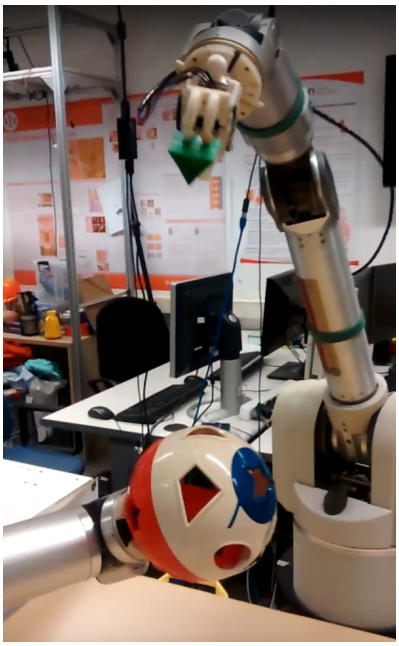


Figure 1.4: Barrett WAM robotic arm inserting a triangular piece through a cavity with the same shape

Our previous work [23] is also relevant. We illustrate how to apply concepts of hierarchical and probabilistic planning to a manipulation task that is comparable with an assembly scenario (the inverse of the problem covered in this document), as shown in Fig. 1.4. Much like in the present work, geometrical concerns play an important role in the planning task.

The work of Kaelbling and Lozano-Pérez [24] deals with hierarchical planning in a way that is very similar to our approach. In a more recent paper [16] they combine this hierarchical planning approach with probabilistic planning in belief space. Very much like us, they base their resolution method in determinizing the SSPP that results from their formulation. The determinization they propose is called α-Cost-Transition-Likelihood. It consists in introducing as many actions as possible outcomes exist in the original domain. Then each action is given a cost that depends on its transition cost and in its probability. This is one of the methods we employ in our own research.

The other determinization strategies that we study are All-Outcome/Single-Outcome [15] and Hindsight Optimization [17, 18]. The former is a baseline determinization technique that simply transforms outcomes to actions, but without assigning any cost. The second one is, perhaps, the most sophisticated of all the considered methods since it approximates the utility of each action in the current state (also known as the Q function in Reinforcement Learning). It needs to invoke the deterministic planner several times before choosing an action.

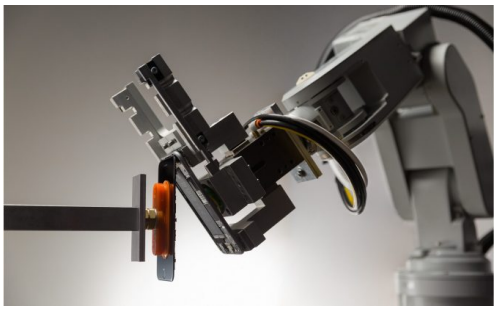


Figure 1.5: Liam robot removing the screen of an iPhone device. Source: `https://youtu.be/AYshVbcEmUc`

We believe that the work of Martínez *et al* is also interesting in the context of our research. It describes an approach for obtaining the operators of a domain via Reinforcement Learning and teacher demonstrations. The NID (*Noise Indeterministic Deictic*) rules formalism is employed to represent the probabilistic outcomes of the operators as disjoint effects, each one with its associated probability. This makes learning new effects through experience and demonstration very convenient. This formalism is adequate for our own purposes because it simply corresponds to an expanded form of the probabilistic PPDDL outcomes. In the future, we may consider completing the specification of our domain with a similar approach, but substituting the teacher demonstrations by physics simulations.

On a more practical level, a recent robotic systems that is more closely related to the application presented here is *Liam* (showed in Fig. 1.5), developed by Apple [25]. Liam is a recycling robot specially conceived to dismantle *iPhone* devices. It is equipped with tools for rescuing the most critical components: the battery, the camera and the logic board. While the details of the robot are not revealed, it would seem that it has been designed for operating in highly controlled environments and following a scripted sequence of actions for retrieving and classifying the different components. In other word, the system is tuned to work with a single type of device, while our goal is to work toward a more general recycling robot.

The work of Torres *et al* [26] falls directly within the recycling application. In fact, it belongs to the Spanish CICYT project (*Sistema Robotizado de Desensamblado Automático basado en Modelos y Visión Artificia*). Much like us, they seek find disassembly sequences for an electromechanical device. They represent these devices as graphs and automatically detect clusters of components and topological dependencies. However, they do not consider stochasticity.

Finally, it is also worth considering the paper of Zhang *et al* on part removal [27]. They propose a method for removing parts from a device with collision free paths. It is more concerned than us with the physical aspects of the problem, rather than with complex long-term goals and interdependences between components. In the future, we would like to incorporate some degree of low-level information into the planner within the Imagine project. PHYS and ASC, presented in Section 1.3, are invaluable tools for this purpose.

CHAPTER 2
THEORY AND BACKGROUND

2.1 MDPs and SSPPs

The concepts of MDPs (*Markov Decision Process*) and SSPPs (*Stochastic Shortest Path Problem*) are a fundamental part of this work. The reason is that we want to account for probabilistic actions and they allow to model such stochasticity. The implemented determinization methods described in Chapter 3 are meant to solve efficiently MDPs in an on-line manner. Therefore, in order to fully understand them, we present here the formal definition of MDPs and SSPPs. We focus on discrete MDPs formulations. Notice that it is possible to consider continuous state and/or actions spaces, although these naturally present more challenges.

Definition 2.1.1. Markov Decision Process A Markov Decision Process is a tuple (S, A, T, R, γ) where:

- S is a finite set of states.
- A is a finite set of actions.
- $T(s, a, s') = \mathbb{P}(S_{t+1} = s' | S_t = s \wedge A_t = a)$ is a probabilistic Markov transition model. For each triplet (s, a, s'), it indicates the probability of transitioning from s to s' when action a is taken in state s.
- $R(s, a, s')$ is a reward function where $R(s, a)$ is the immediate reward for transitioning from state s to state s' through a.
- γ is the discount factor. Rewards are diminished by this factor after each time step.

There are alternative definitions that consider a reward of the form $R(s)$, and also of the form $R(s, a)$. These can be made equivalent with a slight

re-parametrization of the problem. They do not alter the problem in any fundamental way [28].

The reward accumulated over time is:

$$\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$$

, where a_t is the action selected by the agent at each time step. Solving a MDP consists in finding a *policy* that maximizes the total or accumulated reward. A policy is a function $\pi(s) \in A$ that maps each state to the action that the agent should preferentially use in that state.

The discount factor γ gives or subtract importance to the long-term reward. A very low γ may result in a process in which only the most immediate and short-term actions have an effect in the accumulated reward. $\gamma < 1$ is very convenient because it guarantees that the accumulated reward is bounded, as long as all the immediate rewards are not infinite. $\gamma = 0.99$ and $\gamma = 0.95$ are usual choices.

We can assign an utility function to each state. This function gives an idea of the accumulated reward that we can expect to obtain if this state is reached. This utility function is defined as a *Bellman equation*:

$$V(s) = \max_a \left\{ \sum_{s'} \left(R(s, a, s') + \gamma \mathbb{P}(S_{t+1} = s' | S_t = s \wedge A_t = a) V(s') \right) \right\}$$

.

This equation already gives an indication for a couple of dynamic programming algorithms for solving MDPs: *Value Iteration* and *Policy Iteration*. Both algorithms are based on the same idea: iterating over all the states and performing relaxations according to the Bellman equation. In their most basic form, these algorithms compute *off-line* or *full policies*. That is, an exhaustive policy array for all the states. This is fine for problems with small state spaces, but impractical when the state is factored and consists of several state variables. Say, for instance, that in a hypothetical problem the state is composed of 40 binary state variables. Then, the state space would have a cardinality of $2^{40} \approx 10^{12}$, which clearly makes the direct application of Value and Policy iteration infeasible.

Notice that, as per the definition given here, the resolution of MDPs implies maximizing the expected reward rather than finding a sequence of actions. So far, we have not mentioned goals in the classical planning sense. However, we wish to consider problems with specific objectives (e.g. the retrieval of a hard drive's PCB, or the removal of the top lid). SSPPs (*Stochastic Shortest Path Problems*) are a subclass of MDPs. They contain terminating or absorbing states. SSPPs are well-suited for our needs because they allow easy application of determinization methods that do not require exhaustive iteration over all the state space.

Definition 2.1.2. Stochastic Shortest Path Problem A Stochastic Shortest Path Problem is an MDP with absorbing states and negative rewards. The SSPP is specified by a tuple (S, A, T, C, G) as follows:

- S, A and T have the same meaning as in an MDP
- $C(s, a, s')$ is the cost of transitioning from state s to state s' through action a. $C(s, a, s') = -R(s, a, s')$.
- $G \subseteq S$ is the set of absorbing states. $\forall g \in G, a \in A \quad C(g, a, g) = 0 \wedge T(g, a, g) = 1$.
- the discount factor is $\gamma = 1$.

While SSPPs may be solved via conventional MDP techniques such as *Value Iteration*, we can take advantage of their goal-based nature to use more specific and efficient techniques. In the extreme case in which all the transitions are deterministic, SSPPs are equivalent to classical planning problems. Notice that, since all the rewards are negative, we allow $\gamma = 1$ without resulting in an ill-defined problem with positive cycles. The problem essentially consists in reaching one of the goal states with high probability minimizing the transition cost. For a transition cost $C(s, a, s') = 1 \ \forall s, s', a$ this would be the same as minimizing the plan length.

2.2 Specification of the problem in PPDDL

We represent the dynamics of our SSPP using PPDDL (*Probabilistic Planning Domain Description Language*) [6]. As said in Section 1.4, PPDDL is an action-centric language. While exogenous effects and events are difficult to model in PPDDL, it makes it easier to express the effects of conscious choices made by the robot.

The body of a PPDDL domain consists of:

- The list of required *language features* (e.g. conditional effects universal or existential preconditions).
- The *type hierarchy* (objects can have an associated type).
- The *predicate list*. These act as binary state variables. Problem instances are constructed indicating which of these predicates are true. PPDDL operates under the *close-world* assumption, so the not included predicates are considered to be false.
- Optionally, a list of *functional fluents* (numeric state variables). This feature is not widely supported, and we do not use it.

- The list of *actions*, each having a name, a list of *parameters*, the *precondition* and the *effect*.

PPDDL is built on top of PDDL: it has a LISP-like syntax, supports several types of preconditions (e.g. simple predicate check, negative, conjunctive, existential, universal) and effects (e.g. add effects, delete effects, conditional, total). The main additions of PPDDL are the implicit declaration of a `(reward)` numeric variable, should the `:rewards` requirement be listed; and probabilistic effects, if the `:probabilistic-effects` requirement is listed. Fig. 2.1 shows an example PPDDL domain that features two deterministic actions and one probabilistic actions.

It is our intention to fit our domain as much as possible to the premises of the Imagine problem:

- We will use a *single robotic arm*.
- Different *tools* are required to disassemble components such as planar lids, PCBs and screws.
- The robot should be able to grasp the device and turn it over, either to operate on the other side or to let loosen components fall down.
- The points of interest of the different actions are known as *affordances*. An affordance consists in a special point in a component for levering, suctioning or extracting with pliers. It makes sense to consider affordances with different levels of confidence. The confidence determines the success probabilitiy of the associated action. For an example, see Fig. 2.2.

In the remaining of this chapter we will describe how we modeled the dynamics of the recycling domain in PPDDL.

2.2.1 Robot capabilities

We assume the robot to be equipped with a multifunctional gripper. This gripper allows performing bimanual skills with just one manipulator thanks to its two kind of fingers: a couple of parallel fingers for grasping and a SCARA finger to assist in grasping or for manipulation of the device while it is held. The gripper allows to carry several tools and has the potential to use them with two grasping modes: power grab, using all the fingers to perform a stable and powerful grasp; and the SCARA mode, using the third finger to grab the tool while the parallel fingers hold the device.

All of these considerations have been pondered in order to elaborate a domain that describes as good as possible the dynamics of a recycling domain. Whether the robot has equipped one tool, uses a particular mode or is holding the whole device is critical for planning the next actions. We propose the following predicates to describe the robot status and its interaction with the device:

```
(define (domain triangle-tire)
(:requirements :typing :strips :equality :probabilistic-effects :rewards)

(:types location)

(:predicates (vehicle-at ?loc - location)
         (spare-in ?loc - location)
         (road ?from - location ?to - location)
         (not-flattire)
         (hasspare))

(:action move-car
:parameters (?from - location ?to - location)
:precondition (and (vehicle-at ?from) (road ?from ?to) (not-flattire))
:effect (and (decrease (reward) 1) (vehicle-at ?to) (not (vehicle-at ?from))
        (probabilistic 0.5 (not (not-flattire)))))

(:action loadtire
:parameters (?loc - location)
:precondition (and (vehicle-at ?loc) (spare-in ?loc))
:effect (and (decrease (reward) 1) (hasspare) (not (spare-in ?loc))))

(:action changetire
:precondition (hasspare)
:effect (and (decrease (reward) 1) (not (hasspare)) (not-flattire)))
)

(define (problem p01)
(:domain triangle-tire)
(:objects l-1-1 l-1-2 l-1-3 l-2-1 l-2-2 l-2-3 l-3-1 l-3-2 l-3-3 - location)
(:init
        (vehicle-at l-1-1)
        (road l-1-1 l-1-2)
        (road l-1-2 l-1-3)
        (road l-1-1 l-2-1)
        (road l-1-2 l-2-2)
        (road l-2-1 l-1-2)
        (road l-2-2 l-1-3)
        (spare-in l-2-1)
        (spare-in l-2-2)
        (road l-2-1 l-3-1)
        (road l-3-1 l-2-2)
        (spare-in l-3-1)
        (spare-in l-3-1)
        (not-flattire))
(:goal (vehicle-at l-1-3)) (:goal-reward 100) (:metric maximize (reward)))
```

Figure 2.1: Example PPDDL domain and problem. This is the `triangle-tireworld` domain from the IPPC competitions. It depicts a car that lives in world with nodes distributed in a triangular grid. The car has to move from one of the nodes of the grid (in this example, `l-1-1`) to another node (here, `l-1-3`). The car has a 50% chance of getting a flat tire in each movement. It can pick spares from the nodes in which they are available, and fix flat tires if it has previously picked a spare. This is one of the domains regarded as "probabilistically interesting" by Little *et al* [29]

.

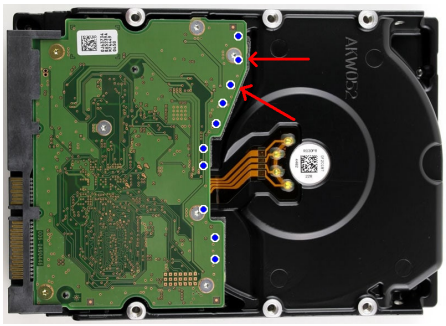


Figure 2.2: Affordances in a HDD PCB. Each of the blue circles is an affordance for a lever-up action, either for removing or loosening the PCB.

- `(current-mode ?m - mode)`: mode of the currently grasped tool. The available modes are `no-mode` (for when the robot is not grabbing any tool), `scara` and `power`.
- `(current-tool ?t - tool)`: indicates the currently equipped tool. We consider the following tools: `flat-sd` (flat screwdriver), `star-sd` (star screwdriver), `suction pads`, `pliers`, a `hammer` and a `cuter`.
- `(valid-mode ?t - tool ?m - mode)`: indicates that `?m` is a valid mode for tool `?t`. Some tools like the screwdriver and the suction pad are valid both in `scara` mode and in `power` mode. However, others like the hammer are valid only in power mode.
- `(current-side ?s - side)`: the side of the device that is currently facing upwards. We assume that the device has a prism-like shape. We consider 6 possible sides: `top`, `bottom`, `left`, `right`, `front` and `back`. Each component of the device is visible from a specific side (e.g. in a hard drive the lever and the platters are at the top, while the PCB is located at the bottom).
- `(opposite-side ?s1 ?s2 - side)`: when a particular side is facing upwards, the loosen components at the opposite face fall down. This predicates is used to mark the opposite side pairs.
- `(held)`: tells whether the device is being held between the parallel fingers. This is necessary to flip it over and to perform actions in `scara` mode.

The predicates presented here do not say anything about the configuration of the different components of the device being recycled. This is covered in the next Section.

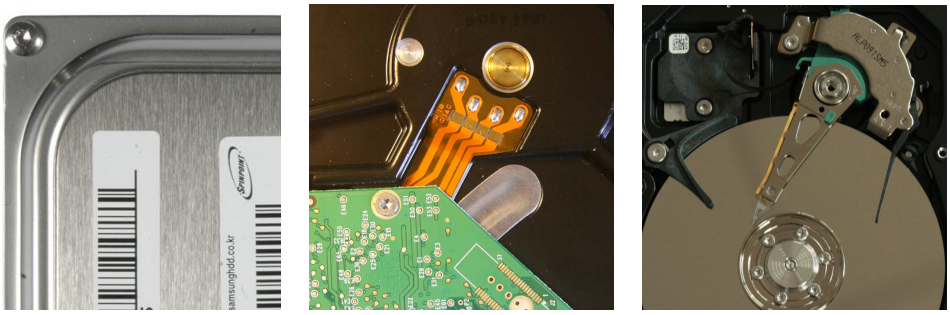
2.2.2 Device representation

While the previous list of predicates is related to the robot capabilities and its interaction with the device, we must also encode in the state the information about the device itself. That is: the detected components and affordances, the screws, connectors, partial occlusions, etc.

Below we show a list with the proposed predicates to represent a device's inner configuration. Fig. 2.3 shows three examples of usage.

- `(at-side ?c - component ?s - side)`: tells the side where a certain component is located at. This side has to be facing upwards in order to interact with the affordances of this component.
- `(broken-component ?c - removable-component)`: the component is in such a bad shape that the robot cannot use its affordances. This often results in a dead end, unless the component is loosen, in which case the device can be turned around in order to let it fall.
- `(broken-tool ?t - tool)`: the tool is broken and it can no longer be used for further manipulation.
- `(connected ?c1 ?c2 - component)`: tells that two components are connected via a cable or a flat connector.
- `(clear ?c - removable-component)`: true for components that neither are connected to anything nor have any screw fixing them.
- `(fixed-by ?c - removable-component ?s - screw)`: true for pairs of components-screws in which the screw keeps the component in place.
- `(has-affordance ?c - removable-component ?a - affordance)`: indicates that a component has associated a certain affordance.
- `(has-confidence ?a - affordance ?c - affordance-confidence)`: indicates the confidence associated to each affordance. We consider three levels of confidence: `low`, `medium` and `high`.
- `(loose ?c - removable-component)`: identifies components that are loose and may fall just by turning the device around.
- `(partially-occludes ?c1 ?c2 - component)`: marks pairs of components in which the first argument occludes the second one. This is useful to identify hierarchies.
- `(removed-non-verified ?c - removable-component)`: identifies components that have been removed, but the robot has not checked behind them to see if they were occluding another component or affordance.
- `(removed-verified ?c - removable-component)`: the component has been removed, and the robot has already checked for previously hidden components and affordances.

- `(stuck ?s - screw)`: the screw is stuck and it is more difficult to retrieve it by simply unscrewing.
- `(valid-sd ?s - screw ?sd - screwdriver)`: identifies which is the valid screwdriver for a certain screw.



(a) (b) (c)

Figure 2.3: (a) Top lid of a hard drive being fixed by a star screw. If the identifiers of the lid and the screw are `lid` and `lid-s0` respectively, then this would be represented as `(fixed-by lid lid-s0)`. (b) Connection between an HDD's PCB and the motor axis. This is represented as `(connected pcb motor-axis)`. Notice that this is a symmetric relationship, so the order of the arguments may be inverted. (c) A hard drive's R/W arm partially occluding a platter. Represented as `(partially-occludes reader platter)`. Notice that when one object occludes another, it may as well be hiding one of the affordances.

.

We consider that these predicates are observable. We assume that we have a working perception system that is able to process the scene and provide them to us, or that there is some middleware to infer them. We can also think of hidden components and relationships. These are not observed by the disassembling agent, but the simulated environment we use for experimentation (see Chapter 5) would need to take them into account since they are part of the ground truth about the device. They are needed to identify those components that are not visible (e.g. the disassembling should not be aware of the components under the HDD lid until the lid is removed).

- `(hides-component ?c1 ?c2 - component)`: tells that the second argument is not visible by the recycling robot because it is completely occluded by the first argument component.
- `(hides-affordance ?c1 - component ?a - affordance)`: tells that the second argument affordance is completely hidden by the first agument component and, therefore, it should not be included into the robot's state.

2.2.3 Actions

Finally, we will briefly comment on the actions available to the robot. We have assumed that the actions for picking and placing the device, switching tools and turning the device around when is held by the robot, are deterministic. On the other hand, we have modeled several actions for unscrewing, bashing and retrieve the components with specialized tools that have probabilistic effects (e.g. successful retrieval of the object, tool break, component break).

Namely, we have included the following actions in the domain:

- `pick-tool`, `put-away-tool`, `grab-device`, `place-device`, `flip`: all of these are deterministic. We believe their names to be self-explanatory.
- `let-fall-down`: also deterministic. Produces that all the loosened components fall down by virtue of turning the device around.
- `cut-connector`: deterministic action for removing the `(connected ?c1 ?c2)` predicate.
- four `unscrew` actions. The probability of successfully unscrewing a screw depends on the grasp mode and whether the screw is stuck or not (hence the 4 actions).
- `bash` action for quickly remove all the screws that are fixing some component, and loosening it. However, it has a low success probability and may break the bashed component. This action only works with the hammer tool.
- six `lever` actions (all the combinations of modes and affordance confidence) for levering a device, each with different success/dead-end probabilities.
- 3 `suck-away` actions, each for a different level of confidence (the mode does not matter here). This action has a very low dead-end probability, but also the success rate (even for highly trusted affordances) is low.
- 3 `extract-with-pliers` actions. The pliers are available only with the power mode, so the 3 actions correspond to the three levels of confidence of the affordance. This action has a very high chance of succeeding, but it also has more risk of breaking the device than the lever action(s).

The complete PPDDL domain specification can be found in Annex A

CHAPTER 3
DETERMINIZATION OF STOCHASTIC PROBLEMS

3.1 Rationale

One may reasonably wonder why would we bother on elaborating a probabilistic version of the problem we want to solve when we are going to determinize it anyway. The idea of constructing a deterministic version of the problem aims at making it easily solvable with a classic planner such as Fast Forward or Fast Downward. In order to select an action, the decision maker invokes a planner with the deterministic version of the problem and the most recent perceived state. It then processes the plan appropriately and decides the most suitable action for the current state (typically, the first action of the plan).

Two of the most straightforward transformations presented here are those implemented by FF-Replan [15]: Single-Outcome Determinization and All-Outcome Determinization. These make little to no use of the probabilistic information when building the deterministic version, and depend entirely on the capability of the planner to find another solution (replan) when there is a deviation from the expected outcome. This strategy, however, fails completely in domains where there is a high probability of falling into a dead end. These are the kind of domains regarded as "probabilistically interesting" by Little *et al* [29].

However, it is possible to translate the probability of the outcomes into some feature of deterministic domains that hints the planner to give preference to the most likely ways of succeeding. The α-Cost-Transition-Likelihood [16] Determinization that we will present here does so via translating the transition probabilities into deterministic costs.

Another possibility is to use the deterministic planner as a subroutine of a method that invokes it repeatedly. Hindsight Optimization [17, 18] does so via outcome sampling and repeated invocations to a deterministic plan to calculate an upper bound to the Q functions (the state-action utility) in the

current state.

However, all the methods presented in this chapter require (or, at the very least, work more easily) with expanded probability outcomes. We will elaborate on that in the next Section. Notice that these techniques are quire general, so we will resort to examples from other domains to illustrate more compactly some of this methods.

3.2 Expansion of probabilistic outcomes

Typically, probabilistic effects in PPDDL are more easily expressed in *factored form*. This means that they are either nested (Fig. 3.1a) or together in a conjunction (Fig. 3.1b).

```
(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))
(decrease (reward) 1))
```

(a)

```
(and
 (probabilistic
  0.25 (and (loose ?comp)
       (when (loose ?comp)
        (removed-non-verified ?comp)))
  0.10 (removed-non-verified ?comp)
  0.12 (broken-component ?comp))
 (probabilistic
  0.12 (broken-tool flat-sd))
 (decrease (reward) 1))
```

(b)

Figure 3.1: (a) Nested probabilistic effects. This example is part of the effect of the `bash` action of the recycling domain (Annex A). (b) Conjunction of probabilistic effects. This example is part of the effect of the `lever-scara-medium-confidence` of the recycling domain (Annex A).

However, all the techniques presented in this chapter are based on introducing one new action per outcome (except Single-Outcome, which picks one). Therefore, it is much more convenient to have the probabilistic effects in expanded form, similarly to NID rules [10].

One could demand the domain designer to write the probabilistic effects already in expanded form. However, that sacrifices the compactness of factored probabilistic effects, and makes the domain design task more tedious and error-prone.

Fortunately, it is straightforward enough to implement an algorithm that performs automatic expansion of probabilistic effects. While it is true that the number of outcomes grows exponentially with the nesting depth and with the number of probabilistic effects in a conjunction, nested effects are typically shallow, and the number of probabilistic effects in conjunctions is usually low.

In Fig. 3.2 we show the result of applying our expansion algorithm to the effects shown in Fig. 3.1.

```
(probabilistic
  0.125 (and
    (forall (?screw - screw)
     (not (fixed-by ?comp ?screw)))
    (forall (?screw - screw ?side_ -
        side)
     (not (at-side ?screw ?side_)))
    (loose ?comp)
    (decrease (reward) 1))
  0.125 (and
    (forall (?screw - screw)
     (not (fixed-by ?comp ?screw)))
    (forall (?screw - screw ?side_ -
        side)
     (not (at-side ?screw ?side_)))
    (decrease (reward) 1))
  0.1 (and
       (broken-component ?comp)
       (decrease (reward) 1))
  0.65 (decrease (reward) 1))
```

(a)

```
(probabilistic
  0.03 (and
              (loose ?comp)
              (when (loose ?comp)
               (removed-non-verified ?
                   comp))
              (broken-tool flat-sd)
              (decrease (reward) 1))
  0.22 (and
         (loose ?comp)
         (when (loose ?comp)
          (removed-non-verified ?comp))
         (decrease (reward) 1))
  0.012 (and
          (removed-non-verified ?comp)
          (broken-tool flat-sd)
          (decrease (reward) 1))
  0.088 (and
          (removed-non-verified ?comp)
          (decrease (reward) 1))
  0.0144 (and
           (broken-component ?comp)
           (broken-tool flat-sd)
           (decrease (reward) 1))
  0.1056 (and
                   (broken-component ?
                       comp)
                   (decrease (reward) 1)
                       )
  0.0636 (and
          (broken-tool flat-sd)
          (decrease (reward) 1))
  0.4664 (decrease (reward) 1))
```

(b)

Figure 3.2: Notice that expanded effects are much more verbose than the original factored form. However, it is much easier to identify the outcomes and their probability. (a) Effect of `bash` after expansion. (b) Effect of `lever-scara-medium-confidence` after expansion.

3.3 All-Outcome Determinization

All-Outcome Determinization consists in building a new domain with one action for each of the outcomes of the original probabilistic domain, discarding completely the probabilities of such outcomes. In addition, all the increments and decrements of rewards are removed. Empty effects result in useless actions, so those can be safely omitted from the deterministic domain. The precondition of the newly introduced actions is the same as that of the original action.

Doing this, we allow the planner to freely choose the outcome that best suits its need for reaching the goal. Whenever a planner finds a solution in the determinized version of the problem, this solution is guaranteed to have a non-zero probability of reaching the goal. In several domains where there are no dead ends or it is unlikely to fall into one, this strategy is good enough.

All-Outcome Determinization produces $n \cdot m$ actions, where n is the number of actions in the original domain and m is the (average) number of outcomes per action.

Let us consider an example. In the recycling domain, all the stochastic actions have more than 4 outcomes, so they are somewhat cumbersome to serve as an illustration for All-Outcome method. Therefore, let us consider the `move-car` action from the Triangle Tireworld (the full domain was given in Fig. 2.1 as an example).

After applying All-Outcome Determinization to the domain, the actions derived from the original `move-car` would look as in Fig. 3.3. This already gives us a hint of one of the main pitfalls of the method: since the planner makes the conscious choice of one outcome over the others, it will never select a harmful outcome. However, harmful outcomes may well occur in a real environment. For this very reason, All-Outcome Determinization is often regarded as "too optimistic".

```
(:action move-car
    :parameters
      (?from - location ?to - location)
    :precondition (and
                    (vehicle-at ?from)
                    (road ?from ?to)
                    (not-flattire))
    :effect (and
              (decrease (reward) 1)
              (vehicle-at ?to)
              (not (vehicle-at ?from))
              (probabilistic 0.5 (not (
                  not-flattire)))))
```

(a)

```
(:action move-car_o0
  :parameters
    (?from ?to - location)
  :precondition (and
                  (vehicle-at ?from)
                  (road ?from ?to)
                  (not-flattire))
  :effect (and
            (vehicle-at ?to)
            (not (vehicle-at ?from))
            (not (not-flattire))))

(:action move-car_o1
  :parameters
    (?from ?to - location)
  :precondition (and
                  (vehicle-at ?from)
                  (road ?from ?to)
                  (not-flattire))
  :effect (and
            (vehicle-at ?to)
            (not (vehicle-at ?from))))
```

(b)

Figure 3.3: (a) Original `move-car` action. Even though it is not in expanded form, it is easy to see that it has only two outcomes: one in which the car moves and gets a flat tire, and another in which the tire is preserved. (b) Deterministic actions derived from the All-Outcome Determinization of `move-car`. Notice that there are 2 actions in total, one for each of the non-empty outcomes of the original action.

3.4 Single-Outcome Determinization

For each action, Single-Outcome Determinization picks only one of the outcomes. The criterion for selecting the outcome varies, although the most frequent choice is to select the most likely one. Other selection criterion is to pick the outcome that makes more predicates true in the state. Much like in `All-Outcome` determinization, rewards are removed from the state.

Since Single-Outcome Determinization commits to a single outcome per action, it has the advantage of resulting in a reduced number of actions with respect to All-Outcome Determinization. On the one hand, the problem is less complex. The impact of this can be notable in problems with many outcomes (like the recycling domain). On the other hand, it potentially throws away outcomes that are necessary to reach the objective. The most immediate consequence of this is that problems that are solvable may become unsolvable under this transformation.

Contrarily to All-Outcome Determinization, the Single-Outcome alternative can use a bit of probabilistic information to construct the deterministic domain, depending on the outcome selection criterion (e.g. this is evident for the most-likely outcome selection criterion).

In Fig. 3.4 we can see an example of Single-Outcome Determinization for the `extract-with-pliers-high-confidence` action of the recycling domain.

```
(:action
    extract-with-pliers-high-confidence
  :parameters
    (?comp - removable-component
     ?pp - pliers-point ?side - side)
  :precondition
    (and
        (not (broken-tool pliers))
        (not (broken-component ?comp))
        (at-side ?comp ?side)
        (has-affordance ?comp ?pp)
        (has-confidence ?pp high)
        (current-side ?side)
        (current-tool pliers)
        (clear ?comp))
  :effect (probabilistic
   0.0425 (and
            (removed-non-verified ?comp)
            (broken-tool pliers)
            (decrease (reward) 1))
   0.8075 (and
            (removed-non-verified ?comp)
            (decrease (reward) 1))
   0.0075 (and
            (broken-component ?comp)
            (broken-tool pliers) (
                decrease (reward) 1))
   0.1425 (and
            (broken-component ?comp)
            (decrease (reward) 1))))
```

(a)

```
(:action
    extract-with-pliers-high-confidence
  :parameters
    (?comp - removable-component
    ?pp - pliers-point ?side - side)
  :precondition
    (and
        (not (broken-tool pliers))
        (not (broken-component ?comp))
        (at-side ?comp ?side)
        (has-affordance ?comp ?pp)
        (has-confidence ?pp high)
        (current-side ?side)
        (current-tool pliers)
        (clear ?comp))
  :effect (and
            (removed-non-verified ?comp)
            (decrease (reward) 1)))
```

(b)

Figure 3.4: (a) Original `extract-with-pliers-high-confidence` action presented in expanded form. (b) Single-Outcome Determinization based on the most-likely-outcome criterion. Even though there are 4 outcomes, only one of them is retained.

3.5 α-Cost-Transition-Likelihood Determinization

α-Cost-Transition-Likelihood Determinization tries to give preference to plans that are more likely to succeed. It is very similar to All-Outcome Determinization in that it creates an action per outcome. However, it does not discard the rewards completely. Instead, it reformulates them as action costs (decrements are transformed to increments, and all the references to the `reward` fluent ar substituted by a `total-cost` fluent). This is akin to the definition of SSPP presented in Chapter 2.

The costs are transformed according to the following expression:

$$C_{new}(s, a, s') = \alpha C_{old}(s, a, s') - \log(\mathbb{P}(s'|s, a))$$

, where α is a parameter that modulates the transition cost.

We can see that this new cost depends on the previous cost and the transition likelihood. The lower the probability, the higher the cost. The α parameter can give more or less relevance to the cost term. $\alpha \to 0$ means that the new cost depends exclusively on the likelihood of the transitions (i.e. the less likely to occur, the higher the cost). Conversely, $\alpha \to \infty$ gives more relevance to the transition cost. If $C_{old}(s, a, s') = 1$, the accumulated cost represents the plan length corrected by the success probability of the plan.

If the planner optimizes over the accumulated cost, we can see that α establishes a trade-off between potentially lengthy but safer plans and short but risky plans. We present a few examples to illustrate this point.

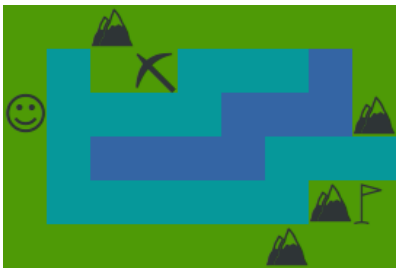


Figure 3.5: Example grid. The green cells depict grass, the cyan cells are shallow water and the blue cells are deep water. The pickaxe, the character, the boulders and the objective are also represented with self-explanatory icons.

The "terrain" example: Let us imagine a rectangular grid. Some of the cells are grass, others shallow water and other deep water. There is a character who starts in a given position of the grid. He has to travel to a goal cell. He can travel through grass with no risk. He can swim through shallow water with little risk (a 5% chance of drowning). He can also swim through deep water with a greater risk (a 20% chance of drowning). To add to the complexity of the problem, some grass cells contain boulders. The character cannot go through these cells. However, some cells may contain a pickaxe that the character can pick and use to break these boulders. Fig. 3.5 shows an example instance of the problem. We want to show the risk/speed trade-off of α-Cost-Transition-Likelihood through this understandable example. The problem is easily enough modeled in PPDDL (shown in Annex B). Fig. 3.6 shows several solutions to this same instance for different values for the α parameter.

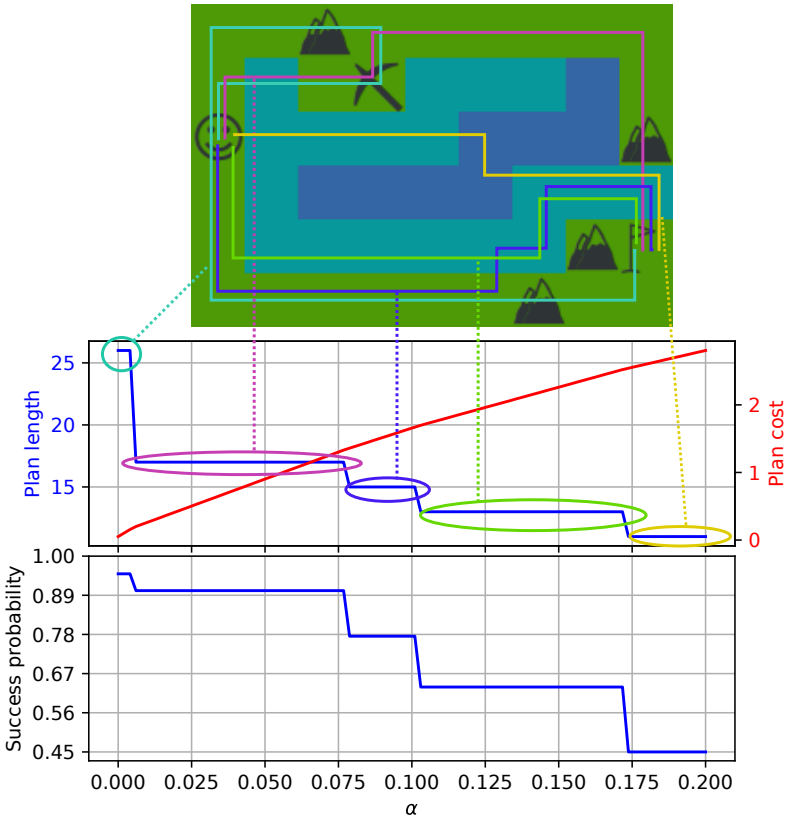


Figure 3.6: Solutions to an instance of the "terrain" problem using α-Cost-Transition-Likelihood Determinization with several values of α.

Clearly, greater αs result in more direct but riskier paths to the goal. We can see that with $\alpha > 0.175$ the character happily accepts swimming through deep water. On the other hand, low α values are associated to longer paths and safer plans. With $\alpha = 0$ the risk is minimal, since the character puts himself only in little danger to get the pickaxe through the shallow water. From that point on, he slowly breaks his way to the target cell and going only through grass.

Example with Imagine domain: We also present an example of the application of $\alpha - Cost - Transition - Likelihood$ Determinization in the recycling domain. We consider a very simple problem in which the robot has to retrieve the PCB of a hard drive. The PCB is located at the bottom, with four screws keeping it in place. Moreover, there is a connector from the PCB to the motor axis. The PCB can be extracted either via levering using the pliers, or bashing it with the hammer to loosen it and then letting it fall. The hard drive is initially resting before the robot with its top side facing upwards. Fig. 3.7 shows that the effect of the α parameter is the same as in the "terrain" example.

Small values lead to long but safe actions, while bigger values result in riskier sequence. Notice, for instance, that the 4th plan uses the `bash` action (the riskiest action in the repertoire) to quickly get rid of the 4 screws that keep the PCB in place, and then the pliers to remove it from the drive.

Triangle Tireworld, an unfriendly domain: Up to this point, α-Cost-Transition-Likelihood has worked as expected. However, it is still somewhat vulnerable to domains with dead ends that are difficult to avoid. This is the case of the Triangle Tireworld domain. Let us remember that the goal is to reach the target node and avoiding getting stuck as result of a flat tire. The probability of getting a flat tire after moving is 50%. It can be replaced if the car has a spare. Spares are distributed among the different cells.

In this case, the α parameter has no influence and the planner has no reason to give preference to neither of the two outcomes because both have the same probability and the same transition cost. In fact, the planner will always suggest the shortest route, even though this may not be the best choice. This is clear in Fig. 3.8. While the safest route is going through the top of the triangle (the `l-2-1` $\rightarrow$ `l-3-1` $\rightarrow$ `l-2-2` $\rightarrow$ `l-1-3` path), for the planner this plan has the same chances of succeeding as the route that goes in a straight line from `l-1-1` to `l-1-3`.

This leads us to an interesting observation: even if the plans obtained by a planner that operates in a determinized domain with α parameterized costs are optimal, these do not induce the best policy. That is, repeatedly obtaining the "optimum" plan (according to the α-Cost-Transition-Likelihood criterion) and executing the first action is not equivalent to executing the optimum policy. The reason is somewhat subtle: even if two plans have the same probability of succeeding and the same accumulated cost, the consequences of failing one of the plans are not necessarily the same as the consequences of failing the second one. Turning back to the Tireworld domain, we can see that if we get a flat tire going through the top of the triangle, we can always pick one of the spares that are distributed along the way and fix the tire. However, this is not the case in the shortest route. In other words, the top route has more tolerance to failure than the straight one.

Notice that we could easily manipulate the domain to make $\alpha - Cost - Transition - Likelihood$ work effectively in this particular case: we can alter the probabilities so that getting a flat tire is more likely than moving without hazard. Then, the cheapest plan with $\alpha = 0$ is the one that always forces a flat tire at each movement, picks a new tire, and proceeds to the next cell. Alternatively, we could artificially increase the transition cost of the outcome of the move action in which the car does not get a flat tire.

This suggests a general approach to improve the robustness of α-Cost-Transition-Likelihood against dead-ends: if we are able to identify the most "beneficial" or "desirable" outcomes, we can artificially increment their transi-

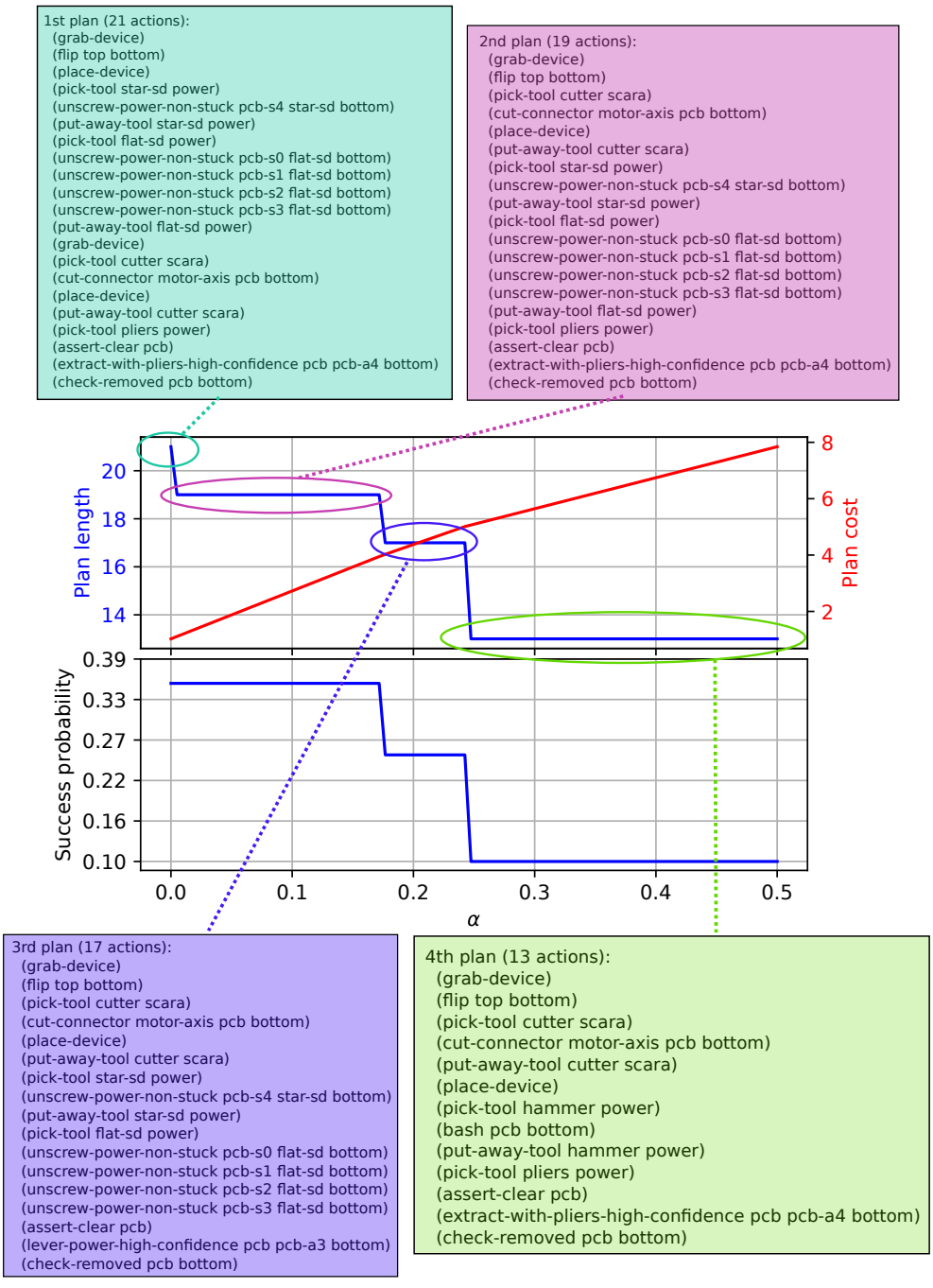

1st plan (21 actions):
(grab-device)
(flip top bottom)
(place-device)
(pick-tool star-sd power)
(unscrew-power-non-stuck pcb-s4 star-sd bottom)
(put-away-tool star-sd power)
(pick-tool flat-sd power)
(unscrew-power-non-stuck pcb-s0 flat-sd bottom)
(unscrew-power-non-stuck pcb-s1 flat-sd bottom)
(unscrew-power-non-stuck pcb-s2 flat-sd bottom)
(unscrew-power-non-stuck pcb-s3 flat-sd bottom)
(put-away-tool flat-sd power)
(grab-device)
(pick-tool cutter scara)
(cut-connector motor-axis pcb bottom)
(place-device)
(put-away-tool cutter scara)
(pick-tool pliers power)
(assert-clear pcb)
(extract-with-pliers-high-confidence pcb pcb-a4 bottom)
(check-removed pcb bottom)
2nd plan (19 actions):
(grab-device)
(flip top bottom)
(pick-tool cutter scara)
(cut-connector motor-axis pcb bottom)
(place-device)
(put-away-tool cutter scara)
(pick-tool star-sd power)
(unscrew-power-non-stuck pcb-s4 star-sd bottom)
(put-away-tool star-sd power)
(pick-tool flat-sd power)
(unscrew-power-non-stuck pcb-s0 flat-sd bottom)
(unscrew-power-non-stuck pcb-s1 flat-sd bottom)
(unscrew-power-non-stuck pcb-s2 flat-sd bottom)
(unscrew-power-non-stuck pcb-s3 flat-sd bottom)
(put-away-tool flat-sd power)
(pick-tool pliers power)
(assert-clear pcb)
(extract-with-pliers-high-confidence pcb pcb-a4 bottom)
(check-removed pcb bottom)
Plan length
Plan cost
20
18
16
14
8
6
4
2
Success probability
0.39
0.33
0.27
0.22
0.16
0.10
0.0
0.1
0.2
0.3
0.4
0.5
α
3rd plan (17 actions):
(grab-device)
(flip top bottom)
(pick-tool cutter scara)
(cut-connector motor-axis pcb bottom)
(place-device)
(put-away-tool cutter scara)
(pick-tool star-sd power)
(unscrew-power-non-stuck pcb-s4 star-sd bottom)
(put-away-tool star-sd power)
(pick-tool flat-sd power)
(unscrew-power-non-stuck pcb-s0 flat-sd bottom)
(unscrew-power-non-stuck pcb-s1 flat-sd bottom)
(unscrew-power-non-stuck pcb-s2 flat-sd bottom)
(unscrew-power-non-stuck pcb-s3 flat-sd bottom)
(assert-clear pcb)
(lever-power-high-confidence pcb pcb-a3 bottom)
(check-removed pcb bottom)
4th plan (13 actions):
(grab-device)
(flip top bottom)
(pick-tool cutter scara)
(cut-connector motor-axis pcb bottom)
(put-away-tool cutter scara)
(place-device)
(pick-tool hammer power)
(bash pcb bottom)
(put-away-tool hammer power)
(pick-tool pliers power)
(assert-clear pcb)
(extract-with-pliers-high-confidence pcb pcb-a4 bottom)
(check-removed pcb bottom)

Figure 3.7

.

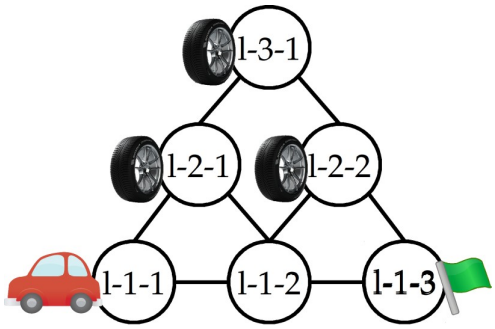


Figure 3.8

tion cost so the planner does not get too optimistic. However, automatically identifying such outcomes is not a trivial task in general.

3.6 Hindsight Determinization

The last determinization method that we consider is not only a modification of the domain, but also an algorithm to calculate a lower bound of the state-action utility (the same Q function that is used in many Reinforcement Learning algorithms) via repeated calls to a deterministic planner.

Each time the action maker has to decide the next action, it calculates an upper bound of the $Q(s, a)$ for the current state and all the available action (or an intelligently chosen subset of all the available actions). Let us call this upper bound $\hat{Q}(s, a)$. $\hat{Q}(s, a)$ is assumed to preserve the same ranking as the true Q function, and is calculated averaging the negative length of several different plans traces that start with that action. The longer the plans, the lower the $\hat{Q}$ score of that action for the given state. To obtain different plans, the outcome of each action is decided in *hindsight* for each time step. For instance, for a given action with 5 outcomes, the planner may decide that if the action is executed at time step 3, it will yield the fourth outcome. In order to fix the outcomes in advance, they are sampled for each time step. The assignment of an outcome for each action at each time step is called *future*. If for a given action and future no plan is found, then a penalty is applied to the $\hat{Q}$ value of that action in order to make it less appealing.

The method that we have outlined here has several advantages over the other determinization methods: it makes the most of the probabilistic information about the outcomes; it invokes a planner several times in order to base the decision on multiple possibilities; and it is very robust against dead ends. However, it also makes an intensive use of computational resources. Some improvements have been suggested to reduce this intensive computation. One of them is computing the $\hat{Q}$ value of only a reduced set of actions. The set of actions is called *Probabilistically Helpful Actions*. One way to compute such set

is to sample several futures, plan for all of them, and retrieve the first action of each one. Another improvement is to identify prefixes that are common in all the plans that start with the selected action. These are usually deterministic actions and can be executed in sequence without resorting to the Hindsight Optimization method for the intermediate states.

The original proposal only sets the outcomes of the actions up to a certain horizon, so one must have an idea of the expected length of the plans before setting this horizon. The authors of the method have modified the implementation of Fast Forward to incorporate Hindsight Optimization using non-stationary action expansion. We propose two method for performing Hindsight Optimization with out-of-the-box planners, using exclusively PDDL features, and that does not fix the horizon of planning. More specifically, we use a *wheel of outcomes*, as shown inf Fig. 3.9.

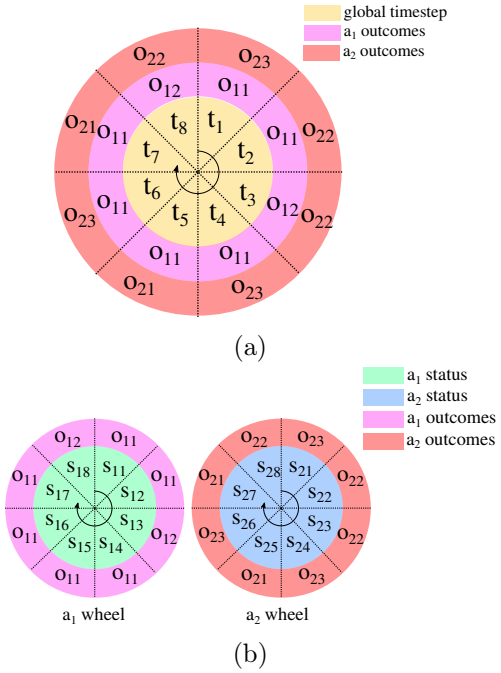


Figure 3.9: (a) Global time step. Each action as an outcome associated to each time step. The execution of any action makes the time step advance, potentially changing the available outcome for the rest of the actions. (b) Local action status. Each action has has associated an outcome for each of its internal statuses. The execution of an action makes the internal status advance, but it will not change the available outcome for the rest of the actions.

The first method is the closest to the author's proposal. It is depicted in Fig.3.9a. The outcome that each action will produce depends on the global time step. However, time steps cycle as in a wheel, so the planner is not constrained to plans that have lengths smaller than the horizon. What is more important, this can be achieved using just PDDL:

- We introduce a `(current-timestep ?t)`.
- Time steps are linked via several `(next-timestep ?t1 ?t2)` predicates. These define the order of the time steps. The "last" time step is succeeded by the "first" one.
- A predicate `(applicable-ai-oj ?t)` for each action a_i and outcome a_j. This predicate tells, for every action, which outcome should be triggered at each time step.

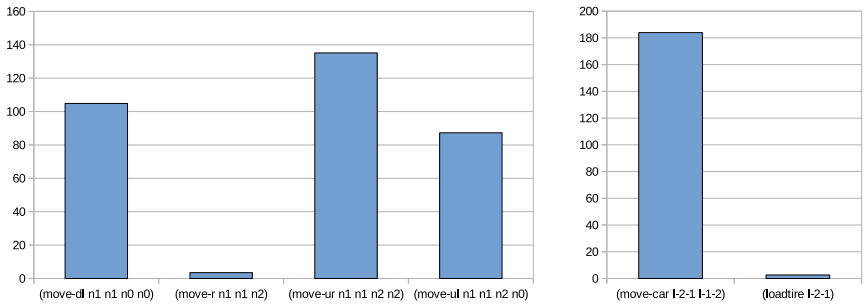
Then, the domain is determinized in an All-Outcome fashion, but modifying the preconditions so that an action outcome can be executed only when is applicable in the current time step. The effects are also modified to make the time step advance. Since each action increments the time step, the execution of an action alter the active outcome of the rest of the actions.

The second method is similar, but it maintains a local status for each action rather than a global one. The execution of an action modifies exclusively its own status, so the planner is forced to trigger bad outcomes if it wants more beneficial effects from the same action. The method is depicted in Fig. 3.9b.

Fig. 3.10 gives an idea of the power of the hindsight optimization method for discriminating good actions from bad ones. It shows the negative $\hat{Q}$ value obtained from two states of the `rectangle-tireworld` and the `triangle-tireworld` domains (therefore, the lower the value, the more appealing the action is). We can observe that the issue that was present with α-Cost-Transition-Likelihood has been mitigated.

In the first case (Fig. 3.10a), the most promising action (and by a great margin) is `(move-r n1 n1 n2)`. This makes sense when we understand the dynamics of the rectangle tireworld domain: the car moves in a rectangular grid and can perform diagonal moves, that allow to reach the goal much quicker, but sacrificing safety. The other three actions correspond to diagonal movements.

In the second case (Fig. 3.10b), the most promising action is to pick a spare from the current position. This makes sense. Let us remind that in this domain each displacement action has associated a risk of getting a flat tire, so it is better to be prepared for the next displacement, even if the car is in good state.



(a) (b)

Figure 3.10: (Negative) score of the assessed actions in a state of (a) the rectangle tireworld domain and (b) the rectangle tireworld domain.

CHAPTER 4

SHORT HORIZON PLANNING

4.1 Rationale

The methods presented in the previous Chapter work with general domains. However, we can take advantage of the structure of our particular problem. The key observation is that the robot can focus on retrieving one of the components, forgetting about the unrelated ones. That is, the planning problem can be simplified in the presence of precedence relationships.

The main source of precedence in our case is partial occlusion. If a component occludes another, it is a good candidate to be removed first. Even if it is possible to remove an occluded component, removing first the occluding object can lead to the discovery of a better affordance than those that have already been detected. Therefore, a good strategy to gain in computational efficiency and to increase the chances of success is to perform topological sorting based on precedence relations, and disassemble first the components with no precedences. The predicates that are not directly related to this component can be removed from the state, easing the task for the planner. That is, we want to enable the robot to focus its attention on a particular component of the device. In this Chapter we consider the problem in its full form: the device is composed of components that may hide each other. While some precedence relations are observable (e.g. `(partially-occludes ...)`), others are not (e.g. a hidden screw fixing a component on the other side). It is our intention to deal successfully with both kind of precedences.

4.2 Simplifying the state

Let us consider an hypothetical hard drive scenario in which the lid and the PCB have already been removed. The only remaining components left to extract are the reader and the platter. The reader is fixed by a screw on the

other side, and at the same time is occluding part of the platter. It turns out that the reader is hiding a high quality suction point. This scenario has been represented as a graph in Fig. 4.1. This kind of representation is useful to represent graphically the distribution of the components, and to operate with its topological properties. Of course, the robot would not see the hidden features, so it would be restricted to the view of Fig. 4.1b.

Now, let us see that the robot could remove the platter much more efficiently if the reader was out of the way. However, when a planner is fed with this state, it may determine that it is better to remove the platter first (let us remember that the planner operates under the assumption of full observability). However, we know that removing first the reader can potentially lead to a much better scenario.

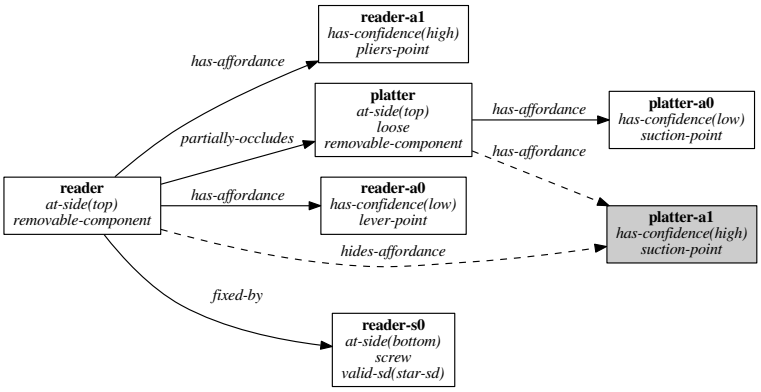
The strategy that we follow is to first identify the components without partial occlusions. We select one of these components for removal (we show in the next Section the criterion used to select the component). Then we use breadth first search to identify the closest neighbors, since these are the ones that are more prone to be relevant in the disassembly of the chosen component. When invoking breadth first search, we do not navigate through the `partially-occluded` edges in order to avoid the accidental inclusion of the components that we want to avoid. The result of doing this in the state of Fig. 4.1b can be shown in Fig. 4.1c. This is, in fact, a form of hierarchical planning. We sort the tasks according to their precedences and solve them in order, without caring about the rest.

4.3 Dealing with unseen precedences

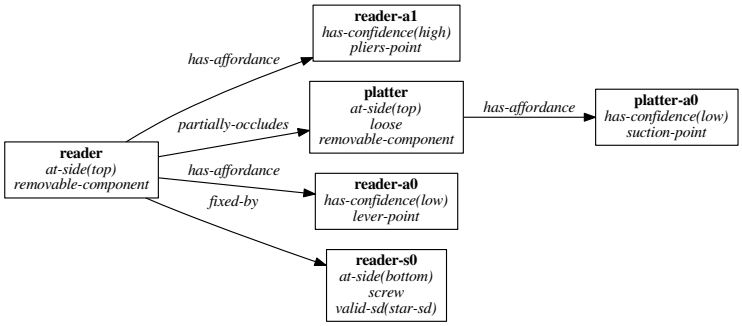
In the previous section we dealt with observable precedences. However, we can expect to encounter situations such as the one from Fig. 4.2. In this particular scenario, the task that the robot has chosen to complete is impossible. This is because it cannot possibly know that there is a hidden screw on the other side of the hard disk keeping the reader in place. However, after a few trials, we can reasonably assume that this task should be tried later.

When choosing one task to complete from the set of tasks with no visible precedences and with no previous experience, the robot may choose any of them. However, if it fails many times during the completion of the task, it can assume that it has picked the wrong component to disassemble, and that it should try another task. The robot can return the component back to the goal stack, and select another one.

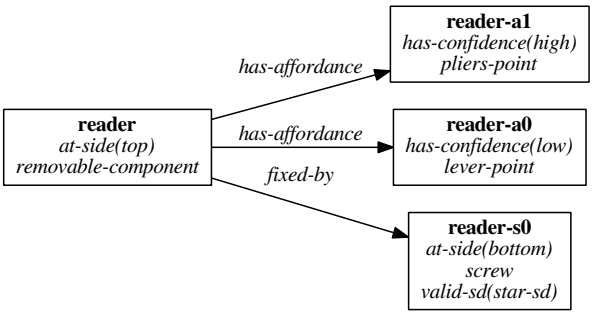
We have implemented this feature giving a score to each component. Whenever the robot repeats the same action several times with no success, we decrease the score of the component and pick another available goal with a higher score.



(a)

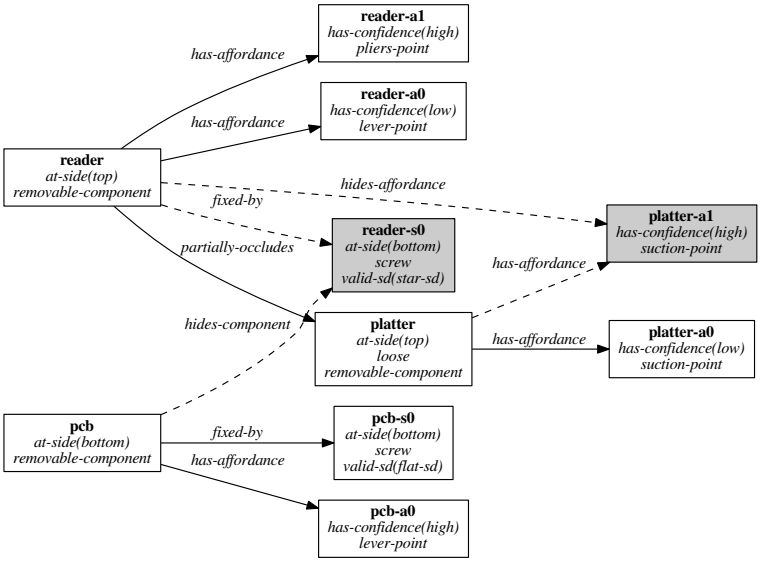


(b)

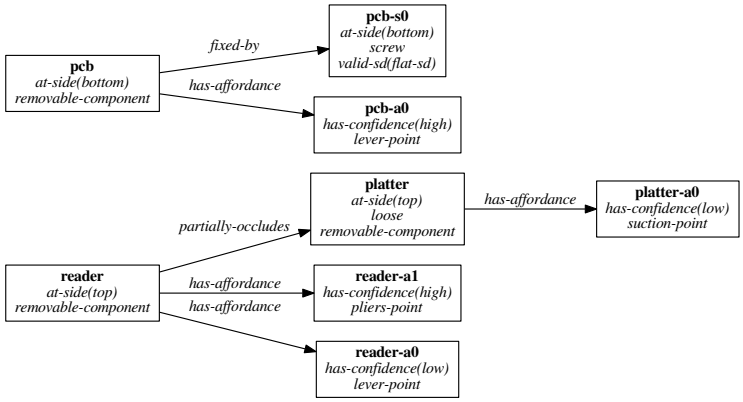


(c)

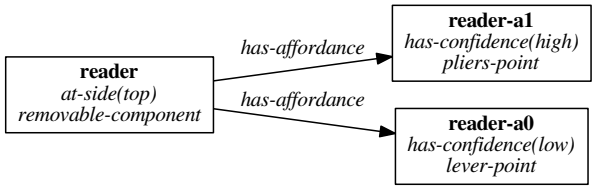
Figure 4.1: (a) Simple hard drive scenario. The hidden nodes are shaded in gray, and the hidden relations are represented as dashed edges. (b) The state that the robot would actually see (i.e. without the hidden nodes/edges). (c) Simplified state after removing the partial occlusions: the platter and its only detected affordance have been removed. The robot can use this simplified state to focus on the removal of the reader.



(a)



(b)



(c)

Figure 4.2: (a) Complex distribution with one hidden precedence: there is one hidden screw below the PCB blocking the reader. (b) The state that the robot would actually see (i.e. without the hidden nodes/edges). Note that, in particular, the reader seems to have no precedence. (c) Simplified state after removing the partial occlusions, focused on the reader. The robot will never succeed removing the it.

CHAPTER 5
EXPERIMENTAL EVALUATION

5.1 Implementation

For the purpose of experimenting with the ideas presented here, we have implemented a planning toolbox in the Python language. Our toolbox contains a (P)PDDL parser and data structures to represent internally the domains and problems as Python classes and manipulate them. We have also implemented a PPDDL simulator that reads problem descriptions and simulates the probabilistic effects of actions. This simulator has been employed to obtain the results that are presented in this section. Our implementation is available in a public Github repositoy[1].

Our tools use the Fast Forward and Fast Downward planners to solve PDDL problems.

On the one hand, we want to perform a quantitative assessment of the determinization techniques described in Chapter 3. To do so, we resort to several problems from the IPPC 2018 (some of them already introduced in the previous sections). We also include some problems from our "terrain" domain (used to illustrate the α-Cost-Transition-Likelihood method) and instances of our own Imagine domain, described in Chapter 2 and shown in Annex A.

On the one hand, we assess qualitatively the techniques for simplifying and limiting the planning horizon. We use a modified version of our simulator to perform these experiments. The modifications consist in removing all the predicates that should not be available to the robot due to occlusions.

5.2 Experiments with determinization techniques

We have considered the following planner configurations:

[1] `https://github.com/sprkrd/planning_tools`

- **ml-ff:** Single-Outcome determinizer that selects the most likely outcome. Uses Fast Forward as a back-up planner.
- **ma-ff:** Same as the previous one, but selects the outcome that sets more predicates to true.
- **ao-ff:** All-Outcome determinizer. Uses Fast Forward as a back-up planner.
- **alph-1.0:** α-Cost-Transition-Likelihood Determinizer with $\alpha = 1$. Uses Fast Downward as a back-up planner, configured to run A* search with the CEA (Context Enhanced Additive) heuristic.
- **alph-0.2:** Same as previous but with $\alpha = 0.2$.
- **alph-0.1:** Same as previous but with $\alpha = 0.1$.
- **alph-0.0:** Same as previous but with $\alpha = 0.0$.
- **ho-lo-30:** Hindsight Optimizer with a wheel of size 30 and local state counter. Uses Fast Forward as a back-up planner.
- **ho-lo-15:** Same as previous but a wheel of size 15.
- **ho-gl-30:** Hindsight Optimizer with a wheel of size 30 and a global time step. Uses Fast Forward as a back-up planner.
- **ho-lo-15:** Same as previous but a wheel of size 15.

The experiments have been conducted as follows: for each domain we have considered several problem instances and run all the determinizing agents executing actions against the simulator. Every agent tries to solve 10 episodes of each problem. The simulator accepts one action per step and then returns the state that results from the execution of that action. All the episodes have a timeout of 180 seconds (3 minutes). We have compiled the following statistics:

- **S**: number of successful episodes.
- **C**: average cost (only for the successful episodes).
- **E**: time spent per episode.

The results are shown in Table 5.1 for the IPPC 2008 problems. The results for the Imagine and "terrain" domains are shown in Table 5.2.

One of our first observations is that there are barely results for the Hindsight Optimization techniques. The reason is that our implementation takes too much time to select an action, and the time out almost always triggered canceling the execution. Note that we do have results for the two tireworld problems, and that they perform better than most of the remaining planners.

Table 5.1: Results of the experiments with the IPPC2008 domains

Problem	**ex-blocksworld**			**zenotravel**		
Determinizer	**S**	**C**	**E**	**S**	**C**	**E**
ml-ff	34/50 (68%)	9.35	0.02	20/20 (100%)	66.60	0.02
ma-ff	10/50 (20%)	5.00	0.03	20/20 (100%)	48.35	0.02
ao-ff	31/50 (62%)	9.06	0.02	20/20 (100%)	61.10	0.02
alph-1.0	29/50 (58%)	8.72	0.14	20/20 (100%)	59.90	0.15
alph-0.2	28/50 (56%)	8.07	0.15	20/20 (100%)	57.65	0.14
alph-0.1	34/50 (68%)	9.06	0.14	20/20 (100%)	70.35	0.15
alph-0.0	33/50 (66%)	8.76	0.13	20/20 (100%)	36.85	0.14
ho-lo-30	-	-	-	-	-	-
ho-lo-15	-	-	-	-	-	-
ho-gl-30	-	-	-	-	-	-
ho-gl-15	-	-	-	-	-	-
Problem	**triangle-tw**			**rectangle-tw**		
Determinizer	**S**	**C**	**E**	**S**	**C**	**E**
ml-ff	50/50 (100%)	21.28	0.10	11/30 (36.67%)	6.27	0.01
ma-ff	50/50 (100%)	21.04	0.09	16/30 (53.33%)	4.94	0.01
ao-ff	8/50 (16%)	2.25	0.01	11/30 (36.67%)	6.09	0.01
alph-1.0	6/50 (12%)	3.33	0.08	13/30 (43.33%)	4.46	0.12
alph-0.2	6/50 (12%)	2.67	0.07	9/30 (30%)	4.22	0.11
alph-0.1	6/50 (12%)	2.33	0.08	9/30 (30%)	4.67	0.12
alph-0.0	7/50 (14%)	2.29	0.07	15/30 (50%)	4.53	0.12
ho-lo-30	49/50 (98%)	20.78	20.01	30/30 (100%)	10.63	40.64
ho-lo-15	46/50 (92%)	21.30	8.45	24/30 (80%)	9.88	11.09
ho-gl-30	49/50 (98%)	21.80	27.74	27/30 (90%)	10.19	33.91
ho-gl-15	47/50 (94%)	21.26	17.71	25/30 (83.33%)	10.12	18.06
Problem	**blocksworld**			**schedule**		
Determinizer	**S**	**C**	**E**	**S**	**C**	**E**
ml-ff	30/30 (100%)	45.17	0.02	0/50 (0%)	-	-
ma-ff	30/30 (100%)	45.93	0.03	30/50 (60%)	24.20	0.02
ao-ff	30/30 (100%)	43.23	0.05	30/50 (60%)	23.20	0.02
alph-1.0	30/30 (100%)	15.40	0.13	50/50 (100%)	27.00	0.17
alph-0.2	30/30 (100%)	15.73	0.14	50/50 (100%)	34.14	0.18
alph-0.1	30/30 (100%)	15.67	0.13	50/50 (100%)	27.18	0.17
alph-0.0	30/30 (100%)	14.97	0.13	50/50 (100%)	30.24	0.17
ho-lo-30	-	-	-	-	-	-
ho-lo-15	-	-	-	-	-	-
ho-gl-30	-	-	-	-	-	-
ho-gl-15	-	-	-	-	-	-

Table 5.2: Results of the experiments with the "terrain" and the recycling domain ("imagine")

Problem	**terrain**			**imagine**		
Determinizer	**S**	**C**	**E**	**S**	**C**	**E**
ml-ff	21/30 (70%)	8.33	0.01	23/50 (46%)	20.30	0.04
ma-ff	10/30 (33.33%)	21.00	0.02	32/50 (64%)	21.81	0.06
ao-ff	14/30 (46.67%)	7.43	0.01	24/50 (48%)	20.29	0.06
alph-1.0	20/30 (66.67%)	7.80	0.10	32/50 (64%)	23.09	0.61
alph-0.2	21/30 (70%)	7.95	0.10	29/50 (58%)	22.90	0.61
alph-0.1	27/30 (90%)	10.33	0.11	35/50 (70%)	26.03	0.77
alph-0.0	30/30 (100%)	20.67	0.11	34/50 (68%)	22.44	0.65
ho-lo-30	-	-	-	-	-	-
ho-lo-15	-	-	-	-	-	-
ho-gl-30	-	-	-	-	-	-
ho-gl-15	-	-	-	-	-	-

In the triangle tireworld problem, the hindsight optimizers are only outperformed by the Single-Outcome approaches. This is the serendipitous effect of the Single-Outcome Determinizers selecting the right outcome out of coincidence. The one using the maximum likelihood criterion chose the worst outcome for the `move` action (the one that always results in a flat tire) by mere chance, since both outcomes have a 50% of probability. The same applies to the Single-Outcome determinizer that selects the outcome with the largest number of add effects, since both outcomes share the same number of additions. However, the Hindsight Optimizers are the clear winners in the rectangle tireworld domain.

α-Cost-Transition-Likelihood exhibits very good performance in the terrain domain and imagine domains. They have a reasonable success rate with a low enough α.

5.3 Hierarchical planning

We have combined all the techniques presented here to build a decision maker capable of deciding intelligent sequences of instructions to fully disassemble a hard drive. Fig. 5.1 shows a plan trace. Preliminary experiments seem promising in the sense that the task decomposition makes sense and episodes are usually computed with success against the simulator.

```
Starting...
[1.195s] Executed (grab-device)
[1.196s] Executed (flip top bottom)
[1.197s] Executed (place-device)
[1.197s] Executed (pick-tool star-sd power)
[1.198s] Executed (unscrew-power-non-stuck pcb-s4 star-sd bottom)
[1.633s] Executed (put-away-tool star-sd power)
[1.633s] Executed (pick-tool flat-sd power)
[1.634s] Executed (unscrew-power-non-stuck pcb-s0 flat-sd bottom)
[1.926s] Executed (unscrew-power-non-stuck pcb-s1 flat-sd bottom)
[2.192s] Executed (unscrew-power-non-stuck pcb-s2 flat-sd bottom)
[2.444s] Executed (unscrew-power-non-stuck pcb-s3 flat-sd bottom)
[2.676s] Executed (assert-clear pcb)
[2.677s] Executed (lever-power-high-confidence pcb pcb-a3 bottom)
[2.678s] Executed (check-removed pcb bottom)
[2.911s] Executed (put-away-tool flat-sd power)
[2.912s] Executed (grab-device)
[2.913s] Executed (flip bottom top)
[2.913s] Executed (place-device)
[2.914s] Executed (pick-tool flat-sd power)
[2.915s] Executed (unscrew-power-non-stuck lid-s0 flat-sd top)
[3.132s] Executed (assert-clear lid)
[3.133s] Executed (lever-power-medium-confidence lid lid-a0 top)
[3.134s] Executed (check-removed lid top)
[3.364s] Executed (put-away-tool flat-sd power)
[3.365s] Executed (grab-device)
[3.366s] Executed (flip top bottom)
[3.366s] Executed (place-device)
[3.367s] Executed (pick-tool star-sd power)
[3.368s] Executed (unscrew-power-non-stuck reader-s0 star-sd bottom)
[3.587s] Executed (put-away-tool star-sd power)
[3.588s] Executed (grab-device)
[3.589s] Executed (flip bottom top)
[3.589s] Executed (place-device)
[3.590s] Executed (pick-tool pliers power)
[3.591s] Executed (assert-clear reader)
[3.592s] Executed (extract-with-pliers-high-confidence reader reader-a1 top)
[3.594s] Executed (check-removed reader top)
[3.823s] Executed (put-away-tool pliers power)
[3.824s] Executed (grab-device)
[3.825s] Executed (flip top bottom)
[3.825s] Executed (assert-clear platter)
[3.826s] Executed (let-fall-down platter top bottom)
[3.827s] Executed (flip bottom top)
[3.828s] Executed (check-removed platter top)
[3.828s] Stopped after 44 step(s). Success!
```

Figure 5.1: Plan trace for the disassembly of a full hard disk.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

We have explored several techniques for dealing with problems from a recycling domain from the planning perspective. We have used the MDP formalization, combined with task decomposition and automatic identification of precedences to bound the complexity o planning.

Perhaps the main point to improve of our project is the implementation of the Hindsight Optimizer, since it seems unable to do timely decisions in most of the problems. However, for our purposes α-Cost-Transition-Likelihood seemed to perform well. In the future we would like to explore options to make this determinization method more robust against dead-ends.

While the simplification of the planning problems, as described in Chapter 4 seems good from a conceptual point of view, we still have to find problems that are complex enough to show its virtues.

BIBLIOGRAPHY

[1] Gerard Canal, Guillem Alenyà, and Carme Torras. "Adapting robot task planning to user preferences: an assistive shoe dressing example". In: *Autonomous Robots* (2018). To appear.

[2] Antonio Andriella et al. "Deciding the different robot roles for patient cognitive training". In: *International Journal of Human-Computer Studies* (2018).

[3] Narcís Palomeras et al. "Toward persistent autonomous intervention in a subsea panel". In: *Autonomous Robots* 40.7 (2016), pp. 1279–1306.

[4] Sylvie C.W. Ong et al. "Planning under uncertainty for robotic tasks with mixed observability". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1053–1068.

[5] Maria Fox and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* (2003).

[6] Håkan LS Younes and Michael L Littman. "PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects". In: *Techn. Rep. CMU-CS-04-162* (2004).

[7] Scott Sanner. "Relational dynamic influence diagram language (RDDL): Language description". In: *Unpublished ms. Australian National University* (2010). URL: `http://rddlsim.googlecode.com/svn-history/r42/trunk/doc/RDDL.pdf`.

[8] Anthony Cassandra. *POMDP File Format*. Online; accessed 10 June 2018. URL: `http://www.pomdp.org/code/pomdp-file-spec.html`.

[9] David Hsu, Wee Sun Lee, et al. *PomdpX File Format (version 1.0)*. Online; accessed 10 June 2018. URL: `http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PomdpXDocumentation`.

[10] David Martínez, Guillem Alenyà, and Carme Torras. "Relational reinforcement learning with guided demonstrations". In: *Artificial Intelligence* 247 (2017), pp. 295–312.

[11] David Martínez Martínez, Guillem Alenya, and Carme Torras. "V-MIN: Efficient Reinforcement Learning through Demonstrations and Relaxed Reward Demands." In: *AAAI*. 2015, pp. 2857–2863.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd Editio. Draft. The MIT Press, June 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[13] Jörg Hoffman and Bernhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14.27 (2001), pp. 253–302. ISSN: 10769757. DOI: 10.1613/jair.855. arXiv: 1106.0675.

[14] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (2006), pp. 191–246.

[15] Sungwook Yoon, Alan Fern, and Robert Givan. "FF-Replan: A baseline for probabilistic planning". In: *17th International Conference on Automated Planning and Scheduling (ICAPS)*. 7. 2007, pp. 352–359. ISBN: 9781577353447 (ISBN). URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-58349118462%7B%5C&%7DpartnerID=40%7B%5C&%7Dmd5=8abd5307b4dee9aa40248bbe6c51d389.

[16] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Integrated task and motion planning in belief space". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227. ISSN: 0278-3649. DOI: 10.1177/0278364913484072. URL: http://ijr.sagepub.com/content/32/9-10/1194.abstract?etoc.

[17] Sungwook Yoon et al. "Probabilistic Planning via Determinization in Hindsight". In: *AAAI* (2008), pp. 1010–1016. URL: http://www.aaai.org/Papers/AAAI/2008/AAAI08-160.pdf.

[18] Sungwook Yoon et al. "Improving determinization in hindsight for online probabilistic planning". In: *20th International Conference on Automated Planning and Scheduling*. 2010, pp. 209–216. ISBN: 9781577354499. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/viewFile/1438/1561.

[19] Andrey Kolobov, Mausam, and Daniel S. Weld. "LRTDP vs. UCT for Online Probabilistic Planning". In: *26th AAAI Conference on Artificial Intelligence*. 2012, pp. 1786–1792. ISBN: 9781577355687.

[20] Thomas Keller and Malte Helmert. "Trial-based Heuristic Tree Search for Finite Horizon MDPs". In: *Proceeding of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS)* (2013), pp. 135–143. DOI: 1011377.3524.

[21] Blai Bonet and Héctor Geffner. "Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming". In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 3. 2003, pp. 12–21. URL: http://www.aaai.org/Papers/ICAPS/2003/ICAPS03-002.pdf.

[22] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *European conference on machine learning* (2006), pp. 282–293. ISSN: 03029743. DOI: 10.1007/11871842_29. URL: http://link.springer.com/10.1007/11871842%7B%5C_%7D29.

[23] Alejandro Suárez-Hernández, Guillem Alenyà, and Carme Torras. "Interleaving hierarchical task planning and motion constraint testing for dual-arm manipulation". In: *International Conference on Intelligent Robots and Systems*. Submitted. 2018.

[24] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *International Conference on Robotics and Automation* (2011), pp. 1470–1477. ISSN: 10504729. DOI: 10.1109/ICRA.2011.5980391. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980391.

[25] Haje J. Kamps. *Apple creates Liam, a robot to rip apart used iPhones*. Online; accessed 10 June 2018. 2016. URL: https://techcrunch.com/2016/03/21/apple-creates-robot-to-rip-apart-iphones/.

[26] F. Torres, S. T. Puente, and R Aracil. "Disassembly Planning Based on Precedence Relations among Assemblies". In: *The International Journal of Advanced Manufacturing Technology* 21.5 (2003), pp. 317–327.

[27] Liangjun Zhang et al. "D-plan: Efficient collision-free path computation for part removal and disassembly". In: *Computer-Aided Design and Applications* 5.6 (2008), pp. 774–786. ISSN: 16864360. DOI: 10.3722/cadaps.2008.774-786.

[28] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016.

[29] Iain Little, Sylvie Thiebaux, et al. "Probabilistic planning vs. replanning". In: *ICAPS Workshop on IPC: Past, Present and Future*. 2007.

APPENDIX A

PPDDL DOMAIN FOR THE RECYCLING APPLICATION

A.1 Robot version

This version of the domain does not contain the `hide-...` predicates, since the robot should not be aware of them:

```
1
2  ; Client version. hide predicates are removed (the client shouldn't be aware
3  ; of them
4
5  (define (domain imagine)
6  (:requirements :adl :rewards :probabilistic-effects)
7  (:types component side mode tool affordance affordance-confidence - object
8  removable-component static-component - component
9  lever-point suction-point pliers-point - affordance
10 screwdriver - tool
11 screw - removable-component
12 )
13
14 (:constants
15 top bottom front back left right - side
16 low medium high - affordance-confidence
17 scara power no-mode - mode
18 flat-sd star-sd - screwdriver
19 hammer suction-tool pliers cutter no-tool - tool
20 )
21
22 (:predicates
23 (has-affordance ?c - removable-component ?a - affordance)
24 (has-confidence ?a - affordance ?c - affordance-confidence)
25 (broken-component ?c - removable-component)
26 (broken-tool ?t - tool)
27 (connected ?c1 ?c2 - component)
28 (clear ?c - removable-component)
29 (current-mode ?m - mode)
30 (current-side ?s - side)
31 (current-tool ?t - tool)
32 (held)
33 (fixed-by ?c - removable-component ?s - screw)
34 (loose ?c - removable-component)
35 (partially-occludes ?c1 ?c2 - component)
36 (removed-non-verified ?c - removable-component)
37 (removed-verified ?c - removable-component)
38 (stuck ?s - screw)
```

```
39 ; static predicates
40 (opposite-side ?s1 ?s2 - side)
41 (at-side ?c - component ?s - side)
42 (valid-mode ?t - tool ?m - mode)
43 (valid-sd ?s - screw ?sd - screwdriver)
44 )
45
46 ;#########################
47 ;# DETERMINISTIC ACTIONS #
48 ;#########################
49
50 (:action check-removed
51 :parameters (?comp - removable-component ?side - side)
52 :precondition (and
53 (current-side ?side)
54 (at-side ?comp ?side)
55 (removed-non-verified ?comp)
56 )
57 :effect (and
58 (forall (?other - component)(not (partially-occludes ?comp ?other)))
59 (not (at-side ?comp ?side))
60 (not (removed-non-verified ?comp))
61 (removed-verified ?comp)
62 )
63 )
64
65 (:action assert-clear
66 :parameters (?comp - removable-component)
67 :precondition (and
68 (not (clear ?comp))
69 (forall (?screw - screw) (not (fixed-by ?comp ?screw)))
70 (forall (?other - component)
71 (and (not (connected ?comp ?other)) (not (connected ?other ?comp))))
72 )
73 :effect (clear ?comp)
74 )
75
76 (:action pick-tool
77 :parameters (?tool - tool ?mode - mode)
78 :precondition (and
79 (not (= ?tool no-tool))
80 (not (= ?mode no-mode))
81 (imply (held) (not (= ?mode power)))
82 (current-tool no-tool)
83 (current-mode no-mode)
84 (valid-mode ?tool ?mode)
85 )
86 :effect (and
87 (not (current-tool no-tool))
88 (not (current-mode no-mode))
89 (current-tool ?tool)
90 (current-mode ?mode)
91 (decrease (reward) 1)
92 )
93 )
94
95 (:action put-away-tool
96 :parameters (?tool - tool ?mode - mode)
97 :precondition (and
98 (not (= ?tool no-tool))
99 (not (= ?mode no-mode))
100 (current-tool ?tool)
101 (current-mode ?mode)
102 )
103 :effect (and
104 (not (current-tool ?tool))
105 (not (current-mode ?mode))
106 (current-tool no-tool)
107 (current-mode no-mode)
108 (decrease (reward) 1)
109 )
110 )
111
112 (:action grab-device
```

```
:parameters ()
:precondition (and
(not (held))
(not (current-mode power))
)
:effect (and (held) (decrease (reward) 1))
)

(:action place-device
:parameters ()
:precondition (held)
:effect (and (not (held)) (decrease (reward) 1))
)

(:action flip
:parameters (?old-side - side ?new-side - side)
:precondition (and (current-side ?old-side) (held))
:effect (and
(not (current-side ?old-side))
(current-side ?new-side)
(decrease (reward) 1)
)
)

(:action let-fall-down
:parameters (?comp - removable-component ?side ?side-opposite - side)
:precondition (and
(held)
(at-side ?comp ?side)
(opposite-side ?side ?side-opposite)
(current-side ?side-opposite)
(loose ?comp)
(clear ?comp)
(forall (?comp_ - component) (not (partially-occludes ?comp_ ?comp)))
)
:effect (and
(removed-non-verified ?comp)
(decrease (reward) 1)
)
)

(:action cut-connector
:parameters (?c1 ?c2 - component ?side - side)
:precondition (and
(current-tool cutter)
(imply (not (held)) (current-mode power))
(connected ?c1 ?c2)
(current-side ?side)
(or (at-side ?c1 ?side) (at-side ?c2 ?side))
)
:effect (and
(not (connected ?c1 ?c2))
(not (connected ?c2 ?c1))
(decrease (reward) 1)
)
)

;########################
;# UNSCREW (4 ACTIONS) #
;########################

(:action unscrew-scara-non-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(not (stuck ?screw))
(current-mode scara)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
)
:effect (and
```

```
(probabilistic
0.60 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.15 (stuck ?screw)
)
(decrease (reward) 1)
)
)

(:action unscrew-scara-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(stuck ?screw)
(current-mode scara)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
)
:effect (and
(probabilistic
0.2 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
)
(decrease (reward) 1)
)
)

(:action unscrew-power-non-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(not (stuck ?screw))
(current-mode power)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
)
:effect (and
(probabilistic
0.85 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.15 (stuck ?screw)
)
(decrease (reward) 1)
)
)

(:action unscrew-power-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(stuck ?screw)
(current-mode power)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
)
:effect (and
(probabilistic
0.75 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
```

```
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.10 (broken-component ?screw)
)
(probabilistic 0.10 (broken-tool ?sd))
(decrease (reward) 1)
)
)

;########
;# BASH #
;########

(: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)
)
)

;######################
;# LEVER (6 ACTIONS) #
;######################

(:action lever-scara-low-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp low)
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.10 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.05 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
)
(probabilistic 0.25 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-scara-medium-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
```

```
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp medium)
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.25 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.10 (removed-non-verified ?comp)
0.12 (broken-component ?comp)
)
(probabilistic 0.12 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-scara-high-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp high)
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.60 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.30 (removed-non-verified ?comp)
0.05 (broken-component ?comp)
)
(probabilistic 0.05 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-power-low-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp low)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.05 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.10 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
)
(probabilistic 0.25 (broken-tool flat-sd))
```

```
(decrease (reward) 1)
)
)

(:action lever-power-medium-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp medium)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.20 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.50 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
)
(probabilistic 0.12 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-power-high-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp high)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic
0.25 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.60 (removed-non-verified ?comp)
0.10 (broken-component ?comp)
)
(probabilistic 0.05 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

;##########################
;# SUCK-AWAY (3 ACTIONS) #
;##########################

(:action suck-away-low-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp low)
```

```
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic 0.05 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

(:action suck-away-medium-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp medium)
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic 0.25 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

(:action suck-away-high-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp high)
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(clear ?comp)
)
:effect (and
(probabilistic 0.50 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

;###################################
;# EXTRACT-WITH-PLIERS (3 ACTIONS) #
;###################################

(:action extract-with-pliers-low-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
(has-affordance ?comp ?pp)
(has-confidence ?pp low)
(current-side ?side)
(current-tool pliers)
(clear ?comp)
)
```

```
:effect (and
(probabilistic 0.50 (removed-non-verified ?comp)
0.50 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)

(:action extract-with-pliers-medium-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
(has-affordance ?comp ?pp)
(has-confidence ?pp medium)
(current-side ?side)
(current-tool pliers)
(clear ?comp)
)
:effect (and
(probabilistic 0.75 (removed-non-verified ?comp)
0.25 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)

(:action extract-with-pliers-high-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
(has-affordance ?comp ?pp)
(has-confidence ?pp high)
(current-side ?side)
(current-tool pliers)
(clear ?comp)
)
:effect (and
(probabilistic 0.85 (removed-non-verified ?comp)
0.15 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)
)
```

A.2 Simulator (complete) version

This version contains all the predicates, including the `hide-...` ones. It is intended for simulation use (the simulator should have a complete and true model of the application).

```
(define (domain imagine)
(:requirements :adl :rewards :probabilistic-effects)
(:types component side mode tool affordance affordance-confidence - object
removable-component - component
lever-point suction-point pliers-point - affordance
screwdriver - tool
screw - removable-component
)

```

```
(:constants
top bottom front back left right - side
low medium high - affordance-confidence
scara power no-mode - mode
flat-sd star-sd - screwdriver
hammer suction-tool pliers cutter no-tool - tool
)

(:predicates
(has-affordance ?c - removable-component ?a - affordance)
(has-confidence ?a - affordance ?c - affordance-confidence)
(broken-component ?c - removable-component)
(broken-tool ?t - tool)
(connected ?c1 ?c2 - component)
(clear ?c - removable-component)
(current-mode ?m - mode)
(current-side ?s - side)
(current-tool ?t - tool)
(held)
(hides-component ?c1 ?c2 - component)
(hides-affordance ?c1 - component ?a - affordance)
(fixed-by ?c - removable-component ?s - screw)
(loose ?c - removable-component)
(partially-occludes ?c1 ?c2 - component)
(removed-non-verified ?c - removable-component)
(removed-verified ?c - removable-component)
(stuck ?s - screw)
; static predicates
(opposite-side ?s1 ?s2 - side)
(at-side ?c - component ?s - side)
(valid-mode ?t - tool ?m - mode)
(valid-sd ?s - screw ?sd - screwdriver)
)

;#########################
;# DETERMINISTIC ACTIONS #
;#########################

(:action check-removed
:parameters (?comp - removable-component ?side - side)
:precondition (and
(current-side ?side)
(at-side ?comp ?side)
(removed-non-verified ?comp)
)
:effect (and
(forall (?other - component)
(and
(not (hides-component ?comp ?other))
(not (partially-occludes ?comp ?other))
))
(forall (?aff - affordance) (not (hides-affordance ?comp ?aff)))
(not (at-side ?comp ?side))
(not (removed-non-verified ?comp))
(removed-verified ?comp)
)
)

(:action assert-clear
:parameters (?comp - removable-component)
:precondition (and
(not (clear ?comp))
(forall (?screw - screw) (not (fixed-by ?comp ?screw)))
(forall (?other - component)
(and (not (connected ?comp ?other)) (not (connected ?other ?comp))))
)
:effect (clear ?comp)
)

(:action pick-tool
:parameters (?tool - tool ?mode - mode)
:precondition (and
(not (= ?tool no-tool))
(not (= ?mode no-mode))
```

```
(imply (held) (not (= ?mode power)))
(current-tool no-tool)
(current-mode no-mode)
(valid-mode ?tool ?mode)
)
:effect (and
(not (current-tool no-tool))
(not (current-mode no-mode))
(current-tool ?tool)
(current-mode ?mode)
(decrease (reward) 1)
)
)

(:action put-away-tool
:parameters (?tool - tool ?mode - mode)
:precondition (and
(not (= ?tool no-tool))
(not (= ?mode no-mode))
(current-tool ?tool)
(current-mode ?mode)
)
:effect (and
(not (current-tool ?tool))
(not (current-mode ?mode))
(current-tool no-tool)
(current-mode no-mode)
(decrease (reward) 1)
)
)

(:action grab-device
:parameters ()
:precondition (and
(not (held))
(not (current-mode power))
)
:effect (and (held) (decrease (reward) 1))
)

(:action place-device
:parameters ()
:precondition (held)
:effect (and (not (held)) (decrease (reward) 1))
)

(:action flip
:parameters (?old-side - side ?new-side - side)
:precondition (and (current-side ?old-side) (held))
:effect (and
(not (current-side ?old-side))
(current-side ?new-side)
(decrease (reward) 1)
)
)

(:action let-fall-down
:parameters (?comp - removable-component ?side ?side-opposite - side)
:precondition (and
(held)
(at-side ?comp ?side)
(opposite-side ?side ?side-opposite)
(current-side ?side-opposite)
(loose ?comp)
(clear ?comp)
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(forall (?comp_ - component) (not (partially-occludes ?comp_ ?comp)))
)
:effect (and
(removed-non-verified ?comp)
(decrease (reward) 1)
)
)

```

```
(:action cut-connector
:parameters (?c1 ?c2 - component ?side - side)
:precondition (and
(current-tool cutter)
(imply (not (held)) (current-mode power))
(connected ?c1 ?c2)
(current-side ?side)
(or (at-side ?c1 ?side) (at-side ?c2 ?side))
)
:effect (and
(not (connected ?c1 ?c2))
(not (connected ?c2 ?c1))
(decrease (reward) 1)
)
)

;########################
;# UNSCREW (4 ACTIONS) #
;########################

(:action unscrew-scara-non-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(not (stuck ?screw))
(current-mode scara)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
(forall (?comp - component) (not (hides-component ?comp ?screw)))
)
:effect (and
(probabilistic
0.60 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.15 (stuck ?screw)
)
(decrease (reward) 1)
)
)

(:action unscrew-scara-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(stuck ?screw)
(current-mode scara)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
(forall (?comp - component) (not (hides-component ?comp ?screw)))
)
:effect (and
(probabilistic
0.2 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
)
(decrease (reward) 1)
)
)

(:action unscrew-power-non-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
```

```
(not (stuck ?screw))
(current-mode power)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
(forall (?comp - component) (not (hides-component ?comp ?screw)))
)
:effect (and
(probabilistic
0.85 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.15 (stuck ?screw)
)
(decrease (reward) 1)
)
)

(:action unscrew-power-stuck
:parameters (?screw - screw ?sd - screwdriver ?side - side)
:precondition (and
(not (broken-component ?screw))
(not (broken-tool ?sd))
(stuck ?screw)
(current-mode power)
(current-side ?side)
(at-side ?screw ?side)
(current-tool ?sd)
(valid-sd ?screw ?sd)
(forall (?comp - component) (not (hides-component ?comp ?screw)))
)
:effect (and
(probabilistic
0.75 (and
(forall (?comp - removable-component) (not (fixed-by ?comp ?screw)))
(not (at-side ?screw ?side))
(removed-verified ?screw))
0.10 (broken-component ?screw)
)
(probabilistic 0.10 (broken-tool ?sd))
(decrease (reward) 1)
)
)

;########
;# BASH #
;########

(: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 (hides-component ?comp_ ?comp)))
(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)
)
)

```

```
;#####################
;# LEVER (6 ACTIONS) #
;#####################

(:action lever-scara-low-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp low)
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.10 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.05 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
)
(probabilistic 0.25 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-scara-medium-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp medium)
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.25 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.10 (removed-non-verified ?comp)
0.12 (broken-component ?comp)
)
(probabilistic 0.12 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-scara-high-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp high)
```

```
(current-side ?side)
(current-tool flat-sd)
(current-mode scara)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.60 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.30 (removed-non-verified ?comp)
0.05 (broken-component ?comp)
)
(probabilistic 0.05 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-power-low-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp low)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.05 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.10 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
)
(probabilistic 0.25 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-power-medium-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp medium)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.20 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.50 (removed-non-verified ?comp)
0.25 (broken-component ?comp)
```

```
)
(probabilistic 0.12 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

(:action lever-power-high-confidence
:parameters (?comp - removable-component
?lp - lever-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool flat-sd))
(at-side ?comp ?side)
(has-affordance ?comp ?lp)
(has-confidence ?lp high)
(current-side ?side)
(current-tool flat-sd)
(current-mode power)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?lp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic
0.25 (and (loose ?comp)
(when (loose ?comp) (removed-non-verified ?comp)))
0.60 (removed-non-verified ?comp)
0.10 (broken-component ?comp)
)
(probabilistic 0.05 (broken-tool flat-sd))
(decrease (reward) 1)
)
)

;#########################
;# SUCK-AWAY (3 ACTIONS) #
;#########################

(:action suck-away-low-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp low)
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?sp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.05 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

(:action suck-away-medium-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp medium)
```

```
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?sp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.25 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

(:action suck-away-high-confidence
:parameters (?comp - removable-component
?sp - suction-point
?side - side)
:precondition (and
(not (broken-component ?comp))
(not (broken-tool suction-tool))
(at-side ?comp ?side)
(has-affordance ?comp ?sp)
(has-confidence ?sp high)
(current-side ?side)
(current-tool suction-tool)
(imply (not (held)) (current-mode power))
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?sp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.50 (removed-non-verified ?comp))
(probabilistic 0.05 (broken-tool suction-tool))
(decrease (reward) 1)
)
)

;###################################
;# EXTRACT-WITH-PLIERS (3 ACTIONS) #
;###################################

(:action extract-with-pliers-low-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
(has-affordance ?comp ?pp)
(has-confidence ?pp low)
(current-side ?side)
(current-tool pliers)
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?pp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.50 (removed-non-verified ?comp)
0.50 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)

(:action extract-with-pliers-medium-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
```

```
(has-affordance ?comp ?pp)
(has-confidence ?pp medium)
(current-side ?side)
(current-tool pliers)
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?pp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.75 (removed-non-verified ?comp)
0.25 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)

(:action extract-with-pliers-high-confidence
:parameters (?comp - removable-component
?pp - pliers-point
?side - side)
:precondition (and
(not (broken-tool pliers))
(not (broken-component ?comp))
(at-side ?comp ?side)
(has-affordance ?comp ?pp)
(has-confidence ?pp high)
(current-side ?side)
(current-tool pliers)
(forall (?comp_ - component) (not (hides-affordance ?comp_ ?pp)))
(forall (?comp_ - component) (not (hides-component ?comp_ ?comp)))
(clear ?comp)
)
:effect (and
(probabilistic 0.85 (removed-non-verified ?comp)
0.15 (broken-component ?comp))
(probabilistic 0.05 (broken-tool pliers))
(decrease (reward) 1)
)
)
)
```

APPENDIX B

PPDDL DOMAIN OF THE "TERRAINS" EXAMPLE

"Terrains" domain together with a small problem instance:

```
; Author: Alejandro Suarez Hernandez

(define (domain terrain)
(:requirements :typing :strips :probabilistic-effects :rewards)
(:types loc - object land shallow-water deep-water - loc)
(:predicates
  (connected ?l1 ?l2 - loc)
  (at ?l - loc)
  (boulder-at ?l - loc)
  (pickaxe-at ?l - loc)
  (flag-at ?l - loc)
  (alive)
  (has-pickaxe)
  (goal-reached)
)

(:action move-to-land
 :parameters (?l1 - loc ?l2 - land)
 :precondition (and
  (alive)
  (at ?l1)
  (or (connected ?l1 ?l2)
  (connected ?l2 ?l1))
  (not (boulder-at ?l2)))
 :effect (and
  (not (at ?l1))
  (at ?l2)
  (decrease (reward) 1))
)

(:action move-to-shallow-water
 :parameters (?l1 - loc ?l2 - shallow-water)
 :precondition (and
  (alive)
  (at ?l1)
  (or (connected ?l1 ?l2) (connected ?l2 ?l1))
  (not (boulder-at ?l2)))
 :effect (and
  (not (at ?l1))
  (at ?l2)
  (probabilistic 0.05 (not (alive)))
  (decrease (reward) 1)
))

(:action move-to-deep-water
```

```
 :parameters (?l1 - loc ?l2 - deep-water)
 :precondition (and
  (alive)
  (at ?l1)
  (or (connected ?l1 ?l2) (connected ?l2 ?l1))
  (not (boulder-at ?l2)))
 :effect (and
  (not (at ?l1))
  (at ?l2)
  (probabilistic 0.2 (not (alive)))
  (decrease (reward) 1))
)

(:action pick-pickaxe
 :parameters (?l - loc)
 :precondition (and
  (alive)
  (not (has-pickaxe))
  (at ?l)
  (pickaxe-at ?l))
 :effect (and
  (has-pickaxe)
  (not (pickaxe-at ?l)))
)

(:action break-boulder
 :parameters (?l1 ?l2 - land)
 :precondition (and
  (alive)
  (has-pickaxe)
  (at ?l1)
  (or (connected ?l1 ?l2) (connected ?l2 ?l1))
  (boulder-at ?l2))
 :effect (and
  (not (boulder-at ?l2))
  (decrease (reward) 2))
)

(:action reach-goal
 :parameters (?l - loc)
 :precondition (and
  (not (goal-reached))
  (alive)
  (at ?l)
  (flag-at ?l)
 )
 :effect (goal-reached)
)

)

(define (problem p01)
(:domain terrain)
(:objects
  x_0_0 x_0_2 x_0_3 x_1_0 x_2_0 x_2_1 x_2_2 x_2_3  - land
  x_0_1 x_1_1  - shallow-water
  x_1_2 x_1_3  - deep-water
)
(:init
  (alive)
  (connected x_0_0 x_1_0)
  (connected x_0_0 x_0_1)
  (connected x_0_1 x_1_1)
  (connected x_0_1 x_0_2)
  (connected x_0_2 x_1_2)
  (connected x_0_2 x_0_3)
  (connected x_0_3 x_1_3)
  (connected x_1_0 x_2_0)
  (connected x_1_0 x_1_1)
  (connected x_1_1 x_2_1)
  (connected x_1_1 x_1_2)
  (connected x_1_2 x_2_2)
  (connected x_1_2 x_1_3)
  (connected x_1_3 x_2_3)
```

```
120   (connected x_2_0 x_2_1)
121   (connected x_2_1 x_2_2)
122   (connected x_2_2 x_2_3)
123   (at x_1_0)
124   (boulder-at x_2_1)
125   (pickaxe-at x_0_3)
126   (flag-at x_2_3)
127   (= (reward) 0)
128 )
129 (:goal (goal-reached))
130 (:metric maximize (reward))
131 )
```