# GOAL REASONING FOR AUTONOMOUS AGENTS USING AUTOMATED PLANNING

by

ALBERTO POZANCO

A dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Advisors:
Prof. Dr. Daniel Borrajo Millán
Prof. Dr. Susana Fernández Arregui

December 21, 2020

*Dedicado a mi abuelo Felipe y todo lo que él representa.*

*"Es de bien nacidos ser agradecido"*
*- Refranero popular*

# ACKNOWLEDGEMENTS

Con la escritura de este documento termina la que ha sido sin duda una de las etapas más bonitas e intensas de mi vida. No sólo en la ciencia sino también en la vida cabalgamos a hombros de gigantes. Así que me gustaría empezar esta tesis agradeciendo a todas las personas que han hecho posible que yo escriba estas páginas.

En primer lugar, a mi familia. Gracias Papá y Mamá por ofrecerme el mejor entorno donde crecer y desarrollarme, no sé si algún día podré devolveros todo vuestro cariño y apoyo. También por darme a María. Fuiste mi compañera favorita de juegos muchos años y aunque ya no cacemos cangrejos en la playa, estoy seguro de que nos quedan muchas aventuras por vivir. Gracias a mis abuelos, por cuidarme cuando era pequeño y darme tan buenos momentos cuando he ido creciendo. En especial a mi *Lolo* Felipe, a quien dedico esta tesis. Eres un espejo donde mirarme y sólo espero que esta situación pase pronto para volver a salir de cañas juntos.

Siempre se dice que los amigos son la familia que se elige, y yo he elegido muy bien. Gracias a David, por ser el primer amigo del que tengo recuerdos. Yo haciendo *ctrl-c, ctrl-v* y tu ya padre de dos criaturas. Espero que les cuides mejor de lo que yo cuidé nuestra cuenta de Bankia. A Manu, por ser mi hermano mayor y estar conmigo en las buenas y en las malas, en el Llano o en los Alpes, en el Escondite o en el Zaguán. La guerra que nos queda por dar, amigo. A Pablo, Marina, Wili y tantos otros que me dejo. Gracias por hacer que volver unos días al pueblo sea el mejor de los planes.

Siempre he defendido que Candeleda es el mejor sitio para crecer. Allí tuve una infancia y adolescencia plenas, y por eso mismo venir a la universidad a Madrid era un paso tan lógico como difícil. Gracias a Sara, por las eternas conversaciones y hacer que salir del pueblo no fuera más que un salto en otro charco. Gracias a toda la gente de la FAM por las fiestas, los viajes y por cenar a la hora de los grillos. En especial a Pelayo y José, compañeros no sólo de habitación sino también de prácticas. A lo mejor nuestro Mario o nuestro Sokoban no eran los mejores del mundo, pero ¿y las risas qué? Realmente no disfruté la informática plenamente hasta que conocí a Carlos. Tu pasión hablando de heurísticas admisibles me convenció de escoger la rama de inteligencia artificial y hacer el trabajo de fin de grado contigo.

A raíz de esa experiencia en 2015 comencé el máster y poco después entré a formar parte del PLG. Gracias a Daniel y Susana por darme esa oportunidad y apoyarme en mi decisión de comenzar una tesis doctoral. Habéis sido unos directores excepcionales, dándome la libertad para explorar los temas que más me interesaban en cada momento y corrigiéndome cuando era necesario. Os debo una caja de bolígrafos rojos. Quiero expresar mi especial gratitud a Daniel, sin duda una de las personas de las que más he aprendido estos años. Gracias por todas las puertas que me has ayudado a abrir.

Pero el PLG no es sólo una tesis o un sitio donde trabajar y aprender. Para mí ha sido muchísimo más. Gracias a Pulido, JC e Isa por darme la más cálida de las bienvenidas allá por febrero de 2016. En especial a Isa, por ser mi maestra Jedi, compañera, casera y amiga. Si esta tesis tuviera apellido sería Cenamor y si tuviese pelo sería rubia. Gracias por todo. Con el tiempo fueron llegando muchos otros al *labo*, con los que seguí viajando y compartiendo buenos momentos. Gracias a Rubén, por trascender el trabajo y convertirte en uno de mis mejores amigos. Tenemos anécdotas para aburrir y espero que sigan creciendo. Gracias a Álex, por compartir mi humor. Gracias a Mauricio, por nuestras conversaciones profundas (y las no tan profundas). Llegarás donde quieras. Y gracias a Alba por todo su apoyo y cariño. Tienes un corazón que no te cabe en el pecho. También tengo mucho que agradecer a mi mongola Yolanda. Diste un empujón tremendo a mi tesis hablándome de la tuya. Gracias por estar en los momentos buenos y en los malos.

Una de las mejores partes de hacer la tesis ha sido tener la oportunidad de viajar a conferencias y conocer un montón de sitios y personas interesantes. Gracias a Álvaro por estar en casi todos esos lugares, siendo amable conmigo y presentándome a todo el mundo. También a Diego, por compartir muchos de esos viajes y experiencias.

Sin duda el viaje más emocionante fue mi estancia en Australia. Gracias a Sebastián, por acogerme y darme tantas facilidades. He aprendido muchísimo de ti y siempre guardaré un grato recuerdo tanto de las conversaciones técnicas en tu despacho, como de las políticas y deportivas en los ratos libres. Gracias también a toda la gente de RMIT y Melbourne que me hizo sentir como en casa: Nir, Melina, Sebastián, John, Julie... Fue un auténtico placer pasar esos meses en la otra punta del mundo con todos vosotros.

No puedo cerrar mi paso por la universidad sin dar las gracias a Yolanda y Esperanza de la cafetería. Los días empiezan mucho mejor cuando te llaman guapo y te sirven un desayuno con la mejor de las sonrisas. Ojalá mucha más gente como vosotras en el mundo.

Let me switch to English to express my gratitude to the amazing people working at the JP Morgan AI Research team. Thank you Daniele for giving me the chance of joining the group first as an intern, and now as my full time job. I would also like to express my deepest gratitude to Parisa, Gonçalo, Mahmoud, and Salwa. Thank you for teaching me so much, for supporting me in the bad times, and sharing with me so many good ones. I look forward to working with you again and start together this new stage of my life.

Last but not least, I would like to thank the planning community. This thesis would not have been possible without all the software and benchmarks you publicly share, the critical reviews that strength the papers, and the conversations and ideas you shared with me during these years. Thank you for being such a friendly and inspiring community.

# PUBLISHED AND SUBMITTED CONTENT

Some of the material in this dissertation has appeared (or will shortly appear) in different journals, conferences and workshops. Here we provide a relation of these papers and in which Chapters they are included. The material from these sources included in this thesis is not singled out with typographic means and references. Conference rankings are according to CORE 2018[1]. Journal metrics are according to FECYT JCR tool[2].

## Conference Papers

1. **Counterplanning using Goal Recognition and Landmarks** Pozanco, A; E-Martín, Y; Fernández, S; and Borrajo, D. In *Proceedings of IJCAI'18*, 4808-4814, Stockholm (Sweden), 2018.
    - *Impact*: $A^+$ conference.
    - *Role*: Main author of the paper, in charge of the idea, writing and coding.
    - *URL*: https://doi.org/10.24963/ijcai.2018/668
    - Partially included in Chapter 5 and 6.
2. **Finding Centroids and Minimum Covering States in Planning** Pozanco, A; E-Martín, Y; Fernández, S; and Borrajo, D. In *Proceedings of ICAPS'19*, 348-352, Berkeley (USA), 2019.
    - *Impact*: $A^+$ conference.
    - *Role*: Main author of the paper, in charge of the idea, writing and coding.
    - *URL*: https://aaai.org/ojs/index.php/ICAPS/article/view/3497
    - Partially included in Chapter 3.
3. **Multi-tier Automated Planning for Adaptive Behavior** Ciolek, D; D'Ippolito, N; Pozanco, A; and Sardina, S. In *Proceedings of ICAPS'20*, 66-74, Nancy (France), 2020.
    - *Impact*: $A^+$ conference.
    - *Role*: Collaborator of the paper, in charge of coding and evaluation.
    - *URL*:
    - Partially included in Appendix B.

[1] http://portal.core.edu.au
[2] https://www.recursoscientificos.fecyt.es/factor/

## Journal Papers

1. **Learning-driven Goal Generation**. Pozanco, A; Fernández, S; and Borrajo, D. *AI Communications, vol. 31, no. 2, pp. 137-150*, 2018.
   - *Impact*: Q4 JCR (0.765).
   - *Role*: Main author of the paper, in charge of writing and coding.
   - *DOI*: 10.3233/AIC-180754
   - Partially included in Chapter 4.

## Workshop Papers

1. **Get me to Safety! Escaping from Risks using Automated Planning**. Pozanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. In *Proceedings of IntEx/GR Workshop, ICAPS'20*, Nancy (France), 2020.
   - *Role*: Main author of the paper, in charge of the idea, writing and coding.
   - *URL*: https://icaps20subpages.icaps-conference.org/workshops/gr/
   - Partially included in Chapter 3.
2. **Counterplanning in Real-Time Strategy Games through Goal Recognition**. Pozanco, A.; Blanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. In *Proceedings of 6th Workshop on Goal Reasoning, IJCAI'18*, Stockholm (Sweden), 2018.
   - *Role*: Main author of the paper, in charge of the idea, writing and coding.
   - *URL*: https://dtdannen.github.io/faim2018grw/counterplanning
   - Partially included in Chapter 5.
3. **Counterplanning using Goal Recognition and Landmarks** Pozanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. In *Proceedings of 5th Workshop on Distributed and Multi-Agent Planning, ICAPS'18*, Delft (Netherlands), 2018
   - *Role*: Main author of the paper, in charge of the idea, writing and coding.
   - *URL*: http://icaps18.icaps-conference.org/dmap/counterplanning
   - Partially included in Chapter 5.

## Submitted Papers

1. **Planning with Distance-based Goals**. Pozanco, A; E-Martín, Y; Fernández, S; and Borrajo, D. *Submitted to Artificial Intelligence Journal (AIJ)*, 2020.
   - *Impact*: Q1 JCR.
   - *Role*: Main author of the paper, in charge of writing and coding.
   - Partially included in Chapter 3.

# OTHER RESEARCH MERITS

During the PhD I have done two research stays, attended to several conferences to present my work, I have been involved in research projects, and published different articles whose content does not appear in the dissertation. The following list enumerates some of these merits.

## Research Stays

1. **RMIT University, Melbourne, Australia (2019)**. 4 months research visit to Prof. Sebastian Sardina, where we explored the synthesis of adaptive behavior for autonomous agents using fully-observable non-deterministic planning. Partially funded by Universidad Carlos III de Madrid through "Ayudas para estancias doctorales".

2. **JP Morgan, London, United Kingdom (2020)**. 3 months research internship at the AI Research Team.

## Conferences and Workshops Attended

- International Conference on Automated Planning and Scheduling (ICAPS). Online, 2020.
  *Oral presentation in IntEx/GR Workshop*

- Symposium on Combinatorial Search (SoCS). Napa (USA), 2019.
  *Oral presentation in main conference*

- International Conference on Automated Planning and Scheduling (ICAPS). Berkeley (USA), 2019.
  *Oral presentation in main conference*
  *Oral presentation in Workshop on the International Planning Competition*

- International Joint Conference on Artificial Intelligence (IJCAI). Stockholm (Sweden), 2018.
  *Oral presentation in main conference*
  *Poster in main conference*
  *Oral presentation in Goal Reasoning Workshop*

- International Conference on Automated Planning and Scheduling (ICAPS). Delft (Netherlands), 2018.
  *Oral presentation in Knowledge Engineering for Planning and Scheduling Workshop*
  *Oral presentation in Distributed and Multi-Agent Planning Workshop*
  *Poster in Doctoral Consortium*

- ICAPS Summer School on Planning Under Uncertainty. Noordwijk (Netherlands), 2018.

- International Joint Conference on Artificial Intelligence (IJCAI). New York (USA), 2016.
  *Oral presentation in Agents in Traffic and Transportation Workshop*
  *Oral presentation in Goal Reasoning Workshop*

### Project Participation

1. **GOALHUB**. Universidad Carlos III de Madrid & Goal Systems (Feb. 2020 - Jan. 2021). Intelligent routing of trains over real-world rail networks.
2. **ARPIA**. Universidad Carlos III de Madrid & Universidad Politécnica de Valencia (Jul. 2018 - Jan. 2021). Activity Recognition and Planning for Intelligent Assistants.
3. **PLICOGOR**. Universidad Carlos III de Madrid & Goal Systems (Feb. 2017 - Feb. 2020). Intelligent Planning System for route generation in collective transport.
4. **GLASS**. Universidad Carlos III de Madrid & Universidad Politécnica de Valencia (Feb. 2016 - Feb. 2017). Goal management for long term autonomy in smart cities.

### Conference Papers

1. **Error Analysis and Correction for Weighted A*'s Suboptimality** Holte, R.C.; Majadas, R.; Pozanco, A.; and Borrajo, D. In *Proceedings of SoCS'19*, 135-139, Napa (USA), 2019.
   - *Impact*: Unranked.
   - *Role*: Collaborator of the paper, in charge of coding and evaluation.
   - *URL*: https://aaai.org/socs/error-analysis

### Journal Papers

1. **On-line Modeling and Planning for Urban Traffic Control**. Pozanco, A; Fernández, S; and Borrajo, D. *Minor revision at Expert Systems*.
   - *Impact*: Q2 JCR (1.505).
   - *Role*: Main author of the paper, in charge of the idea, writing and coding.

### Workshop Papers

1. **Insights from the 2018 IPC Benchmarks**. Cenamor, I.; and Pozanco, A. In *Proceedings of 5th Workshop on the International Planning Competition, ICAPS'19*, Berkeley (USA), 2019.
   - *Role*: Collaborator of the paper, in charge of the idea, writing and coding.
   - *URL*: https://openreview.net/pdf?id=ByeCs5UkcN

2. **Distributed Planning and Model Learning for Urban Traffic Control**. Pozanco, A.; Fernández, S.; and Borrajo, D. In *Proceedings of Workshop on Knowledge Engineering for Planning and Scheduling, ICAPS'18*, Delft (Netherlands), 2018.
    - *Role*: Main author of the paper, in charge of the idea, writing and coding.
    - *URL*: ttp://icaps18.icaps-conference.org/keps/traffic
3. **Urban Traffic Control Assisted by AI Planning and Relational Learning**. Pozanco, A.; Fernández, S.; and Borrajo, D. In *Proceedings of 9th International Workshop on Agents in Traffic and Transportation (IJCAI'16)*, New York (USA), 2016.
    - *Role*: Main author of the paper, in charge of writing and coding.
    - *URL*: http://www.ia.urjc.es/att2016/CRCs/paper6.pdf
4. **On Learning Planning Goals for Traffic Control**. Pozanco, A.; Fernández, S.; and Borrajo, D. In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*, New York (USA), 2016.
    - *Role*: Main author of the paper, in charge of writing and coding.
    - *URL*: http://makro.ink/ijcai2016grw/papers/learning

# ABSTRACT

Automated planning deals with the task of finding a sequence of actions, namely a plan, which achieves a goal from a given initial state. Most planning research consider goals are provided by a external user, and agents just have to find a plan to achieve them. However, there exist many real world domains where agents should not only reason about their actions but also about their goals, generating new ones or changing them according to the perceived environment. In this thesis we aim at broadening the goal reasoning capabilities of planning-based agents, both when acting in isolation and when operating in the same environment as other agents.

In single-agent settings, we firstly explore a special type of planning tasks where we aim at discovering states that fulfill certain cost-based requirements with respect to a given set of goals. By computing these states, agents are able to solve interesting tasks such as find escape plans that move agents in to safe places, hide their true goal to a potential observer, or anticipate dynamically arriving goals. We also show how learning the environment's dynamics may help agents to solve some of these tasks. Experimental results show that these states can be quickly found in practice, making agents able to solve new planning tasks and helping them in solving some existing ones.

In multi-agent settings, we study the automated generation of goals based on other agents' behavior. We focus on competitive scenarios, where we are interested in computing counterplans that prevent opponents from achieving their goals. We frame these tasks as counterplanning, providing theoretical properties of the counterplans that solve them. We also show how agents can benefit from computing some of the states we propose in the single-agent setting to anticipate their opponent's movements, thus increasing the odds of blocking them. Experimental results show how counterplans can be found in different environments ranging from competitive planning domains to real-time strategy games.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LISTINGS

# ACRONYMS

| | |
|---|---|
| AI | Artificial Intelligence |
| AP | Automated Planning |
| BDI | Belief-Desire-Intention |
| CBR | Case-Based Reasoning |
| FOND | Fully-observable Non-deterministic |
| GDA | Goal-directed Autonomy |
| FTP | Fault Tolerant Planning |
| GRS | Goal Related States |
| IPC | International Planning Competition |
| LTL | Linear Temporal Logic |
| MAP | Multi-agent Planning |
| MDP | Markov Decision Process |
| MTP | Multi-tier Problem |
| PAIR | Plan, Activity and Intent Recognition |
| PDDL | Planning Domain Description Language |
| RL | Reinforcement Learning |
| RP | Robust Planning |
| RTS | Real-Time Strategy |
| SAT | Propositional Satisfiability Problem |
| UAV | Unmanned Aerial Vehicle |

Part I

# INTRODUCTION AND BACKGROUND

# 1 INTRODUCTION

*"Intelligence is a very valuable thing, innit,*
*my friend. And usually, it comes far too late."*

*- Alfie Solomons*

In this chapter we introduce the work carried out in this thesis. First, we briefly introduce the research area and motivate the dissertation. Then, we enumerate the objectives of the thesis. Finally, we outline the structure of the rest of the document.

## 1.1 MOTIVATION

As good old Alfie says, intelligence is a very valuable thing. One of its many definitions consider intelligence as a

*"very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings - catching on, making sense of things, or figuring out what to do" (Gottfredson, 1997).*

These mental capabilities are inherent to human beings, but is it possible to extend them to other entities? Artificial Intelligence (AI) is a field of computer science that tries to answer this question by studying the development of *intelligent agents* (Russell & Norvig, 2003), able to exhibit some of these capabilities. In this thesis we focus on one important aspect of intelligence: that of goal-directed behavior. And from the many AI approaches that solve these kind of problems, we will use Automated Planning.

Automated Planning (AP) (Ghallab et al., 2004) solves the task of finding a sequence of actions, namely a *plan*, which achieves a goal from a given initial state. Typically, goals are considered as static entities; they are given at start of planning, and they do not change over time. Although this setting is valid for many research areas within AP, it narrows the application of planning in real world domains.

The overarching motivation of this dissertation is to address the call by Ghallab, Nau, and Traverso (2014; 2016) for an *actor's perspective* on planning. The aim is to put the focus on planning agents, which use automated planning in a broader sense to reason and act based on their environment. Under this framework, the assumption of agents having a pre-defined and static set of goals is too restrictive and limits the autonomy of these agents.

Our aim is to provide agents with some *goal reasoning* capabilities (Cox, 2007; Molineaux et al., 2010), so they augment their autonomy and become able to solve a broader set of tasks. We approach this problem from two perspectives. First, we study different goal reasoning problems when agents act in isolation. We envision how agents can reformulate their goals as their environment changes. In addition, we show how agents can reason about goals in a different way generating plans that do not achieve them but reach states that might be useful. Then, we study goal reasoning in the context of multi-agent settings. In this case we aim at generating goals from scratch based on other agent's behavior. Finally, we extend goal reasoning concepts of the single-agent setting to multi-agent scenarios, showing how the autonomy (and intelligence) of resulting agents is augmented.

## 1.2 OBJECTIVES OF THE THESIS

The ultimate aim of this thesis is to improve the goal reasoning capabilities of planning-based agents. To do so, we study different goal reasoning problems both in single and multi-agent settings. The particular objectives of the thesis are:

- In **single-agent** settings, we are interested in enriching the usual planning task definition where we have a static model and goal definition. Instead of providing agents with an explicit goal they have to achieve, we aim at giving them a set of goals to reason about. By reasoning over a set of goals, agents are able to discover new states that allow them to solve interesting tasks such as: (1) the anticipation of dynamically arriving goals; (2) finding escape plans that put agents in safe places; (3) or hiding their true goal to a potential observer. We consider two complimentary approaches to solve these problems by means of automated planning and machine learning techniques.

- In **multi-agent settings**, we want to study the dynamic generation of goals based on other agents' behavior. We focus on competitive scenarios, where we are interested in computing counterplans that prevent opponents from achieving their goals. We also study how to incorporate some of the single-agent goal reasoning techniques into these multi-agent scenarios.

## 1.3 THESIS OUTLINE

The main body of the thesis is divided into four parts:

- **Part I: Introduction and Background** motivates the problem, sets the objectives of the thesis, and provides the necessary background to correctly understand the work.
  - *Chapter 1: Introduction* presents the motivation that drives the development of this work, sets the objectives of the thesis, and outlines the document's structure.

  - *Chapter 2: Background* provides some formalisms that are orthogonal to the thesis body of work such as Classical Planning, Multi-agent Planning, and Fully Observable Non Deterministic Planning.

- **Part II: Goal Reasoning in Single-Agent Settings** introduces several approaches for agents to reason about their goals when acting in isolation.
  - *Chapter 3: Planning with Distance-based Goals* presents a novel planning task: that of finding states that fulfill some cost-related property with respect to a given set of goals. We highlight the importance of these states for many applications and propose several algorithms that solve this task.
  - *Chapter 4: Learning to Anticipate Planning Goals* presents the use of machine learning techniques to predict the appearance of planning goals. We show how this prediction allows agents to start planning before goals appear, which increases their performance in many different scenarios.

- **Part III: Goal Reasoning in Multi-agent Settings** introduces different techniques for agents to reason about their goals when acting in the same environment as other agents.
  - *Chapter 5: Counterplanning using Goal Recognition and Landmarks* presents a framework that allows agents to dynamically generate their goals based on other agents' behavior. We focus on competitive scenarios, where we are interested in computing counterplans that prevent opponents from achieving their goals.
  - *Chapter 6: Anticipatory Counterplanning* presents an approach that combines Chapter 3, 4 and 6 to maximize the odds of preventing opponents from achieving their goals.

- **Part IV: Conclusions and Future Work** presents our conclusions and proposes avenues for future research.
  - *Chapter 7: Conclusions* discusses the main results and conclusions drawn from the thesis.
  - *Chapter 8: Future Work* outlines some future research directions.

- **Part V: Appendix** includes some works carried out during the thesis that, although having a clear connection with the main ideas introduced in the dissertation, are the fruit of collaboration with other researchers. The appendices also include some extra experiments, results and domains from the chapters included in the dissertation.

# 2 BACKGROUND

*"Our goals can only be reached*
*through a vehicle of a plan,*
*in which we must fervently believe,*
*and upon which we must vigorously act.*
*There is no other route to success."*

*- Pablo Picasso*

In this chapter we provide some background concepts needed to understand the rest of the thesis. First, we introduce Automated Planning and define some of its variations by relaxing assumptions. We start with classical planning, which assumes a single agent acting in a full observable environment and whose actions are deterministic. Then, we present multi-agent planning, where planning for a set of agents acting in the same environment is needed. We will see how (under some assumptions) the world becomes non-deterministic in this case, presenting solution concepts that differ from the ones typically considered in classical planning.

Then, we introduce Goal Reasoning, which strives for agents able to reason about their goals. They can do it in different ways: formulate new goals, self-select which goals to pursue, or transform goals, among many others. We briefly describe some of these works and focus on Goal Recognition, which can be framed within Goal Reasoning as the process agents can follow to either infer other agent's goals; or reason about how their behavior can be inferred by other agents.

## 2.1 AUTOMATED PLANNING

AI is generally divided into two main subcategories: *model-free* and *model-based* techniques. The firsts are usually referred as *learners*, while the latters are known as *solvers* (Geffner, 2018). The differences between *model-free learners* and *model-based solvers* are reminiscent of current accounts in psychology that describe the human mind as made of two interacting systems or processes: a System 1, which is fast, intuitive and specialized; and a System 2, which is slow, deliberative and general (Evans & Stanovich, 2013; Kahneman, 2011).

Automated Planning (Ghallab et al., 2004) falls within the model-based category. In a general definition, it is the task of generating a sequence of actions, namely a *plan*, such that, when applied to a given initial state, it results in a state where some given goals are true. There exist many real-world applications of AP that range from greenhouse logistics (Helmert & Lasinger, 2010) to intermodal transportation (García et al., 2013) or robot operation (Cashmore et al., 2015), among others.

### 2.1.1 *Classical Planning*

There are many planning models depending on their assumptions about the agent(s) and the environment. The most basic model is classical planning, where complete control and knowledge of the environment is assumed:

- Deterministic actions outcome: the effect of the agent's actions is always the same and it is known in advance.
- Fully-observable environment: the environment can only change due to actions performed by the agent, who fully knows the state of the world.

**Definition 2.1.** *Formally, a **single-agent** STRIPS **planning task** can be defined as a tuple* $\Pi = \langle F, A, I, G \rangle$*, where:*

- $F$ *is a set of propositions.*
- $A$ *is a set of instantiated actions.*
- $I \subseteq F$ *is an initial state.*
- $G \subseteq F$ *is a set of propositions.*

A state consists of a set of propositions $s \subseteq F$ that are true at a given time. A state is totally specified if it assigns truth values to all the propositions in $F$, as the initial state $I$ of a planning task. A state is partially specified (partial state) if it assigns truth values to only a subset of the propositions in $F$, as the goals $G$ of a planning task.

**Definition 2.2.** *A state* $s$ *of a planning task* $\Pi$ *is a **goal state** iff all the propositions in* $G = \{p_i, \dots, p_n\}$ *are true in* $s$*, i.e.,* $G \subseteq s$.

Each action $a \in A$ is composed of a set of preconditions (pre($a$)), which represents the literals that must be true in a state to execute an action $a$, and a set of effects (eff($a$)), which represents the literals that become true (add($a$) effects) or false (del($a$) effects) in the state after the execution of action $a$.

The execution of an action $a$ in a state $s$ is defined by a function $\gamma$ such that $\gamma(s, a) = (s \setminus \text{del}(a)) \cup \text{add}(a)$ if $\text{pre}(a) \subseteq s$, and $s$ otherwise ($a$ cannot be applied). The output of a planning task is a sequence of actions, namely plan, $\pi = (a_1, \dots, a_n)$. The execution of a plan $\pi$ in a state $s$ can be defined as:

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \dots, a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases} \tag{2.1}$$

A plan $\pi$ is valid for solving $\Pi$ iff $G \subseteq \Gamma(I, \pi)$. Thus, $\Gamma(I, \pi)$ is a final state that fulfills the property of all propositions in the goal being true. The cost of a plan is commonly defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$, where $c : A \rightarrow \mathbb{R}_{\geqslant 0}$ is a non-negative action cost function. A plan with minimal cost is called optimal. Classical planning turns out to be PSPACE-COMPLETE (Bylander, 1994). However, the use of different heuristics and pruning (among other techniques) allows planners to efficiently solve planning tasks in many cases.

The planning community uses a standard language to specify planning tasks in a domain-independent fashion: the Planning Domain Definition Language (PDDL) (McDermott et al., 1998). Although there exist many versions of PDDL (Haslum et al., 2019), all of them separate the problem definition into two parts: domain and problem. The *domain* expresses the set of available actions, as well as the object types, predicates, and functions; the *problem* contains the initial state and goals of the task. Listing 1 and 2 show part of the domain model and the problem description of a BLOCKS-WORLD planning task as they are represented in PDDL. The initial state of the task is also graphically shown in Figure 2.1. In this well-known domain, an agent can pick-up, drop, stack and unstack blocks with the aim of producing a blocks' configuration that meets the goals.

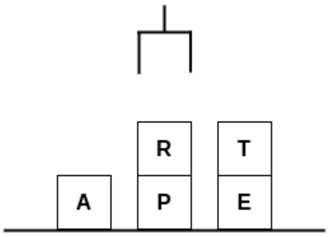


Figure 2.1: Initial state of a BLOCKS-WORLD planning instance.

```
(define (domain blocks)
  (:requirements :strips :typing)
  (:types block)
  (:predicates
    (on ?x1 ?x2 - block)
    (ontable ?x1 - block)
    (clear ?x1 - block)
    (handempty)
    (holding ?x1 - block)
  )

  (:action pick-up
    :parameters (?x1 - block)
    :precondition (and
        (clear ?x1)
        (ontable ?x1)
        (handempty)
    )
    :effect (and
        (not (ontable ?x1))
        (not (clear ?x1))
        (not (handempty))
        (holding ?x1)
    )
  )
```

Listing 2.1: Excerpt of the blocks-world domain model.

```
(define (problem p0)
  (:domain blocks)
  (:objects
    p a t e r - block
  )

  (:init
    (handempty)
    (ontable a)
    (clear a)
    (ontable p)
    (clear r)
    (on r p)
    (ontable e)
    (clear t)
    (on t e)
  )

  (:goal (and
    (clear t)
    (on t e)
    (on e a)
    (ontable a)
  ))
)
```

Listing 2.2: Blocks-world problem instance.

In this case, the goal entails to build the word TEA using the available blocks. A valid (and optimal) plan $\pi$ that solves this task is:

$$\pi = \begin{pmatrix} \text{(unstack t e)}, \\ \text{(stack t r)}, \\ \text{(pick-up e)}, \\ \text{(stack e a)}, \\ \text{(unstack t r)}, \\ \text{(stack t e)} \end{pmatrix} \qquad \text{with} \qquad c(\pi) = 6$$

There exist many different approaches to solving planning problems in the literature. Some approaches compile the planning problem to propositional satisfiability (SAT) (Rintanen, 2004b). Others take advantage of succint data structures, such as in the case of symbolic search (Bryant, 1986; Edelkamp & Helmert, 2001). However, most planners employ heuristic search (Bonet & Geffner, 2001) to compute plans that reach goals from a given initial state. This search is typically conducted in a forward fashion, i.e., states are expanded from the initial state until a goal state is found. However, there also exist alternatives that do the opposite: expand states backwards from the goal to the initial state (Alcázar et al., 2013); or even perform the search in both directions (Torralba et al., 2018). Exploring the state space is impractical in most planning tasks due to large state spaces, regardless the direction of the search. That is why planners usually employ heuristics to drive the search, reducing the number of states explored until a solution is found.

Best-First Search (Hart et al., 1968) is the most frequently used algorithm when performing forward heuristic search. It explores the state space by ranking all the current reachable nodes according to the following heuristic function:

$$f(n) = g(n) + h(n) \tag{2.2}$$

where $g(n)$ is the cost of reaching the state represented by the node $n$ from the initial state; and $h(n)$ is the cost of reaching a goal state from the current state. The algorithm starts expanding nodes from the initial state, generating successors by applying the applicable actions to the current state. These successors are then ranked according to their $f(n)$ value in an open list. At each step, the node with $min(f(n))$ is removed from the open list. This process is repeated until a goal state is found; or the open list is empty. The algorithm is guaranteed to return optimal solutions, i.e., shortest paths, iff the employed heuristic is admissible. A heuristic is admissible iff it never overestimates the actual cost of reaching a goal state. If we consider $h^*$ as the actual optimal cost (perfect estimator), we formally have that a heuristic $h$ is admissible iff:

$$\forall s, h(s) \leqslant h^*(s) \tag{2.3}$$

#### 2.1.1.1 *Landmarks*

In the context of classical automated planning, landmarks were initially defined as sets of propositions that have to be true at some time in every solution plan (Hoffmann et al., 2004). Formally:

**Definition 2.3.** *Given a planning task* $\Pi = \langle F, A, I, G \rangle$, *a formula* $L_\Pi \subset F$ *is a* ***fact landmark*** *of* $\Pi$ *iff* $L_\Pi$ *is true in some state along all valid plans executions that achieve* $G$ *from* $I$.

In the case of the BLOCKS-WORLD example depicted in Figure 2.1 and Listings 1 and 2, `(holding t)` would be a landmark. This is because the agent must hold that block at some point in all the plans that reach a goal state.

Landmarks have been widely used in planning. Some examples include treating landmarks as sub-goals in the search (Porteous et al., 2001); or use landmarks as heuristics (Helmert & Domshlak, 2009; Richter et al., 2008). As we will later see, in this thesis we use landmarks to generate planning goals from scratch.

### 2.1.2 *Multi-agent Planning*

Multi-agent planning (MAP) aims at solving problems in which several agents act in the same environment. In this case, planning capabilities are not centralized in a single agent but distributed among all the agents. Therefore, agents not only need to reason about their goals and actions but how them are affected by the behavior of other agents. Therefore, in some cases, the environment becomes non-deterministic due to the agent not knowing in advance which actions the other agents are executing. For these problems, classical plans are no longer valid solution concepts, and we need to extend these definitions to account for the non-determinism introduced by other agents.

We can divide multi-agent systems (Wooldridge, 2009) into different categories depending on (1) how agents perceive their pairs; and (2) agents' goals. Let us exemplify these different scenarios by using the soccer domain employed by Bowling, Jensen and Veloso (2003) and depicted in Figure 2.2. We have two agents A and B. Each of them occupies one tile of the field. Agent A begins in possession of the ball (depicted with a circle). Each agent can move to each compass direction (N, S, E, and W) provided that the target cell is empty; or can wait, holding its position (H). If the target cell is occupied, then the agent does not move and loses the ball in case of having it. For agent A, losing the ball terminates execution as a failure. The goal for agent A is to score a goal by moving into either of the two goal squares (in green) in possession of the ball. Although this is clearly a MAP problem, there exist different solutions depending on agent B's goals.

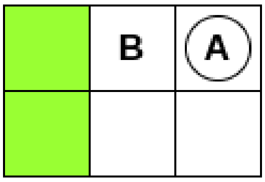


Figure 2.2: Example of a grid where two agents are playing soccer.

Consider the case that A and B are teammates sharing agent A's goal. A simple solution would be as follows: agent A holds at the initial state while its teammate moves south out of its way. Then A moves west into the goal. This plan is guaranteed to reach the goal in a finite number of steps, and agent A should not fear that the other agent will move in front of it, since they are teammates cooperating. This is the case of cooperative MAP, where agents are willing to collaborate in order to achieve a goal, which is typically shared among them (Brafman & Domshlak, 2008). The above example is just a simplification, and do not address crucial issues in cooperative multi-agent planning such as the concurrent execution of actions (Furelos-Blanco & Jonsson, 2019), goal (or subgoals) allocation for each agent (Borrajo, 2013; Borrajo & Fernández, 2019; Luis et al., 2020), communication between the agents (Maliah et al., 2017) or privacy models (Tozicka et al., 2017), among others (Torreño et al., 2018).

Now, consider the case in which B executes actions randomly. A sensible plan would be to hold position until agent B reaches the bottom right state (which it will eventually do, since it is moving randomly). From this position regardless of the action of the other agent, a plan of two consecutive west actions is guaranteed to achieve the goal. Other plans risk agent B's random actions causing it to move in the way, resulting in A loosing the ball and failing. Note that although this plan guarantees reaching the goal, it does not necessarily guarantee achievement in a finite number of steps, as it requires A to hold its position until B moves to the bottom right square. Note that in this case there is not a specific relationship between both agents. However, A should take into account B's possible executions if it wants to succeed.

Neither of these plans though have any guarantees if agent B is actively wanting to prevent A from succeeding. If agent B simply holds it ground, none of the previous plans would make A achieve its goal. In this situation, game-theoretic approaches that provide worst-case guarantees are more appropriate (Jensen et al., 2001). One such plan is for A to randomly select between holding and moving north/south until the other agent is not in front of it. Then, move west into the goal state. This plan has no guarantee of success as the opponent may still move in front of it while it advances toward the goal, causing the ball to be lost. Cimatti et al. (2003) identified three different classes of solutions to problems with non-determinism:

- Weak solutions, which achieve the goal, but without guarantees. In other words, weak plans are *optimistic*, and will only achieve the goal if *everything goes as expected*.
- Strong solutions, which guarantee goal achievement regardless of environment initial or intermediate states.
- Strong-cyclic solutions, which guarantee goal achievement relying on a *fairness* assumption. Roughly speaking, a *fair action* is one in which all effects occur infinitively often when the action is executed infinitively many times in the same state.

For example, it is known that by tossing a (truthful) coin many times, heads will eventually appear, so it would be a fair action.

Although other frameworks exist, we consider a first-person view of planning in a setting where the world is fully known and observable to all agents. Following Bowling, Jensen and Veloso (2003) and Brafman and Domshlak (2008) formulations, we define MAP as follows:

**Definition 2.4.** *A **multi-agent planning task** is a tuple MAP* $= \langle N, F_i, A_i, I_i, G_i \rangle$*, where:*

- $N = \{1, \dots, n\}$ *is a set of agents.*
- $F_i$ *is the set of propositions of agent* $i \in N$.
- $I_i$ *is the initial state of agent* $i \in N$.
- $A_i$ *is the set of actions agent* $i \in N$ *can execute.*
- $G_i$ *is the goal for agent* $i \in N$.

As we have previously seen, solutions to MAP tasks are defined in terms of weak, strong and strong-cyclic solutions, which provide different goal achievement guarantees. However, these solutions vary depending on the assumptions we make about the duration of agent's actions. Most works assume agents act concurrently, with all the actions having the same duration for all the agents (Bowling et al., 2003; Furelos-Blanco & Jonsson, 2019; Muise et al., 2016). Under these assumptions, the most prevalent approach is to form *joint actions* as the cross product of all individual actions: $\mathcal{A} = A_1 \times \ldots \times A_n$. $A_i$ denotes the individual action $a \in A_i$ the agent $i$ executes in the joint action space $\mathcal{A}$ in a given state. We assume $A_i$ contains a *no-op* action, that can be always executed and does not produce any effect, i.e., pre(no-op) = eff(no-op) = ∅. Joint actions constitute a choice from each agent as to what individual action they perform. Bowling *et al.* (2003) assume that the applicability of an individual action does not depend on the individual actions of other agents. However, in general, nothing prevents two actions $a_1$ and $a_2$ of different agents from having conflicting preconditions or effects.

To ensure that joint actions have well-defined effects, it is necessary to impose concurrency constraints that model whether a set of actions can be performed in parallel. There are several concurrency models. Some of them define concurrency constraints on actions that have the same *object(s)* in their preconditions or effects (Crosby et al., 2014). Others define these constraints in terms of *actions* (Boutilier & Brafman, 2001). In this work, we will assume the *propositions*-based concurrency constraints introduced in PDDL 2.1 (Fox & Long, 2003). Two actions $a_1$ and $a_2$ can only be applied concurrently in a state $s$ iff $\text{pre}(a_1) \cup \text{pre}(a_2) \subseteq s$, and they do not interfere. We define two non-interfering actions as follows:

**Definition 2.5.** *Two actions* $a_1, a_2$ *are **non-interfering** iff:*

$$\mathit{pre}(a_1) \cap (\mathit{add}(a_2) \cup \mathit{del}(a_2)) = \mathit{pre}(a_2) \cap (\mathit{add}(a_1) \cup \mathit{del}(a_1)) = \emptyset \tag{2.4}$$

$$\mathit{add}(a_1) \cap \mathit{del}(a_2) = \mathit{add}(a_2) \cap \mathit{del}(a_1) = \emptyset \tag{2.5}$$

We will use $\gamma_J$ to represent the joint execution of two actions.

**Definition 2.6.** *The **joint execution of two actions** $a_1, a_2$ in a state $s$ results in a new state given by:*

$$\gamma_J(s, a_1, a_2) = \begin{cases} (s \setminus \mathit{del}(a_1) \cup \mathit{del}(a_2)) \cup \mathit{add}(a_1) \cup \mathit{add}(a_2) & \textit{if } a_1 \textit{ and } a_2 \textit{ do not interfere} \\ s & \textit{if } a_1 \textit{ and } a_2 \textit{ interfere} \\ \gamma(s, a_1) & \textit{if } a_2 = \textit{no-op} \\ \gamma(s, a_2) & \textit{if } a_1 = \textit{no-op} \end{cases} \tag{2.6}$$

If two actions interfere, we can assume that none of the actions are executed, as defined above, or that one of them has priority, as we will see later. In that case we would change the second part of the inequality to $\gamma(s, a_1)$, if $a_1$ was the highest priority action.

Similarly, we define the joint execution of two plans as the iteration of the joint execution of the actions of those plans.

**Definition 2.7.** *The **joint execution of two plans** $\pi_j, \pi_k$ in a state $s$ results in a new state given by:*

$$\Gamma_J(s, \pi_j, \pi_k) = \begin{cases} \Gamma_J(\gamma_J(s, a_{j,1}, a_{k,1}), (a_{j,2}, a_{k,2}), \dots, (a_{j,n}, a_{k,n}))) & \textit{if } \pi_j, \pi_k \neq \emptyset \\ \Gamma(\gamma(s, a_{k,1}), (a_{k,2}, \dots a_{k,n})) & \textit{if } \pi_j = \emptyset \\ \Gamma(\gamma(s, a_{j,1}), (a_{j,2}, \dots a_{j,n})) & \textit{if } \pi_k = \emptyset \end{cases} \tag{2.7}$$

Given these definitions, we define a strong plan in the context of MAP assuming two agents as follows:

**Definition 2.8.** *A plan $\pi_1$ that solves an agent's planning task $\Pi_1$ is a **strong plan** iff its joint execution with any sequence of actions $\pi_k$ that an agent $\Pi_2$ can execute always achieve the goal $G_1 \in \Pi_1$:*

$$\forall \pi_k \in \Pi_2, G_1 \subseteq \Gamma_J(I_1, \pi_1, \pi_k) \tag{2.8}$$

Any other plan that does not meet this criteria will be weak.

## 2.2 GOAL REASONING

It is generally acknowledged that goal-directed behavior is a hallmark of intelligence (Newell, Simon, et al., 1972). This interpretation has motivated AI research, from early problem solvers to modern automated planners. Goals are a central topic in other research areas such as Belief-Desire-Intention (BDI) agents (Rao & Georgeff, 1995). However, goals have been longer put aside in planning, considering them as static entities provided by external users. This restrictive vision limits the applicability of automated planning in real-world domains, and puts in between the *autonomy* and *intelligence* of planning agents.

Goal reasoning (Aha, 2018; Cox, 2007; Molineaux et al., 2010; Norman & Long, 1995) challenges this interpretation and strives for autonomy of goals. In addition to autonomy of actions, an intelligent agent should be able to reason and deliberate upon the goals it is pursuing. Goal reasoning capabilities may prove useful in many applications where long-term autonomy is required and/or the environment is under constant change such as in underwater or space missions, rescue operations or games, among many others.

Goal reasoning aims at providing solutions to the problems that the following questions arise (Vattam et al., 2013):

- *What* is a goal?
- *Where* does a goal come from?
- *Why* self-formulate a goal?
- *When* is a goal formulated?
- *How* are goals formulated and/or managed?

Answering the first question is easy, since we are assuming planning-agents and hence planning goals, which we have already defined in the previous section. Regarding goal's origin, they are typically given externally by a user. However, goal reasoning focuses on goals that are self generated by agents, since it aims at augmenting agent's autonomy and reasoning capabilities. This answer lead us to also provide an answer to the third question regarding why agents would be interested in self-formulating goals. In the rest of the section, we present the main corpus of work in goal reasoning in the context of planning, which focuses on answering the two last questions: when agents need to formulate new goals, and how are these goals formulated and/or managed.

### 2.2.1 *Goal Formulation Triggers*

Answering *when* agents need to (re-)formulate their goals amounts to decide (1) in which points of the reasoning process may an agent change its goals; and (2) under which circumstances this should occur. Here we present a non-exhaustive list of works that address this problem.

An agent could change its goals when an active plan fails. Plans are generated using *domains*, which are abstract models of the world in which they are meant to be deployed. If these models do not account for all the possible environment's contingencies (as in the case of classical planning), plans may fail upon execution. When a plan fails, most planning approaches try to either repair the plan or re-plan from scratch (Bonet & Geffner, 2011; Fox et al., 2006; Gerevini & Serina, 2000; Nebel & Koehler, 1995). However, replanning is typically performed without changing the goal, assuming that the new environment where the original plan failed does not preclude the original goal achievement. In contrast, goal reasoning systems aim at generating new goals, given the current failure. For example, ARTUE (Klenk et al., 2013) finds discrepancies using a set difference operation between the expected and observed literals. When a discrepancy is detected, its anomaly response mechanisms perform anomaly explanation and goal formulation.

An agent could also change its goals when the environment changes. Instead of focusing solely on the current plan and its execution, agents may monitor the entire environment to determine if new goals should be considered. By monitoring the environment, agents can match the current situation with their expectations (Muñoz-Avila et al., 2019), reacting accordingly. For example, INTRO (Cox, 2007) uses a rule-based model to generate expectations and detect discrepancies in a Wumpus World environment. Kurup et al. (2012) introduce a cognitive model of expectation-driven behavior. It generates future states called expectations, matches them to observed behavior, and reacts when a difference exists between them. Agents can also monitor the environment to detect opportunities, i.e., goals with higher value than the agent's actual goal. In those cases, agents need to decide if they forget the previous goal and try to achieve the new one; or try to replan in order to get bot the previous and the opportunistic goal. Some works that exploit opportunistic goals include robotic applications (González et al., 2015; Schermerhorn et al., 2009) and underwater autonomous missions (Cashmore et al., 2018).

### 2.2.2 *Methods for Goal Formulation*

After answering the *when*, we now try to answer the question on *how* agents can (re-)formulate their own goals.

Most works formulate new goals based on the current state of the world. For example, consider an autonomous car whose initial goal is to drive from point A to point B. At some point over its trip, it realizes that its low-battery indicator is flashing, so the car changes its goal to recharge. Some works use rule-based methods in the form $\langle \text{condition}, \text{goal} \rangle$ to formulate new goals. When the condition is met (low-battery) the new goal (recharge) is triggered. This is the case of ICARUS (Choi, 2010), ARTUE (Klenk et al., 2013), or its extension M-ARTUE (Wilson et al., 2013), among others. In realistic domains it is often infeasible to provide goal formulation knowledge for every situation, i.e., address all possible $\langle \text{condition}, \text{goal} \rangle$ pairs. To overcome this, T-ARTUE (Powell et al., 2011) and EISBot (Weber et al., 2012) learn this knowledge from humans by interacting with them through answers to queries or human demonstrations. Agents can later use these interactions with humans to provide explanations (Cashmore et al., 2019; Chakraborti et al., 2020) on their goal's decisions (Dannenhauer et al., 2018), and/or store this interactions to later generate new goals using a case-based reasoning approach (Muñoz-Avila et al., 2010).

But goal reasoning is not limited to the generation of new goals. It is also related to the problem of selecting which goals an agent should pursue. One can use the same rule-based approach previously described to switch among a set of predefined goals. A more interesting and principled approach for goal selection is that of oversubscription planning (Do et al., 2007; Domshlak & Mirkis, 2015; Smith, 2004). In classical planning, plans have to achieve all the propositions appearing in the goals. Goals are treated as *hard* requirements, and if not such a plan exists, the planning process fails. On the other hand, oversubscription planning relaxes this all-or-nothing constraint, and treat goals as *soft* requirements or rewards. Plans that solve oversubcription planning tasks achieve the *best* subset of soft goals possible. This goal management strategy allows agents to automatically select the goals that maximize the total utility.

### 2.2.3 *Goal Recognition*

Goal recognition is the problem of inferring an agent's goal by observing its behavior. It falls within the bigger scope of plan, activity and intent recognition (PAIR), which has captured the attention of several computer science areas since the early beginnings (Kautz & Allen, 1986). PAIR techniques have been successfully used in many applications and areas such as smart house environments (Roy et al., 2011; Wu et al., 2007), assisted cognition (Pentney et al., 2006), or trajectory prediction (Wiest et al., 2012).

There exist many different approaches for goal recognition: bayesian inference (Albrecht et al., 1997; Charniak & Goldman, 1991), grammars (Geib & Goldman, 2009), and probabilistic solutions (Pynadath & Marsella, 2005), amongst many others (Albrecht & Stone, 2018). One of the most prevalent approaches involves having *plan libraries* that store set of plans (Sukthankar et al., 2014). When a new sequence of observations arrives, it is matched against the plan library. The winning plan (and/or goal) returned by these approaches is the one from the library that best matches the observation sequence. One of the main drawbacks of plan library approaches is that sometimes it is really difficult to build the plan library, i.e., acquire a reasonable number of plans that achieve different goals (Pattison & Long, 2010).

Ramírez and Geffner (2009) proposed a planning-based approach for goal recognition that overcomes the inherent limitations of plan library approaches. In this case, plans are not longer stored but generated as they are needed using a *domain theory* (i.e., a planning domain). Their approach relies on the principle of rationality, where agents are assumed to take optimal or *least sub-optimal* (Ramírez & Geffner, 2010) plans towards their goals. There exist many subsequent works on planning-based goal recognition inspired by Ramírez and Geffner work (E-Martín et al., 2015; Kaminka et al., 2018; Masters & Sardiña, 2019; Pereira et al., 2020). However, since we mainly use Ramírez and Geffner approach in this dissertation, we now review it in detail.

#### 2.2.3.1 *Goal Recognition as Planning*

Ramírez and Geffner (2009) define the goal recognition problem as the *inverse of planning*. The aim is to discriminate an agent's goal from a set of candidate goals, given a set of the agent's performed actions. They formally define a goal recognition problem as follows:

**Definition 2.9.** *A* ***goal recognition problem*** *is a tuple* $T = \langle P, \mathcal{G}, \mathcal{O} \rangle$ *where:*

- $P = \langle F, A, I \rangle$ *is a planning domain and initial conditions;*
- $\mathcal{G}$ *is the set of possible goals* $G$, $G \subseteq F$*; and*
- $\mathcal{O} = (o_1, \dots, o_m)$ *is an observation sequence with each* $o_i$ *being an action in* $A$.

The solution to this problem is a set $\mathcal{G}^*$ made of the goals $G_i \in \mathcal{G}$ having an optimal plan $\pi$ consistent with the observation sequence $\mathcal{O}$. A plan $\pi = a_1, \ldots, a_n$ is said to be consistent with an observation sequence $\mathcal{O} = o_1, \ldots, o_m$ iff there exists a monotonic function $f$ that maps the observation indices into the action indices such that $a_{f(j)} = o_j \forall j \in \{1, \ldots, m\}$.

This approach is limited by the assumption of agents acting optimally. However, agents often take suboptimal paths to achieve their goals. To overcome this limitation, Ramírez and Geffner (2010) extended the work by presenting an alternative probabilistic framework, based on how much the observed actions contribute to achieving a given goal. The problem remains the same as in Definition 2.9, adding now $Pr$ to the tuple $T$, where $Pr$ is a prior probability distribution over the goals in $\mathcal{G}$. The solution now is not a set of goals, but a posterior probability distribution over each goal $G \in \mathcal{G}$. Goals whose plans *best satisfy* observations have a higher probability of being achieved by the agent. The posterior goal probabilities $Pr(G|\mathcal{O})$ can be characterized using Bayes Rule as:

$$Pr(G|\mathcal{O}) = Pr(\mathcal{O}|G)Pr(G) \tag{2.9}$$

where $Pr(G)$ is the prior probability distribution over $G \in \mathcal{G}$, and $Pr(\mathcal{O}|G)$ is the likelihood of observing $\mathcal{O}$ when the goal is $G$. Ramírez and Geffner characterize the likelihood $Pr(\mathcal{O}|G)$ in terms of *cost difference* ($\Delta$) between: (1) the cheapest plan for a goal, given the observed actions already taken by the agent; and (2) the cheapest plan that the agent could have followed to reach the goal, without taking into account the observations:

$$\Delta(G, \mathcal{O}) = Cost(G|\mathcal{O}) - Cost(G|\bar{\mathcal{O}}) \tag{2.10}$$

By comparing cost differences for all $G \in \mathcal{G}$, they propose to generate a probability distribution across $\mathcal{G}$ with the following important property: the lower the cost difference for a particular goal, the higher its probability. This probability distribution is computed using a Boltzman distribution as follows:

$$Pr(G|\mathcal{O}) = \alpha \frac{\epsilon^{-\beta\Delta(G,\mathcal{O})}}{1 + \epsilon^{-\beta\Delta(G,\mathcal{O})}} Pr(G) \tag{2.11}$$

where $\alpha$ is a normalizing constant and $\beta$ is a positive constant. In their paper, Ramírez and Geffner show how the constraints $O$ and $\bar{O}$ can be compiled into a new planning task, so that standard planners can be used to find plans that either account or disregard the observations.

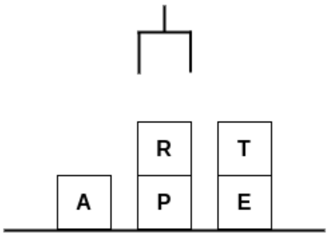


Figure 2.3: Initial state of a BLOCKS-WORDS planning instance.

To better illustrate Ramírez and Gefner (2010) approach, let us go back again to our BLOCKS-WORLD example, depicted again on Figure 2.3. Let us assume the aim of the agent is to build words using the available blocks. Let us consider that the set of possible goals the agent may want to build $\mathcal{G}$ is the following:

$$\mathbf{G_1} = \langle\text{(clear t) (on t e) (on e a) (ontable a)}\rangle \ \text{\#word TEA}$$
$$\mathbf{G_2} = \langle\text{(clear a) (on a t) (on t e) (ontable e)}\rangle \ \text{\#word ATE}$$

The observed sequence of actions is $\mathcal{O} = \{$(pick-up t) (stack t r)$\}$. Given this set of observations, we have the following costs:

$$\mathrm{Cost}(G_1|\mathcal{O}) = 6, \mathrm{Cost}(G_1|\bar{\mathcal{O}}) = 6 \rightarrow \Delta(G_1|\mathcal{O}) = 0$$
$$\mathrm{Cost}(G_2|\mathcal{O}) = 6, \mathrm{Cost}(G_2|\bar{\mathcal{O}}) = 2 \rightarrow \Delta(G_2|\mathcal{O}) = 4$$

The cost difference for $G_1$ is $0$, since the observation is part of an optimal plan to build the word TEA. However, the cost difference for $G_2$ is 4. This is because the word ATE could be built at cost 2 from the initial state, and the observations increase the cost of reaching $G_2$ to 6 (the agent is deviating from achieving that goal). Hence, $G_1$ will have a higher probability of being the true goal the agent is pursuing than $G_2$.

A major drawback of this approach is that it is computationally expensive: we need to compute two plans for every goal in $\mathcal{G}$, one complying with the observations and other without taking them into account. This makes the approach impractical when the number of candidate goals increases.

## 2.3 SUMMARY

In this chapter we have firstly presented and analyzed different automated planning frameworks. As we have seen, each of them makes different assumptions about the environment, but they do have something in common: they use models of the world to generate plans that achieve goals from a given initial state. Over the years, planning research has mainly focused on actions and plans, leaving goals in the background and considering them as an external input that remains static over the execution. This setting is too restrictive and limits the applicability of automated planning in real-world domains, and puts in between the *autonomy* and *intelligence* of planning agents. Goal reasoning tries to overcome this situation by situating goals in the centre of the research agenda. We have presented different works and approaches that try to answer questions on *when* should agents (re-)formulate their goals, and *how* they can do it.

This thesis provides work answering some open questions from the goal reasoning and planning communities.

**Integrated Goal Reasoning and Planning**. Most previous works on goal reasoning understand goal reasoning as an external process to the planning one. As we have seen, most research presents different systems and architectures such as ARTUE (Klenk et al., 2013) or MIDCA (Cox et al., 2016) comprising independent modules with different capabilities. In those architectures, goal reasoning modules are typically outside planning ones, and their interaction is limited in some cases. In that sense and in line with the claims by Paredes and Ruml (2017), we aim at considering goal reasoning as a form of planning, rather than as an independent process.

**Proactively formulating goals**. Most previous works formulate new goals based on the current state of the environment. This approach, although valid for many applications, limits the capabilities of planning agents and make them to behave reactively. We aim at investigating how to make agents proactive, starting to reason about goals that are *likely* to appear in the future. For instance, going back to our autonomous car example, an agent could have learnt over time that she runs out of battery when going from A to B, hence recharging the battery before the indicator starts to flash.

**Multi-agent Goal Reasoning: Integrating Goal Reasoning and Goal Recognition**. Very few works consider other agents' actions when (re-)formulating goals. When they do consider other agents' behavior, they tend to require having rules or predefined libraries that suggest which goal/plan to follow based on the observations. This greatly limits the autonomy of agents. We believe that recent advantages in planning-based goal recognition can be used to infer other agents' goals and plans in order to better (re-)formulate goals, and we will further investigate their connections.

Part II

# GOAL REASONING IN SINGLE-AGENT SETTINGS

# 3 PLANNING WITH DISTANCE-BASED GOALS

> *"Part of the issue of achievement is to be able to set realistic goals, but that's one of the hardest things to do because you don't always know exactly where you're going, and you shouldn't."*
>
> *- George Lucas*

Automated planning deals with the task of finding a sequence of actions, namely a plan, which achieves a goal state from a given initial state. A goal state is a state where a conjunction of propositions is true. Therefore, the stopping criteria of planning algorithms relates to states that fulfill that property. Let us call *final* or *goal states* to the states that fulfill that property. In this chapter, we study other relevant properties of states that define different classes of final states. Instead of considering only one goal (composed of a conjunction of propositions), we will use as input a set of goals, as in planning-based goal recognition works (Ramírez & Geffner, 2009). We will study properties such as finding final states that are as close as possible to the set of goals, or final states that are as far as possible from those goals. We refer to these states as goal-related states. These new properties allow our algorithms to solve classes of real-world problems that have not yet been addressed within the field of automated planning.

In order to solve these new tasks, we define planning algorithms that provide final states fulfilling a given distance(cost)-related property with respect to the set of goals. We will use the terms distance and cost indistinctly. These planning algorithms could also compute plans that fulfill a property related to the set of goals. For instance, in the case of reaching a state as far as possible from a set of goals, the planning algorithms may also return a plan that reaches that state, and at the same time it maximizes the distance of any goal during the whole plan execution.

**Example 3.1.** *Figure 3.1 shows an example of an emergency response domain. Fire flames indicate locations where a fire can dynamically start because of some potential source of ignition. An agent (represented in grey) can move through adjacent white cells. It can also cross the river (depicted in blue cells) with the canoe. Depending on the purpose of the agent, it may reason differently about the set of fires. Let us assume that the agent is a ranger whose objective is to set an emergency camp as close as possible to all fires, in order to minimize the cost (time) of putting out any fire that might arrive. In this case, each goal would be putting out each fire (each goal is made of only one proposition in this example). The final state would be one that minimizes the distance to all these goals, and we refer to it as the centroid state of the task. Now, let us assume that the agent is a family that wants to escape from the fires. In this case, they would want to*

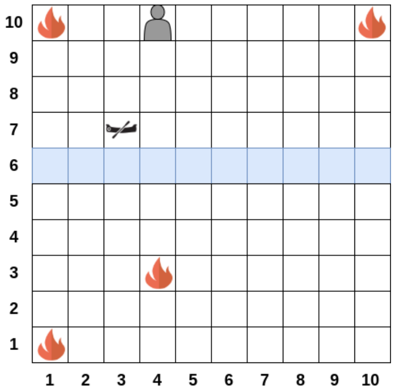


Figure 3.1: Emergency response domain. Fired cells depict potential fires. Blue cells represent a river that people can cross by using the canoe.

*reach the safest possible place and do it by following the safest possible plan. Therefore, the fires can be seen and represented as planning goals and the agent objective (final state) is defined according to them.*

Other domains where finding states fulfilling these properties might be useful include: areas to be patrolled by police (or robots) where potential incidents might happen; source locations for picking up customers by taxis, or packages by delivery companies; choosing a safe plan to protect sensitive electronic data from potential threats in a network; or games where we want to avoid or escape from a set of opponents or dangerous situations.

If the agents in domains we would be interested in take actions in an $n$-dimensional space, we could use classical algorithms that compute centroids, for instance. These domains use the concept of distance between two points to compute solutions to those tasks. However, some domains do not deal with these euclidean spaces, and the distance to a goal is not necessarily determined by the topological distance. Therefore, based on these algorithms, we will use automated planning to compute (or estimate) distances between states (goals).

To our knowledge, this is the first work that computes states that fulfill certain cost-related requirements with respect to a given set of goals. There are some works that deal with similar tasks or where the states we propose may have a direct impact:

- Goal obfuscation (Keren et al., 2016; Kulkarni et al., 2019), where agents are interested in hidding their true goal to an observer. By computing a specific type of plan towards the centroid of the set of goals, we are maximizing the number of goals with respect to which the plan is obfuscated. We do not need to select a subset of goals we want to obfuscate, as previous works need.

- Anticipatory planning (Burns et al., 2012; Fuentetaja et al., 2018), where agents are interested in approaching goals before they arrive. Again, by computing states such as centroids, we can locate agents in states to better reach the goals once they arrive.

- Multi-agent scenarios where agents compete or collaborate with others. By computing some of our states and plans, we can locate agents in states from which they can better block their opponents (Pozanco et al., 2018a; Speicher et al., 2018) or help their peers (Kulkarni et al., 2019; Kvarnström & Doherty, 2010).

The main contributions of this chapter are:

- We define a full collection of distance-based properties of final states, along with some domains and scenarios for which finding them might be useful.
- We assign weights to each goal to incorporate some sort of notion of priority.
- We focus on plans that reach these states, defining some distance-based properties that plans should have.
- We define a common algorithm with different parametrizations for solving the task of computing goal-related states and plans both optimally and suboptimally.
- We evaluate the effectiveness and scalability of our approach on different planning domains.

The rest of the chapter is organized as follows. First, we formally define our collection of goal-related states and plan characteristics. In Section 3.1, we introduce the algorithm we use to compute such states and plans. After that, we present an empirical study in several planning domains in Section 3.3. Finally, we position our contribution in the context of related work in Section 3.4 and close the chapter in Section 3.5 with discussion.

## 3.1 goal-related states

In a geometric space, generating points that fulfill some property with respect to other points is usually a simple task. For instance, the point that minimizes the distance with respect to a set of points (centroid) can be simply computed as an average of the points' coordinates. The task can be considered as simple given that: (1) every point is reachable from all others; (2) point's features (coordinates) are fully specified; and (3) the distance from one point to another can be simply calculated as the Euclidean distance (or any other similar distance) of its coordinates.

In order to generalize this task to non-geometric domains, we can generalize points to the concept of state, where a state is defined in terms of a set of propositions. The state would be $n-$dimensional, where $n$ is the number of propositions whose value can change when solving a given planning task. However, states: (1) are not necessarily reachable from all other states; (2) are not necessarily fully specified (we need to reason about partial states); and (3) the complexity of computing the distance between two states is PSPACE (Bylander, 1994).

Therefore, next we first define some reachability and distance concepts for automated planning. With these definitions at hand, we then introduce our collection of distance-based properties. These properties are based on the sequence of distances of the states with respect to the given set of goals. These distances are computed using (real or estimated) costs to reach one goal from another.

### 3.1.1 *Reachability and Distance Definitions*

For purposes of this work, we modify our previous definition of a goal to allow goal weighting.

**Definition 3.1.** *Given a planning task* $\Pi$, *a* ***weighted planning goal*** *is a tuple* $G_i = \langle p_i, w_i \rangle$, *where* $p_i \subseteq F$ *is a partial state and* $w_i \in \mathbb{R}[0,1]$ *is a number indicating the weight of* $p_i$.

This definition of goal is similar to the one employed in oversubscription planning (Domshlak & Mirkis, 2015; Smith, 2004). However, while oversubscription planning focuses on maximizing the total utility of the achieved soft goals, we are interested in finding a state that fulfills a given property with respect to a set of goals.

Since $p_i$ is a partial state, it is a conjunction of propositions that must be true in a state to be considered as a goal state. For instance, considering the problem depicted in Figure 3.1, a possible goal would be ⟨`(at person t10-10)`,0.8⟩, which indicates that positioning a person in the tile in the tenth column, tenth row has a weight of 0.8. In this case, there is only one proposition in the goal definition.

Moreover, in a classical planning task we only have one goal G to achieve. In our case, we have a **set of goals** we reason about (i.e., several $G_i$), as in planning-based goal recognition (Ramírez & Geffner, 2009). We denote this set of goals as $\mathcal{G}$. A possible set of goals for the problem depicted in Figure 3.1 would be as follows:

$$\mathcal{G} = \{\langle \texttt{(at person t10-10)}, 0.8\rangle, \langle \texttt{(at person t4-7) (at canoe t4-6)}, 1.0\rangle\}$$

That is, having the person at t4-7 and the canoe at t4-6 has a higher utility than having the person at t10-10.

In order to deal with reachability and distance between planning states, we provide the following standard definitions.

**Definition 3.2.** *Given a planning task* $\Pi$, *a* ***state*** $s$ *is* ***reachable*** *from* $I$ *(or simply reachable) iff there is at least one valid plan* $\pi$ *such that its execution from* $I$, $\Gamma(I, \pi)$, *achieves* $s$; *i.e.* $s \subseteq \Gamma(I, \pi)$. $\mathcal{R}_\Pi$ *is the set of all reachable states from* $I$ *in* $\Pi$.

Computing the distance (cost) between two states is not straightforward in automated planning. Moreover, it depends on the distance definition we use. We define the distance between states as follows:

**Definition 3.3.** *Given a planning task* $\Pi$, *the* ***distance between two states*** $s$ *and* $p$ *is the cost of the minimal cost plan* $c(\pi)$ *that achieves* $p$ *from* $s$.

This definition give us the *actual* (optimal) distance between two states. However, sometimes it will be hard to get this value, and therefore we will be interested in estimating it. We will use $\textsc{cost}(s, p, h)$ as a function that computes the cost of achieving $p$ from $s$ using the cost estimator $h$. This function can compute either the actual optimal cost $h^*$ of achieving $p$ from $s$, or an estimated cost (as the one computed by a heuristic function).

Our aim is to find states that fulfill some property related to the distance with respect to the set of goals $\mathcal{G}$. Hence, we compute $\mathcal{D}$, the **sequence of weighted distances** of a state $s$ to each goal in $\mathcal{G}$, using cost estimator $h$, as follows:

$$\mathcal{D}(F, A, s, \mathcal{G}, h) = \{d_i \| (p_i, w_i) \in \mathcal{G}, d_i = \text{cost}(s, p_i, h) \times w_i\} \tag{3.1}$$

We will refer as ComputeDistances($\Pi, \mathcal{G}, s, h$) to the procedure that computes the sequence of weighted distances of a state with respect to all goals in $\mathcal{G}$ . We can compute different statistical measures from this sequence of distances such as the average, or the maximum and minimum elements in the set. We will use later these metrics to define our collection of distance-based properties of states.

Let us return to our running example in Figure 3.1. There are four possible goals that depend on the final intended position of the agent (at a given flame): (10,10), (4,3), (1,1) and (1,10). To slightly simplify the computation, assume that all goals have a weight equal to 1. We use the Manhattan distance as the heuristic function $h$. Hence, we assume the agent can cross the river without using the canoe. If we compute the sequence of weighted distances assuming that the agent is initially located at (4,10), we have:

$$\mathcal{D}(F, A, s, \mathcal{G}, h) = \langle 6, 7, 12, 3 \rangle$$

With these definitions at hand, we are ready to introduce our new planning task, as well as our collection of distance-based properties.

### 3.1.2 *Distance-based Planning Tasks*

In this work we focus on the single-agent classical planning setting previously described, where we make the following assumptions: (1) full observability of the state, which can only change due to the actions performed by the agent; (2) deterministic action outcomes; and (3) the sequence of weighted planning goals $\mathcal{G}$ does not change over time. With these assumptions, we formalize a distance-based planning task as follows:

**Definition 3.4.** *A* ***distance-based planning task*** *is a tuple* $\textsc{dpt} = \langle F, A, I, \mathcal{G} \rangle$*, where* $F$*,* $A$*, and* $I$ *are equivalent to a standard planning task* $\Pi$*; and* $\mathcal{G}$ *is a set of weighted planning goals we want to reason about. The solution to a distance-based planning task is a plan* $\pi$ *whose execution achieves a state* $s$ *that fulfills a cost-related property with respect to the set of goals* $\mathcal{G}$*.*

In order to define some of our states, we will need to reason about all the states in which any of the goals in $\mathcal{G}$ is achieved. We refer to that as the set of individual goal states of a distance-based planning task.

**Definition 3.5.** *Given a distance-based planning task* DPT $= \langle F, A, I, \mathcal{G} \rangle$*, a state* $s$ *is an* ***individual goal state*** *iff (1) it is reachable,* $s \subseteq \mathcal{R}_{\text{DPT}}$*; and (2) there is at least one goal in* $\mathcal{G}$ *that is satisfied,* $\exists G_i = (p_i, w_i) \in \mathcal{G}$ *such that* $p_i \subseteq s$*.* $\mathcal{IG}_{\text{DPT}}$ *is the set of individual goal states of a distance-based planning task.*

Next, we define the cost-related properties we use in this work, which will create our collection of goal-related states and plans.

#### 3.1.2.1 *Distance-based Properties*

We define our collection of goal-related states in terms of some statistical moments over $\mathcal{D}$. We use: the weighted average ($\mu$) over the distances in the sequence; and the maximum (max) and minimum (min) distances in the sequence. The first state we are interested in is the *centroid* of a set of goals. The centroid minimizes the average distance to all the goals in the distance-based planning task. Centroids can have a direct impact in domains such as food delivery, where delivery drivers may be interested in locating themselves close to the most frequented restaurants; or surveillance scenarios, where police (or robots) may want to take some patrolling or control actions in areas where potential incidents might happen. We define planning centroids as follows:

**Definition 3.6.** *A state* $s \in \mathcal{R}_{\text{DPT}}$ *is a* ***centroid state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{R}_{\text{DPT}}, \mu(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \leqslant \mu(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$*. The set of centroid states of a distance-based planning task* DPT *is denoted as Centroids*$^*$(DPT)*.*

In some cases, we might want to approach most of the goals (as the centroid allows), but with the extra requirement of reaching one of the goals in $\mathcal{G}$. Similarly to centroids, we define planning medoids as follows:

**Definition 3.7.** *A state* $s \in \mathcal{IG}_{\text{DPT}}$ *is a* ***medoid state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{IG}_{\text{DPT}}, \mu(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \leqslant \mu(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$*. The set of medoid states of a distance-based planning task* DPT *is denoted as Medoids*$^*$(DPT)*.*

The next property we define is the *minimum covering state*, which minimizes the maximum distance to any goal. The idea behind reaching these states is to not be far from any goal. Minimum covering states might also be useful in the aforementioned scenarios as well as in many multi-agent domains where we do not want to be far from any of the other agents' potential goals in order to block or help them. We define minimum covering states as follows:

**Definition 3.8.** *A state* $s \in \mathcal{R}_{\text{DPT}}$ *is a* ***minimum covering state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{R}_{\text{DPT}}, max(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \leqslant max(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$*. The set of minimum covering states of a distance-based planning task* DPT *is denoted as Minimum-Covering*$^*$(DPT)*.*

As in the case of centroids, we can also restrict the minimum covering states to those already contained in the set of goals $\mathcal{G}$.

**Definition 3.9.** *A state* $s \in \mathcal{IG}_{\text{DPT}}$ *is a* ***minimum covering-m state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{IG}_{\text{DPT}}, max(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \leqslant max(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$*. The set of minimum covering-m states of a distance-based planning task* DPT *is denoted as Minimum-Covering-M*$^*$(DPT)*.*

The four properties of states defined so far try, in different ways, to approach to the goals in $\mathcal{G}$. However, there are domains where it might be useful to find states that are as far from other states as possible. Among others we can find: natural disasters domains, where we want to design escape plans to move people to safe places; games, where we want to avoid or escape from a set of opponents and/or dangerous zones; or choosing the nodes in a network where to store sensitive electronic data to keep it safe from potential threats. An advantage of our distance-based planning task definition is that if we think of goals as *risks*, it is still useful.

**Definition 3.10.** *A state* $s \in \mathcal{R}_{\text{DPT}}$ *is a* ***reverse centroid state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{R}_{\text{DPT}}$, $\mu(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \geqslant \mu(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$. *The set of reverse centroid states of a distance-based planning task* DPT *is denoted as Reverse-Centroids*$^*$(DPT).

**Definition 3.11.** *A state* $s \in \mathcal{I}\mathcal{G}_{\text{DPT}}$ *is a* ***reverse medoid state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{I}\mathcal{G}_{\text{DPT}}$, $\mu(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \geqslant \mu(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$. *The set of reverse medoid states of a distance-based planning task* DPT *is denoted as Reverse-Medoids*$^*$(DPT).

**Definition 3.12.** *A state* $s \in \mathcal{R}_{\text{DPT}}$ *is a* ***reverse minimum covering state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{R}_{\text{DPT}}$, $min(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \geqslant min(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$. *The set of reverse minimum covering states of a distance-based planning task* DPT *is denoted as Reverse-Minimum-Covering*$^*$(DPT).

**Definition 3.13.** *A state* $s \in \mathcal{I}\mathcal{G}_{\text{DPT}}$ *is a* ***reverse minimum covering-m state*** *of a distance-based planning task* DPT *iff* $\forall s' \in \mathcal{I}\mathcal{G}_{\text{DPT}}$, $min(\mathcal{D}(F, A, s, \mathcal{G}, h^*)) \geqslant min(\mathcal{D}(F, A, s', \mathcal{G}, h^*))$. *The set of reverse minimum covering-m states of a distance-based planning task* DPT *is denoted as Reverse-Minimum-Covering-M*$^*$(DPT).

| s | Optimizes | $s \in \mathcal{G}$ |
|---|---|---|
| Centroid | min($\mu(\mathcal{D})$) | × |
| Medoid | min($\mu(\mathcal{D})$) | ✓ |
| Minimum Covering | min(max($\mathcal{D}$)) | × |
| Minimum Covering-m | min(max($\mathcal{D}$)) | ✓ |
| R-Centroid | max($\mu(\mathcal{D})$) | × |
| R-Medoid | max($\mu(\mathcal{D})$) | ✓ |
| R-Minimum Covering | max(min($\mathcal{D}$)) | × |
| R-Minimum Covering-m | max(min($\mathcal{D}$)) | ✓ |

Table 3.1: Summary table of our collection of goal-related properties.

Note that we define the collection of goal-related states as the cartesian product of two variables: (1) the statistical measure over $\mathcal{D}$, average and min/max; and (2) whether we want to minimize or maximize the given measure. By doing this, we are introducing R-Minimum Covering-m, which is a meaningless state that tries to maximize the minimum distance in $\mathcal{D}$ at the same time that requires to be at a distance of $\mathcal{0}$ from at least one goal.Table 3.1 summarizes our collection of goal-related states. For each property, we show which metric it optimizes, and if the returned state must also be a goal state or not.

#### 3.1.2.2 *Plans' Properties*

The states defined in the previous section could be reached through plans that meet different criteria such as being the shortest or the cheapest. We aim to find plans that also fulfill the same distance-based property through all the states traversed by the plan. This can be very useful, for instance, in domains where we do not want to reveal our true goal to an observer (goal obfuscation) (Kulkarni et al., 2019), or domains where we want to be safe during the whole execution (Koenig & Simmons, 1994).

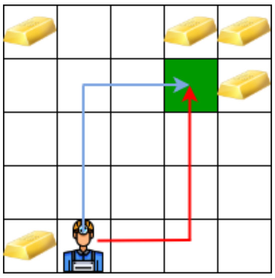


Figure 3.2: Simplified version of a strategy game, where a worker has to gather some resources. Both the blue and red plans have the same cost. However the blue one minimizes the average distance to all the goals along the path.

**Example 3.2.** *Figure 3.2 shows a simplified strategy game where the agent is a worker that may need to gather a set of resources. It is aware that there are some enemies observing its movements; it would therefore like to reveal the least possible information. Under these circumstances, we could see $\mathcal{G}$ as the set of goals the observer believes the agent may want to achieve (the resources), and find its centroid. By doing this: (1) it will get close to gather most of the resources; and (2) if it follows a smart path, it can maximize the number of goals the enemy thinks it is pursuing during most part of the plan, thus minimizing the information leakage. In the above mentioned example, both blue and red plans have the same cost. However, the blue one minimizes the average distance to all the goals along the path. Note that the blue path maximizes goal obfuscation against a worst case observer (Kulkarni et al., 2019), i.e., an observer that can make perfect inferences observing all the actions of the agent. Therefore, less capable observers such as those that can only observe some of the actions, would also be obfuscated by the agent following the blue path. The plan would be obfuscated until the centroid. From that state on, an observer would be able to infer the actual goal of the agent, turning out that centroids are closely related to Last Deceptive Points (Masters & Sardiña, 2017) in goal obfuscation settings.*

**Example 3.3.** *On the other hand, if we see goals as risks, we would want to stay away from them. To do that, we can compute some sort of* escape plans *that keep agents as*

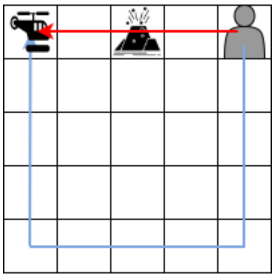


Figure 3.3: Natural disaster domain where a volcano has erupted. The safest place, i.e., the one with the maximum distance to the volcano, is depicted with a helicopter. Although both plans reach the same state, the blue one does it while maximizing the distance to the volcano along the path.

*far as possible from the goals along the execution of all actions in the plan. Consider a natural disaster domain like the one shown in Figure 3.3 where a volcano has erupted in an island. The safest place, i.e., the one with the maximum distance to the volcano, is depicted with an helicopter. When the person reaches this state, she can take an helicopter to leave the island. In this example, both plans do not longer have the same number of actions. The red one is shorter, but makes the agent pass through the risk, while the blue one, although longer, makes the agent reach the helicopter by maximizing the distance to the volcano along the path, thus being a much safer escape plan. These examples have actually been generated by our algorithm using different configurations, as we will later see in the evaluation section.*

We propose another way to compute the cost of a plan that solves a distance-based planning task:

**Definition 3.14.** *The **distance-based cost of a plan** that solves a distance-based planning task* DPT *is the average of a given statistical measure* $m$ *over* $\mathcal{D}$*, for all the states traversed by the plan:*

$$\mathrm{dbc}(\pi, m) = \frac{m(s_0) + \sum_{a_i \in \pi} m(\gamma(s_{i-1}, a_i))}{|\pi| + 1} \tag{3.2}$$

$\pi$ *will be optimal if there is no other plan with lower distance-based cost.*

We use the same metrics we defined for the goal-related states for $m$. For instance, if we compute the centroids of a distance-based planning task, we will use $m = \mu(\mathcal{D})$. If we use any of the reverse properties (those that maximize their distance to the goals), we will transform the metric by substracting a big constant $\kappa$ to it. So, if we compute the reverse centroids of a distance-based planning task, we will use $m = \kappa - \mu(\mathcal{D})$. This allows us to use the same previously-defined cost-minimization algorithms when computing the new plans.

## 3.2 PLANNING WITH DISTANCE-BASED GOALS

In a distance-based planning task, plans need to reach a set of states that is unknown a priori; not even partially as in the case of classical planning. Computing these states requires exploring a vast amount of the reachable state space using a perfect heuristic estimator $h^*$ (see Definitions in Section 3.1.2.1). Since this is a very expensive task, we want to compute plans at the same time that we explore the state space. Hence, we relax our definitions to compute suboptimal states and plans.

We propose to use a common algorithm to compute our collection of states and plans both optimal- and suboptimally, depending on the configuration parameters. We call it GRS, which stands for Goal Related States. Algorithm 1 details the full procedure. It consists of a Best-First Search algorithm that can be tuned by modifying the following parameters:

- The function $h$ used to compute the sequence of distances to the goals in $\mathcal{G}$.
- A parameter $o$ that determines if node reopening is allowed or not.
- The stopping condition of the algorithm, $\tau$.

The algorithm also receives the following inputs:

- A distance-based planning task DPT.
- An input $p$ that determines whether we want to minimize the plan cost $c(\pi)$, or the distance-based cost of a plan in terms of its relationship with the goals $dbc(\pi, m)$, as stated in Definition 3.14.
- An input $m$ that determines the metric we want to minimize, which also means which goal-related state we want to achieve (from Definitions 6 to 13).

The algorithm expands nodes until the stopping condition $\tau$ is met. We store at each search node the planning state, the current $g$, the sequence of distances $\mathcal{D}$, and the path that reaches the node. The sequence of distances $\mathcal{D}$ is computed by ComputeDistances$(F, A, s, \mathcal{G}, h)$. The value of $g$ is computed by ComputeG. This function receives as input: a given node; the parameter $p$ that determines if we want to minimize the plan cost or the distance-based cost of a plan; the metric $m$ over $\mathcal{D}$ we want to minimize; the path towards the node; and the sequence of distances. In case we want to minimize the plan cost, the $g$ of a search node is computed as the sum of its parent's $g$ plus the cost of the applied action, as in a standard Best-First Search algorithm:

$$g_s = g_{\text{parent}} + c(a) \tag{3.3}$$

On the other hand, if we want to minimize the distance-based cost of the plan, we compute $g$ as follows:

$$g_s = \frac{(g_{\text{parent}} \times |\text{path}_s|) + m(s)}{|\text{path}_s| + 1} \tag{3.4}$$

In this case, we are averaging the metric $m$ (computed from the sequence of distances $\mathcal{D}$) of all the states traversed by the plan until reaching $s$.

**Algorithm 1** GRS

```
Require: $h$ (cost estimator), $o$ (node reopening), $\tau$ (stopping condition)
Require: DPT $= \langle F, A, I, \mathcal{G} \rangle$ (distance-based task), $p$ (type of plan), $m$ (metric to minimize)
Ensure: $\mathsf{Best}$ (set of best solutions)
 1: $\mathcal{D}_I \leftarrow$ComputeDistances($F, A, I, \mathcal{G}, h$)
 2: $\mathcal{D}_{\mathsf{Best}} \leftarrow \mathcal{D}_I$
 3: $g_I, g_{cheapest(I)} \leftarrow$ComputeG($I, p, m, \emptyset, \mathcal{D}_I$)
 4: $open, \mathsf{Best} \leftarrow \{\langle I, g_I, \mathcal{D}_I, \emptyset \rangle\}$
 5: while not $\tau$ do
 6:    $\langle n, g_n, \mathcal{D}_n, path_n \rangle \leftarrow$Pop($open, m$)
 7:    $successors \leftarrow$GenerateSuccessors($n$)
 8:    for $s, a$ in $successors$ do
 9:       $\mathcal{D}_s \leftarrow$ComputeDistances($F, A, s, \mathcal{G}, h$)
10:       $g_s \leftarrow$ ComputeG($s, p, m, \{path_n \cup a\}, \mathcal{D}_s$)
11:       if IsNew($s$) or ($o$ and $g_s < g_{cheapest(s)}$) then
12:          $g_{cheapest(s)} \leftarrow g_s$
13:          $path_s \leftarrow path_n + a$
14:          $open \leftarrow open \cup \{\langle s, g_s, \mathcal{D}_s, path_s \rangle\}$
15:          if ComputeMetric($\mathcal{D}_s, m$) < ComputeMetric($\mathcal{D}_{\mathsf{Best}}, m$) then
16:             $\mathsf{Best} \leftarrow \{\langle s, g_s, \mathcal{D}_s, path_s \rangle\}$
17:             $\mathcal{D}_{\mathsf{Best}} \leftarrow \mathcal{D}_s$
18:          else if $\mathcal{D}_s = \mathcal{D}_{\mathsf{Best}}$ then
19:             $\mathsf{Best} \leftarrow \mathsf{Best} \cup \{\langle s, g_s, \mathcal{D}_s, path_s \rangle\}$
20: return $\mathsf{Best}$
```

At each search step, we expand the node which minimizes $m$ (line 6). We generate the successors of such state and compute their sequence of distances and $g$ value (lines 9-10). For the sake of clarity, we assume a smart successors generation function that only generates children that are not already contained in $path_s$. By doing this, we ensure that we are not generating infinite paths with loops. After that, we check if we have to insert the successor into the open list (line 11), which is a priority queue sorted on $m$. We do that either if: (1) the successor is new; or (2) we have found a cheaper path to that state and we allow node reopening (parameter $o$). In such case, the algorithm sets the value of $g$ and the path for the newly generated nodes, and inserts the node in the open list. Then the algorithm checks if the metric value $m$ over the sequence of distances of the current state $\mathcal{D}_s$ is less than or equal to the lowest metric value found so far, stored in $\mathcal{D}_{\mathsf{Best}}$. If so, we update the set of best states found so far $\mathsf{Best}$, and its sequence of distances $\mathcal{D}_{\mathsf{Best}}$ (lines 15-29). When $\tau$ is met, the algorithm returns $\mathsf{Best}$, the set of best solutions found during the search. Such set contains the states and the best plans found that reach those states.

*Theoretical Properties of the Algorithm*

We now prove that our algorithm is sound, complete, and optimal, and under which conditions. First, we show that our task is always solvable.

**Theorem 3.1** (Solution existence)**.** *There is always a solution to distance-based planning task* DPT.

*Proof.* Following Definition 3.4, we have that the solution to DPT is a reachable state that fulfills some distance-based property with respect to a set of goals $\mathcal{G}$. The initial state I is always reachable, since we are already in that state and the algorithm starts searching for a solution from it. In case no other state of the distance-based planning task is reachable, the initial state would be the one that fulfills all previously defined properties. Therefore, there is always a solution for DPT. □

From this, we prove the theoretical properties of GRS*, the optimal version of our algorithm which uses the perfect heuristic estimator as $h$, allows node re-opening $o$ and does not stop until all the reachable state space has been visited $\tau$.

**Theorem 3.2** (Completeness)**.** *If there exists a state* $s$ *that is a solution to* DPT *and a plan* $\pi$ *that solves* DPT, GRS *will find them.*

*Proof.* The algorithm expands nodes by generating all the possible successors of a given state starting from the initial state I. Since there is no pruning, and the termination condition is that the open list is empty, all the reachable state space $\mathcal{R}_{DPT}$ is explored. Also, all states are visited via all the possible paths because we are allowing node re-opening. Therefore, if there is a solution to the DPT, the algorithm will find it. □

**Theorem 3.3** (Soundness)**.** *If the plan* $\pi$ *returned by the algorithm is applied from the initial state* $I$*, it will end up in a state* $s$*, which is the state returned by the algorithm.*

*Proof.* The algorithm starts from the initial state I. At each step, the algorithm generates each successor $st_j$ of the state $st$ by executing an applicable action $a_j \in A$ from $st$ as $st_j = \gamma(st, a_j)$. When the algorithm terminates, the returned plan is the concatenation of actions applied that lead from I to $s$, i.e., $\pi = a_i, \ldots, a_n$, $\Gamma(I, \pi) = s$. Therefore, the plan is valid and the algorithm is sound. □

**Theorem 3.4** (Optimality)**.** *The solution returned by* GRS *optimally solves* DPT *if it is run with the parameters configuration:* $h = h^*$, $o$ *=True and* $\tau = open = \emptyset$.

*Proof.* As we have shown before, the algorithm visits all reachable states and it computes the cost of reaching each goal in $\mathcal{G}$ from each reachable state. By computing this cost using a perfect heuristic estimator $h^*$, each distance in $\mathcal{D}$ represents the actual optimal cost from each state to each goal. Hence, the related metrics, such as the average or the minimum/maximum elements in the set, are also the actual values. When it finishes, the algorithm returns the state and plan that optimize the selected metric. □

We cannot ensure any of these properties for other algorithms that may arise from variations of the different parameters of GRS. For instance, even if a specific version of the algorithm explores all the state space but using a heuristic estimator, it will not be optimal. Although the rest of algorithm's configurations do not share some of these theoretical properties, we will provide some experimental evidences on those properties in the next section, in particular to (sub)optimality.

## 3.3 EVALUATION

We implemented GRS on top of FastDownward (Helmert, 2006). By varying the parameters $h$, $o$, and $\tau$ we obtain five different algorithms, as shown in Table 3.2. These algorithms can also be seen as five different parametrizations of the same algorithm. However, since these parametrizations have completely different theoretical properties, we use the term algorithms instead. We use the FF heuristic, $h_{FF}$ (Hoffmann & Nebel, 2001), to estimate the distance to the goals in the suboptimal versions of GRS, since it is a fast and rather accurate heuristic in many domains. In the optimal version, $h^*$ is computed from the cost of optimal plans to the goals at each search node using $A^*$ with the LMCUT admissible heuristic (Helmert & Domshlak, 2009).

| Version | $h$ | $o$ | $\tau$ |
|---|---|---|---|
| $\text{GRS}^*$ | $h^*$ | Yes | $\mathcal{R}_{DPT}$ |
| $\text{GRS}^f$ | $h_{FF}$ | Yes | $\mathcal{R}_{DPT}$ |
| $\text{GRS}^{f-}$ | $h_{FF}$ | No | $\mathcal{R}_{DPT}$ |
| $\text{GRS}^{hc}$ | $h_{FF}$ | Yes | Greedy + Hill Climbing |
| $\text{GRS}^g$ | $h_{FF}$ | Yes | Greedy |

Table 3.2: Five versions of GRS obtained by varying: (1) the heuristic function $h$; (2) the node reopening policy $o$; and (3) the stoping condition $\tau$.

We employ three different stopping conditions $\tau$. The first one, $\mathcal{R}_{DPT}$, explores all the reachable state space. The second one, Greedy, stops the search when there is no state in the open list with a better (lower) metric value than the best state visited so far. This is used by the greediest version of the algorithm, $\text{GRS}^g$. The third condition, Greedy+Hill Climbing, employed by $\text{GRS}^{hc}$, also performs a greedy search, but it tries to escape from local minima by performing 50 random actions[1] from the state found by $\text{GRS}^g$. It follows a stochastic hill climbing approach that: (1) generates the successors of a state; (2) randomly selects one of them; (3) evaluates it to check if it is better than the best state visited so far – if so, it updates $Best$, which contains the best state observed; and (4) goes back to (1) until it reaches the maximum number of iterations (50).

Most versions of the algorithm reopen nodes to find a cheaper path to the states. However, this computation may be too expensive, specially in the versions that explore all the reachable state space. Hence, we include $\text{GRS}^{f-}$, which behaves like $\text{GRS}^f$, but does not reopen nodes (parameter $o$). This alternative

[1] Best value found during our experimental evaluation.

may potentially reduce the search effort when computing suboptimal solutions (Chen et al., 2019).

These five algorithms take also three parameters $(h, o, \tau)$ that GRS receives as input, and allow reaching the states by differently exploring the search space. By also varying $m$, we can obtain any of our goal related states: centroids, medoids, minimum-covering, minimum-covering-m, reverse centroids, reverse medoids, reverse minimum covering, and reverse minimum-covering-m states. We can also vary $p$, the parameter that indicates the type of plan we want to get. We might want to minimize the cost of the actions, or the distance-based cost of the states traversed. These variations give us 80 different algorithms that generate different states and plans. For instance, we can use GRS$^*$ with $m = \min(\text{avg})$, and $p = \min(c(\pi))$ to compute optimal centroid states through a shortest path (plan with least length). Or we can use GRS$^g$ with $m = \max(\text{avg})$, and $p = \min(dbc(\pi, m))$ to greedily compute reverse centroid states through a plan that maximizes the distance to the goals through the traversed states.

The algorithms we have defined return a set of solutions $Best$. Each solution $S \in Best$ is a tuple $S = \langle s, g_s, \mathcal{D}_s, path_s \rangle$. In case the returned values of $g_s$ and $\mathcal{D}_s$ have been computed using greedy algorithms and/or imperfect heuristic estimators, we ignore these values and compute them using perfect estimators. Thus, we compute $\mathcal{D}_s$ using a perfect heuristic estimator $h^*$, and also compute the distance-based cost of the plan returned in the solution $path_s$. When there exist more than one solution ($|Best| > 1$), the algorithm has found more than one state with the same metric value over $\mathcal{D}$. In this case, we break ties by the (distance-based) plan cost. If ties persist, we report the result of any of the remaining solutions indistinctly.

The experimental evaluation is divided as follows. First, we perform an evaluation of algorithms' suboptimality in small problems taken from three domains to test how the suboptimal versions compare to the optimal one. Second, we increase the size of the problems in a controlled way to test how all versions scale. Third, we continue our quantitative evaluation in large planning instances taken from different International Planning Competitions (IPCs)[2]. Then, we perform a qualitative assessment of the different goal-related states and plans returned by each algorithm.

**Reproducibility**. The experiments were run on Intel(R) Xeon(R) CPU X3470 @ 2.93GHz machines with a time limit of 3600s and a memory limit of 8GB. Both the code and the benchmarks mentioned from now on are already available on-line.[3]

### 3.3.1 *Suboptimality Evaluation*

Our first set of experiments compares the different versions of GRS. We perform a set of experiments in small problems where the optimal version can explore all the reachable state space computing optimal plans to the goals. In those problems, the size of the reachable state space is small, $|\mathcal{R}_{DPT}| \leqslant 1000$. We selected

[2] http://ipc.icaps-conference.org

[3] https://github.com/apozanco/GRS_0.1

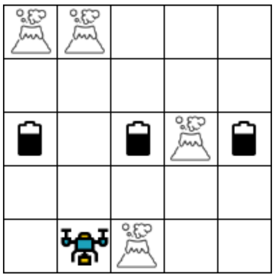


Figure 3.4: UAV domain. Batteries represent recharge stations.

the following three domains, generating 10 random distance-based planning tasks for each one:

- Grid: a path-planning like domain where an agent can move to adjacent cells. The map size is 20 × 20 and 20% of the cells are obstacles that the agent cannot traverse. The goals are four random cells that the agent should reach. The weight of each goal is randomly generated.
- blocks-words: a variation of the well-known blocks-world domain, where an agent can build words using five available blocks (Ramírez & Geffner, 2009). Each problem has three potential goals, which are words that the agent wants to build. The weight of each goal is randomly generated.
- UAV: a domain like the one shown in Figure 3.4, where an Unmanned Aerial Vehicle (UAV) needs to patrol an area with some volcanoes. In particular, the drone is required to take a picture of a volcano once it erupts. The eruption pattern is not known, and the UAV is required to take pictures of the eruption as soon as possible once it occurs. The battery of the UAV is limited, so it will need to recharge it from time to time using the charging stations depicted by the battery icons. The UAV starts with three units of battery, and will consume one unit each time it moves to an adjacent cell of the grid. The map size is 8 × 8 and we place 4 volcanoes and 10 recharge stations randomly in all the problems. Note that in this domain some goals might not be reachable. In those cases, we set the distance to the goal to 1000.

We selected these three domains given that they encompass many others: Grid is the base of any path-planning domain; blocks-words can be seen as a construction domain; and in UAV we have resources that are consumed and might cause deadend situations. For the sake of clarity, in this subsection we will only report results of the five algorithms when computing centroids via shortest paths. However, all the suboptimality results and conclusions can be extrapolated to the rest of states and plans. Table 3.3 summarizes the results of a quantitative evaluation. For each domain, each row shows the average and standard deviation over 10 problems of the results obtained by each of the five versions. Each column represents different measures of quality and performance:

| **Centroids - Shortest Path** | | | | | |
|---|---|---|---|---|---|
| Domain | Version | $\mu$ | $\Delta(\mu)$ | $c(\pi)$ | t |
| Grid | GRS$^*$ | 7.8±2.3 | -0.4±0.2 | 9.6±4.4 | 2681.1±165.3 |
| | GRS$^f$ | 7.8±2.2 | -0.4±0.2 | 11.7±7.7 | 1.2±0.1 |
| | GRS$^{f-}$ | 7.8±2.2 | -0.4±0.2 | 11.7±7.7 | 1.1±0.0 |
| | GRS$^{hc}$ | 8.0±2.5 | -0.4±0.2 | 10.8±7.2 | 0.7±0.0 |
| | GRS$^g$ | 8.1±2.6 | -0.4±0.2 | 10.3±7.7 | 0.8±0.0 |
| Blocks Words | GRS$^*$ | 5.1±1.2 | -0.4±0.2 | 7.3±3.0 | 848.2±14.2 |
| | GRS$^f$ | 5.5±1.4 | -0.4±0.2 | 10.0±3.3 | 0.7±0.1 |
| | GRS$^{f-}$ | 5.5±1.4 | -0.4±0.2 | 14.4±6.4 | 0.6±0.1 |
| | GRS$^{hc}$ | 7.9±1.9 | -0.1±0.2 | 5.6±2.9 | 0.2±0.0 |
| | GRS$^g$ | 8.7±2.5 | -0.0±0.2 | 3.2±1.6 | 0.2±0.0 |
| UAV | GRS$^*$ | 251.7±352.4 | -0.2±0.2 | 3.3±1.7 | 144.5±47.5 |
| | GRS$^f$ | 251.8±352.5 | -0.2±0.2 | 3.0±1.4 | 0.3±0.0 |
| | GRS$^{f-}$ | 251.8±352.5 | -0.2±0.2 | 3.0±1.4 | 0.3±0.0 |
| | GRS$^{hc}$ | 252.1±353.0 | -0.2±0.2 | 2.3±1.2 | 0.3±0.0 |
| | GRS$^g$ | 252.1±353.0 | -0.2±0.2 | 2.3±1.2 | 0.3±0.0 |

Table 3.3: Comparison of different versions of the algorithm on easy problems when computing centroids via a shortest path. For each domain, each row shows the results obtained by each version. Columns show averages and standard deviations over the set of problems for: weighted average cost to the goals of returned state ($\mu$); improvement of $\mu$ in the returned state with respect to the initial state ($\Delta(\mu)$); cost of the returned plan ($c(\pi)$); and running time (t).

- $\mu$: weighted average cost to the goals in $\mathcal{G}$, for the state $s$ returned by the algorithm;
- $\Delta(\mu)$: ratio of the difference between $\mu$ of the returned state and the $\mu$ of $I$; and $\mu$ of $I$. We compute it as follows:

$$\Delta(\mu) = \frac{\mu(s) - \mu(I)}{\mu(I)} \tag{3.5}$$

  Therefore, negative numbers mean that returned states are closer to the goals than $I$, while positive numbers mean that returned states are further from goals.

- $c(\pi)$: cost of the returned plan; and
- $t$: time in seconds to return a solution.

The optimal version of the algorithm, GRS$^*$ obtains the best results in terms of quality of solution, as expected. It returns states that are around $20 - 40\%$

percent closer to the goals than the initial state. However, its execution times are around three orders of magnitude higher than the ones of the suboptimal versions. As a reminder, it explores all the reachable state space from the initial state, computing an optimal plan to each goal in every state.

GRS$^{f}$ and GRS$^{f-}$ also explore the reachable state space, but estimating the distance to each goal (using a heuristic) rather than computing the cost of an optimal plan to the goals. These algorithms obtain results in about one second that are close to or even optimal. As we can see, avoiding node reopening can slightly speed-up the computation (around 0.1 seconds in two domains), obtaining a very similar quality. However, the plans that achieve them have a higher cost in some cases.

The greediest versions of the algorithm, GRS$^{hc}$ and GRS$^{g}$ are faster, but the states they return are farther from the optimum in some cases. Their associated plans tend to be shorter ($c(\pi)$), meaning that they can fall in local minima. We can also see how the Hill Climbing version of the algorithm improves the results of the greedy search in some domains.

We can also observe this behavior in Figure 3.5, where we plot how the suboptimal versions compare to the optimal ones with respect to the $\mu$ of the returned state. As we can see, GRS$^{f}$ and GRS$^{f-}$ , the versions that explore all the reachable state space using a heuristic to compute the distance to the goals, obtain results that are at most 1.3 times suboptimal in BLOCKS-WORDS. The greediest version, GRS$^{g}$, has the worst performance, but rarely returns centroid states that are more than 2.0 times suboptimal.

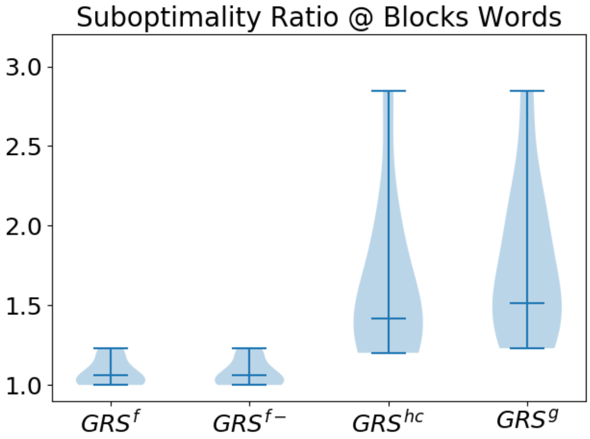


Figure 3.5: Degradation of GRS$^{f}$, GRS$^{f-}$, GRS$^{hc}$, and GRS$^{g}$ with respect to GRS$^{*}$ when computing centroids following a shortest path in BLOCKS-WORDS. The central horizontal line indicates the median of the distribution, while the other two lines indicate the maximum and minimum values. A wider blue shadow indicates that more points have that value.

Suboptimality results trends are similar in all domains. However, in the GRID domain suboptimal algorithms behave better. We conjecture that this is due to the number of local minima in the problems of each domain. To confirm our hypothesis, we generated problems with increasing percentages of obstacles in GRID. Results are depicted in Figure 3.6, where we compare the results of GRS$^{*}$

with GRS$^g$. In the absence of obstacles, the greediest version is able to obtain optimal centroids. The reason is that the FF heuristic is a perfect estimator in that domain. As the percentage of obstacles increases, and hence the number of local minima, some solutions start moving away from the optimum. The median is 1.0 for 0, 5, and 10% of obstacles, and it only increases up to 1.2 in the case of 20% of obstacles. This means GRS$^g$ is returning optimal solutions most of the times.

Suboptimal algorithms behave worse in UAV, where the FF heuristic is less accurate since it considers that the drone is always fully charged due to the delete relaxation. However, the results of the greediest algorithm are at most 3 times suboptimal in problems where all the goals are reachable. In problems where some goals are not reachable this difference vanishes, given that we add a large value of 1000 as the distance to unreachable goals. That is also why we get average distance values of around 250 in this domain, meaning that 1 out of the 4 goals is not achievable from the drone initial position.

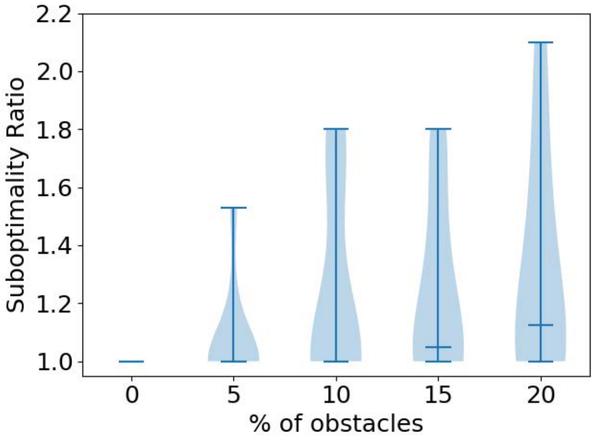


Figure 3.6: Degradation of GRS$^g$ with respect to GRS$^*$ when increasing the percentage of obstacles in GRID. The central horizontal line indicates the median of the distribution, while the other two lines indicate the maximum and minimum values. A wider blue shadow indicates that more points have that value.

Overall, we can say that suboptimal versions present a good balance between quality and time, obtaining fast average results that are close to the optimum in some cases.

### 3.3.2 *Scalability Evaluation*

We generated a set of GRID problems with increasing reachable states and goals to test how the different algorithms scale. We increased the reachable state space $|\mathcal{R}|$ by considering grids of 5 × 5, 10 × 10, 20 × 20, and 50 × 50 with no obstacles; and increase the number of goals, $|\mathcal{G}|$, in powers of 2 from 2 to 16. We generated 10 problems for each combination of grid size and number of goals (16 combinations), and compute the reverse centroids on them. The results we report hereafter are averaged among these problems. The optimal version of the algorithm only solves 9 out of the 16 combinations. The versions that also explore all the reachable state space solve 14 out of 16. They fail in the problems with a $50 \times 50$ grid with 8 and 16 risks. The greediest versions are able to solve all problems in all cases.

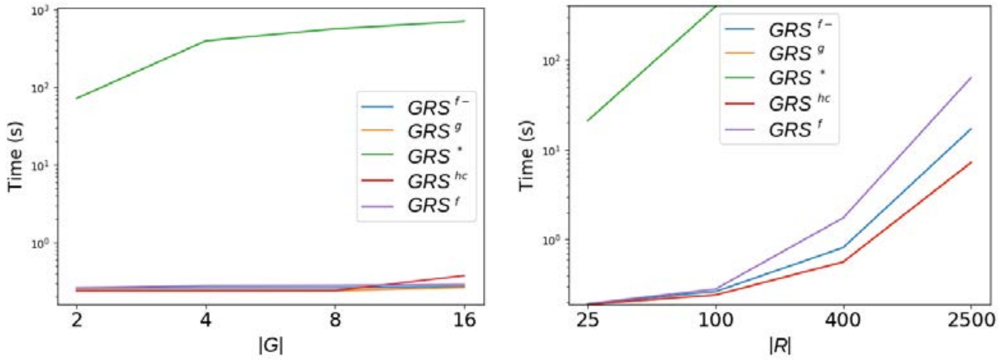


Figure 3.7: Time in logarithmic scale needed for each algorithm to return a solution. The left image shows the results in $10 \times 10$ problems of GRID when we increase the number of goals. The right image shows the results when we fix the number of goals to 4 and increase the grid size, where $|\mathcal{R}|$ goes from 25 ($5 \times 5$) to 2500 ($50 \times 50$).

Figure 3.7 shows how the different algorithms scale when we fix either the number of goals or the number of reachable states. We measure the time needed for each algorithm to return a solution. The left image shows the results in a $10 \times 10$ problems of GRID when we increase the number of goals. The right image shows the results when we fix the number of goals to 4 and increase the square size from 5 to 50. As expected, the execution times of the optimal algorithm are various orders of magnitude higher than the ones of the suboptimal versions. In the left image we can see that only the optimal algorithm gets affected by increasing the number of goals.

We also observed that the plan length decreases as the number of goals increases, with plans being up to 60% longer when we have two goals compared to when we have 16 goals. If the state space is full of goals that are randomly distributed, it is more likely that the centroids (as well as any other goal-related states) are not far from the initial state, and thus the solutions are shorter. In the right image we can see that all the algorithms are affected by increasing the number of reachable states. The execution time grows exponentially for the suboptimal versions, but they are still able to solve all problems.

### 3.3.3 *Quantitative Evaluation of* GRS

As we have seen, suboptimal versions can compute states and plans that are quite close to the optimal ones in a reasonable amount of time. However, we have not seen the differences between our collection of goal-related states so far. Figure 3.8 graphically shows the results obtained after solving 10 instances of BLOCKS-WORDS via a shortest path with GRS*. The different states are represented in the x-axis, while the y-axis depicts different metrics over the sequence of distances $\mathcal{D}$. For each state, the green plot shows the values of $\mu$, while the red plot shows the maximum element in the set (the minimum in the case of r-centroid, r-medoid, r-minimum-covering, and r-minimum-covering-m).

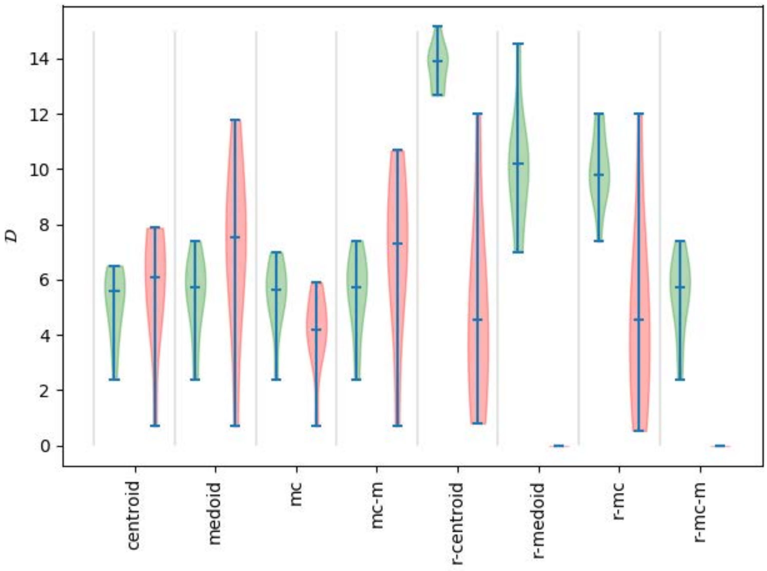


Figure 3.8: Results after solving 10 instances of BLOCKS-WORDS via a shortest path with GRS*. The x-axis shows the different states, while the y-axis depicts different metrics over the sequence of distances $\mathcal{D}$. For each state, the green plot shows the $\mu$ results, while the red plot shows the maximum element in the set (the minimum in the case of the inverse states). For each boxplot, the central horizontal line indicates the median of the distribution, while the other two lines indicate the maximum and minimum values. A wider shadow indicates that more points have that value.

As expected, we obtain states with lower average ($\mu$) distance to the goals when computing centroids and medoids, and higher values when computing reverse states such as r-centroid. Also, we get states with lower maximum distance to the goals when computing minimum covering states.

In both cases, when we force the returned states to be within the set of goals (medoids and related states), we obtain worse average and min/max results. For instance, the average distance to the goals is a bit higher in the case of medoids than it is when computing centroids. In the case of r-medoids and r-minimum-covering-m, we always obtain a value of zero in the minimum element in $\mathcal{D}$. This makes sense, since these states are restricted to conform a goal of $\mathcal{G}$, thus having a distance of $0$ to it (the worst possible value). R-medoids and r-minimum-covering-m are therefore of not much interest, and we only define and compute them to consider all the states given by the cross product of statistical measures and their minimization/maximization.

### 3.3.4 *Large Planning Instances*

In the previous sections, we have used small instances for which the optimum can often be computed. However, the set of reachable states of a standard planning task is much larger. So, to conclude our quantitative evaluation, we test the suboptimal versions of the algorithm in planning domains taken from different IPCs. We conducted experiments in five well-known planning domains: FERRY, GRIPPER, HANOI, LOGISTICS, and TERMES. LOGISTICS, and TERMES. We selected them because they are sufficiently different from each other, and we think computing some of our proposed states in those domains might be interesting. For example, we think that centroids might be really useful in domains such as LOGISTICS and TERMES. For each domain, we generate 10 distance-based planning tasks based on the original IPC problems. For each original IPC problem, we generate a goal's set of size between 2 and 4. Each goal in the goal's set is composed of 1 to 4 predicates. In addition to the metrics in Table 3.3, we define the following ones:

- minmax: minimum or maximum value in $\mathcal{D}$ for the returned state $S$. We report: the maximum element in the set for those states that want to approach the goals, such as centroids or minimum-covering states; and the minimum element for those that want to stay away from them, such as r-centroids or r-minimum-covering states.

- $\Delta(\text{minmax})$. Ratio of the difference between the minmax of the returned state and the minmax of I; and minmax of I. We compute it as follows:

$$\Delta(\text{minmax}) = \frac{\text{minmax}(s) - \text{minmax}(I)}{\text{minmax}(I)}$$

  When computing states that get closer to the goals (such as centroids or minimum-covering states), a negative number indicates that the maximum distance to all the goals decreases. In other words, the state is not far from any of the goals. When computing states that get far from the goals (such as r-medoids or r-minimum-covering states), a positive number indicates that the minimum distance to all the goals increases. In other words, the state is not close to any of the goals.

- $\text{dbc}(\pi, m)$: distance-based cost of the returned plan according to the metric $m$, which is the same metric that the one minimized/maximized by the given state.

- #s: number of instances solved within the given time and memory bounds.

We conducted experiments for each domain and all the 64 combinations of goal-related state (8 states), plan type (2 types of plans), and algorithm version (4 suboptimal algorithms). For clarity purposes, we only show the results of a couple of combinations and domains in Tables 3.4 and 3.5.

In this case, we have non-restricted planning tasks and hence the number of reachable states is usually large. As we can see, in LOGISTICS all the algorithms

| Domain | State | Plan type | Version | $\mu$ | $\Delta(\mu)$ | minmax | $\Delta$(minmax) | $c(\pi)$ | $dbc(\pi, m)$ | t | #s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| logistics | centroid | shortest | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 23.0±3.7 | 5.3±0.2 | 66.6±54.1 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 31.2±6.4 | 4.7±0.5 | 11.6±8.0 | 10 |
| | | | $GRS^{hc}$ | 5.7±4.2 | -0.5±0.4 | 5.1±3.5 | -0.5±0.3 | 20.9±8.7 | 8.1±2.8 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 6.3±4.6 | -0.5±0.4 | 5.4±3.7 | -0.4±0.3 | 12.2±12.0 | 8.6±3.0 | 0.2±0.0 | 10 |
| | | metric-related | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 33.5±6.6 | 4.4±0.4 | 11.6±8.0 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 33.5±6.6 | 4.4±0.4 | 11.7±8.0 | 10 |
| | | | $GRS^{hc}$ | 5.7±4.1 | -0.5±0.4 | 5.0±3.4 | -0.5±0.3 | 15.8±11.0 | 8.1±2.9 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 6.3±4.6 | -0.5±0.4 | 5.4±3.7 | -0.4±0.3 | 13.0±13.0 | 8.5±3.2 | 0.2±0.0 | 10 |
| | medoid | shortest | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 23.0±3.7 | 5.3±0.2 | 78.8±60.8 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 31.4±6.7 | 4.7±0.5 | 11.4±7.8 | 10 |
| | | | $GRS^{hc}$ | 5.2±3.9 | -0.6±0.3 | 5.0±3.6 | -0.5±0.3 | 17.9±9.6 | 8.0±2.8 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 6.3±4.6 | -0.5±0.4 | 5.4±3.7 | -0.4±0.3 | 12.2±12.0 | 8.6±3.0 | 0.2±0.0 | 10 |
| | | metric-related | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 23.0±3.7 | 5.3±0.2 | 80.8±50.3 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 33.7±6.9 | 4.4±0.4 | 11.5±7.9 | 10 |
| | | | $GRS^{hc}$ | 6.0±4.4 | -0.5±0.4 | 5.5±3.9 | -0.4±0.4 | 18.0±12.5 | 8.1±2.9 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 6.3±4.6 | -0.5±0.4 | 5.4±3.7 | -0.4±0.3 | 13.0±13.0 | 8.5±3.2 | 0.3±0.0 | 10 |
| | minimum-covering | shortest | $GRS^{f}$ | 0.5±0.5 | -1.0±0.0 | 0.7±0.6 | -0.9±0.1 | 28.8±5.8 | 4.6±0.8 | 37.4±28.2 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.5 | -1.0±0.0 | 0.7±0.6 | -0.9±0.1 | 28.8±5.8 | 4.6±0.8 | 10.6±7.8 | 10 |
| | | | $GRS^{hc}$ | 2.6±3.7 | -0.8±0.3 | 2.3±3.2 | -0.8±0.3 | 21.5±10.7 | 6.0±2.5 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 2.6±3.7 | -0.8±0.3 | 2.3±3.2 | -0.8±0.3 | 21.5±10.7 | 6.0±2.5 | 0.2±0.0 | 10 |
| | | metric-related | $GRS^{f}$ | 0.5±0.5 | -1.0±0.0 | 0.7±0.6 | -0.9±0.1 | 31.1±6.1 | 4.3±0.8 | 10.7±7.8 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.5 | -1.0±0.0 | 0.7±0.6 | -0.9±0.1 | 31.1±6.1 | 4.3±0.8 | 10.8±8.0 | 10 |
| | | | $GRS^{hc}$ | 2.6±3.7 | -0.8±0.3 | 2.3±3.2 | -0.8±0.3 | 23.3±11.6 | 5.8±2.7 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 2.6±3.7 | -0.8±0.3 | 2.3±3.2 | -0.8±0.3 | 23.3±11.6 | 5.8±2.7 | 0.2±0.1 | 10 |
| | minimum-covering-m | shortest | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 23.0±3.7 | 5.3±0.2 | 59.8±45.1 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 29.4±5.9 | 4.5±0.8 | 11.4±7.9 | 10 |
| | | | $GRS^{hc}$ | 2.5±3.5 | -0.8±0.3 | 2.6±3.3 | -0.7±0.3 | 23.0±9.4 | 5.9±2.4 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 2.7±3.9 | -0.8±0.3 | 2.6±3.3 | -0.7±0.3 | 21.1±11.2 | 6.2±2.6 | 0.2±0.0 | 10 |
| | | metric-related | $GRS^{f}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 23.0±3.7 | 5.3±0.2 | 57.8±40.1 | 10 |
| | | | $GRS^{f-}$ | 0.5±0.4 | -1.0±0.0 | 0.8±0.7 | -0.9±0.1 | 31.7±6.1 | 4.2±0.7 | 11.4±7.8 | 10 |
| | | | $GRS^{hc}$ | 3.6±4.3 | -0.7±0.4 | 3.6±4.0 | -0.6±0.4 | 22.3±13.0 | 6.5±3.0 | 0.2±0.0 | 10 |
| | | | $GRS^{g}$ | 3.7±4.5 | -0.7±0.4 | 3.4±3.8 | -0.7±0.4 | 21.3±14.2 | 6.6±3.1 | 0.4±0.0 | 10 |

Table 3.4: Comparison among different versions of the algorithm computing different states in the Logistics domain. Each row shows the average and the standard deviation over the set of problems for: weighted average cost to the goals of the returned state ($\mu$); ratio of difference of the returned state with respect to the initial state ($\Delta(\mu)$); minimum value in $\mathcal{D}$ for the returned state (minmax); ratio of difference of the returned state with respect to the initial state ($\Delta$(minmax)); cost of the returned plan ($c(\pi)$); distance-based cost of the returned plan ($dbc(\pi, m)$); running time of the algorithm (t); and number of solved tasks (#s).

| Domain | State | Plan type | Version | $\mu$ | $\Delta(\mu)$ | minmax | $\Delta$(minmax) | $c(\pi)$ | $dbc(\pi, m)$ | t | #s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TERMES | CENTROID | SHORTEST | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.0±0.6 | 11.9±6.1 | 0.3±0.0 | 5 |
| | | METRIC-RELATED | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 5.4±3.4 | 10.9±5.8 | 0.4±0.1 | 5 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.2±1.0 | 11.9±6.1 | 0.3±0.0 | 5 |
| | MEDOID | SHORTEST | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.0±0.6 | 11.9±6.1 | 0.3±0.0 | 5 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.0±0.6 | 11.9±6.1 | 0.3±0.0 | 5 |
| | | METRIC-RELATED | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 1.8±0.4 | 12.7±6.5 | 0.3±0.0 | 4 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 1.8±0.4 | 12.7±6.5 | 0.3±0.0 | 4 |
| | MINIMUM-COVERING | SHORTEST | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.1±0.3 | -1.0±0.0 | 0.4±0.7 | -1.0±0.1 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | | | GRS$^{g}$ | 0.1±0.3 | -1.0±0.0 | 0.4±0.7 | -1.0±0.1 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | | METRIC-RELATED | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.1±0.2 | -1.0±0.0 | 0.2±0.3 | -1.0±0.0 | 3.6±1.4 | 11.5±6.1 | 0.3±0.0 | 5 |
| | | | GRS$^{g}$ | 0.1±0.3 | -1.0±0.0 | 0.4±0.7 | -1.0±0.1 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | MINIMUM-COVERING-M | SHORTEST | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.4±1.0 | 11.9±6.0 | 0.3±0.0 | 5 |
| | | METRIC-RELATED | GRS$^{f}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{f-}$ | - | - | - | – | - | - | - | - |
| | | | GRS$^{hc}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.3±1.1 | 12.6±6.5 | 0.3±0.0 | 4 |
| | | | GRS$^{g}$ | 0.0±0.0 | -1.0±0.0 | 0.0±0.0 | -1.0±0.0 | 2.3±1.1 | 12.6±6.5 | 0.3±0.0 | 4 |

Table 3.5: Comparison among different versions of the algorithm computing different states in the TERMES domain. Each row shows the average and the standard deviation over the set of problems for: weighted average cost to the goals of the returned state ($\mu$); ratio of difference of the returned state with respect to the initial state ($\Delta(\mu)$); minimum value in $\mathcal{D}$ for the returned state (minmax); ratio of difference of the returned state with respect to the initial state ($\Delta$(minmax)); cost of the returned plan ($c(\pi)$); distance-based cost of the returned plan ($dbc(\pi, m)$); running time of the algorithm (t); and number of solved tasks (#s).

solve the 10 distance-based planning tasks. However, this is not the case in the rest of domains. In most cases, only GRS$^{hc}$ and GRS$^{g}$, which do not explore all the state space, are able to solve the tasks. As observed in the previous experiments, greediest algorithms obtain worse results, but return solutions in less than a second in most cases. On the other hand, GRS$^{f}$ and GRS$^{f-}$ obtain better results, but their execution times are higher and cannot solve the tasks within the given time in many cases. As expected, we obtain lower $\mu$ values when computing centroids, and higher $\mu$ values when computing r-centroids. The same applies to the minmax metric when computing minimum-covering states and r-minimum-covering states.

There are some domains such as TERMES where algorithms are able to return states with an average distance to the goals equal to 0. This means that there are some states where all the goals of the distance-based planning task are true. This makes sense because these tasks come from solvable IPC problems, where there exist at least one state where all the goal propositions are true. This was not the case in the BLOCKS-WORDS tasks, where the agent can build only one of the words at the same time.

The fact that we constructed our distance-based planning tasks from the IPC tasks also affects the results we get for the different states. For instance, in the GRIPPER domain, the greediest algorithms quickly compute the states that want to escape from goals, such as r-centroids, because the goals are already *far* in the original IPC task. So, the initial state or one of its most immediate successors is the one that is furthest from the goals. On the other hand, it is really costly to find centroids or medoids in these very same tasks, since those states that minimize the average distance to the goals are much further from the initial state.

### 3.3.5 *Qualitative Assessment of* GRS

Finally, we show how safest states and escape plans look like in different domains, and how the different algorithms behave. Regarding the **states**, it is easy to visualize them in path-planning domains such as GRID. However, we might wonder how they look like in other domains. Figure 3.9 shows a graphical representation of all the defined goal related states in a BLOCKS-WORDS distance-based planning task.

As we can see, the four states that try to somehow approach the goals (centroid, medoid, minimum-covering and minimum-covering-m) put most of the blocks on the table. This is the strategy that allows to build most of the words at a lower cost. On the other hand, if we want to stay away from the goals, the strategy seems to be to stack most of the blocks in a big tower. From these states, it is hard to build any of the goal words, being necessary to unstack the blocks to later stack them in the proper order. In other domains such as LOGISTICS, plans that reach centroids tend to move the packages close to their destination and/or move the trucks and planes to places where they can pick up/drop the packages with less cost.

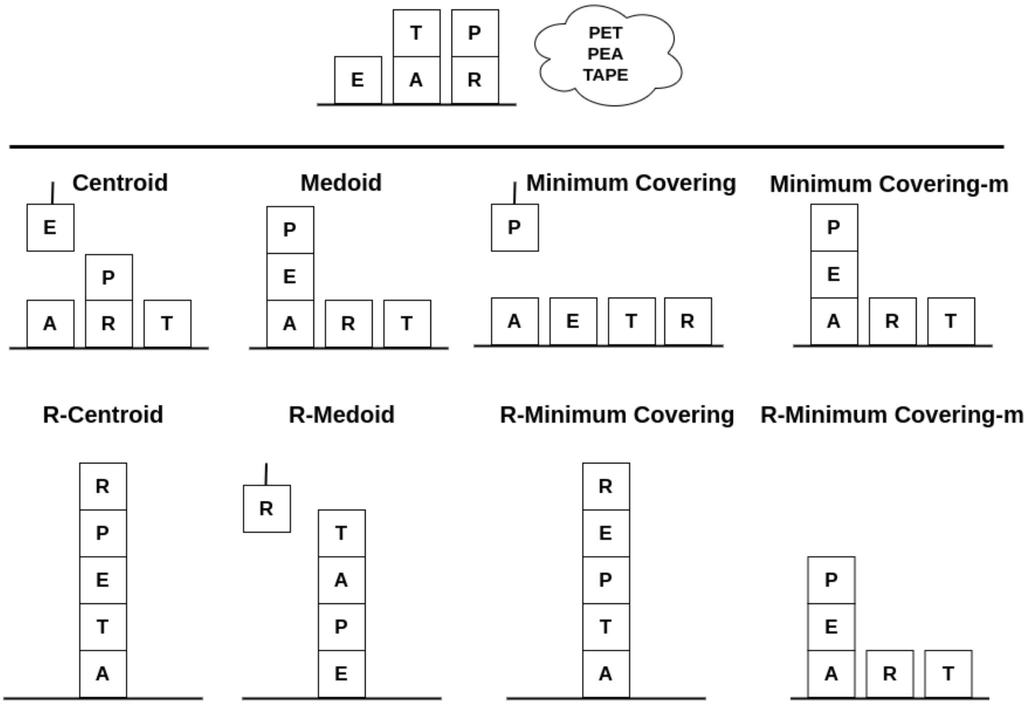


Figure 3.9: Set of goal-related states of a blocks-words distance-based planning task. The upper image above the horizontal line depicts the initial state and the set of goals. The images below the horizontal line show the different goal-related states as returned by the optimal version of the algorithm grs$^*$

.

In the UAV domain the best option for the UAV would be to reach either: (1) the centroid state, if it wants to be close to most of the potential eruptions; or (2) the minimum-covering state if it does not want to be far from any potential eruption. Both states are shown in Figure 3.10 for a particular task. In the centroid state (left image), the UAV is fully charged at the charging station located in the middle of the map. The average cost to take a picture of a potential eruption is 2.75. In the minimum-covering state (right image), the UAV has two units of battery. The average distance to the eruptions in this case is a bit greater, 3. However, while in the centroid the UAV is at distance 5 from taking a picture of the volcano located at the top-left corner of the map (it will need to recharge again in the way), the maximum distance in the minimum-covering state is 4 both to that volcano and the one located at the bottom.

The states shown in Figure 3.9 are computed using the optimal version of the algorithm, which is not very useful in practice. Thanks to our evaluation of suboptimality, we saw that suboptimal algorithms are typically not far from the optimum. So we also wanted to know how the states and plans computed by the suboptimal versions of the algorithm look like. Figure 3.11 shows a graphical representation of the states and plans returned by the four suboptimal versions of grs in the natural disaster domain of our running example when computing r-centroids through a minimum cost plan.

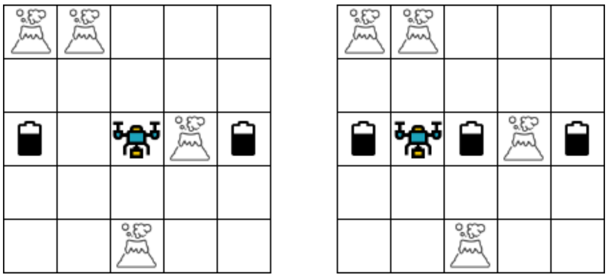
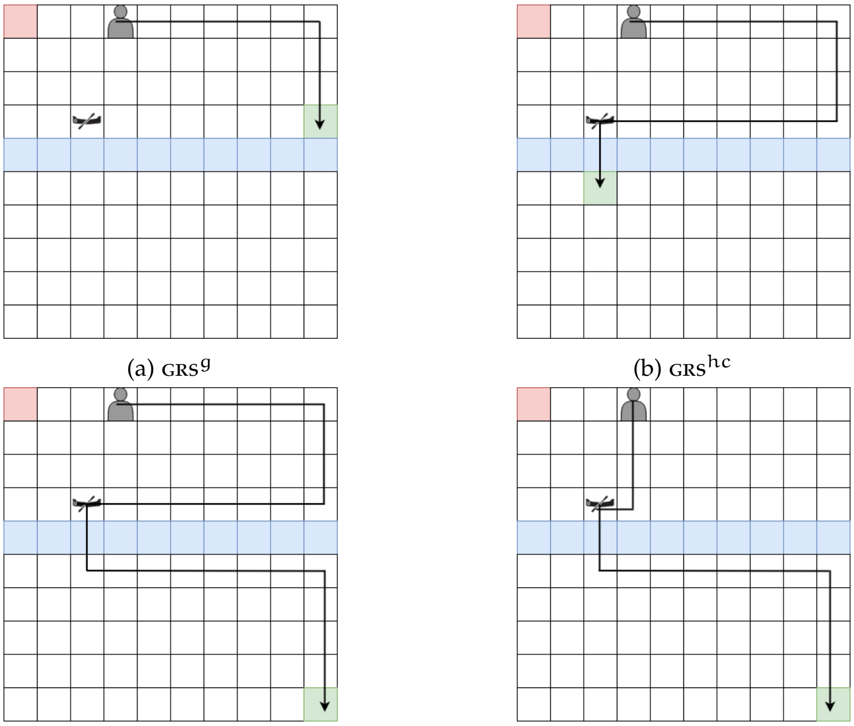


Figure 3.10: Optimal centroid (left) and minimum-covering (right) states in a UAV surveillance task.

As we can see, the greediest algorithm, GRS$^g$, falls into local minima. It is not able to discover that using the boat can make the person move further away from the goal (in red), and it generates a greedy plan to move her to the farthest place from the goal on the north side of the river. GRS$^{hc}$ performs a local search from the state returned by GRS$^g$ and finds a better plan that involves crossing the river with the boat. On the other hand, GRS$^{f-}$ and GRS$^f$, which explore all the reachable state space, find the optimal state of the distance-based planning task. However, GRS$^f$ computes an optimal plan, while GRS$^{f-}$ returns a suboptimal one. There exist many other optimal paths in terms of cost in this particular case, but GRS$^f$ is able to return the one that also maximizes the distance to the goal over the complete execution. This is also the case of the optimal version, GRS$^*$, which returns the same escape plan as GRS$^f$.

Finally we wanted to see the differences on the returned **plans**, depending on whether we minimize the plan cost or its distance-based cost with respect to the set of goals of the task. For instance, consider a BLOCKS-WORDS instance like the one shown in Figure 3.12, where we compute the medoid using GRS$^*$. The plan with minimum cost comprises 10 actions, while the plan with minimum distance-based cost comprises 12 actions. By taking a deeper look to the plans, we can observe their differences. Both plans start executing the same actions: `unstack t e`, `put-down t`, `unstack e a`, `put-down e`, `unstack a p`. They differ in the next action: while the shortest plan prescribes `stack a t`, the metric-related plan prescribes `put-down a`. If we examine both states, we observe that the one returned by the shortest plan has a cost of 12 of reaching PATER, the goal with higher weight. On the other hand, the state returned by the metric-related plan has a cost of 10 of achieving that high priority goal, minimizing the weighted average distance to the goals in $\mathcal{G}$. The rest of the plan is similar and both plans reach the same final state: building the word RAT. However, the metric-related plan minimizes the average cost to the goals along the path.

There are many similar examples along all the domains where we observe this small differences in the plans. But we could also observe bigger differences if we twist and force the problems a little bit, like the result obtained in the problem shown in Figure 3.3.



(a) GRS$^{g}$

(b) GRS$^{hc}$

(c) GRS$^{f-}$

(d) GRS$^{f}$

Figure 3.11: R-centroid (in green) and plans that minimize the distance-based cost (black arrow) computed using different versions of GRS in the natural disaster domain.

## 3.4 RELATED WORK

To our knowledge, this is the first work that focuses on computing states that fulfill certain cost-related properties with respect to a given set of goals. In this section, we examine the relevant literature related to our approach for planning with distance-based goal definition. We show both, how our work is situated within previous works, and how our technique may be useful for solving some other tasks.

Recently, the concept of Goal Reasoning has received increased attention (Aha, 2018; Cox, 2007; Molineaux et al., 2010; Vattam et al., 2013). Agents no longer

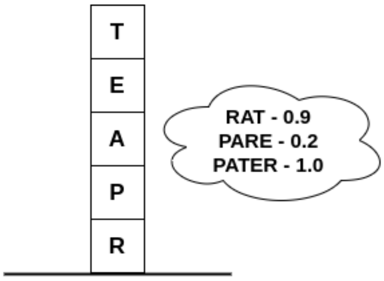


Figure 3.12: Initial state and set of goals with their associated weight of a BLOCKS-WORDS distance-based planning task.

receive as input a fixed set of goals they have to achieve. By contrast, they are encouraged to reason about their goals over time, changing them as the *environment* changes. Some of these approaches even try to anticipate goals prior to their arrival (Burns et al., 2012; Fuentetaja et al., 2018; Pozanco et al., 2018b), in order to minimize the time (cost) of achieving a goal when it dynamically arrives. This task is known as Anticipatory Planning, and we will discuss it in Chapter 4. Our notion of planning centroids and medoids has a direct impact on it. By approaching these states, agents are getting closer to most of the potential goals that may appear. There are two main differences between our approach and these other works. On the one hand, none of the above mentioned works solves any of the planning tasks we define. On the other hand, they consider dynamic goals, while we are currently considering a set of static goals. While our approach could also consider goals that arrive dynamically, it is out of the scope of this chapter, leaving it for the next chapter.

Other works also have similar formalisms to represent goals, and/or consider a set of goals to reason about. A remarkable case is that of oversubscription planning (Domshlak & Mirkis, 2015; Smith, 2004), which achieves a subset of goals as valuable as possible within a fixed allowance of the total action cost. Opposed to this approach, we are not interested in maximizing the utility of the achieved goals, but some other cost-based properties with respect to the set of goals $\mathcal{G}$. Also, in the case of oversubscription planning, $\mathcal{G}$ would typically be composed of a single proposition, while we admit any size on each element of that set.

In recent years, automated planning has shown its effectiveness in solving goal-recognition problems (E-Martín et al., 2015; Pereira et al., 2020; Ramírez & Geffner, 2009), where the aim is to infer the actual goal of an agent by observing the actions it performs in the environment. All these works consider a set of goals $\mathcal{G}$ agents may want to achieve, and perform different reasoning processes over the particular goals in the set.

There exist many applications based on planning-based goal recognition where our set of states may be useful. In a single-agent setting, one example is goal obfuscation (Keren et al., 2016; Kulkarni et al., 2019; Masters & Sardiña, 2017). The objective in this kind of tasks is to minimize the information leakage of the execution of a plan. In other words, the aim is to maximize the part of the plan in which a potential observer is unaware of the true goal the agent is pursuing. Although approaches differ, all these works solve a similar task when computing the obfuscated plans: they try to compute states that would be part of any plan leading to a specific subset of goals. Keren et al. (2016) propose a solution that obfuscates a goal by choosing another goal within $\mathcal{G}$ which has the maximum non-distinct observation sequence (plan) in common with the true goal. Kulkarni et al. (2019) achieves goal obfuscation by choosing a subset of $k$ goals from $\mathcal{G}$ (including the true goal). Then, they perform a search in the belief state space, returning a plan that is $k-$ambiguous, i.e., it is consistent with $k$ goals. Our approach differs from them in that we consider the whole set of candidate goals $\mathcal{G}$. By computing the centroid of $\mathcal{G}$, and a plan that minimizes the average cost to the goals in $\mathcal{G}$ along the path, we are ensuring goal obfuscation with respect to the maximum number of goals, without the need of selecting a value of k, storing belief states, nor post-processing the output as these other approaches do. Following our approach, we are also ensuring goal obfuscation against a worst case observer, i.e., an observer that can make perfect inferences observing all the actions of the agent. We showed it in Figure 3.2. The agent would be approaching most of the goals through a plan that makes the observer doubt regarding which is its true goal.

In a multi-agent setting, some works use goal-recognition to automatically generate goals given other agents' actions. In a competitive setting, there is the work on counterplanning (Pozanco et al., 2018a), which tries to generate goals and plans that prevent opponent from achieving their goals (see Chapter 5). In this approach, the agent considers a set of candidate goals the opponent might be trying to achieve. With this information, the agent: (1) uses goal recognition to infer the opponent goal; (2) extracts the landmarks (Hoffmann et al., 2004) involved in the opponent achieving that goal; and (3) generates a plan to delete that landmark before the enemy achieves it. However, sometimes the agent infers the actual goal of the opponent when it is too late to reach the landmark before the opponent. By early approaching states such as the minimum-covering of the candidate goals, the agent would put herself in a better position to block the enemy. We will explore this approach in Chapter 6.

We can treat the states and plans that move away from the goals as escape plans. Some works in the literature deal with finding escape or evacuation plans. However, they often compute the solutions over grids and networks where they solve path finding tasks. Moreover, they are domain-dependent, focusing on specific tasks such as evacuation of people in fire scenarios (Olsen et al., 2015) or vehicle routing in critical contexts such as homeland defense preparation (Lu et al., 2005). In contrast, our approach is domain-independent and can find different states and plans in many other types of scenarios. Other works also focus on computing risk-sensitive solutions in a domain-independent way using state space search (Jeantet & Spanjaard, 2008; Koenig & Simmons, 1994; Perny et al., 2007). These approaches are directed by a specific goal. They have an initial goal and try to find a plan that maximizes the probability of its achievement or minimizes the expected execution cost of the plan. Most of these approaches are based on utility theory (Morgenstern & Von Neumann, 1953) or rank-dependent utility (Quiggin, 2012). In contrast to these works, our approach is not goal-directed; that is, we do not have a initial goal to pursue.

Our work is also related to the work on defining and reasoning with path constraints, as in the case of using Linear Temporal Logic (LTL) (Baier et al., 2009). Usually, path constraints refer to some state-related constraint, such as some set of literals being true/false at some point during the execution of the plan. In our case, the plan-related properties relate to a set of goals and the properties link with metrics such as averages or maximum/minimum distance, which are hard to be defined in LTL.

## 3.5 SUMMARY

This chapter presents the formal definition of a new kind of planning task, that of finding states and plans that fulfill some distance (cost) requirements with respect to a set of goals. We called this task distance-based planning, since the goal state (final state for our algorithms) must fulfill a distance-based property with respect to the set of goals. We defined a collection of eight different types of goal-related states, highlighting domains where computing each of them might be interesting. We also introduced a new concept of plan cost different from the typical one: minimize a given goal-distance metric along the plan's traversed states. We remarked how these type of plans might be useful in trending planning topics and applications such as goal recognition and its variations of goal legibility and goal obfuscation (Chakraborti et al., 2019; Kulkarni et al., 2019). Then, we described a common algorithm with five different parametrizations to solve distance-based planning tasks, by varying suboptimality requirements. Finally, we extensively evaluated our approach in several planning domains, reporting both quantitative and qualitative results.

Experimental results showed that AP can effectively solve these distance-based planning tasks. Among its virtues, the presented approach: (1) is domain-independent; (2) can be easily extended with different goal related states; and (3) it is straightforward to extract explanations about the returned states and plans, which might be useful in real-world applications (Chakraborti et al., 2019).

Optimally computing our collection of goal related states and plans using GRS turns out to be unfeasible in practice, since it is necessary to explore all the reachable state space, computing an optimal plan to each goal. However, we have shown that suboptimal versions obtain results close to the optimal one very fast, also scaling up to larger instances. Thus, we can run a greedy algorithm if we want to rapidly reach a state close/far from the goals. This can be very useful in on-line settings such as strategy games. But we can also run either $\text{GRS}^{f}$ or $\text{GRS}^{f-}$ if we have more time and want high quality states and plans.

We are aware that we are paying an efficiency burden when using a common algorithm to compute all the different states. A better way to proceed when computing centroids would be to perform backward search from the goals. However, we were more interested in defining a new planning task, highlighting its importance and providing a common algorithm able to compute a large set of states and plans. In future work we would like to explore different algorithms to compute each state as well as define others, since the algorithm is mostly agnostic to that extent. As we have shown, the behavior of the suboptimal versions is not only related to the presence of local minima but also to the use of accurate heuristics closer to $h^*$. In this chapter, we used the FF heuristic to estimate the distances to goals, but in future work we would like to compare algorithm's performance by using other heuristic estimators. We would also like to define an anytime version of the algorithm, so users can stop it at their will, returning the best states/plans found so far. Finally, we are also interested in allowing users to insert preferences into the algorithm such as “a plan with a maximum distance to a particular goal during all the plan".

# 4 LEARNING TO ANTICIPATE PLANNING GOALS

*"Instinct is a funny thing."*

*- Polly Gray*

In recent years the interest in creating autonomous agents for numerous real world applications has greatly increased. Applications range from surveillance purposes to control tasks (Ai-Chang et al., 2004; Cenamor et al., 2017; de la Asunción et al., 2005; García et al., 2013; Ruml et al., 2011). In these cases the closed-world and static-goals assumptions common in automated planning do not hold any more and reasoning about goals and the changes in the environment becomes essential (Talamadupula et al., 2010; Vattam et al., 2013). Therefore, new approaches explicitly deal with dynamic goals. A notable example is the concept of Goal-Driven Autonomy (GDA), inspired by Cox's work (2007) and detailed in (Klenk et al., 2013) that we introduced in the Background section. GDA is a conceptual model that explicitly considers goal reasoning as a key component of the deliberative reasoning process of autonomous agents. It allows us to design and deploy autonomous agents that can explicitly reason about their goals, identifying when they need to be updated or changed through environment monitoring.

The first works on GDA generated goals following a set of pre-programmed rules that are triggered under some state's conditions (Coddington et al., 2005). A human must code all the goal-triggering rules before the system's execution. Some recent works on goal reasoning learn goal formulation without human interaction, extending agents autonomy (Jaidee et al., 2013). All these previous works rely on goal reasoning based on the current state of the world. We want to extend the agent's performance by creating a system that is able to generate and handle not only the current existing goals, but also the possible upcoming ones, given the current and recent states of the environment.

From an automated planning point of view, previous approaches trigger planning episodes when the current state and/or goals change (most often state changes). We refer to this paradigm as Reactive Planning, since its behaviour is triggered to react to changes in the environment. Our system is based on Anticipatory Planning (Burns et al., 2012) that takes into account the possible upcoming – but not currently existing – goals along with the current goals when triggering the planning process.

The main contributions of this chapter include:

- The design of a learning system that can predict the appearance of new goals in the near future. The learning system is capable of learning goal's appearance off-line and on-line by collecting learning examples from the plans' execution. In an on-line setting it is able to handle concept drift; when the conditions of goal's appearance change dynamically.

- The design of a goal management system that takes into account the learned goal predictive models to generate new goals.
- The integration of the new goal management system within a cognitive architecture that already integrates planning, execution, monitoring and replanning capabilities. We call this new approach Learning-driven Goal Generation Anticipatory Planning (LGG-AP).
- The evaluation of its performance in a typical domain for goal management, comparing LGG-AP against Reactive Planning, that only reasons with the current goals.
- The analysis of the impact of different relevant parameters that influence the anticipation of future goals.

The rest of the chapter is organized as follows. First, in Section 4.1 we enumerate the domain characteristics where LGG-AP can be applied. Then, we describe an architecture that integrates goal reasoning with automated planning in Section 4.2. After that, we introduce the learning component that allows the agent to generate future goals based on the current and past states in Section 4.3. In Sections 4.4 and 4.5, we detail our experimental setting and provide experimental results in a surveillance domain respectively. Finally, we position our contribution in the context of related work in Section 4.6 and conclude with a summary of the chapter in Section 4.7.

## 4.1 ANTICIPATORY DOMAINS

In many real world domains agents can improve their performance if they can anticipate and reason about the arrival of future goals. Agents can generate (Cox, 2013) some (future) goals and start trying to achieve them sooner. In order to define the anticipatory behavior, we will first provide the following definitions related to goals.

**Definition 4.1.** *A predicate* $p$ *is a* ***goal predicate*** *in a planning domain* $D$ *if any of its instantiations (groundings) appears in the goals' list of any of its problems* $P$.

Given any domain $D = \langle F, A \rangle$, in theory any predicate can appear instantiated as a goal of any problem. However, in most domains there is a subset of predicates that are the ones that appear instantiated as problems' goals. For instance, consider a Taxi domain in which a fleet of taxis have to serve a set of customers' pick-up requests. The domain model defines predicates like `at-car`, `at-customer`, `car-empty` and `in`, but most problems will only use `at-customer` in the goals description. Thus, `at-customer` would be a goal predicate for the Taxi domain. We assume that our system knows the set of all goal predicates, $\mathcal{P}$. It can be either given by the user or automatically computed by looking at a set of problems. That allows us to define the following set.

**Definition 4.2.** *We refer as* ***all-goals set of a set of predicates*** $\mathcal{P}$, $\mathcal{G}_{\mathcal{P}}$ *to the set of all instantiated goals that can be generated by instantiating predicates in* $\mathcal{P}$ *with world objects.*

Goals in $\mathcal{G}_{\mathcal{P}}$ can appear at any time step from the beginning until the end of the agent's execution. The agent knows which goals could be given (or generated by the agent), but it does not know if or when they will arrive. For example, in the Taxi domain, the all-goals set $\mathcal{G}_{\{\text{at-customer}\}}$ of an agent's execution would consist of all potential instantiations of the goal predicate `at-customer`. So, it would be `at-customer(A,locX)`, `at-customer(B,locY)`,. . . .

**Definition 4.3.** *A goal* $g \in \mathcal{G}_{\mathcal{P}}$ *is a* ***known goal*** *if it has appeared (given or generated) at any time step.* $\mathcal{K} \subseteq \mathcal{G}_{\mathcal{P}}$ *is the set of known goals.*

Following the previous Taxi domain example, if at a given time step only the goal `at-customer(A,locX)` has appeared, it would be a known goal, conforming $\mathcal{K}$. We now define two other types of goals: active goals and predicted goals.

**Definition 4.4.** *A* ***goal*** $g$ ***is active*** *if the agent is trying to achieve it in the next planning episode.* $G \subseteq \mathcal{K}$ *is the set of active goals.*

**Definition 4.5.** *A* ***goal*** $g$ ***is predicted*** *if the agent's inner model foresees its appearance in the near future. At the moment a goal is predicted, it becomes active and known.* $G_l$ *is the set of predicted goals.*

The agent should generate plans to achieve those goals in G. Some of them are known goals that the agent has not achieved yet. But, new goals $g \in \mathcal{G}_{\mathcal{P}}, g \notin G$ could be added over time. The reason why a goal is added to G can vary: a user adds $g$ to the current set of goals at a given time step; a procedure in the planning agent decides to generate it based on the current state of the world, as in some GDA approaches (Coddington et al., 2005); or a model predicts its appearance in the near future, as we deal with in this chapter.

We will make some assumptions on the properties that domains should meet for Anticipatory Planning to be useful.

- For each goal $g \in \mathcal{G}_{\mathcal{P}}$, the agent does not know whether $g$ will appear or when $g$ will appear.
- There is at least one goal $g \in \mathcal{G}_{\mathcal{P}}$ that follows an appearance pattern; i.e., its appearance depends on some features related to the observable state. This is needed so that learning to predict new goals makes sense.
- There is full observability on the appearance of goals. At each time step, the agent can observe the set of new goals that have appeared or have been generated by itself, updating the set of active goals $G^t$ to $G^{t+1}$.
- When a goal is active, it will not disappear from G until it is achieved by the execution of the plan.
- An increasing penalty is paid at each time step for all goals $g_i$ that have already appeared, $g_i \in G \setminus G_l$, and have not been achieved. Although predicted goals $G_l$ are active, since the agent is trying to achieve them, the agent only pays the penalty for those goals in G that have already appeared. The total penalty for each goal, $p(g)$ can be computed following Equation 4.1.

$$p(g) = \begin{cases} 0 & \text{if } g \notin G \\ k_g \times \delta & \text{if } g \in G \setminus G_l \end{cases} \tag{4.1}$$

where $k_g$ is the penalty associated to the goal $g$ and $\delta$ is the difference between the arrival time of the goal and the time when it is achieved by the execution of some action. The total penalty of the whole planning-execution cycle, $P$, is the sum of the penalties paid by each goal that has appeared over the complete planning-execution cycle, $p(g)$.

As we have mentioned, there are many domains where those assumptions hold. For instance, take a company that provides products to some warehouses. Suppose that each warehouse will generate a new goal of having a given product when they do not have stock of that product. If the company is able to predict when warehouses will run out of a product, it can plan to supply the product before the warehouses ask for it, offering a better service and saving time for the warehouse.

Yet another domain is the case of a smart city that wants to control its urban traffic (Pozanco et al., 2016). In that work the goals consist on decreasing the density level of the busy streets through changing the green and red phases of the traffic lights. If one is able to learn a predictive model that suggests the appearance of congestions in the near future, it is possible to incorporate these predicted goals into the set of active goals. Then, it is possible to start the planning process sooner, improving the behavior of the system and leading to less waiting time for the cars and pollution levels for the city.

These domains can be seen as goal maintenance problems (van Riemsdijk et al., 2008), where some predicates must hold during execution. But, Anticipatory Planning offers some advantages. First, techniques that perform goal maintenance trigger actions to achieve maintenance goals as soon as the goals do not hold (Pokahr et al., 2003). For example, when the warehouse runs out of a product, the agent will immediately try to achieve the goal of having a specific quantity of the product. Other recent works define proactive agents. An agent will take actions not only in response to a maintenance goal not holding, but also in anticipation of the maintenance goal being violated (Duff et al., 2006). The agent reasons about the effects of several actions in the near future and will not take any action that violates a maintenance goal. That is, the agent takes into account that all the maintenance goals hold during the planning process. This can improve the rational behavior of agent systems, but these maintenance goals cannot deal with exogenous events, as our Anticipatory Planning approach does. We store information about the environment and build a model that predicts the appearance of goals (or maintenance goals that will be violated) in the future based on these exogenous events.

Other example domains, which can not be seen from a maintenance goal point of view, are surveillance tasks, as those of police, guards, or drones. If we can anticipate where the security breach will appear (where each security breach will generate a new goal to address it), they can arrive at the place earlier and patrol (or execute a set of actions) where the predictive model suggests.

In this chapter, we will focus on Unmanned Aerial Vehicle (UAV) domains. UAV's usage is growing in recent years due to their low price, increasing set of programming tools, and versatility. The possible uses of UAV's range from military approaches to different surveillance purposes. In this case we propose a domain in which an UAV performs surveillance tasks on an area that has been discretized in a grid.

The UAV has to serve a set of requests coming from a surveillance center. Each goal is a request of performing a given set of tasks (as taking images) in a specific cell of the grid. To service a request, the UAV must move from its current position to the cell of the request. For example, a request can be taking an image or making some kind of measurement. So, the goal would be `(taken-image <cell-id>)` or `(measured <cell-id>)`. Since several observation requests can appear at the same time, the UAV is provided with a planning component in order to find the best plan to cover all the requests (goals) with the minimum penalty. The goals do not disappear until they are achieved and can be achieved at any time. This differs from previous works that assume a finite horizon (Burns et al., 2012; Fuentetaja et al., 2018).

All these domains share the same assumptions, so we expect similar results in terms of improvement of performance over a system that does not anticipate to their appearance. Since we had good results in the traffic domain, we wanted to analyze here whether they generalize to domains with similar assumptions on goals. In the following section we describe the architecture that allows the agent to generate and formulate its own goals based on learning, as well as performing anticipatory planning and executing the generated plans.

## 4.2 ARCHITECTURE

Given that the agent needs to integrate learning, goal management, planning, and execution, we have based our work in a domain-independent architecture that we had developed, PELEA (Guzmán et al., 2012). We have instantiated PELEA in LGG-AP in order to add goals prediction capabilities to the architecture, goal management based on learning goals appearance and anticipatory planning. A high level view of LGG-AP is depicted in Figure 4.1. First, we will briefly describe how the architecture works. In the next section, we will describe in more detail the main contribution of this chapter: the goal generation component based on learning goal prediction models. We will also describe in the following section the simulator we have built in order to test the system's performance.

Initially, the `Execution` module receives a planning task, as a *domain* and *problem* file, including the initial state and goals (initial active goals G). These files are sent to `Monitoring` and `Planning` components, so that the `Planning` component can generate a *plan* for the initial planning task. The plan consists of actions that the `Execution` module will translate into *low level actions* that will be sent to the `Environment`. As an example, the action `move(UAV,cell1,cell2)` will be translated into the different power values to each of its four engines.

The `Execution` module periodically receives a set of *observations* from the `Environment`. These observations are translated into a new PDDL problem that

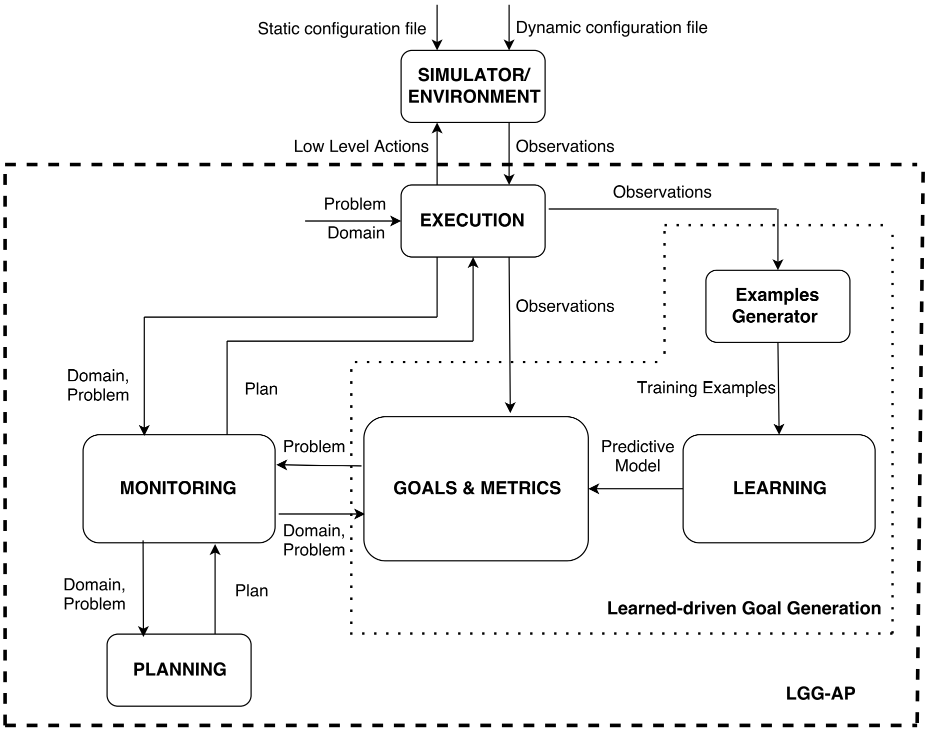


Figure 4.1: Planning architecture that includes goal formulation and goal learning capabilities.

is sent to the Monitoring component. Execution of actions can be stochastic, but in this chapter we will only focus on a specific source of uncertainty; the set of new goals that appears at each time step. If the received state or goals (present in the *problem*) do not match the ones that the Monitoring module expected (e.g., a new goal appears and becomes active), this module will call the Planning component in order to find a new plan with the state and goals (a new problem). Execution also sends the *observations* to the Examples Generator and the Goals & Metrics components.

At each time step, and before calling the Planning module, the Goals & Metrics component receives a *problem* from the Monitoring component, a set of *observations* from Execution and a *predictive model* from the Learning component. The Goals & Metrics component is in charge of deciding, given a problem, which goals the system should pursue.[1] It can be instantiated in several ways, depending on the goal management approach we want to use. For instance, it can reason about the current subset of goals to plan for in case of oversubscription planning (Smith, 2004) (there is no plan that can achieve all goals with the current resources). We will later describe how it works in our case.

[1] It can also change the metrics to optimize for, though we are not changing metrics here.

## 4.3 learning-driven goal generation

In this section we describe the Learning-Driven Goal Generation, corresponding to the modules within the dotted line shown in Figure 4.1. This component allows the architecture to formulate new goals when the system predicts they will appear in the near future. There are three steps in this process: the generation of learning examples; the construction of the predictive model; and the formulation of new goals.

### 4.3.1 *Generation of Examples*

Our base idea is to predict future goals based on current and past observations. *Observations* include measurable features about the state and checking whether new goals have appeared. Examples of state features in the case of the UAV can be temperature readings, seismic activity, or any other measurement that our UAV can observe at each cell. Therefore, we will use the state features to generate the attributes of each *training example*. The check about the appearance of a new goal at each cell generates the class of each example. In the case of the UAV, the check at each cell and time step returns true if the surveillance center requested an observation in that cell and time step.

The `Examples Generator` module translates observations into examples in two steps: collecting observations and generating the examples. First, at each time step $t$ during execution, a set of *observations* $O_t$ are collected, one per cell in the grid; $O_t = \{o_{i,t} | i \in [1, m]\}$, where $m$ is the number of cells. Each observation $o_{i,t} \in O_t$ takes the form of:

$$o_{i,t} = \langle \mathtt{cell} - \mathtt{id}_i, f_{1,i,t}, f_{2,i,t}, \ldots, f_{n,i,t}, g_{i,t} \rangle$$

`cell-id$_i$` identifies the corresponding cell, $f_{j,i,t} (j \in [1, n])$ represents the value of feature $f_j$ at cell $i$ at time $t$, $n$ is the number of measurable features, and $g_{i,t}$ will be either true (if the goal has appeared at time $t$ at cell $i$) or false otherwise.

The learning task consists on building a predictive model for each cell $i$ of whether the system expects a goal to appear at $i$ in the near future. Therefore, each example should contain information about several observations in the past $S$ time steps, as well as the information on whether the goal has appeared in exactly $H$ time steps into the future. $S$ defines the number of previous time steps that will be taken into account when making the prediction, and $H$ refers to the prediction horizon. So, our learning system uses two parameters whose values we will study in the experimental section.

Second, a *training example* can be generated for each cell $i$ and time step $t$ from a set of environment *observations*:

$$\mathcal{EO}_{i,t} = \langle O_{i,t-S}, O_{i,t-S+1}, \ldots O_{i,t-1}, O_{i,t}, O_{i,t+H} \rangle \tag{4.2}$$

Each $O_{i,k} (k \in [t - S, t]$ or $k = t + H)$ is composed of the feature values observed of the $n$ features at cell $i$ and time $k$. The union of the values of all observations from $O_{i,t-S}$ to $O_{i,t}$ (corresponding to the measurements from $S$ time steps ago until the current time $t$) will be the attribute values of each example. And the

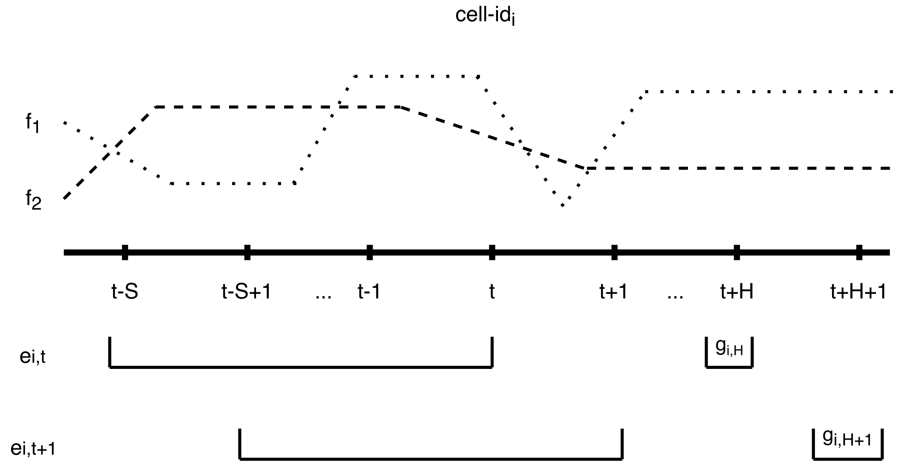


Figure 4.2: Generation of two training examples $e_{i,t}$ and $e_{i,t+1}$ at a cell $i$ from observations. The attributes of the training examples correspond to the value of two variables $f_1$ and $f_2$ in the previous time steps. The class of the training examples indicates whether there is a goal $g$ or not at cell $i$ in the future time step $H$.

goal value in $O_{i,t+H}$, $g_{i,t+H}$, will be the class of the example. This is shown in Figure 4.2. Hence, each training example (at cell $i$ and time step $t$) can be defined as:

$$\begin{aligned} e_{i,t} = \langle cell-id_i, & f_{1,i,t-S}, \dots f_{1,i,t}, \\ & f_{2,i,t-S}, \dots f_{2,i,t}, \dots \\ & f_{n,i,t-S}, \dots f_{n,i,t}, \\ & g_{i,t+H} \rangle \end{aligned}$$

The set of *training examples* will be:

$$E = \{e_{i,t} | i \in [1, m], t \in [0, T]\} \tag{4.3}$$

where $T$ is the number of simulation steps. Therefore, the number of features in each example will be $(S+1) \times n$ ($n$ features, and $S+1$ time steps). And the number of examples will be $m \times T$ (number of cells times the number of simulation steps). The example generation process is independent of the employed representation. In our work on the traffic domain, we used a relational (predicate logic) representation where observations were described in terms of a set of literals. Each example was defined as a set of grounded predicates indicating the current and past states of the world and the class of the training example. And we employed a relational learning algorithm that can extract knowledge from these structured examples. Now, we use an attribute-value representation instead, where the input data set is typically represented as a single table. Each row in the table is an example and each column is an attribute that represents one particular property of each example.

### 4.3.2 *Building a Goal Predictive Model*

Once the system has the examples, it can use a learning algorithm to generate a *predictive model* of future goal appearances. We learn a single model for all cells. In case different cells have different behaviors with respect to goal appearances, the learning system can use the `cell-id` as a feature that can help on making the predictions.

Our approach does not need a particular learning algorithm. The only restrictions are that it has to handle the representation formalism of the examples, and it has to deal with classification tasks (class is discrete). As we mentioned, in the case of traffic we used a relational representation. Then, we had to use a relational learning algorithm as TILDE (Blockeel & Raedt, 1998), which learns relational decision trees from structured examples based on predicates. Now, we use attribute-value representation, so we can use any standard learning technique that solves classification tasks. The input of the learning algorithm is the set of training examples generated in the previous step. The output is a *predictive model*, $L$. The resulting *predictive model* is provided as input to the `Goals & Metrics` module.

In the case of an off-line learning setting, the system would collect all examples from one or several runs and learn from them. In an on-line setting, as the one we consider, examples come over time and the system learns every time new examples come. Given that the reasons (pattern) for the appearance of goals can change over time, there might be a concept drift. Therefore, we add new examples and re-train at each time step.

### 4.3.3 *Generation of Predicted Goals*

In the last step of learning-driven goal generation, the `Goals & Metrics` module receives as input at each time step $t$: the current planning task, $\Pi = \{F, A, I, G\}$; a predictive model of goal appearance, $L$; and a set of environment observations in the previous time steps at every cell in the grid, $\mathcal{E}\mathcal{O}_t = \{O_i = \langle o_{1,i}, o_{2,i}, \ldots o_{m,i} \rangle, i \in [t - S, t]\}$. If $L$ predicts future goals $G_l$ (in exactly $H$ steps into the future) based on the observations, the goals in $G_l$ are added to the current set of goals, generating a new set of goals, $G' = G \cup G_l$. The output of this step is an updated planning task that incorporates the new goals, $\Pi' = \{F, A, I, G'\}$.

## 4.4 EXPERIMENTAL SETTING

In this section we cover the simulator we have implemented for testing our approach, as well as the input variables and metrics used in the comparison.

### 4.4.1 *UAV Simulator*

We developed a simulator to test our approach. It allows us to define different stochastic scenarios under diverse settings. The simulator receives two files:

- Static configuration file: it contains the static characteristics of the simulation, such as the map size. For the experiments, we have generated a $10 \times 10$ grid area like the one shown in Figure 4.3.

- Dynamic configuration file: it contains the dynamic characteristics of the simulation. In particular, for each time step $t$, it contains $O_t$; that is, the set of observations $o_{i,t}$ at time $t$ and cell $i$. As described above, these observations include the values for each feature $f_j$ at time $t$ and cell $i$, $f_{j,i,t}$ and the observation of goal appearance, $g_{i,t}$. This scheme allows us to define different scenarios with specific patterns of observations and goal appearances. We consider there could be two kinds of features: those related to the state (e.g. position of the UAV or the images taken); and the ones corresponding to exogenous events of the environment (e.g. the seismic activity and the earth temperature at each cell and time step). The values of these parameters are explained later.

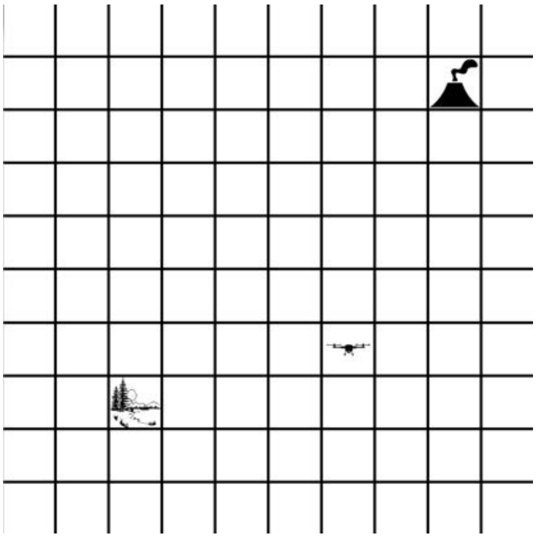


Figure 4.3: Screenshot from the surveillance UAV simulator. In this example scenario there is an active volcano on the top right side of the grid and a lake on the bottom left.

At each time step $t$, the simulator will receive one action from the `Execution` module. After checking for its validity (it can be applied in the current state), it will change the state of the simulation. Then, it returns to the `Execution` module the observations at time $t$, $O_t$. For each cell, these observations include the values of state variables, the values of features related to exogenous events, and the appearance of a goal. The simulator also returns some metrics about the goal achievement process during the simulation. For each goal, it returns its current penalty (described in Equation 4.1), which we will use later to present the results.

In the experiments, we have defined two kinds of goal appearance patterns: random and pattern-based. Some goals appear randomly. Other goals follow an appearance pattern that depends on the observations. We will define in each experiment what pattern we are using.

### 4.4.2 *Experimental Variables*

First we introduce the parameters we will vary through the simulation and then we present the employed metrics. We will study the impact of varying the main parameters that can affect the LGG-AP performance:

- Anticipation horizon $H$: we use the values one, three and five.
- Noise in the appearance of the predicted goals: we use the values 0%, 20% and 40%. The noise level represents the probability that a goal does not appear even if the pattern indicates it. We stop at 40%, because with higher values no appearance pattern would be generated.
- Goal ratio: ratio between the number of goals that follow an appearance pattern and the total number of goals. The latter is the sum of the randomly generated ones and the ones that follow a pattern. Both numbers are computed over the whole simulation. We test five different values: $0.7, 0.5, 0.3, 0.2$ and $0.1$, corresponding to a high percentage of pattern-based goals down to a low percentage. We fixed the number of goals that follow an appearance pattern to 100. When the ratio decreases, there are more randomly generated goals in the grid.
- Number of goal appearance patterns that occur at the same time step (one per cell maximum): from one to five.
- Number of agents: only one UAV pursuing the goals.
- Exogenous events produced by the simulator: seismic activity and earth temperature. These parameters take values from zero to three. A volcano eruption is correlated with high values of both parameters, i.e., it is likely to occur. The frequency of an eruption will depend on these values and will be varied over the experiments to test the system's capabilities.

We run each simulation for $T = 2000$ time steps. Since some goals will be generated randomly (subset of $\mathcal{G}_{\mathcal{P}}$ that does not follow an appearance pattern), we run each simulation 10 times to obtain an average penalty. We compare LGG-AP, that updates the set of active goals with the learned goals $G^{t+1} = G^t \cup G_l$, against a Reactive Planning approach that only pursues the current active goals $G$ (not performing any update in the set of goals due to any reasoning process on future goals).

In the experiments the agent pays an increasing penalty of one for each time step when a goal that has already appeared (is active) has not been achieved. The penalty values $P$ obtained by each approach at the end of the simulation depend on the number of goals. Thus, analysis of results can be more difficult to perform. We propose a metric that normalizes the penalty paid by each approach. We denote this metric with $\mathbb{P}$. It is computed as:

$$\mathbb{P} = \frac{(P_r - P_{\text{LGG-AP}})}{AG} \tag{4.4}$$

where $P_r$ is the penalty paid by the Reactive Planning approach, $P_{\text{LGG-AP}}$ is the penalty paid by LGG-AP, and $AG$ is the number of all goals that have appeared during the simulation. The resulting number $\mathbb{P}$ can be seen as the saved penalty for each goal $g \in AG$ by using LGG-AP compared to the penalty paid by the Reactive Planning approach. All the experiments were run on a Ubuntu machine with Intel Core i7-4510U running at 2.00 GHz. The results are presented in the next section.

## 4.5 EXPERIMENTS AND RESULTS

In the first experiment we modify three of the parameters that can affect LGG-AP: anticipation horizon, noise in the appearance of predicted goals and the goals ratio. We compare the performance of LGG-AP against the Reactive Planning approach. In the second experiment we study the performance of LGG-AP if we introduce more than one goal appearance pattern. The third experiment focuses on analyzing the concept drift capabilities of our system, testing the agent's adaptation to new conditions, on-line learning and exploiting a new predictive model.

### 4.5.1 *Influence of Parameter Settings in* LGG-AP

We begin with a one step anticipation pattern ($H = 1$) with 0% of noise. Figure 4.4 shows a sample of the observations generated in the volcano cell. In the remaining cells, we provided a random pattern of observations. A pattern-based goal is only generated in the volcano cell, when the value of the two environment features (seismic activity and temperature) is three. In this case, whenever a goal is generated in this cell, the value of both variables in the previous time step was two.

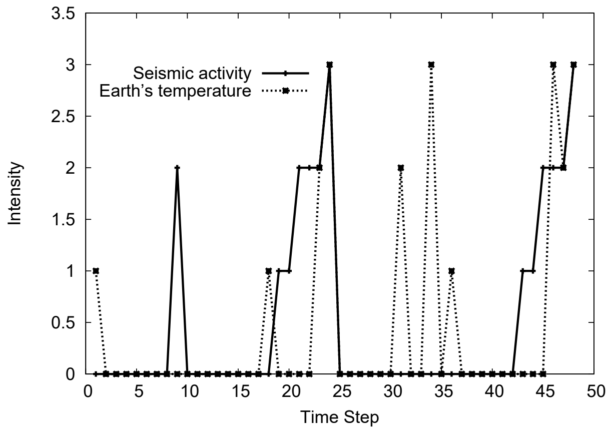


Figure 4.4: Sample of observations generated in the volcano cell.

Firstly, we want to test the performance of the two approaches over time, and the results are shown in Figure 4.5. When LGG-AP collects enough data to build a model, it is able to exploit it correctly by anticipating where the goal will be generated and moving the UAV to the potential future goal location (volcano cell). Thus, it outperforms the Reactive Planning approach. The agent's performance is the same using both approaches until LGG-AP is able to build an accurate predictive model around the time step 350. From then on, LGG-AP outperforms Reactive Planning in all the simulation.

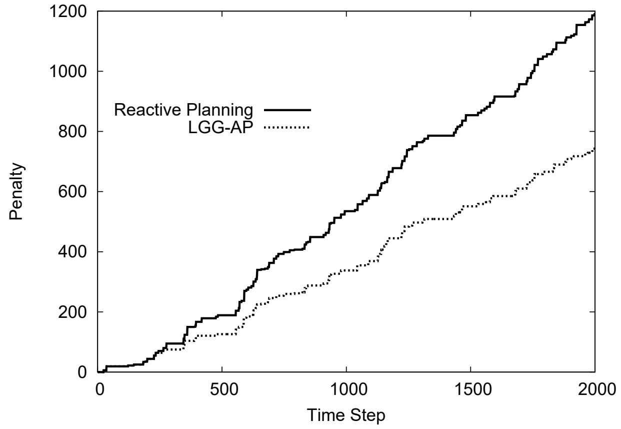


Figure 4.5: Accumulated penalty of LGG-AP and Reactive Planning. The penalty at each time step of goals is $k_g = 1$, $H = 1$ and noise is 0%.

We then conduct several experiments where we modify the parameters described in the experimental setting. The results are shown in Figure 4.6. The difference between LGG-AP and Reactive Planning is bigger when the predicted model is more accurate, as expected. With 0% and 20% noise levels, $\mathbb{P}$ increases as the goal ratio decreases (there are more random goals), until a point where there are many goals where most of them are random. At this point LGG-AP performance becomes similar to that of Reactive Planning, given that both approaches have more opportunities (goals) to decrease the penalty paid. Anticipating provides a smaller advantage.

As the noise levels are bigger, $\mathbb{P}$ reaches its best value with less goals and more percentage of them related with the goal appearance pattern. If the goals were introduced randomly without following any appearance pattern, the learning component would not be able to extract any model that correctly predicts goals in the future. In these cases the goal's set of LGG-AP and Reactive Planning would be the same since the set of future goals would be empty. Consequently, their performance would be similar, taking into account only the current goals.

We obtain better results anticipating the goals one or three time steps rather than five. In this case $\mathbb{P}$ values are higher when there are fewer goals. The best anticipation value is closely related to the size of the grid. The experiments show that for this particular $10 \times 10$ grid, going for the goals five time steps before their appearance is not as good as doing it three or one time steps before.

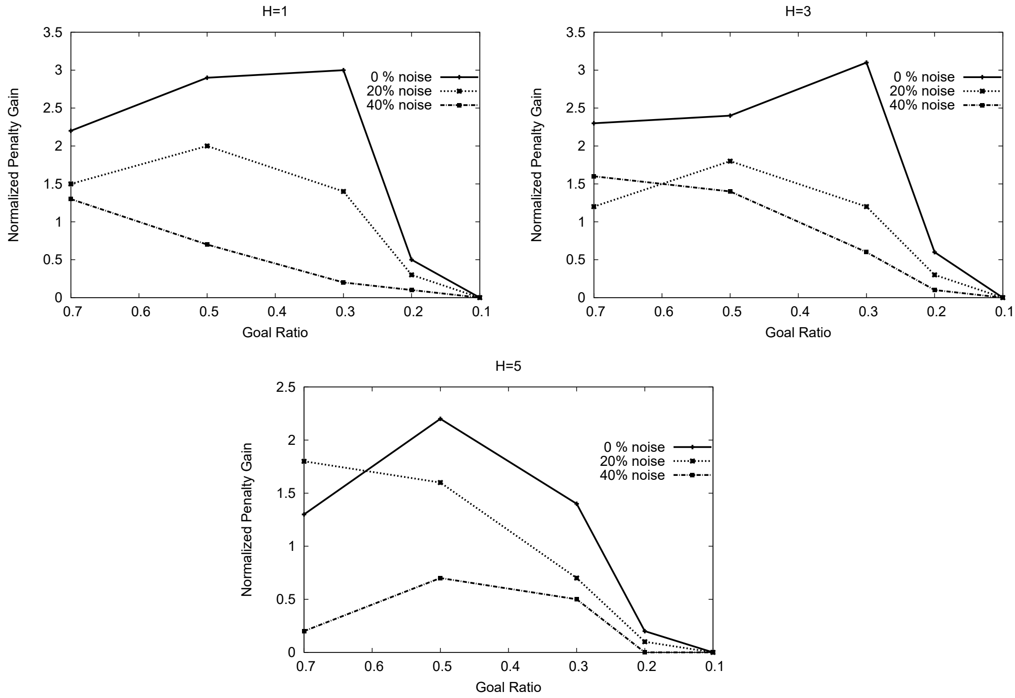


Figure 4.6: From left to right, top to bottom: one, three and five time steps anticipation. The x-axis represents the ratio between the number of goals that follow an appearance pattern and the total number of goals. The y-axis represents the value of $\mathbb{P}$ for the different noise levels in the prediction.

Even in the worst case, in which the agent learns an inaccurate predictive model, LGG-AP performance is at least as good as Reactive Planning, independently of the number of goals to achieve or the number of steps that goals are anticipated. This difference in the value of $\mathbb{P}$ between Reactive and LGG-AP is very relevant, since it denotes the difference in terms of penalty per goal, assuming that all goals have an homogeneous associated penalty equal to one. This is not the case in most domains, even in the one that we are presenting here. The real penalty related to a volcano eruption is not the same as the one for not taking a picture of a crop zone. In these cases, the benefit of using LGG-AP instead of Reactive Planning would increase considerably. As an example, in case we had an homogeneous penalty of 1000 for each goal, and 50 goals, a difference of three (as in Figure 4.6) would mean a penalty of $1000 \times 50 \times 3 = 150000$.

### 4.5.2 *Analysis of the Number of Patterns*

In this experiment we study how the number of goal appearance patterns influence LGG-AP. We introduce from one to five volcanoes uniformly distributed over the grid. We compare the $\mathbb{P}$ value fixing the previous parameters to: 0% noise level, 0.5 goal ratio and $H = 3$. The results are shown in Figure 4.7, which follows the same peak behavior as the previous ones. The system is able to capture every appearance pattern, flying to the places where goals will appear in the future. LGG-AP always outperforms Reactive Planning. $\mathbb{P}$ raises until it reaches the best value when there are three patterns in the grid (three volcanoes). When there are many patterns, we can observe the same behavior as when there are many goals and LGG-AP performance decreases. But the difference with the Reactive Planning approach is still outstanding in any case.

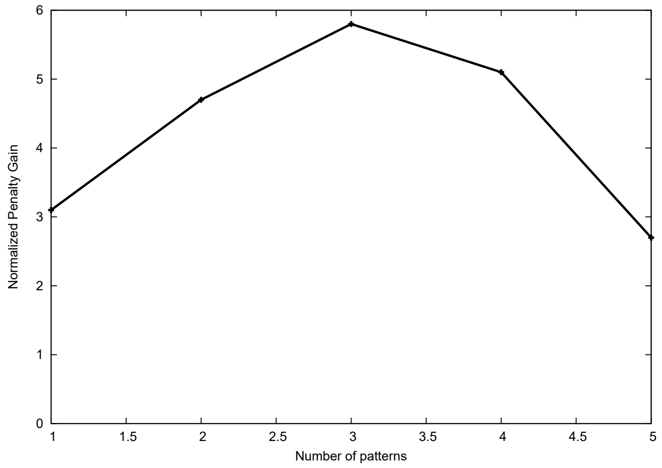


Figure 4.7: Normalized penalty obtained by using different number of goal appearance patterns. The x-axis represents the number of appearance patterns that could occur at the same time. The y-axis represents the value of $\mathbb{P}$.

### 4.5.3 *Ability to Handle Concept Drift*

In the previous experiments we have shown that LGG-AP outperforms Reactive Planning in terms of the penalty they pay. In those cases, the agent learns patterns that do not change over time and exploits them. In most real world domains these goal appearance patterns change and the agent should discover these new patterns, adapting to them. This is the concept drift paradigm (Widmer & Kubat, 1996).

To test the capability of adapting to new goal patterns, we generate a one time step anticipation pattern in the volcano cell and we change it to a random pattern at time step 850. At that time step, pattern-based goals will not appear any more in the volcano cell and pattern-based goals will appear in the lake's cell. The appearance of goals in the lake will be correlated only with the temperature variable unlike the previous pattern that also depends on the seismic value.

As we can see in Figure 4.8, LGG-AP starts outperforming Reactive Planning from the beginning of the execution. Around time step 850, when the pattern changes, LGG-AP deteriorates its performance for a period of time. At this point the agent is building two predictive models in parallel: the one that suggests the appearance of goals in the volcano area, which is starting to decrease its accuracy, and the new one that only predicts goals in the lake. These two patterns conflict for a while, i.e., occur simultaneously, leading to similar performance between Anticipatory and Reactive planning. This occurs until the learning algo-

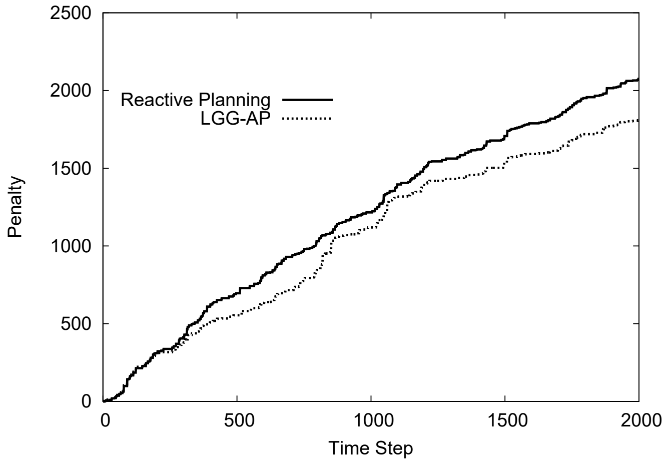


Figure 4.8: Accumulated penalty paid by Reactive Planning and LGG-AP. The goal appearance pattern changes at time step 850.

rithm discards the old pattern due to its inaccuracy and returns a new predictive model. After time step 1200, LGG-AP is able to recover and again outperforms Reactive Planning.

### 4.5.4 *Centroids and Anticipatory Planning: Combining* GRS *and* LGG-AP

Our final experiment aims at combining GRS and LGG-AP. Using LGG-AP, the drone will start approaching a goal once it enters in the set of active goals. After achieving the goal, the agent will only move iff that set is not empty. A goal can be in that set either if (1) it has already appeared; or (2) LGG-AP predicts its appearance. If none of this occurs, the drone will stand still until a new goal is introduced into the set of active goals. We conjecture that a better way to proceed would be to start moving to those points that are closer to the places where goals are *predicted* to appear, even if they will not appear in the short time. These points are the centroids of the planning task, which can be computed using GRS. By heading to them the drone will be closer to the predicted goals, paying even less penalty than only using LGG-AP.

We generated 10 random problems following the experimental setting outlined in Section 4.5.2, where LGG-AP learns more than one goal appearance pattern. In this case we fix the number of goals that follows a pattern to three. For each problem, we let the simulation run for 2000 time steps, reporting the accumulated penalty paid for each approach: Reactive Planning, LGG-AP, and LGG-AP + GRS. Centroids are computed online using the optimal version of GRS. The goals considered for the centroid computation are those for which LGG-AP have learnt a model. All goals considered by GRS have the same associated weight.

Figure 4.9 shows the comparison between the three approaches in terms of the accumulated penalty paid by each of them at the end of each simulation. As we conjectured, combining LGG-AP and GRS outperforms both using LGG-AP only

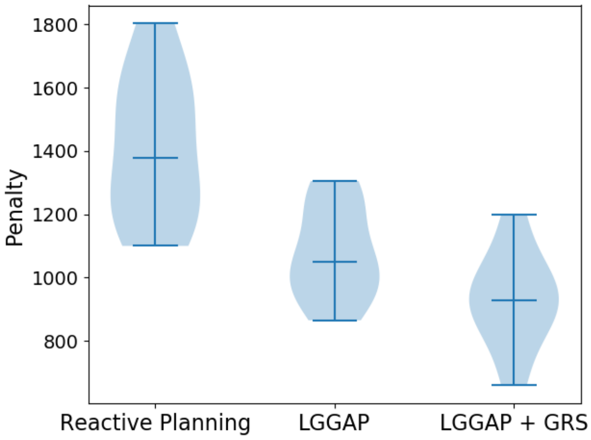


Figure 4.9: Accumulated penalty paid by Reactive Planning, LGG-AP and LGG-AP +GRS. The central horizontal line indicates the median of the distribution, while the other two lines indicate the maximum and minimum values. A wider blue shadow indicates that more points have that value.

and Reactive Planning. It reduces the penalty paid compared to LGG-AP in all the problems, with the margin depending on the appearance of the randomly distributed goals. The only overhead required by this last approach is to compute the centroids of the problem. This can be done using the optimal version of GRS when: (1) the problem is small; (2) the centroid can be computed offline, i.e., we do not require real-time response; and (3) the goals' appearance patterns do not constantly change over time. In case these premises are not fulfilled, it is always possible to use a greedier version of GRS, which will compute the centroids faster. Even if the returned centroid is not optimal, it will help the agent to better locate itself to later achieve the goals, as we discussed in the previous Chapter.

## 4.6 RELATED WORK

Most works in the context of goal reasoning have focused on the Goal-Driven Autonomy (GDA) conceptual model (Cox, 2007; Klenk et al., 2013). A GDA agent generates a plan to achieve a given goal together with its expectations; i.e., the set of constraints that are predicted to hold in the partial states generated when executing the plan. The agent monitors the environment for discrepancies between its expectations and its observations during execution. If the expectations do not match the observed states or if the current plan fails, the GDA agent can formulate a new goal (Cox et al., 2016; Maynord et al., 2013). The first works on GDA formulated new goals using rule-based *principles*, which describe situations where specific goals should be generated (Coddington et al., 2005). These rules were hand-crafted by a domain expert.

Few works have studied the addition of learning capabilities to the agents in the goal reasoning process. Powell *et al.* extended the ARTUE agent (Klenk et al., 2013) with the ability to learn goal selection knowledge through interaction with an expert (Powell et al., 2011). They framed this as a case-based supervised learning task that employs active learning. Unlike their approach, we do not need the interaction with a human to formulate goals, since we learn this ability by collecting examples from the agent execution in an environment, that can be real or simulated.

Without the need of human interaction, Jaidee *et al.* summarized some work on creating GDA agents capable of automatically acquiring knowledge using Case-Base Reasoning (CBR) and Reinforcement Learning (RL) methods (Jaidee et al., 2011). In this case, the problem domains are Real-Time Strategy (RTS) games, more specifically DOM and Wargus. Weber *et al.* implemented a method that also uses CBR and intent recognition in order to build GDA agents that learn from demonstration (Weber et al., 2012). They applied the approach to build an agent for the RTS game StarCraft. Molineaux and Aha employed a variant of FOIL (Quinlan, 1990) to learn models of unknown exogenous events in partially observable, deterministic environments and showed how they can be used by a GDA agent (Molineaux & Aha, 2014). They implemented this learning method in FOOLMETWICE, an extension of ARTUE. Maynord *et al.* employ TILDE (Blockeel & Raedt, 1998), a relational learning algorithm, to learn a decision tree for goal prediction in the blocksworld planning domain (Maynord et al., 2013). Finally, Gopalakrishnan *et al.* (2016) learn goals from planning traces in planning domains. Our work differs from those in the sense that they are not generating and reasoning with possible upcoming goals, as we do. While they learn from world states in isolation, we take into account the time context, as we are planning with goals predicted by an on-line learning model.

Regarding the concept of Anticipatory Planning we are addressing in this chapter, while the idea of using Automated Planning taking into account possible upcoming goals comes from previously presented works (Burns et al., 2012; Fuentetaja et al., 2018), our work differs from theirs in some aspects. While they assume they know a priori the goal arrival distribution, we are learning it through the collection of examples from the system's execution. Another difference is that they use a special planner that reasons internally with the upcoming goals distribution and its penalties. We propose to use a classical planner, incorporating either the current or the possible upcoming goals and replanning when a new goal appears.

## 4.7 SUMMARY

In this chapter we have presented an architecture that allows the design and implementation of autonomous agents with learning capabilities. Using this architecture for a small UAV domain, we have shown that an agent can discover opportunities and adapt its behavior as the surrounding environment changes following a concept drift approach.

We have gone further in the goal reasoning concept, letting the agent not only reason with the current state of the world but also with the possible near future. If the agent discovers a goal before it really appears, it can start the planning process sooner, improving its performance.

Finally, we have enumerated the requirements that a domain must fulfill in order to successfully apply Anticipatory Planning. We have presented a list of such domains and selected one of them to perform the experiments. Through a surveillance UAV domain, we have carried out some experiments in order to discuss the main characteristics and parameters that affect LGG-AP. The results show that LGG-AP works, in the worst case, as well as Reactive Planning, outperforming the reactive approach in the rest of scenarios. We obtain the best results when the agent has time to reason about the future goals instead of just be acting all the time. This happens when there are many goals in the grid whether they come from several goal appearance patterns or they are randomly generated.

As we previously discussed in Chapter 3, some of the concepts we introduced there, have a close relationship with the Anticipatory Planning tasks considered in this chapter. If we have a set of predicted goals, i.e., those which we think may appear in the near future, we can compute their centroid and start moving to that state before any goal becomes active. By doing this, we would put the agent in a better position, paying less penalty once a goal appears. As we have shown in the experimental evaluation, both approaches are orthogonal and improves agents' performance. We will further explore their synergies in Chapter 6 in the context of multi-agent settings.

# Part III

# GOAL REASONING IN MULTI-AGENT SETTINGS

# 5 COUNTERPLANNING USING GOAL RECOGNITION AND LANDMARKS

> *"The supreme art of war is to subdue the enemy without fighting."*
>
> *- Sun Tzu, The Art of War*

Consider a police control domain, like the one shown in Figure 5.1. A terrorist has committed an attack and wants to escape, while the police aim at stopping the terrorist before she leaves the city. The terrorist has carefully designed her escape plan buying bus tickets; she therefore needs to go to the bus station and take the bus. Before that, she needs to make a call to a partner. However, she is afraid that her phone is tapped by the police, so she needs to make the call from any of the phone booths distributed over the city. Once she reaches the bus station having made the call from a non-tapped phone, she will be out of reach of the police. On the other side, the police do not know which means of transport the terrorist is going to use to escape. However, they do know that the terrorist may want to leave the city by using train, bus, or plane. The police can move around the city using a patrol car and set controls in the white tiles. They can also tap the phone booths from the police station. The police have control over some of the city cameras, located at different key points around the city, which helps them identify the terrorist's executed actions. In this situation, a valid approach for the police would be to: (1) observe the terrorist's movements to infer which means of transport she is trying to use to escape (train, bus, or airplane); and (2) generate a plan to stop her in her way to the corresponding station. Given the threat posed by the terrorist, the police want to generate plans that guarantee that the terrorist will be stopped regardless of her movements. These plans may stop the terrorist as soon as possible to avoid panic breaking out, or may minimize the cost of the police operation.

This police control domain is just an example of a competitive scenario where an agent is trying to prevent an opponent from achieving her goals. This is a common task in domains such as security (Boddy et al., 2005; Hoffmann, 2015; Pita et al., 2008; Tambe, 2012), real-time strategy games (Ontañón et al., 2013), or military applications (Borck et al., 2015). However, most approaches to solve these counterplanning problems are domain-dependent. On the goal recognition side, they use plan libraries (Kabanza et al., 2010), rules (Carbonell, 1981), or behavior libraries (Borck et al., 2015; Bowling et al., 2004) to detect their opponent's goals. On the action reasoning side, they use stored policies (Carbonell, 1981), ask humans for guidance following a mixed-initiative paradigm (Jarvis et al., 2004), or require heavy knowledge engineering processes such as HTN-based approaches (Willmott et al., 2001).

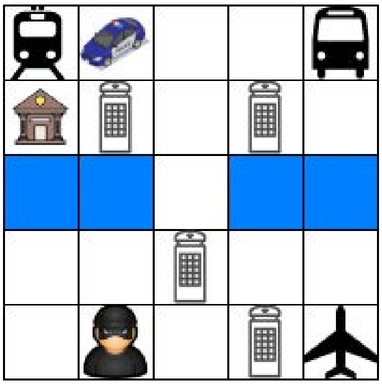


Figure 5.1: Police control domain where the police wants to stop a terrorist that is trying to leave the city after the terrorist has committed an attack. Blue tiles depict a river that agents cannot traverse.

In this chapter we propose a novel approach that is fully automated and domain-independent, both on the goal recognition and the planning sides, to generate strategies (plans) that prevent opponent agents from achieving their goals. It is based on goal recognition, landmarks, and classical automated planning:

1. Goal recognition aims to infer an agent's goals from a set of observations. In this work, we use Ramírez and Geffner (2009; 2010) planning-based goal recognition framework to infer an opponent's goal G given a set of observations.

2. Fact landmarks are propositions that must be true in all valid solution plans (Hoffmann et al., 2004). In this work, we use landmarks to identify subgoals involved in the achievement of g. We filter the set of landmarks and select a *counterplanning landmark*, which is a landmark where the opponent could be stopped and therefore unable to reach her goals.

3. Generate a plan, namely *counterplan*, to achieve the *counterplanning landmark*, and therefore to prevent the opponent's goal achievement.

The rest of the chapter is organized as follows. Section 5.1 defines some planning formalisms and functions we use in this chapter. Section 5.2 describes the counterplanning setting and formally defines counterplanning tasks. Section 5.3 introduces our fully automatic domain-independent approach to solve counterplanning tasks. Section 5.4 presents an empirical study in different competitive planning domains. Section 5.5 shows some experiments in which we use a variation of our counterplanning approach that works in Starcraft, a real-time strategy game. Section 5.6 discusses related work in the area. Finally, Section 5.7 provides some conclusions and highlights possible future works.

## 5.1 PRELIMINARIES

In this chapter we will often use the term *step* when talking about a plan.

**Definition 5.1.** *Given a plan $\pi = (a_1, \ldots, a_x, \ldots, a_n)$, we say that a given **fact** $f_i \in F$ **is needed at a (time) step** $x \in \mathbb{N}$ if $f_i \in pre(a_x)$*

**Definition 5.2.** *Given a plan $\pi = (a_1, \ldots, a_x, \ldots, a_n)$ and a **fact** $f_i \in F$, we say $x$ is the **last step** in which $f_i$ is **needed** in $\pi$ iff $\forall x' \in [x+1, n], f_i \notin pre(a_{x'})$.*

Moreover, sometimes we will be interested in computing not only one but a set of plans that solves a planning task $\Pi$. We refer to $\mathcal{P}_\Pi$ as the set of all plans that solve a planning task $\Pi$, and $\mathcal{P}^*_\Pi$ as the set of all plans that optimally solve $\Pi$.

During the chapter, we will also use the following functions to compute some of the solutions and planning concepts described in Chapter 2.

- Planner($\Pi$) to refer to an algorithm that computes an optimal plan $\pi$ from a planning task $\Pi$.
- StrongPlanner($\Pi$) to refer to an algorithm that computes an strong optimal plan $\pi$ from a planning task $\Pi$. This planner checks the non-interfering constraints (Definition 2.5) at each node expansion, ensuring the opponent agent cannot violate them with less or equal cost than the $g$ value of the node to expand.
- RecognizeGoals($F, A, I, \mathcal{G}, \mathcal{O}, r$) to refer to an algorithm that solves a goal recognition task (see Definition 2.9) assuming equal $Pr$ for each goal. The parameter $r$ indicates the configuration used to solve the compiled tasks. This function returns a set containing the goal(s) in $\mathcal{G}$ with higher probability.
- ExtractLandmarks($F, A, I, G$) to refer to an algorithm that computes a set of conjunctive fact landmarks (see Definition 2.3) $\mathcal{L}_\Pi$ from a planning task $\Pi$.

Finally, in Definition 2.6 we assumed that the state remains the same when two actions interfere. In this chapter we will harden this definition and will assume that the opponent will be able to execute its action when two actions interfere.

## 5.2 DOMAIN-INDEPENDENT COUNTERPLANNING

In this section we introduce our counterplanning setting. Firstly, we describe a *counterplanning episode* in terms of the set of assumptions we make:

- We consider two planning agents acting in the same environment. A seeking agent, SEEK, that wants to achieve a goal; and a preventing agent, PREV, that wants to stop the seeker from achieving its goal.
- SEEK will try to achieve a goal $G_{\text{SEEK}}$. That goal will not change during the counterplanning episode.
- PREV knows SEEK's model, i.e., its actions. However, PREV does not know SEEK's actual goal $G_{\text{SEEK}}$ or plan $\pi_{\text{SEEK}}$.

- PREV and SEEK models are coupled, i.e., they share some propositions $p \in F$. More specifically, PREV can delete(add) some propositions appearing in SEEK's actions preconditions.

- PREV knows a set of potential goals that SEEK might try to achieve, $\mathcal{G}_{\text{SEEK}}$. SEEK actual goal is always within that set, $G_{\text{SEEK}} \subseteq \mathcal{G}_{\text{SEEK}}$.

- SEEK is assumed to be rational; i.e., it is assumed to take optimal paths to its goals. However, it might change its optimal behavior upon observing PREV starting to move, thus being able to execute any suboptimal plan from that moment on.

- Both agents have full observability of other's actions.

- PREV's goal is initially set to empty; hence, it does not have an initial plan. She will try to formulate a goal and a plan during the counterplanning episode to prevent SEEK from achieving $G_{\text{SEEK}}$.

Most of these assumptions are common either in goal recognition research or real world applications. For instance, in most real world domains where counterplanning can be useful (e.g. police control, cyber security, strategy games,. . . ), the preventing agent knows her enemy's model and a set of potential goals that she is interested in. The rationality assumption is common in goal recognition research (Ramírez & Geffner, 2009).

Considering the above mentioned assumptions, we can formally define a counterplanning task.

**Definition 5.3.** *A **counterplanning task** is defined by a tuple* $\mathcal{C} = \langle \Pi_{\text{SEEK}}, \Pi_{\text{PREV}}, \mathcal{O}_{\text{PREV}}, \mathcal{G}_{\text{SEEK}} \rangle$ *where:*

- $\Pi_{\text{SEEK}} = \langle F_{\text{SEEK}}, A_{\text{SEEK}}, I_{\text{SEEK}}, G_{\text{SEEK}} \rangle$ *is the planning task of* SEEK*.*

- $\Pi_{\text{PREV}} = \langle F_{\text{PREV}}, A_{\text{PREV}}, I_{\text{PREV}}, G_{\text{PREV}} \rangle$ *is the planning task of* PREV*.*

- $\mathcal{O}_{\text{PREV}} = (o_1, \dots, o_m)$ *is a set of observations in the form of executed actions that* PREV *receives from the execution of* SEEK*'s plan* $\pi_{\text{SEEK}} = (o_1, \dots, o_m, a_{m+1}, \dots, a_k)$*. The notation differentiates between observations (previously executed* SEEK*'s actions),* $o_i$*, and future actions to be executed by* SEEK*,* $a_j$*.*

- $\mathcal{G}_{\text{SEEK}}$ *is the set of goals that* PREV *currently thinks* SEEK *can be potentially pursuing.*

The meaning of *currently* in the definition of $\mathcal{G}_{\text{SEEK}}$ indicates that this set changes according to the set of observations $\mathcal{O}_{\text{PREV}}$. In fact, given that we are assuming rational (optimal) agent behavior to achieve its goal, $\mathcal{G}_{\text{SEEK}}$ monotonically decreases with each observation $o \in \mathcal{O}_{\text{PREV}}$ (2009). In other words, PREV will consider less (or equal) SEEK's potential goals as SEEK executes more actions of her plan.

As we have discussed, at the beginning of the counterplanning task PREV has not performed any action (her goal and plan are empty). Therefore, the new composite state $I_c$ after receiving the set of observations $\mathcal{O}_{\text{PREV}}$ is defined as $I_c = \Gamma(I_{\text{SEEK}} \cup I_{\text{PREV}}, \mathcal{O}_{\text{PREV}})$. The solution to a counterplanning task is a preventing agent's plan $\pi_{\text{PREV}}$, namely a counterplan. We define valid counterplans[1] as follows:

**Definition 5.4.** *A counterplan* $\pi_{\text{PREV}}$ *is a **valid counterplan** iff its joint execution from* $I_c$ *with the remaining of* SEEK*'s actual plan* $\pi_{\text{SEEK}}$*, results in a state* $I'_c$ *from which* SEEK *cannot achieve any of the goals in* $\mathcal{G}_{\text{SEEK}}$*, and therefore its actual goal* $G_{\text{SEEK}}$*. Formally:*

$$I'_c = \Gamma_J(I_c, \pi_{\text{PREV}}, \pi_{\text{SEEK}} \setminus \{\mathcal{O}_{\text{PREV}}\}) \tag{5.1}$$

$$\forall G_i \in \mathcal{G}_{\text{SEEK}}, \nexists \pi'_{\text{SEEK}} \mid G_i \subseteq \Gamma(I'_c, \pi'_{\text{SEEK}}) \tag{5.2}$$

Note that the definition of a valid counterplan is quite strict: it must prevent SEEK from achieving any of the goals PREV thinks she is currently trying to achieve. Moreover, it will only be a valid counterplan with respect to the *actual* plan SEEK is executing $\pi_{\text{SEEK}}$, which PREV does not know. Therefore, the validity of a counterplan can only be tested *a posteriori*. Going back to our running example, in the limit case where the terrorist has not started moving, $\mathcal{O}_{\text{PREV}} = \emptyset$, the police would need to compute a counterplan that blocks the achievement of any of the terrorist's goals. In case such a counterplan does not exist, the police should wait until they infer the terrorist's true goal by observing more actions. Other approaches would involve *betting* for one of the goals and setting a control at one of the stations. However, we are aiming at domains such as police control where we want to make sure, i.e., guarantee, the opponent is stopped. In the rest of the chapter, we will use the terms *block*, *stop*, or *prevent* as synonyms of an agent executing a counterplan.

### 5.2.1 *Counterplanning Landmarks*

In automated planning, the only way of ensuring that a goal is not achievable is to thwart any of the planning landmarks involved in it (as a reminder, goals are landmarks by definition). If those landmarks cannot be achieved again as we are assuming here, this would prevent SEEK from achieving the goal regardless the plan it follows.

**Proposition 5.1.** *There is no plan* $\pi$ *that achieves a goal* $G_i \in \Pi$ *if any conjunctive landmark* $L_i \in \mathcal{L}_\Pi$ *is negated.*

*Proof.* Given that landmarks have to be true in each $\pi$ that solves $\Pi$, if any landmark is negated, then there will be no plan that achieves the goal. □

Note that we are assuming SEEK can execute any (suboptimal) plan upon PREV starts executing its plan. If we were assuming optimal SEEK behavior until it reaches its goal, we would be reducing the possible plans SEEK can take, thus increasing the number of *landmarks*, i.e., the predicates that we could delete (add) to prevent SEEK from achieving its goal. PREV does not know SEEK's actual goal but a set of potential goals she might be trying to achieve $\mathcal{G}_{\text{SEEK}}$.

[1] In the rest of the section we will use the terms counterplan and valid counterplans indistinctly.

**Definition 5.5.** *Given a counterplanning task $\mathcal{C}$, we refer to the* ***set of all the potential planning tasks*** *that* PREV *currently thinks* SEEK *might be solving as $\mathcal{X}_{\text{SEEK}}$.*

$$\mathcal{X}_{\text{SEEK}} = \{\langle F_{\text{SEEK}}, A_{\text{SEEK}}, I_{\text{SEEK}}, G_i \rangle : G_i \in \mathcal{G}_{\text{SEEK}}\} \tag{5.3}$$

Therefore, PREV must find a counterplan that deletes (or adds) any of the fact landmarks that are *common* in all the planning tasks in $\mathcal{X}_{\text{SEEK}}$. We refer to this set of landmarks as counterplanning landmarks.

**Definition 5.6.** *Given a counterplanning task $\mathcal{C}$, a fact in $L_i \in F_{\text{SEEK}}$ is a* ***counterplanning landmark*** *iff:*

- $L_i \in \mathcal{L}_{\Pi_j}, \forall\, \Pi_j \in \mathcal{X}_{\text{SEEK}}$*; and*
- *If $L_i$ is a positive literal, $\exists a \in A_{\text{PREV}}$ such that $L_i \in del(a)$. If $L_i$ is a negative literal, $\exists a \in A_{\text{PREV}}$ such that $L_i \in add(a)$.*

*We refer to the set of counterplanning landmarks of a counterplanning task as $CPL_{\mathcal{C}}$, and* EXTRACTCPL*($\mathcal{C}$) as the function that computes them.*

PREV will set any of these counterplanning landmarks as her goal $G_{\text{PREV}}$, computing a counterplan that deletes (adds) it, making impossible for SEEK to achieve her goal $G_{\text{SEEK}}$. However, some of the counterplanning landmarks might be *closer* to SEEK than PREV. This means that we cannot ensure PREV will be able to delete them before (in less steps) SEEK stops needing them.

Following the notions of weak and strong plans outlined in the Background Section, we differentiate between weak and strong counterplanning landmarks.

**Definition 5.7.** *A counterplanning landmark $L_i \in CPL_{\mathcal{C}}$ is a* ***strong counterplanning landmark*** *iff* PREV *can always delete (add) it applying less actions from $I_c$ than the last step of an optimal plan in which* SEEK *needs $L_i$. Given $\mathcal{P}^*_{\Pi_{\text{SEEK}}}$, which contains all the optimal plans that achieve any of the goals $G_i \in \mathcal{G}$; a function* LASTSTEP*($L_i$,$\pi$) that returns the last step in which $L_i$ appears in any precondition of a plan $\pi$; and a function* STRONGPLANNER *that returns a strong optimal plan, we formally define strong counterplanning landmarks as follows:*

$$\nexists \pi_{\text{SEEK}} \in \mathcal{P}^*_{\Pi_{\text{SEEK}}} \mid \text{LASTSTEP}(L_i, \pi_{\text{SEEK}}) < c(\text{STRONGPLANNER}(F_{\text{PREV}}, A_{\text{PREV}}, I_c, \neg L_i)) \tag{5.4}$$

These will be the counterplanning landmarks that PREV will be able to delete before SEEK stops needing them, regardless the plan SEEK follows. We refer to the set of strong counterplanning landmarks of a counterplanning task as SCPL, and EXTRACTSCPL($\mathcal{C}$, CPL$_{\mathcal{C}}$) as the function that computes them. This function is detailed in Algorithm 2.

We start computing all the SEEK optimal plans to achieve each goal in $\mathcal{G}_{\text{SEEK}}$ through the COMPUTEOPTIMALPLANS function (lines 2-4). Then, we iterate over the counterplanning landmarks, computing the minimum number of steps, i.e., the optimal cost of a PREV's plan to delete that counterplanning landmark $L_i$ (line 7). The plan is computed through the STRONGPLANNER function, which

**Algorithm 2** ExtractSCPL

```
Require: Counterplanning task $\mathcal{C}$, counterplanning landmarks $CPL_{\mathcal{C}}$
Ensure: Strong Counterplanning Landmarks SCPL
 1: SCPL $\leftarrow \emptyset$
 2: for $G_i \in \mathcal{G}_{\textsc{seek}}$ do
 3:     $\Pi \leftarrow \langle F_{\textsc{seek}}, A_{\textsc{seek}}, I_c, G_i \rangle$
 4:     $\mathcal{P}_{\Pi} \leftarrow$ComputeOptimalPlans($\Pi$)
 5:     for $L_i \in CPL_{\mathcal{C}}$ do
 6:         minStepSeeker $\leftarrow \infty$
 7:         minStepPreventer $\leftarrow$ c(StrongPlanner($F_{\textsc{prev}}, A_{\textsc{prev}}, I_c, \neg L_i$))
 8:         if minStepPreventer $\neq \infty$ then
 9:             minStepSeeker $\leftarrow$ GetMinLastStep($L_i$,$\mathcal{P}_{\Pi}$)
10:         if minStepPreventer $<$ minStepSeeker then
11:             SCPL $\leftarrow$ SCPL $\cup \langle \neg L_i$, minStepSeeker, minStepPreventer$\rangle$
12:         else
13:             $CPL_{\mathcal{C}} \leftarrow CPL_{\mathcal{C}} \setminus \{L_i\}$
14: SCPL $\leftarrow$ GetMinLastStepSeeker(SCPL)
15: return SCPL
```

runs an optimal planner that only returns strong plans. If the call to StrongPlanner returns an empty plan, i.e., there is no strong plan to get $\neg L_i$, its cost will be infinite and we will not include that landmark into the set of strong counterplanning landmarks. Therefore, minStepPreventer would store the sooner (minimum number of steps) prev can safely falsify the given counterplanning landmark.

We do something similar for seek, but in this case we cannot compute an optimal plan to get the landmark. This is because some landmarks are already true in $I_c$, and the cost of an optimal plan that achieves them would be zero. This is the case of many landmarks of planning tasks such as those involving resources that are consumed as actions are executed. What we need is to compute the last time step of any optimal plan in which seek requires the given landmark to achieve the goal, and take the minimum step among all the plans. This would be the last state in which prev could block seek if selecting that landmark. We perform this computation in the GetMinLastStep function, storing its result in the minStepSeeker variable. Finally, we compare if prev can falsify $L_i$ before seek stops needing it by comparing minStepPreventer and minStepSeeker in line 10. If that is the case, we incorporate $\neg L_i$ to the set of strong counterplanning landmarks SCPL. We add the minimum step at which seek stops needing it and prev can falsify it in order to later reason about which SCPL prev should delete (add). Otherwise, we remove $L_i$ from the set of counterplanning landmarks in line 13. We can safely remove $L_i$ because if there exists one goal for which seek stops needing $L_i$ before prev can delete it, $L_i$ will not be a strong counterplanning landmark, and therefore we do not have to reason about it in next iterations of the algorithm, i.e., for other goals.

Once we have performed this process for every goal, we might have different minimum seeker time steps for the same counterplanning landmarks in

SCPL. This is because SEEK might stop needing some landmarks before, i.e., in earlier steps, in some goals and later in others. To solve this, we get the minimum seeker last time step for each $L_i \in$ SCPL (line 15). All these decisions: (1) compute all optimal plans; (2) get the minimum last step at which SEEK stops needing $L_i$ in any goal $G_i \in \mathcal{G}$, are made to keep some stopping guarantees, considering a worst case SEEK agent following the plan that stops needing the landmarks as soon as possible. Therefore, SCPL $\subseteq$ CPL$_\mathcal{C}$, and all the counterplanning landmarks (including the strong ones) are weak counterplanning landmarks WCPL, in the same way as strong plans are also valid weak plans.

**Theorem 5.1.** *Given a counterplanning task $\mathcal{C}$, if there exists a landmark $L_i \in$ CPL$_\mathcal{C}$ that* PREV *can always delete (add) before* SEEK *stops needing it, i.e., a strong counterplanning landmark,* EXTRACTSCPL *will find it.*

*Proof.* Let us assume that EXTRACTSCPL cannot find that strong counterplanning landmark $L_i$. This could only happen in the following cases:

- EXTRACTSCPL misses reasoning about $L_i$. But this is not possible, since $L_i \in$ CPL$_\mathcal{C}$ and the algorithm is iterating over all $L_j \in$ CPL$_\mathcal{C}$.

- EXTRACTSCPL is overestimating the cost of a PREV's plan to delete $L_i$. But this is not possible, since it is using a strong optimal planner that will return a minimum cost plan to achieve $\neg L_i$.

- EXTRACTSCPL is overestimating the last step in which SEEK needs $L_i$. But this is not possible, since by computing all optimal plans, it is getting a lower bound on the last step SEEK might stop needing $L_i$ (last step $L_i$ appears in any optimal plan). By doing this for all the goals $G_i \in \mathcal{G}_{\text{SEEK}}$ and getting the minimum, EXTRACTSCPL ensures it is not overestimating the last step in which SEEK needs $L_i$.

Since EXTRACTSCPL is reasoning over all the counterplanning landmarks, and it is not overestimating the cost of any agent to delete/stop needing $L_i$, EXTRACTSCPL will find it. □

### 5.2.2 *Counterplan's Properties*

In the same way as there are multiple plans that solve a classical planning task, there might exist different counterplans that solve a counterplanning task. We characterize counterplans by focusing on two key aspects: optimality and stop guarantees. While the first aspect is related to the generation of PREV's goal $G_{\text{PREV}}$ from the set of CPL, the latter is related to the PREV's plan that achieves the generated goal, $\pi_{\text{PREV}}$.

**Counterplan's optimality**. In this work we devise two types of counterplan's optimality metrics. Counterplans can thwart opponent's plans: (1) as soon as possible, i.e., letting SEEK to execute the least number of actions; or (2) at the lowest cost for PREV. We will select a different goal for PREV among the CPL's depending on which metric we are interested in optimizing.

- Blocking the opponent as soon as possible. The aim of PREV is to stop the opponent by generating a counterplan that thwarts SEEK's plan by reducing as much as possible the number of actions she can perform. This approach can be useful in domains such as security, where preventing agents want to stop any thread as soon as possible. PREV will set her goal to the negation of the strong counterplanning landmark which is closest to SEEK's current state. In other words, the (negated) proposition $p_i \in \text{SCPL}$ with minimum minStepSeeker value[2].

$$G_{\text{PREV}} \leftarrow x \mid \langle x, y, z \rangle = \text{argmin}_{\langle \neg L_i, minSeeker, minPrev \rangle \in \text{SCPL}}(minSeek) \tag{5.5}$$

We will later refer to this metric as *asap*.

- Blocking the opponent at the lowest possible cost. The aim of PREV is to stop the opponent by generating a counterplan with the lowest cost (or number of actions in case of unitary costs). This approach can be useful in domains such as real-time strategy games, where PREV's resources are limited. PREV will set her goal to the negation of the strong counterplanning landmark which is closest to her state. In other words, the (negated) proposition $p_i \in \text{SCPL}$ with minimum minStepPreventer value.

$$G_{\text{PREV}} \leftarrow x \mid \langle x, y, z \rangle = \text{argmin}_{\langle \neg L_i, minSeek, minPrev \rangle \in \text{SCPL}}(minPrev) \tag{5.6}$$

We will later refer to this metric as *preventing*.

**Stopping guarantees**. In this work we devise two types of counterplans in terms of their stopping guarantees. The stopping guarantees of a counterplan are not only related to the plan itself, but also to the goal PREV selects. If the goal is generated among the weak counterplanning landmarks, the counterplan will necessarily be weak, given that there is no guarantee PREV can delete the landmark before SEEK stops needing it. On the other hand, if the goal is generated among the strong counterplanning landmarks, the counterplan might be weak or strong.

**Definition 5.8.** *A valid counterplan $\pi_{\text{PREV}}$ is a **strong counterplan** iff its joint execution from $I_c$ with any plan SEEK can execute $\pi'_{\text{SEEK}}$, results in a state $I'_c$ from which SEEK cannot achieve any of the goals in $\mathcal{G}_{\text{SEEK}}$, and therefore its actual goal $G_{\text{SEEK}}$. Given $\mathcal{I}'_c$, which refers to the set of all possible states that we can get from the joint execution of $\pi_{\text{PREV}}$ and any SEEK's plan $\pi'_{\text{SEEK}}$ from $I_c$, we formally define a strong counterplan as follows:*

$$\mathcal{I}'_c = \Gamma_J(I_c, \pi_{\text{PREV}}, \pi'_{\text{SEEK}}) \;\; \forall \pi'_{\text{SEEK}} \in \mathcal{P}_{\Pi_{\text{SEEK}}} \tag{5.7}$$

$$\forall I'_c \in \mathcal{I}'_c, \forall G_i \in \mathcal{G}_{\text{SEEK}}, \nexists \pi'_{\text{SEEK}} \mid G_i \subseteq \Gamma(I'_c, \pi'_{\text{SEEK}}) \tag{5.8}$$

As we have discussed (Proposition 5.1 and Definition 5.7), this is only possible if:

$$\exists \neg L_i \in \text{SCPL} \mid \neg L_i \subseteq \Gamma_J(I_c, \pi_{\text{PREV}}, \pi'_{\text{SEEK}}) \tag{5.9}$$

[2] For the sake of clarity, we use minSeek and minPrev to refer to minStepSeeker and minStepPreventer respectively.

In the rest of cases, the counterplan will be weak and hence PREV will only be able to stop SEEK under some circumstances, i.e., depending on the actions SEEK executes. The main difference between a valid counterplan and a strong counterplan is that valid counterplans only need to prevent SEEK from achieving the goals in $\mathcal{G}$ from the state $I'_c$ resulting from executing $\pi_{\text{PREV}}$ and SEEK's actual plan $\pi_{\text{SEEK}}$. On the other hand, strong counterplans need to prevent SEEK from achieving the goals in $\mathcal{G}$ from *any* state $\mathcal{I}'_c$ (i.e., a set of states) that could result from the joint execution of $\pi_{\text{PREV}}$ and *any* plan SEEK can generate.

## 5.3 COMPUTING COUNTERPLANS

We now introduce DICP (Domain-independent Counterplanning), our algorithm to compute counterplans which is shown in Algorithm 3. This algorithm is always run by PREV. It receives a counterplanning task $\mathcal{C}$ as input, and the following parameters that determine how it will be solved:

- A parameter $s$ that determines whether we want to compute strong or weak counterplanning landmarks when setting PREV's goal.
- A parameter $p$ that determines whether we want to compute strong or weak plans.
- A parameter $m$ that indicates whether we want to block the opponent at the lowest cost or as soon as possible.
- A parameter $r$ that indicates the configuration used to solve the goal recognition tasks.

**Algorithm 3** DICP

```
Require: $\mathcal{C}$
Require: $s,p,m,r$
Ensure: $\pi_{\text{PREV}}$
$\pi_{\text{PREV}} \leftarrow \emptyset$
$G_{\text{PREV}} \leftarrow \emptyset$
$\mathcal{G}_{\text{SEEK}} \leftarrow \text{RecognizeGoals}(F_{\text{SEEK}}, A_{\text{SEEK}}, I_{\text{SEEK}}, \mathcal{G}_{\text{SEEK}}, \mathcal{O}_{\text{PREV}}, r)$
$I_c = \Gamma(I_{\text{SEEK}} \cup I_{\text{PREV}}, \mathcal{O}_{\text{PREV}})$
$CPL \leftarrow \text{ExtractCPL}(\mathcal{C})$
if $CPL \neq \emptyset$ then
    if $s = \texttt{strong}$ then
        $CPL \leftarrow \text{ExtractSCPL}(\mathcal{C}, CPL)$
    while $\pi_{\text{PREV}} = \emptyset$ and $CPL \neq \emptyset$ do
        $G_{\text{PREV}} \leftarrow \text{SelectGoal}(CPL, m, s)$
        $\pi_{\text{PREV}} \leftarrow \text{Planner}(F_{\text{PREV}}, A_{\text{PREV}}, I_c, G_{\text{PREV}}, p)$
        if $\pi_{\text{PREV}} = \emptyset$ then
            $CPL \leftarrow CPL \setminus G_{\text{PREV}}$
return $\pi_{\text{PREV}}$
```

The algorithm first solves a planning-based goal recognition task in order to infer the goal SEEK is pursuing. This is done in the RECOGNIZEGOALS function (line 3). Given a planning domain, initial conditions, a set of candidate goals, a set of observations, and a planner, this function updates the set of candidate goals $\mathcal{G}_{\text{SEEK}}$. Then, the current composite state $I_c$ is updated by advancing the state from $I_{\text{SEEK}}$ applying all actions corresponding to the observations in $\mathcal{O}_{\text{PREV}}$ (line 4). After that, we extract the set of counterplanning landmarks (see Definition 5.6). If this set is not empty, a counterplan exists and we proceed to find it. Otherwise, the counterplanning task is unsolvable and we return an empty plan.

In case we are interested in ensuring SEEK will be stopped, we will set $s$ to `strong` and will compute the strong counterplanning landmarks of $\mathcal{C}$. This is done by the EXTRACTSCPL function, which we already detailed in Algorithm 2. The next step is to (1) select a goal $G_{\text{PREV}}$ from the CPL; and (2) generate a counterplan $\pi_{\text{PREV}}$ that achieves $G_{\text{PREV}}$. We repeat this process until a counterplan is found or no counterplanning landmarks are left in lines 9-13. First, we select $G_{\text{PREV}}$ using the SELECTGOAL function. This function receives as input the set of counterplanning landmarks (which will be composed of strong or weak counterplanning landmarks based on $s$), the metric $m$ we want to optimize, and the parameter $s$. If $s =$ `strong`, SELECTGOAL will set $G_{\text{PREV}}$ using equations 5.5 or 5.6 depending on $m$. Otherwise, SELECTGOAL will randomly choose one of the counterplanning landmarks in CPL to be $G_{\text{PREV}}$. Finally, we use an optimal planner to compute a counterplan that achieves $G_{\text{PREV}}$. We overload the definition of the PLANNER function, adding the parameter $p$ as input. If $p =$ `strong`, the generated counterplan will be strong, guaranteeing that $G_{\text{PREV}}$ will be achieved regardless of SEEK's actions. Otherwise, the generated counterplan will be weak and will only achieve $G_{\text{PREV}}$ in some cases, i.e., if SEEK does not execute certain actions.

### 5.3.1 *Theoretical Properties of the Algorithm*

We now prove that our algorithm is sound, complete, and optimal. We can only provide this theoretical properties when DICP computes strong counterplans, i.e., it is run with $s$ and $p$ set to `strong`, and $r$ is Ramírez and Geffner (2009) approach.

**Theorem 5.2** (Completeness)**.** *Given a counterplanning task $\mathcal{C}$, if there exist a strong counterplan $\pi_{\text{PREV}}$ that solves it,* DICP *will find it.*

*Proof.* Following Definition 5.4, a valid counterplan must prevent SEEK from achieving any of the goals in $\mathcal{G}$ from $I_c$. The only way of ensuring this is to delete (add) a landmark which is common to all the potential goals before SEEK stops needing it (see Definitions 2.3, 5.1, 5.6, 5.7). From Theorem 5.1 we can ensure that if such a landmark exists, Algorithm 2 will find it. Since we are iterating over the set of strong counterplanning landmarks, we will always find a strong counterplan $\pi_{\text{PREV}}$ if it exists. □

**Theorem 5.3** (Soundness)**.** *The plan returned by* DICP *is a valid counterplan.*

*Proof.* If DICP returns a non-empty plan, it will achieve $G_{\text{PREV}}$ through a strong plan $\pi_{\text{PREV}}$. We are setting $G_{\text{PREV}}$ as one of the strong counterplanning landmarks, so $\pi_{\text{PREV}}$ will delete (add) one of the landmarks SEEK needs to achieve its true goal $G_{\text{SEEK}}$ before it stops needing it (Theorem 5.1). Since landmarks must be true along all valid plans that achieve a goal, SEEK will not be able to reach $G_{\text{SEEK}}$. □

**Theorem 5.4** (Optimality)**.** *The plan returned by* DICP *is optimal with respect to the given metric* $m$*.*

*Proof.* When computing the set of strong counterplanning landmarks, we store the sooner SEEK stops needing the landmark $L_i$, and the sooner PREV can achieve $\neg L_i$. The SELECTGOAL function then selects the counterplanning landmark that minimizes $m$ from SCPL. Since we iterate over SCPL until a counterplan is found, DICP will return the plan that minimizes $m$, either minimum steps for SEEK or PREV. □

We cannot ensure any of these properties for other algorithms that may arise from variations of the different parameters of DICP. For instance, even if we compute the set of strong counterplanning landmarks, the returned counterplan might not be sound if the plan that achieves $G_{\text{PREV}}$ is a weak one. In Section 3.3, we conduct an experimental evaluation to show the behavior of the optimal version of DICP. We will also vary some of the parameters and relax some constraints to test how suboptimal versions compare to the optimal in different competitive domains.

### 5.3.2 DICP *Running Example*

We illustrate now all the previous concepts through our POLICE CONTROL running example. We will vary the counterplanning tasks to show the behavior of DICP. In all the examples, the terrorist will be SEEK, and the police will be PREV. Both the police and the terrorist can `move` to their adjacent non-blue tiles. In order to move to a tile, it has to be `(free)` and `(connected)` to the current agent's tile. After the execution of a `move` action, the agent's position changes, the current tile is no longer `(free)`, and the previous tile becomes free again. The police can also `tap-booths` when being at a police station. On the other hand, the terrorist can `make-call` when being at a phone booth that has not been tapped. When the terrorist reaches a terminal having made the call, she will leave the city and the counterplanning episode will have finished.

*Counterplanning Task 1: Strong vs Weak Counterplanning Landmarks*

In the first counterplanning task we want to study (depicted in Figure 5.2), SEEK has not moved and therefore $\mathcal{O}_{\text{PREV}} = \emptyset$. Since the police has not observed any

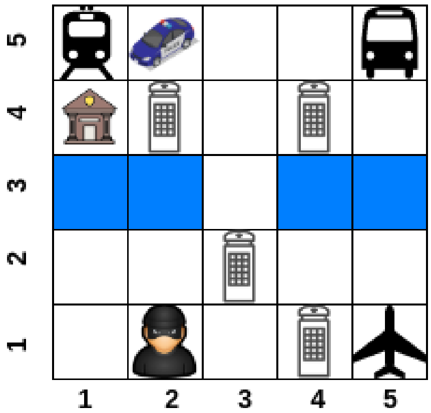


Figure 5.2: Counterplanning task in the POLICE CONTROL domain. SEEK has not moved, and therefore $\mathcal{O}_{\text{PREV}} = \emptyset$.

terrorist's actions, the call to RECOGNIZEGOAL returns that all goals are equally likely:

$$\mathcal{G}_{\text{SEEK}} = \langle\{\texttt{(at-terrorist c1\_5)(call-made)}\},\{\texttt{(at-terrorist c5\_5)(call\_made)}\},$$
$$\{\texttt{(at-terrorist c5\_1)(call\_made)}\}\rangle$$

The next step is to extract the counterplanning landmarks that are common to the three goals. In this case `phone-available` is the only common counterplanning landmark, which PREV can falsify by heading to the police station and executing `tap-phone-booths`.

If we run DICP with s = strong, the next step is to extract the strong counterplanning landmarks. To do that, we compute all the optimal SEEK plans to achieve any of the goals in $\mathcal{G}$, getting from them the minimum last time step in which SEEK stops needing `phone-available`. Note that there exists multiple optimal plans to achieve any of the goals, depending on (1) the path the terrorist follows; and (2) the booth from which she decides to make the call. The sooner step SEEK stops needing `phone-available` in any plan is three, by making the call from any of the phone booths located at two steps from her location in the south part of the map. On the other hand, the sooner PREV can achieve `(not (phone-available))` is also three, by disconnecting the booths from the police station located at two steps from her location. In this case, PREV cannot delete the landmark before SEEK stops needing it (line 10 of Algorithm 2), so it is not a strong counterplanning landmark. Since there are no more counterplanning landmarks, DICP will stop and return an empty plan, meaning there is not strong counterplanning landmark that guarantees the terrorist is blocked.

If we run DICP with s = weak, it would set $G_{\text{PREV}}$ =`(not (phone-available))`. A strong or weak plan (depending on p) that disconnects the booths from the police station would be computed, but this counterplan cannot guarantee that the SEEK is blocked. It will only block SEEK in the cases in which she decides to go to the train or bus stations, also making the call from any of the booths located in the north side of the map.

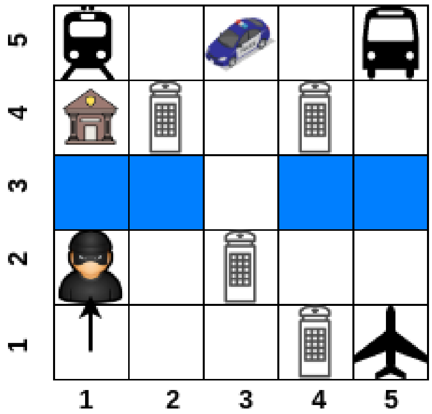


Figure 5.3: Counterplanning task in the POLICE CONTROL domain. SEEK has not moved, and therefore $\mathcal{O}_{\text{PREV}} = \langle$(move c1-1 c1-2)$\rangle$.

*Counterplanning Task 2: Counterplanning Landmarks Optimality*

In the second counterplanning task we want to study (depicted in Figure 5.3), SEEK has moved north from her new location at the bottom-left part of the map. The call to RECOGNIZEGOAL returns that the terrorist escaping through the bus and train stations are equally likely. We can discard the airport since we are assuming SEEK rational behavior. The set of counterplanning landmarks now include some tiles in addition to `phone-available`: `free c3-2`, `free c3-3`, and `free c3-4`. Making the phone booths unavailable and negating `free c3-2` are not strong counterplanning landmarks for the same reasons we discussed in the previous case. On the other hand, the police could safely set a control at the other two tiles, preventing the terrorist from reaching any of the candidate stations. In this case, PREV will move to `free c3-4` if she is interested in blocking SEEK at the lowest possible cost; or to `free c3-3` if she is interested in blocking SEEK as soon as possible, i.e., allowing SEEK to only execute the lowest number of actions of her plan.

## 5.4 EVALUATION IN COMPETITIVE PLANNING DOMAINS

In this first evaluation, we want to test the behavior of DICP in different competitive planning domains. In particular, we aim at testing the scalability of the algorithm, as well as studying what the best conditions for the existence of counterplans are, and comparing how the solutions returned by relaxed versions of DICP compare to the optimal one.

### 5.4.1 *Experimental Setting*

DICP can be instantiated with many different combinations of planners, goal recognition techniques, and parameters. However, we selected a representative set of configurations that lead to the following four versions of the algorithm[3].

[3] All versions of DICP can run with $m$ set to lowest cost or as soon as possible.

Each of them has different strengths and weaknesses we will discuss through the evaluation.

- DICP$^*$: optimal version of DICP run with $s, p =$ strong, and an the optimal goal recognition approach presented in Ramírez and Geffner (2009). in the RECOGNIZEGOALS function. This version guarantees that SEEK will be blocked (if possible) by computing a strong plan that achieves a strong counterplanning landmark.

- DICP$^o$: this version also runs with $s, p =$ strong, but uses Ramírez and Geffner (2010) probabilistic goal recognition approach in the RECOGNIZEGOALS function. It uses an optimal planner to solve the compiled goal recognition tasks.

- DICP$^w$: weak version of DICP run with $s, p =$ weak, and a suboptimal planner to solve the probabilistic goal recognition tasks (2010) in the RECOGNIZEGOALS function. This version will only block SEEK in some cases.

- DICP$^m$: middle ground between DICP$^o$ and DICP$^w$. This version uses a suboptimal planner in the RECOGNIZEGOALS function and $p =$ weak like DICP$^w$. It computes SCPL like DICP$^o$ ($s =$ strong), but using a lighter version of EXCTRACTSCPL: it (1) only computes one optimal plan for SEEK (line 4 of Algorithm 3); and (2) computes the cost of PREV using a weak planner (line 7 of Algorithm 3). Like DICP$^w$, this version will only block SEEK in some cases.

We conducted experiments in the following competitive planning domains. For each domain, we generated 10 counterplanning tasks.

- POLICE CONTROL. Our running example domain, in which a terrorist (SEEK) wants to escape from a city. The police (PREV) can set controls over the city as well as disconnect some phone booths to block the terrorist escape plan. The maps are $10 \times 10$ grids with 25% of obstacles and 10 randomly distributed booths. The set of candidate goals is 3.

- PAINTED BLOCKS-WORDS. In this domain a robotic arm (SEEK) is trying to build some words using a set of available blocks. It can stack and unstack blocks as long as their top is not painted. There is another agent (PREV) that can paint the top part of clear blocks, i.e., blocks that do not have other blocks on top of them. The paint needed to paint each block is randomly distributed over several connected rooms. PREV is also randomly placed in one of these rooms. To paint a block, PREV needs to have the paint and be in the room where SEEK is building the words. Problems in this domain contain 8 blocks and 5 rooms. The blocks are initially piled randomly into several towers. The set of candidate goals is 5, i.e., SEEK might be building five words. The words SEEK needs to build range from 3 to 6 blocks (letters).

- COUNTER LOGISTICS. In this domain an agent (SEEK) can use a fleet of trucks and planes to deliver some packages. It can load the packages into the vehicles as long as they are operative. There is another agent (PREV) that can damage the vehicles rendering them inoperative. To damage the vehicles, PREV needs to collect some artifacts that are distributed over the logistics network. Once it is located at the same place that the vehicle with the artifact, it can damage the vehicle. We used a subset of the problems employed by Ramírez and Geffner (2009), modifying them to randomly placed two artifacts and two PREV trucks in the logistic network.

- ROVERS & MARTIANS. This is a game in which a robotic agent called rover (SEEK) has to conduct several experiments on Mars. It has to navigate to different locations, collecting samples, analyzing them and communicating the results. There is another agent, a martian (PREV) that does not want intruders on its planet. It can screw up the rover's experiments by stealing the samples or interrupting its communications. While the rover can only move through visible locations (i.e., may need to apply a set of actions to go from A to B), the martian can move between any two points of the map (i.e., it can move from A to B applying just one action). Problems in this domain contain 10 locations and 6 samples, in addition to different number of objectives and cameras. Both agents are randomly distributed over the map. The set of candidate goals is 6, i.e., the rover might be trying to get/communicate the results of 6 different experiments.

For all the counterplanning tasks, we select one goal $G_i \in \mathcal{G}_{\text{SEEK}}$ and set is as SEEK's true goal. Then we compute an optimal plan to achieve it ($\pi_{\text{SEEK}}$) using $A^*$ with the LMCUT admissible heuristic (Helmert & Domshlak, 2009) in Fast-Downward (Helmert, 2006). We will refer to this configuration as OPT-LMCUT. This optimal planner configuration is also the one we use in the PLANNER and STRONGPLANNER functions. The set of observed actions $\%Obs$ that $\mathcal{O}_{\text{PREV}}$ will contain is taken to be a subset of $\pi_{\text{SEEK}}$, ranging from the first 10% of the actions up to 70% of the actions. Higher percentages of observations mean that more actions of SEEK's plan have already been executed, and thus observed by PREV. We did not include experiments where the observed sequence is higher than 70% because in these cases SEEK is so close to its goal that it cannot be blocked.

Our fully automatic domain-independent counterplanning approach works with any goal recognition technique. For purposes of this evaluation, we have selected Ramírez and Geffner (2009; 2010) approaches with different planners. For the optimal goal recognition we use HSP* (Haslum & Geffner, 2000), the planner used by Ramírez and Geffner (2009) approach. For the probabilistic goal recognition (Ramírez & Geffner, 2010) we use two different planners: OPT-LMCUT, the aforementioned optimal planner; and LAMA GREEDY (Richter & Westphal, 2010), a satisficing planner that stops the search after it finds the first solution. Finally, we use Katz, Sohrabi and Udrea (2020) top-quality planner to compute all the optimal plans of a given planning task in the COMPUTEOPTIMALPLANS function.

We will measure the following quality and performance metrics in the different experiments:

- $|\mathcal{G}_{\text{SEEK}}|$: number of goals in the candidate goal's set.
- $|\mathcal{L}|$: number of common landmarks among the most likely goals.

- |CPL|: number of counterplanning landmarks among the most likely goals.

- Q: 1 if the actual goal $G_{\text{SEEK}}$ was found to be among the most likely goals $\mathcal{G}_{\text{SEEK}}$ after calling RecognizeGoals, 0 otherwise.

- E: 1 if SEEK is stopped, 0 otherwise. To compute this number, we jointly execute $\pi_{\text{SEEK}}$ and $\pi_{\text{PREV}}$. If the joint execution of both plans does not allow SEEK to achieve its goal, $E = 1$.

- %E: ratio of times a counterplan $\pi_{\text{SEEK}}$ succeeds and blocks SEEK. This number is computed as $\frac{\text{successful counterplans}}{\text{generated counterplans}}$.

- $\%\pi_{\text{SEEK}}$: ratio of $\pi_{\text{SEEK}}$ that SEEK can execute until it is blocked by PREV. Lower numbers are better, and 1 means that SEEK has not been blocked. We only report this ratio when the counterplan successfully blocks SEEK.

- $|\pi_{\text{PREV}}|$: counterplan length (cost) for PREV. We only report the length of the counterplans that successfully block SEEK.

- $t_{\mathcal{C}}$: time to return a solution for a counterplanning task.

**Reproducibility**. The experiments were run on an Intel Core i5-8400 CPU 2.80GHz machines with a time limit of 1800s and a memory limit of 8GB. The full domains for SEEK and PREV are shown in Appendix D.

### 5.4.2 *Evaluation Results*

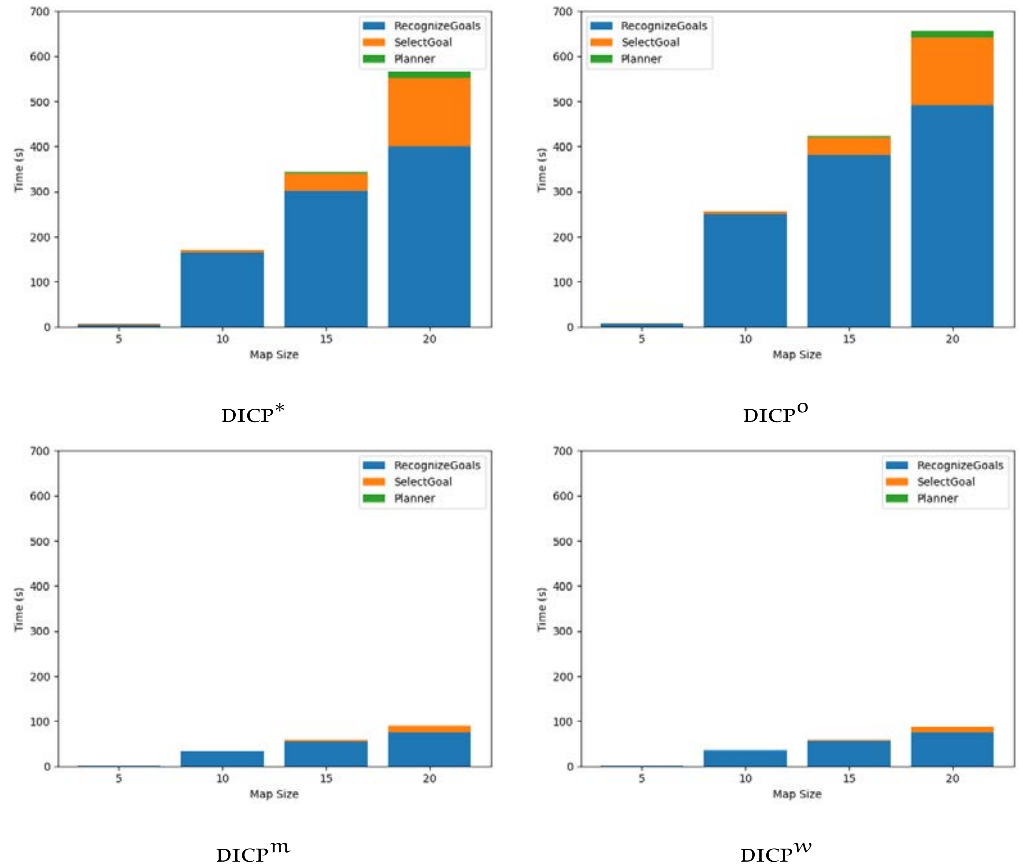
*Scalability Evaluation*

W want to test first how the algorithm scales and the amount of execution time needed by each component of DICP. Theoretically, the scalability of our algorithm depends on two main aspects:

- The size of the problem. The bigger the problem, the harder will be to (1) solve the compiled goal recognition tasks; (2) extract the landmarks of each goal; (3) compute all the optimal plans in the case of DICP*; and (4) compute the optimal plans (either strong or weaks) that achieve $G_{\text{PREV}}$.

- The number of candidate goals $\mathcal{G}_{\text{SEEK}}$. Having more goals implies (1) more tasks to be solved in the planning-based goal recognition; and (2) more planning tasks from which to extract landmarks.

To test which of these aspects influences the execution times of DICP more, we conducted two types of experiments in the POLICE CONTROL domain. Although we only report scalability results in this domain, the obtained results can be extrapolated to other domains. In the first experiment we fixed the number of candidate goals to three and increased the size of the problem from $5 \times 5$ to $20 \times 20$ grids. In the second experiment we fixed the size of the map to $10 \times 10$ and increased the number of candidate goals from 2 to 5. In both cases we kept the other variables (such as the percentage of obstacles) the same and generated 10 random tasks for each configuration with 10% of observations. Then we solved those tasks with each version of DICP and stored the aggregated solving times. Figure 5.4 shows the results we obtained for the first experiment, and Figure 5.5 shows the results of the second one. The total size of the bar depicts the total time employed by the algorithm. Each color shows how each part of the algorithm contributes to the total solving time. We show in blue the time used by RecognizeGoals when inferring the opponent's goal. We show in orange the time needed to select a goal from the set of (strong) counterplanning landmarks. This comprises the time employed by the ExtractCPL, ExtractSCPL and SelectGoal functions. Finally, we show in green the time used by Planner to return a counterplan that achieves $G_{\text{PREV}}$.



DICP$^*$

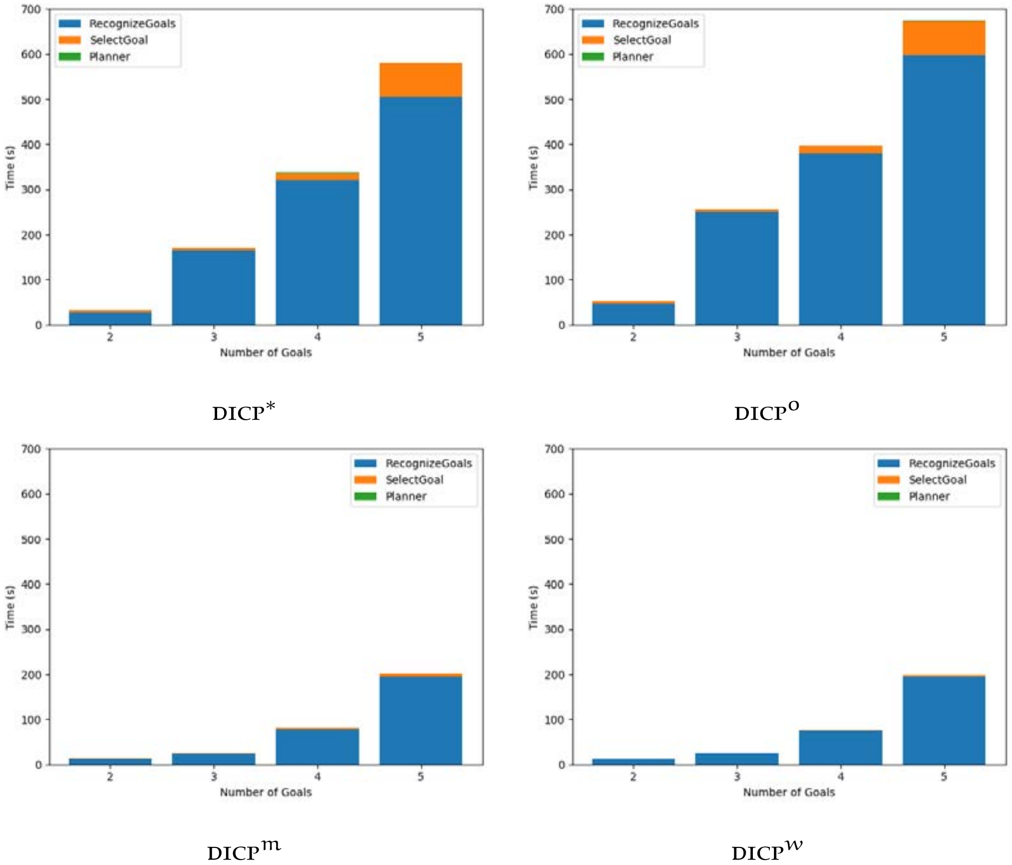
DICP$^o$

DICP$^m$

DICP$^w$

Figure 5.4: From left to right, top to bottom: aggregated solving times as the problem size grows for DICP$^*$, DICP$^o$, DICP$^m$, and DICP$^w$. In blue, the time used by RecognizeGoals. In orange, the time used by ExtractCPL, ExtractSCPL and SelectGoal. In green, the time used by Planner.

As we can see, DICP$^*$ and DICP$^o$ are much slower than DICP$^m$ and DICP$^w$. They also scale worse; while we can see an almost linear trend in the case of DICP$^m$ and DICP$^w$ when we increase the size of the problems, the solving time grows exponentially for DICP$^*$ and DICP$^o$. In fact, when we run these two versions in larger problems, they cannot solve many counterplanning tasks in POLICE CONTROL maps that exceed a size of $50 \times 50$, while the other versions can.



dicp$^*$

dicp$^o$

dicp$^m$

dicp$^w$

Figure 5.5: From left to right, top to bottom: aggregated solving times as the number of candidate goals grows for dicp$^*$, dicp$^o$, dicp$^m$, and dicp$^w$. In blue, the time used by RecognizeGoals. In orange, the time used by ExtractCPL, ExtractSCPL and SelectGoal. In green, the time used by Planner.

Another relevant fact we can extract from the results is the enormous weight the goal recognition part has compared to the other parts of the algorithm. In all the cases, this is the part that contributes the most to the total solving time; and it makes sense, given that RecognizeGoals solves two planning problems for each candidate goal. This is especially true in the case of DICP$^*$ and DICP$^o$, which use an optimal planner to solve these problems. Overall we can say that increasing the number of candidate goals affects all the algorithms more than increasing the size of the problem, which mainly influences DICP$^*$ and DICP$^o$. While computing the counterplans does not seem to take long compared to the other parts of the algorithm, the goal selection from the set of (strong) counterplanning landmarks starts taking some time as the problems increase in size. In fact, it takes around 200 seconds in the case of DICP$^*$ and DICP$^o$ solving counterplanning tasks in $20 \times 20$ maps. We can also observe this growth of SelectGoal's time in the greedier versions, but it is less steep given than they are not computing all optimal plans. Finally, we can see that the solving times for DICP$^m$ and DICP$^w$ are very similar, although as we shall see below their behavior in terms of stopping guarantees is quite different.

### DICP's *Behavior Evaluation*

Tables 5.1, 5.2, 5.3, and 5.4 summarize the results each version of DICP obtains on each domain. As we can see in all the domains, the higher percentage of observations, the higher Q values. The goal recognition task becomes easier as more actions have been observed, as vastly reported in goal recognition works. As expected, DICP$^*$ always guesses the goal right, given that we are assuming optimal SEEK behavior towards its goal. Higher Q values also imply higher %E. In other words, if we are able to correctly guess SEEK's goal, we increase the chances that the generated counterplan blocks SEEK. The experimental results at this point validate our theoretical claims: DICP$^*$ (1) always guess SEEK's goal right, $Q = 1$; and (2) if it generates a counterplan, the counterplan always blocks SEEK, $\%E = 1$.

We are randomly generating the problems, and thus some of them might not have a solution. For instance, in POLICE CONTROL there are some problems in which the police is too far from the terrorist candidate goals; and in PAINTED BLOCKS-WORDS the painting needed to paint a block can be in a room far from PREV's initial location. Despite this, our algorithms are able to block SEEK in most solvable problems. On average, we block SEEK 20% of times in POLICE CONTROL and around 30% of times in PAINTED BLOCKS-WORDS. This percentage is much lower in the case of COUNTER LOGISTICS, specially in the versions that compute strong counterplanning landmarks, which are not able to generate valid counterplans in any of the counterplanning tasks. This is because in this domain SEEK can always move one of the vehicles at cost one, i.e., in one step, making impossible for PREV to delete the landmark in fewer steps. On the other hand, DICP$^w$ is able to find some valid counterplans when less than half of SEEK's plan has been observed. The opposite is true in the case of ROVERS & MARTIANS, where PREV blocks SEEK around 70% of times, regardless the algorithm used. This is due to the competitive advantage the martian has over the rover in this

| Domain | $\lvert \mathcal{G}_{\text{SEEK}} \rvert$ | Algorithm | $m$ | %Obs | $\lvert\mathcal{L}\rvert$ | $\lvert$CPL$\rvert$ | Q | E | %E | $\%\pi_{\text{SEEK}}$ | $\lvert\pi_{\text{PREV}}\rvert$ | $t_{\mathcal{C}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| POLICE CONTROL | 3 | DICP$^*$ | asap | 0.1 | 4.6 | 1.8 | 1.0 | 0.2 | 1.0 | 0.9 | 1.2 | 171.3 |
| | | | | 0.3 | 4.2 | 1.6 | 1.0 | 0.2 | 1.0 | 0.8 | 1.3 | 185.1 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 1.0 | 0.7 | 1.3 | 225.3 |
| | | | | 0.7 | 3.2 | 1.1 | 1.0 | 0.1 | 1.0 | 0.9 | 1.0 | 254.0 |
| | | | preventing | 0.1 | 4.6 | 1.8 | 1.0 | 0.2 | 1.0 | 0.9 | 1.2 | 173.4 |
| | | | | 0.3 | 4.2 | 1.6 | 1.0 | 0.2 | 1.0 | 0.8 | 1.3 | 190.6 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 1.0 | 0.7 | 1.3 | 222.7 |
| | | | | 0.7 | 3.2 | 1.1 | 1.0 | 0.1 | 1.0 | 0.9 | 1.0 | 253.9 |
| | | DICP$^o$ | asap | 0.1 | 4.6 | 1.8 | 1.0 | 0.2 | 1.0 | 0.9 | 1.2 | 261.4 |
| | | | | 0.3 | 4.2 | 1.6 | 1.0 | 0.2 | 1.0 | 0.8 | 1.3 | 272.1 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 1.0 | 0.7 | 1.3 | 310.8 |
| | | | | 0.7 | 3.2 | 1.1 | 1.0 | 0.1 | 1.0 | 0.9 | 1.0 | 344.8 |
| | | | preventing | 0.1 | 4.6 | 1.8 | 1.0 | 0.2 | 1.0 | 0.9 | 1.2 | 256.8 |
| | | | | 0.3 | 4.2 | 1.6 | 1.0 | 0.2 | 1.0 | 0.8 | 1.3 | 263.1 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 1.0 | 0.7 | 1.3 | 307.0 |
| | | | | 0.7 | 3.2 | 1.1 | 1.0 | 0.1 | 1.0 | 0.9 | 1.0 | 336.6 |
| | | DICP$^m$ | asap | 0.1 | 5.8 | 2.4 | 0.7 | 0.1 | 0.3 | 0.8 | 2.6 | 40.0 |
| | | | | 0.3 | 3.6 | 1.3 | 1.0 | 0.1 | 1.0 | 0.8 | 1.1 | 43.7 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 1.0 | 0.9 | 1.2 | 46.0 |
| | | | | 0.7 | 3.2 | 1.1 | 0.9 | 0.1 | 1.0 | 0.9 | 1.0 | 50.2 |
| | | | preventing | 0.1 | 5.8 | 2.4 | 0.6 | 0.1 | 0.3 | 0.8 | 2.2 | 39.9 |
| | | | | 0.3 | 3.6 | 1.3 | 1.0 | 0.1 | 1.0 | 0.8 | 1.1 | 42.9 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.3 | 1.0 | 0.8 | 1.4 | 46.0 |
| | | | | 0.7 | 3.2 | 1.1 | 0.9 | 0.1 | 1.0 | 0.9 | 1.0 | 50.0 |
| | | DICP$^w$ | asap | 0.1 | 5.8 | 2.4 | 0.7 | 0.2 | 0.3 | 0.7 | 3.6 | 39.6 |
| | | | | 0.3 | 3.6 | 1.3 | 1.0 | 0.2 | 0.7 | 0.6 | 1.7 | 44.4 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 0.5 | 0.9 | 1.6 | 45.3 |
| | | | | 0.7 | 3.2 | 1.1 | 0.9 | 0.1 | 0.3 | 0.9 | 3.0 | 49.9 |
| | | | preventing | 0.1 | 5.8 | 2.4 | 0.6 | 0.2 | 0.3 | 0.8 | 3.5 | 41.4 |
| | | | | 0.3 | 3.6 | 1.3 | 1.0 | 0.2 | 0.7 | 0.7 | 1.9 | 43.3 |
| | | | | 0.5 | 3.8 | 1.4 | 1.0 | 0.2 | 0.5 | 0.9 | 1.7 | 45.9 |
| | | | | 0.7 | 3.2 | 1.1 | 0.9 | 0.1 | 0.3 | 0.9 | 3.0 | 50.8 |

Table 5.1: Comparison of the four versions of DICP in the POLICE CONTROL domain. Numbers shown are all averages over the set of problems as described in the experimental setting. The metrics measured are: size of $\mathcal{G}_{\text{SEEK}}$, algorithm used, metric to minimize $m$, percentage of observations, number of common landmarks $|\mathcal{L}|$, number of counterplanning landmarks |CPL|, goal recognition accuracy $Q$, counterplanning accuracy $E$, percentage of accuracy, percentage of SEEK plan allowed, and counterplanning total time.

| Domain | $\lvert\mathcal{G}_{\text{SEEK}}\rvert$ | Algorithm | $m$ | %Obs | $\lvert\mathcal{L}\rvert$ | $\lvert$CPL$\rvert$ | Q | E | %E | $\%\pi_{\text{SEEK}}$ | $\lvert\pi_{\text{PREV}}\rvert$ | $t_{\mathcal{C}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PAINTED BLOCKS WORDS | 5 | DICP$^*$ | asap | 0.1 | 9.1 | 2.7 | 1.0 | 0.1 | 1.0 | 0.9 | 3.0 | 101.5 |
| | | | | 0.3 | 11.1 | 3.2 | 1.0 | 0.1 | 1.0 | 0.8 | 5.0 | 102.3 |
| | | | | 0.5 | 11.4 | 3.2 | 1.0 | 0.1 | 1.0 | 0.9 | 4.0 | 112.7 |
| | | | | 0.7 | 10.6 | 2.7 | 1.0 | 0.2 | 1.0 | 0.9 | 3.5 | 132.3 |
| | | | preventing | 0.1 | 9.1 | 2.7 | 1.0 | 0.1 | 1.0 | 0.9 | 3.0 | 102.3 |
| | | | | 0.3 | 11.1 | 3.2 | 1.0 | 0.1 | 1.0 | 0.9 | 3.0 | 101.9 |
| | | | | 0.5 | 11.4 | 3.2 | 1.0 | 0.2 | 1.0 | 0.9 | 3.3 | 112.6 |
| | | | | 0.7 | 10.6 | 2.7 | 1.0 | 0.2 | 1.0 | 0.9 | 3.5 | 131.6 |
| | | DICP$^o$ | asap | 0.1 | 14.3 | 4.2 | 0.7 | 0.2 | 0.7 | 0.6 | 4.3 | 398.5 |
| | | | | 0.3 | 12.4 | 3.6 | 0.8 | 0.1 | 0.3 | 0.8 | 4.3 | 193.2 |
| | | | | 0.5 | 11.4 | 3.2 | 1.0 | 0.1 | 0.3 | 0.9 | 4.0 | 192.0 |
| | | | | 0.7 | 10.6 | 2.7 | 1.0 | 0.2 | 1.0 | 0.9 | 3.5 | 195.3 |
| | | | preventing | 0.1 | 14.3 | 4.2 | 0.7 | 0.2 | 0.7 | 0.6 | 4.3 | 399.5 |
| | | | | 0.3 | 12.4 | 3.6 | 0.8 | 0.1 | 0.3 | 0.9 | 3.7 | 192.5 |
| | | | | 0.5 | 11.4 | 3.2 | 1.0 | 0.2 | 0.7 | 0.9 | 3.3 | 194.0 |
| | | | | 0.7 | 10.6 | 2.7 | 1.0 | 0.2 | 1.0 | 0.9 | 3.5 | 194.6 |
| | | DICP$^m$ | asap | 0.1 | 18.1 | 5.1 | 0.6 | 0.3 | 0.5 | 0.7 | 3.8 | 67.2 |
| | | | | 0.3 | 17.5 | 4.8 | 0.5 | 0.1 | 0.2 | 0.9 | 4.0 | 46.8 |
| | | | | 0.5 | 15.2 | 4.1 | 0.5 | 0.4 | 0.7 | 0.8 | 3.3 | 46.6 |
| | | | | 0.7 | 11.7 | 3.0 | 1.0 | 0.2 | 1.0 | 0.9 | 3.5 | 47.4 |
| | | | preventing | 0.1 | 18.1 | 5.1 | 0.6 | 0.2 | 0.5 | 0.6 | 4.3 | 67.8 |
| | | | | 0.3 | 17.5 | 4.8 | 0.5 | 0.1 | 0.2 | 0.9 | 4.4 | 47.1 |
| | | | | 0.5 | 15.2 | 4.1 | 0.5 | 0.3 | 0.4 | 0.8 | 3.3 | 46.6 |
| | | | | 0.7 | 11.7 | 3.0 | 1.0 | 0.1 | 1.0 | 0.9 | 3.0 | 47.9 |
| | | DICP$^w$ | asap | 0.1 | 18.1 | 5.1 | 0.6 | 0.4 | 0.4 | 0.6 | 4.4 | 67.0 |
| | | | | 0.3 | 17.5 | 4.8 | 0.5 | 0.4 | 0.4 | 0.8 | 4.2 | 46.5 |
| | | | | 0.5 | 15.2 | 4.1 | 0.5 | 0.3 | 0.3 | 0.8 | 4.6 | 46.2 |
| | | | | 0.7 | 11.7 | 3.0 | 1.0 | 0.1 | 0.1 | 0.9 | 4.2 | 47.7 |
| | | | preventing | 0.1 | 18.1 | 5.1 | 0.6 | 0.6 | 0.5 | 0.6 | 4.2 | 67.0 |
| | | | | 0.3 | 17.5 | 4.8 | 0.5 | 0.2 | 0.2 | 0.8 | 4.4 | 46.4 |
| | | | | 0.5 | 15.2 | 4.1 | 0.5 | 0.2 | 0.2 | 0.7 | 4.3 | 46.1 |
| | | | | 0.7 | 11.7 | 3.0 | 1.0 | 0.2 | 0.2 | 0.9 | 4.3 | 47.6 |

Table 5.2: Comparison of the four versions of DICP in the PAINTED BLOCKS-WORDS domain. Numbers shown are all averages over the set of problems as described in the experimental setting. The metrics measured are: size of $\mathcal{G}_{\text{SEEK}}$, algorithm used, metric to minimize $m$, percentage of observations, number of common landmarks $\lvert\mathcal{L}\rvert$, number of counterplanning landmarks $\lvert$CPL$\rvert$, goal recognition accuracy $Q$, counterplanning accuracy $E$, percentage of accuracy, percentage of SEEK plan allowed, and counterplanning total time.

| Domain | $\mid \mathcal{G}_{\text{SEEK}} \mid$ | Algorithm | $m$ | %Obs | $\mid \mathcal{L} \mid$ | $\mid$CPL$\mid$ | Q | E | %E | $\%\pi_{\text{SEEK}}$ | $\mid\pi_{\text{PREV}}\mid$ | $t_{\mathcal{C}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COUNTER LOGISTICS | 10 | DICP$^*$ | asap | 0.1 | 9.0 | 0.8 | 1.0 | 0.0 | - | - | - | 44.7 |
| | | | | 0.3 | 11.6 | 1.3 | 1.0 | 0.0 | - | - | - | 101.2 |
| | | | | 0.5 | 9.4 | 1.7 | 1.0 | 0.0 | - | - | - | 147.9 |
| | | | | 0.7 | 6.8 | 1.2 | 1.0 | 0.0 | - | - | - | 96.5 |
| | | | preventing | 0.1 | 9.0 | 0.8 | 1.0 | 0.0 | - | - | - | 44.8 |
| | | | | 0.3 | 11.6 | 1.3 | 1.0 | 0.0 | - | - | - | 101.5 |
| | | | | 0.5 | 9.4 | 1.7 | 1.0 | 0.0 | - | - | - | 147.9 |
| | | | | 0.7 | 6.8 | 1.2 | 1.0 | 0.0 | - | - | - | 98.2 |
| | | DICP$^o$ | asap | 0.1 | 9.0 | 0.8 | 1.0 | 0.0 | - | - | - | 45.8 |
| | | | | 0.3 | 11.6 | 1.3 | 1.0 | 0.0 | - | - | - | 108.2 |
| | | | | 0.5 | 9.4 | 1.7 | 1.0 | 0.0 | - | - | - | 152.0 |
| | | | | 0.7 | 6.8 | 1.2 | 1.0 | 0.0 | - | - | - | 100.0 |
| | | | preventing | 0.1 | 9.0 | 0.8 | 1.0 | 0.0 | - | - | - | 45.8 |
| | | | | 0.3 | 11.6 | 1.3 | 1.0 | 0.0 | - | - | - | 107.5 |
| | | | | 0.5 | 9.4 | 1.7 | 1.0 | 0.0 | - | - | - | 152.7 |
| | | | | 0.7 | 6.8 | 1.2 | 1.0 | 0.0 | - | - | - | 99.4 |
| | | DICP$^m$ | asap | 0.1 | 9.5 | 0.8 | 0.7 | 0.1 | 1.0 | 0.5 | 6.0 | 1.7 |
| | | | | 0.3 | 10.9 | 1.3 | 0.8 | 0.1 | 0.3 | 0.7 | 6.7 | 2.0 |
| | | | | 0.5 | 10.1 | 1.7 | 0.8 | 0.0 | 0.0 | - | 5.0 | 2.1 |
| | | | | 0.7 | 7.1 | 1.2 | 0.9 | 0.0 | - | - | - | 2.4 |
| | | | preventing | 0.1 | 9.5 | 0.8 | 0.7 | 0.1 | 1.0 | 0.5 | 6.0 | 1.8 |
| | | | | 0.3 | 10.9 | 1.3 | 0.8 | 0.1 | 0.3 | 0.7 | 6.7 | 2.0 |
| | | | | 0.5 | 10.1 | 1.7 | 0.8 | 0.0 | 0.0 | - | 5.0 | 2.2 |
| | | | | 0.7 | 7.1 | 1.2 | 0.9 | 0.0 | - | - | - | 2.5 |
| | | DICP$^w$ | asap | 0.1 | 9.5 | 0.8 | 0.7 | 0.2 | 0.5 | 0.4 | 6.5 | 1.7 |
| | | | | 0.3 | 10.9 | 1.3 | 0.8 | 0.4 | 0.4 | 0.7 | 6.6 | 1.9 |
| | | | | 0.5 | 10.1 | 1.7 | 0.8 | 0.0 | 0.0 | - | 6.1 | 2.1 |
| | | | | 0.7 | 7.1 | 1.2 | 0.9 | 0.0 | 0.0 | - | 6.1 | 2.4 |
| | | | preventing | 0.1 | 9.5 | 0.8 | 0.7 | 0.2 | 0.5 | 0.4 | 6.5 | 1.7 |
| | | | | 0.3 | 10.9 | 1.3 | 0.8 | 0.4 | 0.4 | 0.7 | 6.6 | 1.9 |
| | | | | 0.5 | 10.1 | 1.7 | 0.8 | 0.0 | 0.0 | - | 6.1 | 2.1 |
| | | | | 0.7 | 7.1 | 1.2 | 0.9 | 0.0 | 0.0 | - | 6.1 | 2.4 |

Table 5.3: Comparison of the four versions of DICP in the COUNTER LOGISTICS domain. Numbers shown are all averages over the set of problems as described in the experimental setting. The metrics measured are: size of $\mathcal{G}_{\text{SEEK}}$, algorithm used, metric to minimize $m$, percentage of observations, number of common landmarks $|\mathcal{L}|$, number of counterplanning landmarks |CPL|, goal recognition accuracy $Q$, counterplanning accuracy $E$, percentage of accuracy, percentage of SEEK plan allowed, and counterplanning total time.

| Domain | $\mid \mathcal{G}_{\text{SEEK}} \mid$ | Algorithm | $m$ | %Obs | $\mid \mathcal{L} \mid$ | $\mid$CPL$\mid$ | Q | E | %E | $\%\pi_{\text{SEEK}}$ | $\lvert\pi_{\text{PREV}}\rvert$ | $t_{\mathcal{C}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROVERS & MARTIANS | 6 | DICP$^*$ | asap | 0.1 | 4.1 | 2.2 | 1.0 | 0.6 | 1.0 | 0.4 | 1.3 | 98.5 |
| | | | | 0.3 | 3.7 | 2.1 | 1.0 | 0.7 | 1.0 | 0.6 | 1.2 | 101.3 |
| | | | | 0.5 | 2.8 | 1.8 | 1.0 | 0.7 | 1.0 | 0.7 | 1.0 | 122.4 |
| | | | | 0.7 | 2.2 | 1.2 | 1.0 | 0.5 | 1.0 | 0.9 | 1.0 | 141.0 |
| | | | preventing | 0.1 | 4.1 | 2.2 | 1.0 | 0.6 | 1.0 | 0.5 | 1.2 | 98.6 |
| | | | | 0.3 | 3.7 | 2.1 | 1.0 | 0.7 | 1.0 | 0.6 | 1.2 | 105.4 |
| | | | | 0.5 | 2.8 | 1.8 | 1.0 | 0.7 | 1.0 | 0.7 | 1.0 | 121.6 |
| | | | | 0.7 | 2.2 | 1.2 | 1.0 | 0.5 | 1.0 | 0.9 | 1.0 | 141.6 |
| | | DICP$^o$ | asap | 0.1 | 4.1 | 2.2 | 1.0 | 0.6 | 1.0 | 0.4 | 1.3 | 105.5 |
| | | | | 0.3 | 3.7 | 2.1 | 1.0 | 0.7 | 1.0 | 0.6 | 1.2 | 124.2 |
| | | | | 0.5 | 2.8 | 1.8 | 1.0 | 0.7 | 1.0 | 0.7 | 1.0 | 145.4 |
| | | | | 0.7 | 2.2 | 1.2 | 1.0 | 0.5 | 1.0 | 0.9 | 1.0 | 177.3 |
| | | | preventing | 0.1 | 4.1 | 2.2 | 1.0 | 0.6 | 1.0 | 0.5 | 1.2 | 102.8 |
| | | | | 0.3 | 3.7 | 2.1 | 1.0 | 0.7 | 1.0 | 0.6 | 1.2 | 125.4 |
| | | | | 0.5 | 2.8 | 1.8 | 1.0 | 0.7 | 1.0 | 0.7 | 1.0 | 141.3 |
| | | | | 0.7 | 2.2 | 1.2 | 1.0 | 0.5 | 1.0 | 0.9 | 1.0 | 181.6 |
| | | DICP$^m$ | asap | 0.1 | 4.2 | 2.1 | 0.5 | 0.7 | 1.0 | 0.4 | 1.3 | 47.2 |
| | | | | 0.3 | 3.7 | 2.0 | 0.7 | 0.8 | 1.0 | 0.6 | 1.2 | 57.3 |
| | | | | 0.5 | 3.2 | 1.9 | 0.7 | 0.8 | 1.0 | 0.6 | 1.0 | 68.9 |
| | | | | 0.7 | 2.1 | 1.2 | 0.9 | 0.5 | 1.0 | 0.9 | 1.0 | 80.0 |
| | | | preventing | 0.1 | 4.2 | 2.1 | 0.5 | 0.7 | 1.0 | 0.4 | 1.3 | 47.2 |
| | | | | 0.3 | 3.7 | 2.0 | 0.7 | 0.8 | 1.0 | 0.6 | 1.2 | 56.6 |
| | | | | 0.5 | 3.2 | 1.9 | 0.7 | 0.8 | 1.0 | 0.7 | 1.0 | 69.0 |
| | | | | 0.7 | 2.1 | 1.2 | 0.9 | 0.5 | 1.0 | 0.9 | 1.0 | 80.1 |
| | | DICP$^w$ | asap | 0.1 | 4.2 | 2.1 | 0.5 | 0.7 | 0.9 | 0.4 | 1.3 | 47.0 |
| | | | | 0.3 | 3.7 | 2.0 | 0.7 | 0.8 | 0.9 | 0.6 | 1.2 | 57.5 |
| | | | | 0.5 | 3.2 | 1.9 | 0.7 | 0.8 | 0.8 | 0.6 | 1.0 | 69.1 |
| | | | | 0.7 | 2.1 | 1.2 | 0.9 | 0.6 | 0.7 | 0.9 | 1.0 | 81.7 |
| | | | preventing | 0.1 | 4.2 | 2.1 | 0.5 | 0.8 | 0.9 | 0.4 | 1.2 | 42.0 |
| | | | | 0.3 | 3.7 | 2.0 | 0.7 | 0.8 | 0.9 | 0.6 | 1.2 | 55.4 |
| | | | | 0.5 | 3.2 | 1.9 | 0.7 | 0.8 | 0.9 | 0.6 | 1.0 | 68.9 |
| | | | | 0.7 | 2.1 | 1.2 | 0.9 | 0.7 | 0.8 | 0.9 | 1.0 | 80.3 |

Table 5.4: Comparison of the four versions of DICP in the ROVERS & MARTIANS domain. Numbers shown are all averages over the set of problems as described in the experimental setting. The metrics measured are: size of $\mathcal{G}_{\text{SEEK}}$, algorithm used, metric to minimize $m$, percentage of observations, number of common landmarks $|\mathcal{L}|$, number of counterplanning landmarks $|$CPL$|$, goal recognition accuracy $Q$, counterplanning accuracy $E$, percentage of accuracy, percentage of SEEK plan allowed, and counterplanning total time.

domain; while SEEK needs to move through several location to collect samples, PREV can move to any point much faster, thus being able to block SEEK's plan in most counterplanning tasks by only executing one action.

The capacity of blocking an opponent is also closely related to the number of counterplanning landmarks (|CPL|), which depends on the number of (common) landmarks $|\mathcal{L}|$ and the capacity of PREV's model to negate them. A higher number of counterplanning landmarks leads to more valid counterplans (higher $E$ values). This is the case of PAINTED BLOCKS-WORDS, which has more counterplanning landmarks on average. Moreover, we tend to observe higher $E$ values when we have observed less than half of SEEK's plan. When we try to generate a counterplan with 70% of SEEK's plan already executed, it is less likely that a valid counterplan can be generated; the number of remaining counterplanning landmarks is lower in those cases, i.e., SEEK is too close to its goal for us to stop it.

In all the configurations and domains, counterplans that tries to block the enemy as soon as possible $m =$ asap tend to be larger (higher $|\pi_{\text{PREV}}|$), allowing less of SEEK's plan to be executed (lower $\%\pi_{\text{SEEK}}$) compared to the plans returned when $m =$ preventing which try to block SEEK in the least number of steps.

Finally, we can notice some differences if we compare algorithms' behavior. As we have said, DICP$^*$ never fails: if it returns a counterplan, it is guaranteed to block SEEK. DICP$^o$ is close to DICP$^*$ in that aspect (%$E$), specially in POLICE CONTROL. However, it requires more time to solve the goal recognition task than DICP$^*$. In PAINTED BLOCKS-WORDS, it is able to generate valid counterplans in a couple of tasks where DICP$^*$ cannot. This is because DICP$^o$ fails guessing the right goal, but finds a landmark that can be deleted before SEEK stops needing it if optimal plans to the actual SEEK goal are not taken into account. In other words, the counterplan generated in those cases is only valid for that particular SEEK's plan, but not for any plan that it could generate. Overall, DICP$^w$ is the one that blocks SEEK more often. This difference is specially remarkable in PAINTED BLOCKS-WORDS, where it can block SEEK up to 60% of times. However, most of the counterplans that DICP$^w$ generate do not prevent SEEK from achieving its goal. Its average %$E$ is close to 0.2, meaning that only 20% of the PREV's plans it returns are valid counterplans. DICP$^m$ shows a good trade-off among all the versions: (1) it is as fast as DICP$^w$ and is able to stop SEEK a similar number of times ($E$); and (2) the counterplans it generates are closer in terms of %$E$ to the ones of the much slower versions such as DICP$^*$ or DICP$^o$.

## 5.5 EVALUATION IN REAL-TIME STRATEGY GAMES

We also wanted to test our counterplanning approach in more realistic scenarios such as Real-Time Strategy (RTS) games. Games have always been interesting and great test beds for developing and trying new AI capabilities because of their well-defined set of rules, clear aims, and the possibility of evaluating results in an objective way. RTS games are a particularly difficult type of games with many analogies to real world problems. They differ from traditional board games in several aspects: they could be partially observable; their state space is usually enormous; all players make their moves simultaneously; and they have a small amount of time to decide the next action. These features present many challenging subproblems for AI research, such as decision making under uncertainty, collaboration, opponent modeling, or adversarial real-time planning (Buro, 2003).

In this section, we slightly modify our counterplanning approach to generate counterplans quicker so the technique can be used in RTS games.

### 5.5.1 *Cost Estimation Gradient Goal Recognition*

Up to now, we have employed Ramírez and Geffner (2009; 2010) goal recognition approaches to identify opponent's goals. However, they are often slow and therefore cannot be used to identify goals in RTS games, where the actions are performed almost instantly. This is the reason why we propose a lighter planning-based goal recognition approach based on cost estimation gradient. It is described in Algorithm 4. The algorithm takes as inputs the current seeking agent's problem and domain $(F, A, I)$, a set of hypothesis $\mathcal{G}$, and the last observation $(o)$. It estimates the cost of achieving each of the candidate goals, both from the previous input state $I$ and the current state $I'$ given the last observed action $I'$. This cost estimation can be performed using any heuristic or actually solving the problem and obtaining a plan. We compute this cost in parallel for each $g \in \mathcal{G}_{\text{SEEK}}$ to better scale-up. The algorithm returns a list containing the cost estimation difference $\Delta_c$ between the new and the previous state for all the goals in $\mathcal{G}$.

**Algorithm 4** RecognizeGoalsFast

```
Require: $F, A, I, \mathcal{G}, o$
Ensure: $\mathcal{G}'$
$\mathcal{G}' \leftarrow \emptyset$
$I' \leftarrow$ Update$(F, A, I, o)$
for $G_i \in \mathcal{G}$ do
    $c_{G_i} \leftarrow$ EstimateCost$(F, A, I, G_i)$
    $c\prime_{G_i} \leftarrow$ EstimateCost$(F, A, I', G_i)$
    if $c\prime_{G_i} < c_{G_i}$ then
        $\mathcal{G}' \leftarrow \mathcal{G}' \cup \{G_i\}$
return $\mathcal{G}'$
```

**Example 5.1.** *An example of this process is shown in Figure 5.6. In this case, an agent (represented as a triangle) might want to achieve G1 or G2. From its initial state, the* EstimateCost() *function returns 6 for G1 and G2 respectively. After observing the first action of the agent, the function returns 5 and 7 respectively. Since the new value has decreased for G1, that will be the goal that conform $\mathcal{G}'$, which will be returned by the algorithm.*

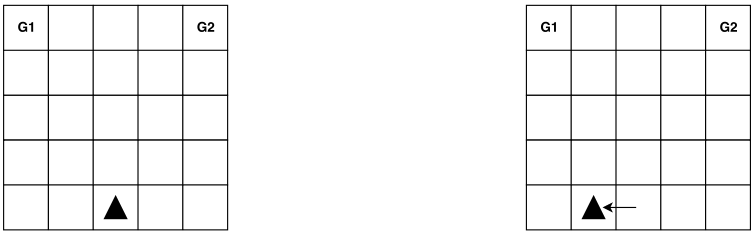


Figure 5.6: Goal recognition example. The agent is depicted by a triangle. Its goal can be either achieve G1 or G2.

### 5.5.2 *Real-Time Counterplanning*

Algorithm 5 shows the high-level algorithm used to solve a counterplanning task from the perspective of prev in a real-time setting. While in dicp we received as input a sequence of observations, in rtdicp (Algorithm 5) we receive the observations one at a time. In this case, the algorithm asks for the last observed action, sending it to RecognizeGoalsFast along with a planning domain, initial conditions, and a set of candidate goals $\mathcal{G}_{\text{seek}}$. The call to this function returns the updated set of candidate goals $G_{\text{seek}}$. Then, the composite state is updated by applying $o_{\text{seek}}$. Next, it extracts the set of counterplanning landmarks CPL in line 6. As in dicp, if there are no common counterplanning landmarks, the counterplanning task cannot be solved. Otherwise, the algorithm selects $G_{\text{prev}}$ from the set of counterplanning landmarks. At this point we simplify the computations we made in dicp for the sake of algorithm's speed. We iterate over each counterplanning landmark in CPL, checking if prev can delete it before seek can achieve it (lines 8-12). If that is the case, the computed counterplan is returned. As we previously discussed, this formulation will not allow prev to find counterplans in which the given landmark is already true in $I_{\text{seek}}$.

rtdicp cannot guarantee that the enemy will be blocked, since we are not computing strong counterplanning landmarks or strong plans. However, the generated counterplans will prevent seek from achieving its goals in many cases as we will see in the experimental evaluation.

### 5.5.3 *Modelling StarCraft as Planning*

*StarCraft* is a military science fiction real-time strategy video game developed and published by Blizzard Entertainment. The objective of a standard *StarCraft* match is to defeat your opponents by eliminating their units and buildings. Although there are different races featured in the game, for simplicity we will only take into account the *Terran* race. The game-play of *StarCraft* is based on resource management and base construction. These elements are the key to create an army that can overpower the enemy and, therefore, succeed in the game. A player relies on two kind of basic resources: minerals and vespene gas. The first one is necessary to build new structures and produce new units. The second one is needed for more advanced units and structures. Both of the resources can be collected by a worker unit from the nodes placed in the map and be

**Algorithm 5** RTDICP

```
Require: $\mathcal{C}$
Ensure: $\pi_{\text{PREV}}$
$\pi_{\text{PREV}} \leftarrow \emptyset$
$CPL \leftarrow F_{\text{SEEK}}$
while $\pi_{\text{PREV}} = \emptyset$ or $\text{GetObservation}() \neq \emptyset$ do
    $o_{\text{PREV}} \leftarrow \text{GetObservation}()$
    $\mathcal{G}_{\text{SEEK}} \leftarrow \text{RecognizeGoals}(F_{\text{SEEK}}, A_{\text{SEEK}}, I_{\text{SEEK}}, \mathcal{G}_{\text{SEEK}}, o_{\text{PREV}})$
    $I_c = \Gamma(I_{\text{SEEK}} \cup I_{\text{PREV}}, o_{\text{PREV}})$
    $CPL \leftarrow \text{ExtractCPL}(\mathcal{C})$
    while $CPL \neq \emptyset$ do
        for $L_i \in CPL$ do
            $\pi_{\text{SEEK}} \leftarrow \text{Planner}(\Pi_{\text{SEEK}} = \langle F_{\text{SEEK}}, A_{\text{SEEK}}, I_{\text{SEEK}}, L_i \rangle)$
            $\pi'_{\text{PREV}} \leftarrow \text{Planner}(\Pi_{\text{PREV}} = \langle F_{\text{PREV}}, A_{\text{PREV}}, I_{\text{PREV}}, \neg L_i \rangle)$
            if $c(\pi_{\text{SEEK}}) >= c(\pi_{\text{PREV}})$ then
                $\pi_{\text{PREV}} \leftarrow \pi'_{\text{PREV}}$
            else
                $CPL.\text{remove}(L_i)$
return $\pi_{\text{PREV}}$
```

brought to the main building. The resource is stored and becomes available for its use. However, the vespene gas can only be collected from the source (a vespene gas geyser) by building an extraction structure on top of it (for the Terran race, this structure is called a refinery). Different buildings or units require different amounts of minerals or vespene gas to be produced. When the sufficient amount has being collected, the player can initiate the creation process, which takes a specific time to be completed depending on the kind of structure or unit. Once the units with attack capability have been created, the player can attack the enemy. In the experiments, we consider that all the actions are executed with cost (time) equal to one.

In a standard *StarCraft* match, most of the map is hidden to the player by the "fog of war". This feature does not let the player know the position of the enemy until that area has been explored. The areas where there is a player's building or unit are revealed only to this player. For purposes of this work, the fog of war has been disabled during the experiments, since the agent needs total visibility of the actions of the enemy to produce a counterplan.

We have reduced the standard match to a set of mini-games in order to simplify the complexity of a whole *StarCraft* match. In these mini-games, both players will be represented only by one unit and the objective of each mini-game will be different. In order to use the counterplanning technique, we need to translate the features and actions of *StarCraft* to a planning domain and the state of the game to a planning problem.

For the planning domain[4], we need to replicate the actual game dynamics using a set of objects, predicates, and actions. We use the objects *tile* and *building*.

[4] The full planning domain is shown in Appendix D

The object *tile* is used to identify the different positions in the actual map. Each tile corresponds to a *TilePosition*, as it is called in *StarCraft*. The object *building* is used to identify the different structures that can be constructed during the game. For that purpose, we define a different type of object *building* for every type of structure in the game. The defined predicates are shown below:

- `connected`: this predicate represents the connection between tiles in the map.
- `at`: this predicate indicates the position of a unit. We have two predicates of this kind to identify both units in the mini-game: `at` and `at-enemy`.
- `empty`: indicates if a tile is empty.
- `at-vespene`: indicates the position of a vespene gas geyser.
- `at-mineral`: indicates the position of a mineral field.
- `at-building`: indicates the position of a structure. There is a predicate of this kind for each type of structure in the game and for both players. As an example, we define: `at-barracks`, `at-barracks-enemy`.
- `carrying-mineral`: indicates if a unit carries a chunk of mineral. We have two predicates of this kind to indentify both units in the mini-game: `carrying-mineral` and `carrying-mineral-enemy`.
- `carrying-vespene`: indicates if a unit carries vespene gas. We have two predicates of this kind to indentify both units in the mini-game: `carrying-vespene` and `carrying-vespene-enemy`.
- `have-mineral`: indicates that a unit has stored mineral. We have two predicates of this kind to indentify both units in the mini-game: `have-mineral` and `have-mineral-enemy`.
- `have-vespene`: indicates that a unit has stored vespene gas. We have two predicates of this kind to indentify both units in the mini-game: `have-vespene` and `have-vespene-enemy`.
- `alive`: indicates that a unit is alive. We have two predicates of this kind to identify both units in the mini-game: `alive` and `alive-enemy`.
- `building-built`: indicates that a certain kind of structure has been built. There is a predicate of this kind for each type of structure in the game and for both players. As an example we define: `refinery-built`, `refinery-built-enemy`.
- `unit-trained`: indicates that a certain kind of unit has been created. There is a predicate of this kind for each type of unit in the game: `marine-trained`, `firebat-trained`, `...`

Along with these predicates, a set of planning actions have been defined to represent all the possible actions that a unit can perform during the game. These actions are:

- `move`: a unit moves between two tiles that are connected. A PDDL representation of this action is shown in Listing 5.1.
- `attack-unit`: a unit attacks an enemy's unit. We indicate a tile, and our unit attacks from that tile in its range of attack.
- `attack-structure`: a unit attacks a certain structure. There is an action of this kind for each type of structure in the game due to its different sizes.
- `gather-resource`: a unit gathers a resource from a source. It is represented through two actions: `gather-mineral` and `gather-vespene`. This is due to the differences between the two actions. Minerals can be extracted directly from the source, but in order to gather vespene gas, a refinery must be built on top of the vespene geyser beforehand.
- `store-resource`: a unit, which is currently carrying a resource, stores it at the command center.
- `take`: a unit takes a chunk of resource present in the map. These types of objects have been placed on the map for some of the mini-games.
- `build-structure`: a unit builds a new structure in a place with enough space in the map. There is an action of this kind for each type of structure in the game. E.g. `build-refinery`, `build-academy`, . . .
- `train-unit`: creating a new unit. There is an action of this kind for each type of unit in the game. E.g. `train-marine`, `train-firebat`, . . .

As in our previous experiments, we assume deterministic action outcomes. That is, the only changes in the environment state are the ones described by the effects of the actions in the planning domain model. For example, in the `attack-unit` action we assume that the other unit dies after the attack.

```
(:action move
 :parameters (?x1 - tile ?x2 - tile)
 :precondition (and
    (at ?x1)
    (connected ?x1 ?x2)
    (empty ?x2))
 :effect (and
    (not (at ?x1))
    (at ?x2)
    (not (empty ?x2))
    (empty ?x1)))
```

Listing 5.1: Example of the move action in the PDDL format. It moves the friendly unit from one tile to another.

We need to know the state of the game and translate it to a PDDL file to define the planning problem. We get this information at each time step using the Brood War Application Programming Interface (BWAPI)[5]. BWAPI lets us read all the relevant information about the game state, like the organization of the map or the positions of our own units, the opponent units, and other units or structures that appear on the map. This information is saved into a PDDL

[5] https://bwapi.github.io/

file defining the problem and into two other files that contain the observations about the opponent's units and the possible goals that the opponent could be trying to achieve in the future.

In order to create the problem, we start detecting all of the *TilePositions* that form our map for the mini-game. We use tiles that units cannot step in like water or high ground to model the shape of our custom maps for the mini-games. Every tile is named using its coordinates in the game. Then, we write the state of our players (`alive` by default), and the positions of each player unit and every building. A unit only occupies one tile, but a building needs more space, so it is represented by several `at-building` predicates. Then, we proceed to specify the state of every tile, indicating as empty the tiles that are not occupied by unit or a building, and declaring the connections among them. The predicate `connected` is uni-directional, so we need two predicates `connected` for each pair of tiles to represent that they are fully connected in both directions. We only consider horizontal and vertical connections to simplify the problem.

Finally, we identify the different goals that the enemy could be targeting, such as taking the chunks of mineral, attacking one of our buildings, or creating a certain type of structure or unit.

### 5.5.4 *Results*

We have created five different *StarCraft* mini-games to test our approach. Each of the games is slightly more difficult than the previous one and introduces some variants, such as resources gathering, building construction, or troops confrontations. In all the mini-games there are two players, SEEK and PREV, which handle Terran units. The seeking agent has a pre-computed plan $\pi_{\text{SEEK}}$ that it will follow to achieve its actual goal $G_{\text{SEEK}}$. This goal is hidden for the preventing agent. The mini-games finish when the seeking player has achieved its initial goal, or if the preventing player stops it. At each mini-game we measure:

- $|\mathcal{G}_{\text{SEEK}}|$: number of goals in the candidate goal's set.
- $|\mathcal{L}_{\Pi_{\text{SEEK}}}|$: number of common landmarks for the seeking agent planning task.
- $|\text{CPL}|$: number of counterplanning landmarks among the most likely goals.
- $\%\mathsf{Obs}$: percentage of observed actions from the total SEEK's plan needed to perform the goal inference.
- $Q$: $1$ if the actual goal $G_{\text{SEEK}}$ was found to be among the most likely goals $\mathcal{G}_{\text{SEEK}}$ after calling RECOGNIZEGOALSFAST, $0$ otherwise.
- $E$: $1$ if SEEK is stopped, $0$ otherwise. To compute this number, we jointly execute $\pi_{\text{SEEK}}$ and $\pi_{\text{PREV}}$. If the joint execution of both plans does not allow SEEK to achieve its goal, $E = 1$.
- $\%\pi_{\text{SEEK}}$: ratio of $\pi_{\text{SEEK}}$ that SEEK can execute until it is blocked by PREV. Lower numbers are better, and $1$ means that SEEK has not been blocked.

- $t_Q$: time in seconds taken for solving the goal recognition problem. It is accumulated each iteration that a counterplan cannot be generated.
- $t_{\mathcal{L}}$: time in seconds taken for computing the landmarks.
- $t_{\mathcal{C}}$: total time in seconds taken for producing a counterplan.
- $|\pi_{\text{SEEK}}|$: number of actions in the seeking agent's plan.
- $|\pi_{\text{PREV}}|$: number of actions in the generated counterplan.

Since *StarCraft* runs in Windows and the planners run in Ubuntu, we communicate both parts of the architecture via a TCP/IP connection. The StarCraft part of the architecture runs on a Windows 10 machine with Intel Core i5-4210U running at 1.7 GHz. The counterplanning algorithm runs on an Ubuntu machine with Intel Core 2 Quad Q8400 running at 2.66 GHz. We use the LMCUT heuristic (Helmert & Domshlak, 2009) for cost estimation and the PROBE planner (Lipovetzky et al., 2014) for plans computation. We selected them for their speed, crucial for our real-time setting. The next subsections describe the mini-games and the results obtained.

*Take the Gem*

The first mini-game takes places in a 64x20 map. Each player only controls one unit. The seeking agent is located at the middle bottom of the map, while the preventing agent is located at the middle top side. There are two possible goals ($|\mathcal{G}| = 2$) that SEEK may want to achieve: the top-left gem $g_1$ or the top-right gem $g_2$. This initial situation is depicted in Figure 5.7.

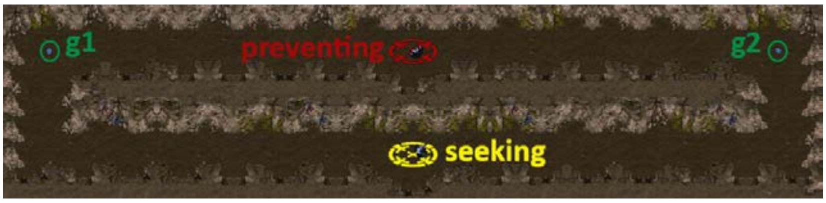


Figure 5.7: Take the gem mini-game. g1 and g2 indicate the two SEEK's possible goals. The actual goal of SEEK is to take the gem at g1.

The actual SEEK's goal is to achieve g1. After its first movement, PREV is able to correctly guess it and start to compute the involved landmarks. There are three landmarks: `at-enemy s31-18`, the initial location of SEEK, which is always a landmark; `at-enemy s3-3`, the position where the enemy needs to be in order to achieve $g_1$; and `at-target s3-3`, the position of the gem. From this set of landmarks, the only counterplanning landmark that PREV can falsify is `at-target s3-3`, given that there is an action in its domain that deletes the `at-target` predicate. After the costs calculation to ensure that it can reach that landmark before the opponent does it, `not (at-target s3-3)` becomes PREV's goal, generating a counterplan $\pi_{\text{PREV}}$ to achieve it. Figure 5.8 shows the moment when PREV takes the gem before its opponent. A video showing the experiment can be accessed through the following link https://youtu.be/mxbNir6sESk.

Figure 5.8: Moment when PREV takes the gem before its opponent.

*Gathering or Attacking?*

The second mini-game takes place in a 64x64 map with Y shape as depicted in Figure 5.9. There are two mineral fields, one at the top-right and another at the top-left side of the map; the latter is the actual goal of SEEK. There are two command centers: one at the top (PREV) and another one at the bottom (SEEK). PREV thinks that the opponent may want to gather resources from any of the mineral fields or attack its command center. Our counterplanning algorithm is able to find a common landmark that blocks the achievement of any of the candidate goals. So a counterplan is generated to send a unit to the bottleneck spot placed in the middle of the Y. A video showing the experiment can be accessed through the following link https://youtu.be/T8Y8cbNqvFk.

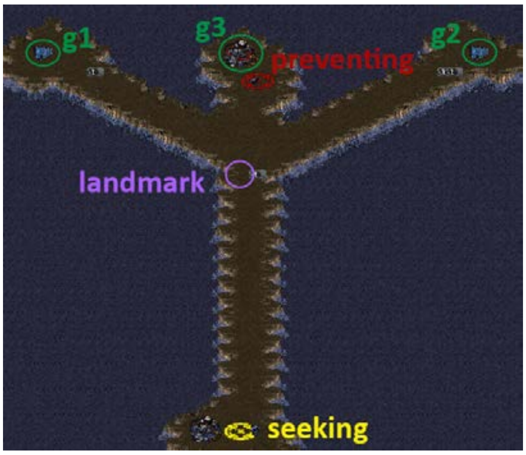


Figure 5.9: Resource gathering or attacking mini-game. The actual goal of SEEK is to gather the resources at g1. The landmark point indicates where PREV's unit will stand to attack SEEK's unit and block its goal achievement process.

*Gathering in the Maze*

The third mini-game takes place in a 64x64 maze-like map as depicted in Figure 5.10. In this case, there are three mineral fields placed in the map, one at the top-left, one on the right and one almost at the center of the map. As before, SEEK is placed at the bottom and PREV at the top of the map. A set of bottleneck spots have been also placed around the map to allow blocking the path of the enemy. PREV thinks that the goals that SEEK are pursuing are gathering mineral from one of the mineral fields. In this experiment, PREV is not able to find a common landmark to block the enemy, since the shape of the maze allows SEEK two different paths to any point of the map. If the goal is the mineral field at the center of the map, PREV cannot infer this until the seeking agent has completed almost 80% of its whole plan. That is when SEEK has reached the spot at the middle part on the left of the map and begins to go right. At that time, PREV cannot block it. A video showing the experiment can be accessed through the following link https://youtu.be/5pUmkSZcpXY.

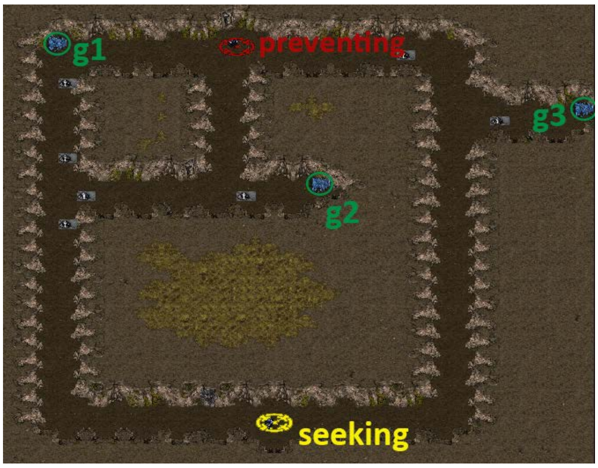


Figure 5.10: Resource gathering in a maze mini-game. The actual goal of SEEK is to gather the minerals at g2.

*Base Construction*

The fourth mini-game takes place in a 64x26 map as depicted in Figure 5.11. There is one mineral field placed on the right and one vespene geyser placed at the top-left of the map. The positions of the two agents are the same as in the previous experiments. There are also a command center and a barrack for the seeking agent, which lets it build the structures that we are considering as possible goals for this experiment: building a supply depot or a factory. When SEEK starts to go to the left, PREV considers that it is going to the vespene geyser placed on top and tries to counterplan by blocking its opponent way to the bottleneck point, which is a landmark. PREV believes that SEEK's final goal is to build a factory. In order to achieve that, SEEK needs to build a refinery on top of the vespene geyser, extract the vespene, store it in the command center and, finally, build the factory. PREV gets to the landmark point when the seeking agent is about to reach the same point, blocking its way. SEEK has only completed 1.9% of its plan.

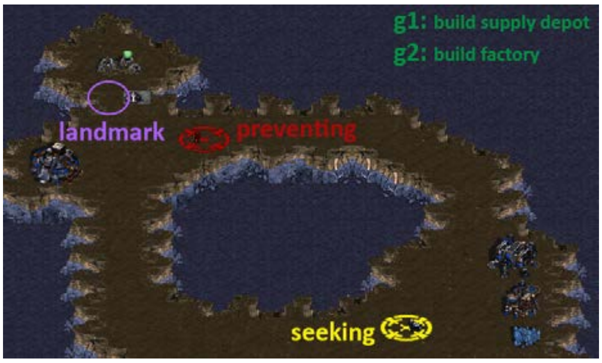


Figure 5.11: Base construction mini-game. The actual goal is to build a factory. The landmark point indicates where PREV is going to stand to block the way and attack.

### *Training Army*

The fifth mini-game takes place in the exact same map that was used for the fourth experiment as depicted in Figure 5.12. For this experiment we have also placed an academy for SEEK, which allows it to train different kinds of units. This time, the possible goals that SEEK might be pursuing are training a marine or training a firebat. For the first goal, it needs to gather mineral, store it in the command center, and then train the marine using the barracks. For the second goal, SEEK needs also to gather and store vespene gas. Independently of SEEK's actions, PREV will figure out that the common landmark that can invalidate both goals is to destroy the barracks where the units are trained. PREV attacks the barracks when SEEK has only completed 2.6% of its plan. A video showing the experiment can be accessed through the following link https://youtu.be/MwvqnZocJpk.

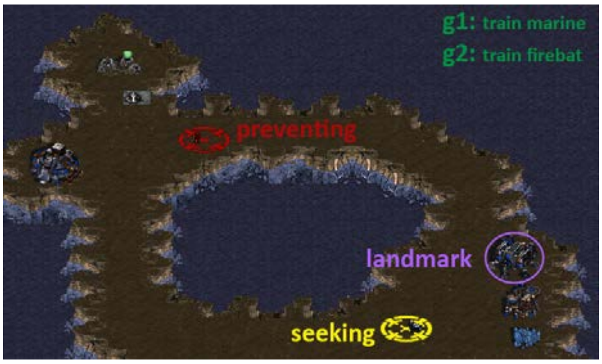


Figure 5.12: Training army mini-game. PREV detects a common landmark for both goals: destroying the barracks. The actual SEEK's goal is to train a firebat.

### *Mini-games Summary*

Overall we have shown how we can use RTDICP to generate large counterplans in a RTS game like Starcraft. Table 5.5 summarizes the quantitative results of each experiment. As in the case of the competitive planning domains of the

| Mini-game | $\lvert\mathcal{G}_{\text{SEEK}}\rvert$ | $\lvert\mathcal{L}_{\Pi_{\text{SEEK}}}\rvert$ | $\lvert$CPL$\rvert$ | %Obs | Q | E | $\%\pi_{\text{SEEK}}$ | $t_{\mathcal{Q}}$ | $t_{\mathcal{L}}$ | $t_{\mathcal{C}}$ | $\lvert\pi_{\text{SEEK}}\rvert$ | $\lvert\pi_{\text{PREV}}\rvert$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Take the gem | 2 | 3 | 1 | 2.2 | 1 | 1 | 0.8 | 1.8 | 1.9 | 4.8 | 44 | 30 |
| Gathering or attacking? | 3 | 6 | 3 | 1.4 | 1 | 1 | 0.2 | 1.1 | 1.7 | 3.4 | 70 | 12 |
| Gathering in the maze | 3 | 3 | 1 | 79.5 | 1 | 0 | $\infty$ | 19.2 | 1.5 | - | 78 | - |
| Base construction | 2 | 11 | 3 | 1.9 | 1 | 1 | 0.6 | 12.7 | 2.7 | 24.4 | 51 | 9 |
| Training army | 2 | 6 | 2 | 2.6 | 1 | 1 | 0.5 | 3.1 | 2.2 | 4.3 | 75 | 30 |

Table 5.5: Table that summarizes the results obtained in the five mini-games.

previous section, having more counterplanning landmarks often implies more opportunities to block the opponent. In all the mini-games but one, we are able to infer SEEK's goal with few actions. This allows PREV to start generating the counterplan when SEEK is far from its goal, increasing the odds of blocking it. In *Gathering in the maze* we are not able to infer SEEK's goal until the final part of its plan, and thus we cannot block it. Even though we have reduced the time needed to perform the goal recognition, it is still the part that contributes the most to the total counterplanning time.

## 5.6 RELATED WORK

In this section, we examine the relevant literature related to our counterplanning approach, showing how it is situated within previous works. First, we put it in the context of adversarial planning; then, we show how the work is strongly related to goal reasoning.

### *Adversarial Planning*

The name of our technique is inspired by the work of (Carbonell, 1981). In this case, a domain-dependent decision making process computes strategies to block opponent agents in domains like political conflicts solving (Carbonell, 1978). The author characterizes two main types of such strategies: diversionary counterplanning, where an agent diverts an opponent from achieving its initial goal; and obstructive counterplanning, where an agent makes the opponent's goal state unreachable. Our domain-independent counterplanning approach relies on the latter.

The related work often frames similar problems as Stackelberg security games from game theory (Paruchuri et al., 2008). In this framework, the leader (defender) moves first, followed by the follower (attacker), who will choose her action after observing the leader's choice. A solution (a Stackelberg equilibrium (Stackelberg, 1952)) is a leader/follower move pair that minimizes the defender's loss given optimal (worst case) attacker play. These games assume that the attacker will observe the defender's moves and choose an optimal response. This paradigm appears to fit many real-world situations, and has been successfully applied in airport patrolling (Pita et al., 2008) or protection of critical infrastructures like power grids (Brown et al., 2005), to name a few. Most works relying on game theory are domain-dependent. A remarkable exception is the work by Speicher *et al.* (2018), where the authors introduce Stackelberg planning. They formulate both the leader and the follower as planning agents and use a classical planner to find a Pareto frontier of Stackelberg equilibriums. They assume that agents do not share any action. However, this paradigm does not fit our problem, since we are interested in simultaneous games, where agents execute actions concurrently.

In adversarial scenarios where agents execute concurrently, Jensen and Veloso (2000) present two algorithms: optimistic and strong cyclic adversarial planning. These algorithms return universal plans, which can be seen as policies that associate the best action to apply given a state for achieving the goal. However, they assume the opponent's goal is known, while we use planning-based goal recognition to infer it from a set of candidate goals.

### *Goal Reasoning*

Recently, there has been increasing interest in the study and generation of agents capable of reasoning about their own and opponents' goals as well as their environment (Aha, 2018; Cox, 2007; Vattam et al., 2013). Some works follow the Goal-Driven Autonomy (GDA) process, which integrates a diverse set of AI components such as HTN planning or explanation generation (Molineaux et al., 2010; Weber et al., 2010).

Focusing only on adversarial settings, researchers often test their goal reasoning systems and agents in strategy games. This is because in these games, agents need to constantly reason about their opponent's actions and intents, changing its current goals and strategies accordingly. In these games, strategy selection is the most employed technique (Ontañón et al., 2013) when deciding which strategy to follow. It is based on choosing a strategy from a set of predefined policies given the current state. Multiple approaches for strategy selection exist, ranging from game-theoretic approaches (Sailer et al., 2007; Tavares et al., 2016), fuzzy rules (Preuss et al., 2013), or case-based reasoning (Aha et al., 2005; Jaidee et al., 2013; Wender & Watson, 2014). A common drawback of these approaches is that they must have access to a strategy set, which must be previously given by a domain expert or generated from games or simulations.

Some other works have explored strategy construction in strategy games (Churchill et al., 2012; Stanescu et al., 2014; Uriarte & Ontañón, 2014). To deal with its huge search space, hierarchical and abstract game state representations are used along game-tree search algorithms such as *minimax* or *Monte Carlo Tree Search* to build the strategies. The drawback of these approaches is that they require heavy knowledge engineering processes to generate the abstraction levels and the goals involved in each of them. Moreover, these approaches do not take into account opponent's intentions.

The strategies can be selected or generated given the current game state. However, few works focus on predicting the opponent's movements or strategies to enhance agents' reasoning process. Weber and Mateas (2009) proposed a data mining approach to strategy prediction, Kabanza *et al.* (2010) recognize opponent's intentions using plan libraries, and Stanescu and Čertický (2016) employ answer set programming to predict the units produced by the opponent. However, none of the aforementioned works is capable of observing the opponent and autonomously synthesize a good plan from scratch to counter the opponent strategy. This lack of adaptation has been identified as one of the major challenges in AI applied to strategy games (Ontañón et al., 2013). And precisely this is what we are proposing with DICP: a domain-independent approach to generate goals and plans given the opponent's actions.

## 5.7 SUMMARY

We have presented a novel fully automatic domain-independent approach for counterplanning, which is based on classical planning techniques. We have formally defined the counterplanning task involving two planning agents: a SEEK agent that seeks to achieve some goals; and a PREV agent that tries to prevent its opponent from achieving its goals. To successfully block an agent in a domain-independent way, we: (1) recognize the opponent's goals by observing its performed actions; (2) identify the counterplanning landmarks of its planning task; and (3) generate a sequence of actions to block its goal achievement process. We introduced DICP, an algorithm with four different versions that provide different opponent's stopping guarantees. Results in competitive planning domains show the ability of our algorithms to compute valid counterplans that minimize metrics such as the number of steps that SEEK can execute or the length of PREV's plan. Empirical evidences show the existence of counterplans is strongly related to: (1) the ability of inferring the opponent's goal with few observations; and (2) the number of counterplanning landmarks and their distance to both agents.

Most versions of DICP are not fast enough to be used in real-time settings, largely due to the slowness of the goal recognition part. Moreover, they assume a set of observations is given at once, instead of receiving them one at a time as occurs in many real-world scenarios. To address this we introduced RTDICP, a version of DICP that can work in *almost* real-time by processing observations one at a time and using a lighter goal recognition approach. We tested RTDICP in Starcraft, a well-known real-time strategy game. Results on different minigames of increasing difficulty show how we can effectively block opponents in such environments.

# 6 ANTICIPATORY COUNTERPLANNING

> *"More is lost by indecision than by wrong decision."*
>
> *- Carmela Soprano*

As we have briefly discussed in the previous chapter, many counterplanning tasks are not solvable given that SEEK is closer to all the landmarks involved in $\mathcal{G}_{\text{SEEK}}$ than PREV. If this happens when SEEK has not moved, i.e., a counterplanning task with $\mathcal{O}_{\text{PREV}} = \emptyset$, there is little PREV can do to block its opponent. However, in some counterplanning episodes this PREV's *handicap* comes from the fact that PREV stands still observing SEEK actions. Even if PREV was able to stop SEEK at the beginning of the counterplanning episode (with $\mathcal{O}_{\text{PREV}} = \emptyset$) now it cannot, since SEEK is now closer to all the landmarks.

We can see these two cases in Figure 6.1. The counterplanning task depicted in the left figure is unsolvable regardless of PREV actions; there is no landmark that PREV can falsify before SEEK stops needing it. On the other hand, the counterplanning task in the right would have a solution if the agent had started moving in the *right* direction at the same time as SEEK executed its first action, instead of standing still and watching.

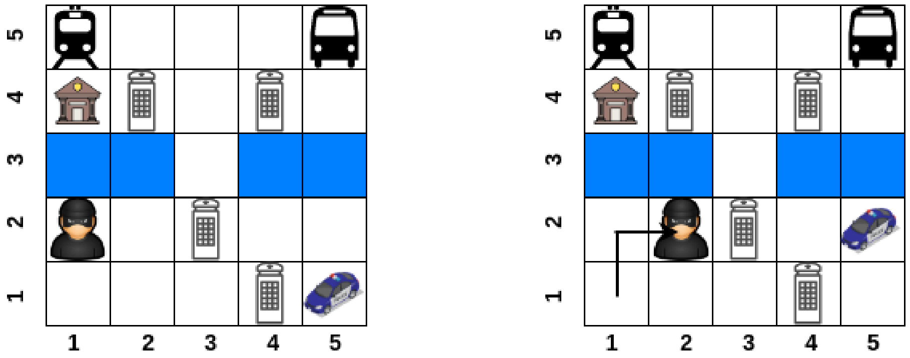


Figure 6.1: Unsolvable counterplanning tasks in the POLICE CONTROL domain. The tasks on the left is unsolvable regardless PREV actions. The task would be solvable if PREV had started moving at the same time as SEEK.

In the previous chapter we assumed counterplanning tasks where the observations were given *all at once* in $\mathcal{O}$. We relaxed this in the RTS games evaluation, assuming PREV receives observations *one at a time* to perform the goal recognition. However, up to now we were assuming PREV only moves when it has inferred SEEK's goal and generated a counterplan. In this chapter we want to go further and increase PREV's chances of blocking its opponent. To do that, we study (1) which are the *right* actions PREV should start executing; and (2) when it should stop executing those actions and start computing a counterplan.

## 6.1 CONNECTIONS BETWEEN GRS AND COUNTERPLANNING

Solving a counterplanning task can be summarized as *finding a counterplanning landmark that* PREV *can delete before* SEEK *can stop needing it*. But these landmarks are often *closer* to PREV than SEEK. In DICP and RTDICP, PREV does not execute any action until there is at least one counterplanning landmark common to all the likely goals. A better strategy would be to start approaching the counterplanning landmarks of all the likely goals, regardless if PREV has already inferred or not SEEK's goal. By doing this, PREV would position itself in a better state from which its odds of blocking SEEK would increase. But, what particular state should PREV start approaching? And what plan should it follow?

We already answered these questions in Chapter 3. In this case, PREV should compute a centroid or minimum-covering state of the counterplanning landmarks (Pozanco et al., 2019). By heading to these states PREV will maximize its odds of blocking SEEK. The next question is, when should PREV stop approaching the centroid of the counterplanning landmarks and start generating a goal and its corresponding counterplan? There are many alternatives, but we propose here to stop heading to the centroid/minimum-covering state as soon as there exists a counterplan that achieves a counterplanning landmark. The full procedure is detailed in Algorithm 6, which details ADICP (Anticipatory Domain-independent Counterplanning), a variation of DICP that solves a counterplanning task with increasing observations in a similar fashion as RTDICP. Initially, $\mathcal{O}_{\text{PREV}} = \emptyset$.

It receives the same inputs as DICP plus $t$, a parameter that determines if we want to compute centroid or minimum-covering states. The algorithm loops until no more observations are received, i.e., SEEK has reached its goal or its plan has been blocked. First $\mathcal{O}_{\text{PREV}}$ is updated with the last observed action. Then we update $\mathcal{G}_{\text{SEEK}}$ and extract the set of strong counterplanning landmarks. We only want to compute the centroid of those counterplanning landmarks that PREV can *still* delete. In this case we slightly change the counterplanning landmarks extraction process by using the EXTRACTLISTOFCPL and EXTRACTLISTOFSCPL functions. They differ from the standard functions we previously used in that they also return the individual (strong) counterplanning landmarks, i.e., the counterplanning landmarks of each goal $G_i \in \mathcal{G}$ in addition to the usual set of common (strong) counterplanning landmarks. If the set of common strong counterplanning landmarks is empty, i.e., there is no common strong counterplanning landmark for any of the most likely goals, we call the GRSCP function (line 22) which returns the next action to be executed by PREV based on the centroid computation. Otherwise, we start the counterplanning process, selecting $G_{\text{PREV}}$ from CPList and generating a plan to achieve it. If the returned plan is empty, we remove that counterplanning landmark from CPL and call GRSCP again. This will make PREV execute the action prescribed by GRSCP until a counterplan $\pi_{\text{PREV}}$ is found.

Algorithm 7 details how GRSCP works. It receives as input PREV's planning task $\Pi_{\text{PREV}}$, the list of strong counterplanning landmarks CPList, the current composite state $I_c$, and the goal related state to compute $t$. The algorithm ranks

**Algorithm 6** ADICP

```
Require: $\mathcal{C}$
Require: s,p,m,t
Ensure: $\pi_{PREV}$
$\pi_{PREV} \leftarrow \emptyset$
$G_{PREV} \leftarrow \emptyset$
$I_c = I_{SEEK} \cup I_{PREV}$
Start = True
while GetObservation() $\neq \emptyset$ or Start = True do
    Start = False
    $o_{PREV} \leftarrow$GetObservation()
    $\mathcal{O}_{PREV} \leftarrow \mathcal{O}_{PREV}$.INSERT$(o_{PREV})$
    $\mathcal{G}_{SEEK} \leftarrow$ RecognizeGoals$(F_{SEEK}, A_{SEEK}, I_{SEEK}, \mathcal{G}_{SEEK}, \mathcal{O}_{PREV})$
    CPL, CPList $\leftarrow$ ExtractListOfCPL$(\mathcal{C})$
    CPL, CPList $\leftarrow$ ExtractListOfSCPL$(\mathcal{C}$, CPL, CPList$)$
    if CPL $\neq \emptyset$ then
        $I_c = \Gamma(I_c, o_{PREV})$
        while $\pi_{PREV} = \emptyset$ and CPL $\neq \emptyset$ do
            $G_{PREV} \leftarrow$ SelectGoal(CPL, $m, s)$
            $\pi_{PREV} \leftarrow$ Planner$(F_{PREV}, A_{PREV}, I_c, G_{PREV}, p)$
            if $\pi_{PREV} = \emptyset$ then
                CPL $\leftarrow$ CPL $\setminus G_{PREV}$
                $\pi'_{PREV} \leftarrow$ GRSCP$(\Pi_{PREV}$CPList$, I_c, t)$
                $I_c = \Gamma(I_c, \pi'_{PREV})$
    else
        $\pi'_{PREV} \leftarrow$ GRSCP$(\Pi_{PREV},$CPList$, I_c, t)$
        $I_c = \Gamma(I_c, o_{PREV}, \pi'_{PREV})$
return $\pi_{PREV}$
```

**Algorithm 7** GRSCP

```
Require: $\Pi_{PREV}$,CPList,$I_c$,t
Ensure: $\pi_{PREV}$
$\pi_{PREV} \leftarrow \emptyset$
$\mathcal{G} \leftarrow$Rank(CPList)
$\pi_{PREV} \leftarrow GRS^{g}(F_{PREV}, A_{PREV}, I_c, \mathcal{G})$
return $\pi_{PREV}$
```

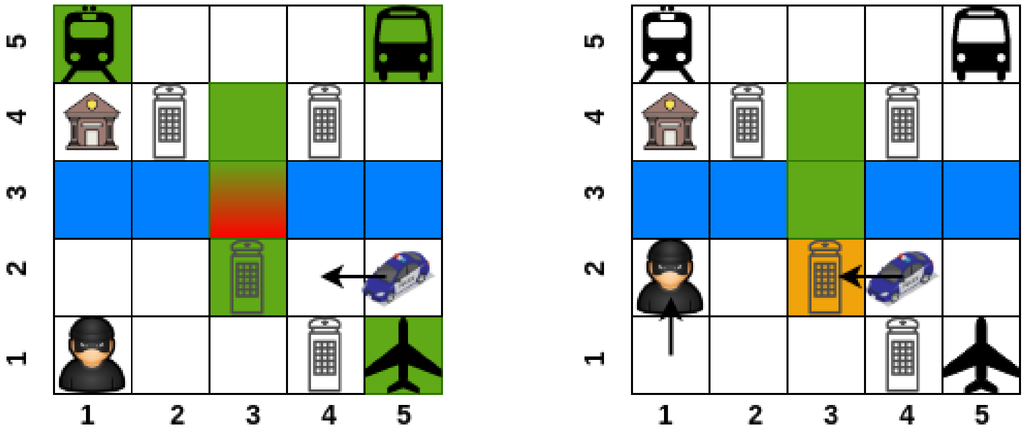


Figure 6.2: Behavior of ADICP in a POLICE CONTROL counterplanning task. Green cells depict individual strong counterplanning landmarks for each goal. Cells in red depict the centroid of the strong counterplanning landmarks. The orange cell in the right image represents the goal selected by PREV to block SEEK.

the counterplanning landmarks in CPList to generate the set of candidate goals GRS$^{\mathcal{G}}$ receives as input. This ranking is computed using the following formula in the Rank function (line 3). We use SET as a function that maps a list into a set.

$$\mathcal{G} = \left\{ \langle G_i, w_i \rangle \mid w_i = \frac{\text{COUNT}(G_i, \text{CPList})}{|\text{CPList}|} \ \forall G_i \in \text{SET}(\text{CPList}) \right\} \tag{6.1}$$

In this way, counterplanning landmarks appearing in multiple goals will have a higher weight in $\mathcal{G}$, and therefore GRS$^{\mathcal{G}}$ will prioritize approaching them. Finally, we slightly change the behavior of GRS. Given that the actions executed by SEEK might change the set of likely goals and therefore the counterplanning landmarks, we are not interested in finding an optimal centroid nor the plan to achieve it because they might become useless in the next iteration of ADICP. In this case we only want to execute one action that makes PREV approach most of the counterplanning landmarks. Thus, we will stop GRS$^{\mathcal{G}}$ when the first action has been computed. This will be the action returned by GRSCP.

Let us exemplify how the algorithm works in the POLICE CONTROL counterplanning task depicted in Figure 6.2. The start of the counterplanning episode (and the algorithm) is shown in the left image. SEEK has not performed any action (no observation has been received), so all the goals are equally likely. The algorithm then extracts the list and set of strong counterplanning landmarks. Since there is not common strong counterplanning landmark among the most likely goals (line 12), no valid counterplan can be produced yet, and the algorithm calls GRSCP with the list of individual strong counterplanning landmarks CPList (line 22). The green cells depict the individual strong counterplanning landmarks. The size of CPList is 9, i.e., there are 9 individual strong counterplanning landmarks since some are repeated, i.e., they are common to several goals. The counterplanning landmarks at each station appear only once in CPList. Therefore, we assign them a weight of $\frac{1}{9} = 0.11$. On the other hand, the three counterplanning landmarks in the middle of the image

($\langle$(not (free c3_2),(not (free c3_3),(not (free c3_4)$\rangle$) appear twice in CPL, for SEEK escaping through the train and bus stations. Therefore, their associated weight is $\frac{2}{9} = 0.22$. This would be the set of weighted goals returned by the RANK function from the list of counterplanning landmarks:

$$\begin{aligned}\mathcal{G} = \{&\langle\texttt{(not (free c1\_5))}, 0.11\rangle, \langle\texttt{(not (free c5\_5))}, 0.11\rangle, \langle\texttt{(not (free c5\_1))}, 0.11\rangle,\\ &\langle\texttt{(not (free c3\_4))}, 0.22\rangle, \langle\texttt{(not (free c3\_3))}, 0.22\rangle, \langle\texttt{(not (free c3\_2))}, 0.22\rangle\}\end{aligned}$$

Then, GRSCP calls $\text{GRS}^{\mathcal{G}}$, which returns the next action that PREV should execute: moving east from c5_2 to c4_2[1]. This action makes the agent to closer to most of the individual strong counterplanning landmarks. After that, the composite state is updated with SEEK's observed action $o_{\text{PREV}} = $ move c1_1 c1_2 and the action prescribed by GRSCP, $\pi'_{\text{PREV}} = $ move c5_2 c4_2. This update ends the first iteration of ADICP, and the algorithm asks for a new observation.

The first observation received is shown in the right image of Figure 6.2. SEEK has moved north, so now the most likely goals are the terrorist escaping through the bus and train stations. In this case there are strong counterplanning landmarks common to all the most likely goals (CPL $\neq \emptyset$), and thus we can start the counterplanning process. The first counterplanning landmark that PREV can delete is also the closer to SEEK, free c3_2, so its negation is set as $G_{\text{PREV}}$. Finally, a plan to achieve $G_{\text{PREV}}$ is computed and the algorithm ends returning the counterplan $\pi_{\text{PREV}} = \langle$move c4_2 c3_2$\rangle$. Note that neither this counterplan nor any other would have been existed if PREV would not have moved towards the centroid of the counterplanning landmarks in the first place.

## 6.2 EVALUATION

We randomly generated 100 counterplanning tasks on POLICE CONTROL, PAINTED BLOCKS-WORDS, and ROVERS & MARTIANS in the same way as we did in Section 5.4. We did not conducted experiments in COUNTER LOGISTICS because it does not have strong counterplanning landmarks. In this case, we provide one observation at a time to each algorithm. The counterplanning episode finishes when no more observations are received. This can occur when SEEK achieves its goal or PREV blocks its plan. We compare three different algorithms:

- DICP*, the optimal version of DICP. It will only generate actions when the set of strong counterplanning landmarks is not empty.
- ADICP*. It will start heading to the centroid / minimum-covering state of the strong counterplanning landmarks until it can generate a strong counterplan.
- RANDOMADICP, a variation to ADICP* that executes random actions rather than actions that approach the centroids/minimum-covering states.

In all versions, $m$ is set to stop SEEK as soon as possible. Table 6.1 summarizes the results of our evaluation. We only report the values of those metrics that

[1] Although the modified version of $\text{GRS}^{\mathcal{G}}$ only returns the next action, we show in red the centroid of the counterplanning landmarks towards PREV is heading.

| Domain | Algorithm | $E$ | $\%\pi_{\text{SEEK}}$ |
|---|---|---|---|
| POLICE CONTROL | DICP* | 0.2 | 0.8 |
| | ADICP* | 0.5 | 0.7 |
| | RANDOMADICP | 0.3 | 0.8 |
| PAINTED BLOCKS-WORDS | DICP* | 0.2 | 0.9 |
| | ADICP* | 0.7 | 0.6 |
| | RANDOMADICP | 0.4 | 0.8 |
| ROVERS & MARTIANS | DICP* | 0.6 | 0.5 |
| | ADICP* | 0.7 | 0.5 |
| | RANDOMADICP | 0.6 | 0.5 |

Table 6.1: Comparison of DICP*, ADICP*, and RANDOMADICP in POLICE CONTROL, PAINTED BLOCKS-WORDS, and ROVERS & MARTIANS. The metrics measured are counterplanning accuracy $E$, and percentage of SEEK plan allowed.

are relevant for this experiment. For example, since we are using the optimal versions of the algorithm, $\%E$ is always 1. The time employed by ADICP in computing the next action to be executed is negligible, so $t_{\mathcal{C}}$ is also equal for all the algorithms.

As we can see, ADICP* outperforms DICP* in all domains. It is able to return strong counterplans in more than twice as many counterplanning tasks in POLICE CONTROL and PAINTED BLOCKS-WORDS. The difference is smaller in ROVERS & MARTIANS. This is because in that domain PREV is already close to most counterplanning landmarks, and therefore the difference between DICP* and ADICP* is often small.

ADICP* also allows PREV to block SEEK sooner in most cases, allowing it to execute less steps of its plan. On average, ADICP* make 3.3 calls to GRSCP in POLICE CONTROL, 1.3 in PAINTED BLOCKS-WORDS, and 1.1 in ROVERS & MARTIANS in the cases where $E = 1$. This means that after those number of executed actions, the algorithm is able to find a strong counterplanning landmark. Taking a look at these actions returned by GRSCP, they try to approximate the police to the landmarks of all the goals in $\mathcal{G}_{\text{SEEK}}$ in POLICE CONTROL as we saw in the previous section. In the case of PAINTED BLOCKS-WORDS, PREV tries to move around the rooms, positioning itself in a better room from which to collect the needed paint. In ROVERS & MARTIANS the martian starts heading to samples' locations.

Note that even RANDOMADICP achieves better results than DICP. In other words, in most counterplanning tasks it is slightly better to randomly move until a strong counterplanning landmark is inferred rather than standing still just observing. By executing random actions, PREV is able to solve more tasks on average, although some times the executed actions make the counterplanning task unsolvable.

Table 6.1 only reports results in which ADICP* tries to approach the centroid of the individual strong counterplanning landmarks. We got almost the same

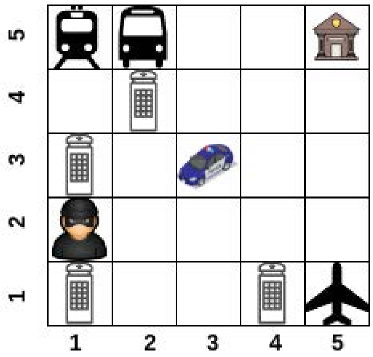


Figure 6.3: POLICE CONTROL counterplanning task where computing centroids or minimum-covering states changes the behavior of ADICP.

results in our randomly generated counterplanning tasks when t was set to centroids or minimum-covering states. Although it seems difficult to randomly generate tasks where the behavior of the algorithm changes with the value of t, we can hand-craft tasks such as the one depicted in Figure 6.3 where they do differ. Let us assume that the real goal of SEEK is to escape through the airport. In this case, if ADICP is run with t = centroid, the first action GRSCP would prescribe would be to move north or west. If PREV executes one of these actions, it will not be able to generate a strong counterplan to stop SEEK in the next iteration when SEEK moves south, being the airport now the most likely goal. On the other hand, if ADICP is run with t = minimum-covering, the plan prescribed by GRSCP would be empty, i.e., do not execute any action since PREV is already in the minimum-covering state of all individual strong counterplanning landmarks. By standing still in this case, PREV will be able to generate a strong counterplan in the next iteration, negating `free c5_1`.

## 6.3 SUMMARY

As we have seen in the latest chapters of this document, many counterplanning tasks do not have a solution due to the initial state of SEEK and PREV. However, many other tasks are not solvable only because of the inactivity of PREV just observing SEEK actions until a (strong) counterplanning landmark is found.

In this chapter we have moved to a more realistic setting, demonstrating how we can increase the chances of finding valid counterplans by starting to act at the same time as SEEK. Experimental results show how we stop opponents more often by just executing random actions until we can infer a common counterplanning landmark among the most likely goals. We can increase even more the stopping odds by acting *in the right way* rather than randomly. We proposed that this right way comes from the centroids and minimum-covering states computation we presented in Chapter 3. By executing actions towards those states, we are able to duplicate the number of counterplanning tasks we can solve.

Part IV

# CONCLUSIONS AND FUTURE WORK

# 7 CONCLUSIONS

> *"It's good to be in something from the ground floor. I came too late for that and I know. But lately, I'm getting the feeling that I came in at the end. The best is over."*
>
> *- Tony Soprano*

Throughout this thesis, and following the goal reasoning paradigm, we have switched the predominant question planning aims to answer from *which sequence of actions should I follow to achieve my goal?* to *which goals should I achieve?* We introduced and formalized new problems in which autonomous agents need to answer this question, both when acting in isolation and when acting with other agents. Although we drew the main conclusions at the end of each chapter, we would like to summarize them here, also highlighting the main contributions presented in this thesis.

1. In **single-agent** settings, we allow agents to reason about a set of goals. This results in the following specific contributions:

    - We introduced *Distance-based planning* in Chapter 3, a novel planning task that aims at finding states that fulfill some cost-related property with respect to a given set of goals. We defined eight different states and two kind of plans that agents can follow to reach them. We also showed the benefits of computing such states and plans in many applications ranging from goal obfuscation to anticipatory planning. Moreover, we presented GRS, a forward search planning algorithm to solve Distance-based planning tasks. Experimental results showed that even though optimally solving these tasks is unfeasible in practice, suboptimal versions of GRS can produce fast solutions that are close to the optimal on average.
    - We introduced LGG-AP, a planning architecture that allows agents to reason not only with a current set of goals, but also with the ones that might appear in the near future. This task is known as anticipatory planning, and previous works assumed that a distribution of those goals over time was known beforehand. In Chapter 4 we showed how this distribution can be learned from scratch by monitoring the environment, inferring goal appearance patterns and starting to plan for goals before they actually appear. Experimental results showed how this approach outperforms reactive planning in an UAV surveillance domain where goals can be predicted. We also showed evidences on how the combination of GRS and LGG-AP can enhance agents' anticipatory capabilities.

2. In **multi-agent** settings, we have studied the dynamic generation of goals based on other agents' behavior, putting the focus in competitive scenarios. This results in the following specific contributions:

    - We introduced *Domain-independent Counterplanning* in Chapter 5, a novel planning approach to prevent opponents' from achieving their goals. By using our technique, agents automatically formulate their goals based on (1) the actions performed by the opponent; and (2) the planning landmarks involved in the opponent's goal achievement process. Unlike other approaches, we are not selecting goals and/or plans to follow from a library when we observe a particular plan/behavior of the opponent, but generating them from scratch. Moreover, we are not producing a plan that denies all opponent's goals, which is often unfeasible in practice. Our approach is *closer* to the real world, since we are incorporating goal recognition techniques to better filter the set of goals the opponent might be trying to achieve, maximizing the odds of blocking him/her. We formalize counterplanning tasks, defining the existence of counterplans and some of their properties. We conducted experiments both in planning domains and Starcraft, a RTS game, showing how our approach can actually be used to block other agents in competitive environments.

    - Finally, we showed how we can use GRS to augment the odds of blocking opponents. Instead of starting with an empty goal and generating it upon receiving opponent's observations, agents can compute the centroid (or minimum covering state) of the opponent's candidate goals, starting to approach it before they infer the enemy's goal. Once agents infer it, they can start computing a counterplan, but now being in a better position to block the opponent. We presented this approach in Chapter 6, calling it Anticipatory Counterplanning since it unifies the two main bodies of work developed in this thesis: distance-based planning and counterplanning. By unifying both techniques, we came up with a way agents can generate multiple goals/plans and change them as they receive observations from opponents.

# 8 FUTURE WORK

> *"The past is always with us. Where we come from, what we go through, how we go through it; all this matters. Like at the end of the book, you know, boats and tides and all. It's like you can change up, right, you can say you're somebody new, you can give yourself a whole new story."*
>
> *- D'Angelo Barksdale*

Most of our contributions in this thesis are related to new problem definitions. Therefore, in some cases we did not have the necessary time to explore all the alternatives and potential of our techniques, as well as to conduct a more in depth study of their clear synergies. Here we outline some of the research avenues we devise to extend and strength the work presented in this thesis.

### *Distance-based Planning*

We are using GRS, a forward search algorithm with different possible configurations, to solve Distance-based planning tasks. It strongly depends on the heuristic we use to estimate the distances to the goals. In fact, we demonstrate that we can only solve distance-based tasks optimally, i.e., computing optimal goal-related states, when using a perfect heuristic estimator. As we proved, we cannot use admissible heuristics to preserve the optimality of the algorithm. This is because in Distance-based planning we are interested on ranking states based on an aggregation of multiple cost estimations (one for each goal), and not on estimating the cost of reaching a single goal. An admissible heuristic when aggregated can incorrectly rank states, and therefore we would lose optimality. We would like to explore and devise heuristics that still preserve the *correct ranking* of states when aggregating the cost estimation to several independent goals. If such heuristics exist, we could compute optimal goal-related states much faster. On the other hand, we would also like to experimentally test how the use of other heuristics affects the solutions returned by GRS.

In our current approach, GRS uses the same metric when computing the state and the plan. But there might be cases in which it would be interesting to compute the state that minimizes a particular metric, and then reach it through a plan that minimizes/maximizes a different one. For instance, we could be interested in reaching a centroid state through a plan that minimizes the maximum distance to any of the goals.

Finally, GRS might not be the most efficient algorithm to compute some of the states we propose. For example in the case of centroids, a better way to proceed could be to perform backward search from the goals. On top of that, one could combine backward and symbolic search to reduce the time needed to compute the centroids of a Distance-based planning tasks. Finding more efficient ways of computing the rest of states such as minimum-covering or r-centroids is not as clear, so more research on this would be needed.

### *Domain-independent Counterplanning*

Our definition of counterplanning episode and counterplanning task is somehow strict: we require to know the opponent's model, initial state, and a set of candidate goals. Moreover, we require to observe every action of the opponent in the correct order and without noise, and we are assuming it is acting optimally. These assumptions, even though common in many domains, might restrict the applicability of our technique in many others. We would like to explore the progressive relaxation of the above mentioned assumptions in future work.

Another crucial assumption we make is that agents act concurrently applying one action at each time step. This allows us to simplify some definitions such as the joint execution of two actions, or strong plans. However, it would be interesting to broaden them to be able to deal with temporal actions having arbitrary duration and different preconditions and effects dynamics.

The solvability of a counterplanning task strongly depends on the initial state of both agents with respect to the landmarks, in the same way that inferring the correct goal of an agent depends on its position with respect to the goals. In future work we would like to study the applicability of techniques similar to Goal Recognition Design (Keren et al., 2020) in the context of counterplanning to better generate scenarios where blocking opponent agents becomes easier.

We are also interested in further exploring the synergies between Distance-based planning and Domain-independent Counterplanning that we started studying in Chapter 6. Our current approach only reasons about strong counterplanning landmarks, which might be expensive to compute in some cases and unpractical in some domains. We would like to test how different relaxations such as computing the centroid states over SEEK goals affect the behavior of the algorithm. Moreover, we would like to add the notion of *strong centroids* (and other goal-related states) when computing them in multi-agent settings such as counterplanning.

Finally, we would also like to study how to adapt our counterplanning approach to be used in collaborative scenarios. We could use goal recognition techniques and landmarks computation to identify spots where other agents might need help, trying to provide them those landmarks in earlier steps of their plans.

# Part V

# APPENDIX

# A

# APPENDIX OUTLINE

Throughout this dissertation, we have provided different approaches that allow agents to reason about their goals both in single and multi-agent settings. In the subsequent appendices we include some works carried out during the thesis that, although having a clear connection with the main ideas introduced here, are the fruit of collaboration with other people. The appendices also include some extra experiments, results and domains from the chapters included in the dissertation.

# B

# MULTI-TIER AUTOMATED PLANNING FOR ADAPTIVE BEHAVIOR

The work presented in this appendix is the result of collaboration with Sebastián Sardina and others during my research internship at RMIT. The main part of this appendix has been published in ICAPS'20 as a full paper with the title "Multi-tier Automated Planning for Adaptive Behavior".

> *"Adapt or perish, now as ever, is Nature's inexcusable imperative."*
>
> *- H.G. Wells*

In automated planning planners typically generate a plan to achieve a *given goal* from an initial state, using a *model* of the environment, i.e., a planning domain. As we have already discussed in the thesis, this limits the autonomy of planning-based agents.

Up to now we were assuming that planning domains perfectly model the environment's dynamics. However, this is not the case in many real-world situations. As any model, planning domains are never "complete" and they inevitable make assumptions on the dynamics of the environment. In other words, the knowledge engineer must specify which is the expected outcome (effects) after applying an action into the environment. A limitation of standard planning formalism is that they do not account for deviations from such assumptions, and hence are not well prepared for integration within an execution framework. A common approach to handle discrepancies between what the planner expected and what happened at run-time is to simply perform *re-planning* or *plan-repair* (Fox et al., 2006). But, why would the system keep reasoning about the same model of the world that has been proven "wrong"? And should the system keep trying to achieve the same goal whose achievement has already failed?

In this chapter, and inspired by work in Software Engineering (D'Ippolito et al., 2014), we introduce a "generalized" planning framework that aims to better account for the uncertainties at design/engineering time. Concretely, rather than fixing the level of risk and objectives, we envision the specification of various assumption levels, as well as different goals for each assumption level. This is achieved by allowing the knowledge engineer to specify a family of planning domains, each carrying a set of assumptions. For example, in an fully idealized model (domain) of the BLOCKS WORLD domain, a robotic arm always successfully grabs blocks, whereas in less idealized models the gripper may fail when picking, maybe missing or even breaking the blocks. Depending on the assumptions imposed on the gripper operation, one may aim for different types of block towers (goals).

The aim of the framework is to synthesize not one but a *collection* of interrelated policies embedding, together, *adaptive* behavior. So, as the environment violates assumptions, the agent should gracefully "degrade" to less refined planning models. Since such models carry less assumptions on the environment, less functionalities can generally be offered. If the gripper may break a block while picking it, building a complete block may just not be achievable. So, with model degradation, comes goal degradation, usually to a less ambitious (and often less demanding) one. We call the above framework *multi-tier adaptive planning* and is the topic of this chapter.

## B.1 NO-RUNNING EXAMPLE

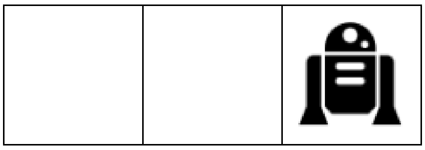


Figure B.1: No-running example.

Consider a robot moving in a $1 \times n$ grid like the one shown in Figure B.1, and similar to the dust-cleaning robot example in (Bonet & Geffner, 2015). The robot can *walk* one cell at a time or it can *run* covering multiple cells in one shot. Unfortunately, neither the physical shape of the corridor nor the physical guarantees of the robot's actuators are fully known to the designer. Because of that, in some scenarios, some cells may be impossible to navigate and the robot may get damaged or even broken. So, the knowledge engineer considers various possible assumption levels on the environment's behavior together with corresponding adequate objectives, as shown in Figure B.2.

In the most idealized model $\mathcal{D}_3$, the designer assumes that both walking and running actions succeed with no negative side-effects. The goal there is for the robot to reach a destination cell located at the end of the corridor (`c0`) and intact. In a less idealized $\mathcal{D}_2$, both running and walking actions still cause the robot to advance, but no assumption can be made on their side effects and movement may cause minor damages. The robot should then just aim to reach the target destination `c0`. Finally, in the least idealized $\mathcal{D}_1$, a walking action may sometimes cause the robot to get minor damages without even advancing, and even worse, a running action may get the robot *broken* and render it unusable. Under those weaker assumptions, the robot should return to base location `c2` for servicing.

Under the above multi-tier specification, the robot tries its best, but adapts its behavior as it discovers some assumptions may not hold. To do so, the robot initially assumes the most idealized world model and thus works for the most ambitious goal: reach destination undamaged. But, upon observing an inconsistency with the assumptions, it must *adapt* both the model of the environment as well as the objective being pursued. For example, if the robot succeeds in advancing when moving but gets a minor damage, it should *degrade* to $\mathcal{D}_2$. If it

```
(:action walk
  :parameters
    (?o - Cell ?d - Cell)
  :precondition (and (at ?o)
    (adj ?o ?d) (not (broken)))
  :effect (and
    (not (at ?o)) (at ?d)))

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (and
    (not (at c2)) (at c0)))

(:goal (and (at c0)
            (not (scratch))
            (not (broken))))
```

Listing B.1: Model $\mathcal{D}_3$.

```
(:action walk
 :parameters
   (?o - Cell ?d - Cell)
 :precondition (and (at ?o)
   (adj ?o ?d) (not (broken)))
 :effect (oneof
   (and (not (at ?o)) (at ?d))
   (and (not (at ?o)) (at ?d) (scratch))))

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (oneof
    (and (not (at c2)) (at c0))
    (and (not (at c2)) (at c0) (scratch))))

(:goal (and (at c0) (not (broken))))
```

Listing B.2: Model $\mathcal{D}_2$.

```
(:action walk
  :parameters (?o - Cell ?d - Cell)
  :precondition (and (at ?o)
    (adj ?o ?d) (not (broken)))
  :effect (oneof
    (and (not (at ?o)) (at ?d))
    (and (not (at ?o)) (at ?d) (scratch))
    (scratch) ))

(:action run
  :precondition
    (and (at c2) (not (broken)))
  :effect (oneof
    (and (not (at c2)) (at c0))
    (and (not (at c2)) (at c0) (scratch))
    (broken) ))

(:goal (and (at c2) (not broken)))
```

Listing B.3: Model $\mathcal{D}_1$.

Figure B.2: Actions walk left and run left in the three models.

actually fails to move at all, it should *degrade* to $\mathcal{D}_1$ to operate under such level weaker assumptions.

A solution to this scenario must, on the one hand, strive for the best possible solution and, on the other hand, be open to a potential future degradation. Concretely, the robot should never attempt to perform an action that may prevent graceful degradation. In our example, while, in principle, running would be the most efficient way to reach the destination, it may cause a catastrophic failure in tier level 1, precluding the goals of every tier. Thus, the robot must be conservative and should cautiously move by walking. Hence, we shall call this example the *no-running example*.

In what follows, we propose a framework to specify problems like this one in the realms of automated planning, define its solution concept, and show how to compute solutions with existing planning technology.

## B.2 BACKGROUND

There are many real-world scenarios where the dynamics of the environment are not well-known in advanced, hindering a succint definition of actions' effects. Several planning approaches have been proposed to model and plan with non-deterministic transition systems where the environment is still fully-observable. One of the most prevalent is probabilistic planning, which associates a non-zero probability to each alternative action's outcome. Most probabilistic planning techniques are based on Markov Decision Processes (MDPs) (Puterman, 1994) formulations (Kolobov et al., 2009), although very successful probabilistic planners use *simpler* approaches based on a determinization of the input task (Yoon et al., 2007).

In some cases the stochasticity of a system (1) is not well understood, i.e., we can not provide accurate probabilities to each action outcome; or (2) its understood is deemed unnecessary. In those cases, Fully-Observable Non-Deterministic (FOND) planning (Cimatti et al., 2003) offers a valid framework similar to probabilistic planning, but with the exception that there is no commitment to the probability of the different action's effects taking place.

We formally define a ***FOND planning task*** in a similar fashion as classical planning tasks (Definition 1), with the exception of actions. Now, action's effects are not a set of literals that become true or false in the state after the execution of action $a$. Instead, $eff(a) = e_1 | \ldots | e_n$ with $n \geqslant 1$, is the (non-deterministic) effect of the action $a$. Each $e_i$ is a (set of) conditional effects $C \rightarrow E$, with $C$ being a Boolean formula over the set of propositions $F$, and $E$ a set (conjunction) of propositions. The intended meaning is that *one* of the $e_i$ effects ensues non-deterministically, by the environment's choice. We will use $S$ to denote the set of all possible states of a FOND planning task.

Solutions to FOND planning tasks are no longer sequential plans but policies that map states into actions. A ***policy controller*** is a function $\pi : S \rightarrow 2^A$ that maps state $s \in S$ to a set of (executable) actions $\pi(s)$. A policy $\mathcal{C}$ executed from state $s \in S$ on a given domain $D$ (understanding $F$ and $A$ as domain) defines a set of possible executions $Ex_\pi(D, s)$ of the form $\lambda = s_0 o_0 s_1 \ldots s_i o_i s_{i+1} \ldots$, where $s_0 = s$, $o_i \in \pi(s_i)$, $\mathrm{pre}(o_i) \subseteq s_i$, and $s_{i+1}$ is a possible successor state when $o_i$ is executed in state $s_i$ w.r.t. domain $D$, for all $i \geqslant 0$.

Cimatti et al. (2003) identified three different classes of solutions to FOND problems:

- Weak solutions, which achieve the goal, but without guarantees. In other words, weak plans are *optimistic*, and will only achieve the goal if *everything goes as expected*.

- Strong solutions, which guarantee goal achievement regardless the environment casuistry.

- Strong-cyclic solutions, which guarantee goal achievement relying on a *fairness* assumption.

Roughly speaking, a *fair action* is one in which all effects occur infinitively often when the action is executed infinitively many times in the same state. For example, it is known that by tossing a (truthful) coin many times, heads will eventually appear, so it would be a fair action. On the other hand, there is no guarantee of the coin falling on its edge, even if the coin is tossed infinitely many times, so it would not be a fair action. When all actions are assumed fair, a strong-cyclic plan guarantees that the agent, by *retrying*, eventually achieves the goal; i.e., heads will appear if the coin is tossed many times. In turn, when fairness can not be assumed, a plan with acyclic executions that reaches the goal in a bounded number of steps (a strong policy) is required.

The ***Dual FOND*** (or FOND+) hybrid variation has recently been introduced to deal with domains that have both fair and unfair actions (Camacho & McIlraith, 2016; Geffner & Geffner, 2018). In those cases, strong-cyclic solutions amounts to only use the fair actions of the domain. This framework is useful for domains in which we are able to separate *faulty* behaviors from *normative* ones in the actions' dynamics.

Let us put an example of a domain having both fair and unfair actions by borrowing the BLOCKS WORLD example by Camacho and McIlraith (2016), which is depicted in Figure B.3. Initially, block $A$ is on top of block $B$, and the goal is to put $A$ on the table. The considered actions of the domain are simplifications over the standard BLOCKS WORLD actions. The agent can pick up blocks from other blocks (or from the table) and put them on top of other blocks (or on the table).

```
(:action pick-up
    :parameters (?x1 - block)
    :precondition (and
        (handempty)
        (clear ?x1)
    )
    :effect (oneof
        (and (not (handempty)) (not (clear ?x1)))
        (and (clear ?x1) (ontable ?x1))
    )
)
```

Listing B.4: Non-deterministic action in the BLOCKS WORLD domain.

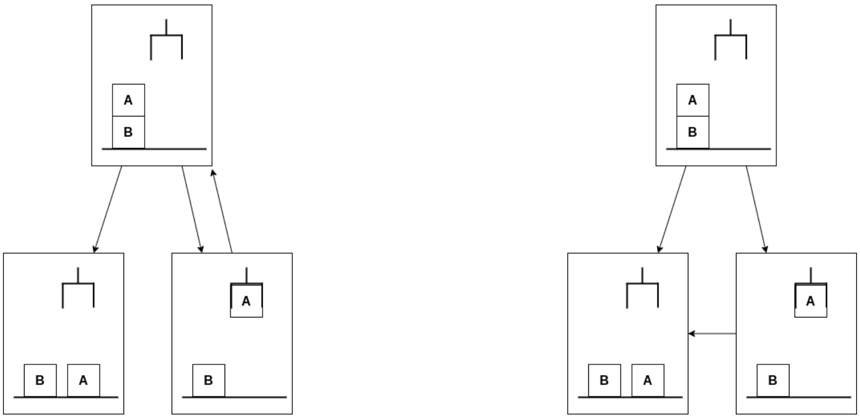


Figure B.3: Different policies for the BLOCKS WORLD domain. The left image shows a strong-cyclic solution in FOND. The right image shows a strong-cyclic solution in Dual FOND.

The `pick-up` action is non-deterministic in this case, and the gripper may drop the block onto the table (second non-deterministic effect), but is guaranteed to eventually pick-up the block (first non-deterministic effect) if tried repeated times. This action is depicted in Listing 4 in order to better understand it, and also to show the `oneof` statement, which is the only addition to PDDL needed to represent FOND tasks. The `put-on` action is deterministic and puts the block held by the gripper on top of another block/table.

Strong-cyclic FOND solutions that assume fairness may rely on the eventual occurrence of the faulty effect of `pick-up` in order to reach the goal. The left image of Figure B.3 shows the strong-cyclic solution that only uses the `pick-up` action. The policy picks up block A and puts it on top of B repeatedly until block A is dropped on the table. Although valid under the FOND framework, it is not a good solution, since it is relying on a failure to achieve the goal. A better way to proceed would be to use the `put-on` action in case pick-up succeeds, since it is deterministic and also offers a strong-cyclic plan, as depicted in the right part of Figure B.3.

FOND (and Dual FOND) planning is again more expressive and expensive than classical planning (Rintanen, 2004a). However, effective optimized techniques and solvers have been developed in the last years (Geffner & Geffner, 2018; Muise et al., 2014).

## B.3 MULTI-TIER PLANNING

Following (D'Ippolito et al., 2014), we propose a multi-tier automated planning framework in which the knowledge engineer can specify a ranked set of assumptions about the environment and a corresponding set of objective goals. We formally define a multi-tier planning task as follows.

**Definition B.1.** *A* **multi-tier planning domain** *(MTD) is a tuple $\langle \Omega, \leqslant \rangle$ such that:*

1. *$\Omega$ is a set of FOND planning domains over the same propositions $F$ and actions $A$, and every action has the same preconditions across all domains in $\Omega$;*

2. *$\leqslant$ is a partial-order relation over $\Omega$ such that $\mathcal{D}_1 \leqslant \mathcal{D}_2$ implies $Ex(\mathcal{D}_2, s) \subseteq Ex(\mathcal{D}_1, s)$ for all states $s \in S$; and*

3. *$\leqslant$ has a greatest element in $\Omega$, denoted $\hat{\mathcal{D}}$, as well as a minimum element.*

The first condition states that an MTD is just a collection of planning domains over the same vocabulary. While the action names and their preconditions ought to be the same in all domains, the effects of each operator may differ across domains. Such differences in operator effects will reflect *different assumptions on the environment*. However, the differences cannot be arbitrary: they are to reflect model "refinements." This is achieved by the second condition, which specifies that domains in lower tiers of the hierarchy ($\mathcal{D}_1$) must produce the same behaviors as higher models ($\mathcal{D}_2$), and possibly more. The intuition is that higher-level models are "refinements" of lower-level models, posing possibly more assumptions on the environment (e.g., by actions having fewer non-deterministic effects), hence permitting fewer execution runs.

As in standard planning, a problem instance task adds a specific initial situation and a (set of) objectives.

**Definition B.2.** *A* **multi-tier planning problem** *(MTP) is a tuple $\mathcal{M} = \langle\langle\Omega, \leqslant\rangle, s_I, \mathcal{G}\rangle$ where $\langle\Omega, \leqslant\rangle$ is an MTD, $s_I$ is $\mathcal{M}$'s initial state, and $\mathcal{G}$ is a function mapping each domain $\mathcal{D}$ in $\Omega$ to a goal $G(\mathcal{D})$ (or just $G_{\mathcal{D}}$).*

Observe that unlike standard planning approaches, we allow the designer to specify various goals, depending on the assumed environment. Often, the weaker the assumptions, the lower level of functionality that may be guaranteed (2014).

**Example B.1.** *Figure B.2 depicts an MTP for the no-running example previously described, in which the modeler specifies three planning domains. The two actions `walk` and `run` are modeled at each of the three tiers. The actions' schema and preconditions are the same across all tiers. However, the effects differ progressively, as assumptions are relaxed from higher tiers to lower tiers. For example, in third tier model $\mathcal{D}_3$, the most idealized model, it is assumed that movement actions always succeed, whereas in the lowest tier model $\mathcal{D}_1$, actions may fail and may even do so catastrophically. Also observe the goals for each tier may differ, since when assumptions are relaxed, more ambitious goals may not be achievable. For example, while in $\mathcal{D}_3$ and $\mathcal{D}_2$ we want the robot to reach the end of the corridor, in the lowest tier model $\mathcal{D}_1$ it is enough for us that the robot stays in the initial position. Because effects are incrementally relaxed across tiers, runs at lowers tiers are strict super sets of those in upper tiers.*

Finally, we define a structure that associates a specific policy to each domain in an MTD, prescribing what behavior should ensue from the executor under the different models.

**Definition B.3.** *A* **multi-tier controller** *(MTC) for an MTD $\langle\Omega, \leqslant\rangle$ is a function $\mathcal{C} : \Omega \mapsto (S \mapsto 2^O)$ mapping each domain $\mathcal{D} \in \Omega$ to a specific policy $\mathcal{C}(\mathcal{D})$ (or just $\mathcal{C}_{\mathcal{D}}$).*

The challenge now is to formally capture when an MTC amounts to a "solution" strategy for a multi-tier planning problem. To do so, it is important to first understand how the MTC structure is meant to be deployed in the environment. Intuitively, at any point in time, the executor is operating relative on some planning domain (i.e., model) of the world $\mathcal{D}$ from the ones available in $\Omega$, by carrying out its corresponding policy $\mathcal{C}(\mathcal{D})$ so as to bring about the level's goal $G(\mathcal{D})$. Initially, the executor deploys policy $\mathcal{C}(\hat{\mathcal{D}})$ from the initial problem state $s_I$ on the most idealized domain $\hat{\mathcal{D}}$, aiming at achieving the most ambitious goal $G(\hat{\mathcal{D}})$. However, if at any point during execution, an inconsistency with the current model $\mathcal{D}_i$ is observed, the executor ought to switch to an alternative domain $\mathcal{D}_j \in \Omega$ such that $\mathcal{D}_j \leqslant \mathcal{D}_i$. Technically, an inconsistency amounts to observing an actual state $s$ that cannot be explained with planning domain $\mathcal{D}_i$. Of course, once the executor switches downwards—referred as *degradation*—the model it operates on to a more permissive one (i.e., one with weaker assumptions), the objective sought, and hence the strategy, must be changed too. A smart executor, though, aims to degrade gracefully, that is, as little as possible, switching to a planning domain that retains as many assumptions as possible about the environment (and the most ambitious goal).

Let us now develop the solution concept for MTPs. We define the set of triggering states for a domain in an MTD as those states in which the executor, when deployed in a given multi-tier controller as per the above operational scheme, may need to start operating under such domain. As expected, the initial state $s_I$ is the triggering state for the highest level, most idealized, domain $\hat{\mathcal{D}}$. For other domains, a triggering state amounts to a degradation step.

**Definition B.4.** *Let $\mathcal{C}$ be an MTC for a MTD $\langle\Omega, \leqslant\rangle$, and let $s$ be a state (over variables $V_\Omega$). We inductively define the set of* **triggering initial states** *for each planning domain $\mathcal{D} \in \Omega$ under $\mathcal{C}$, denoted $Init(\mathcal{D}, \mathcal{C})$, as follows:*

1. *$Init(\hat{\mathcal{D}}, \mathcal{C}) = \{s_I\}$;*
2. *if $\mathcal{D}$ is not the maximum in $\Omega$ (i.e., $\mathcal{D} \neq \hat{\mathcal{D}}$), then*

$$\begin{aligned}
&Init(\mathcal{D}, \mathcal{C}) = \\
&\quad \{s \mid \mathcal{D} < \mathcal{D}', s' \in Init(\mathcal{D}'), \\
&\qquad \lambda \in Ex_{\mathcal{C}_{\mathcal{D}'}}(\mathcal{D}', s'), o = \mathcal{C}_{\mathcal{D}'}(last(\lambda)), \\
&\qquad \lambda o s \in Ex(\mathcal{D}, s') \setminus \bigcup_{\mathcal{D}'' : \mathcal{D} < \mathcal{D}''} Ex(\mathcal{D}'', s').\}
\end{aligned}$$

Let us explain the second case. Suppose the executor has so far been carrying out policy $\mathcal{C}_{\mathcal{D}'}$ on a domain model $\mathcal{D}'$, from some (triggering) state $s'$. Suppose this has yielded execution run $\lambda$ (consistent with $\mathcal{D}'$). However, when executing the next prescribed operator $o$ (as per the corresponding policy $\mathcal{C}_{\mathcal{D}'}$ for $\mathcal{D}'$), the resulting evolution to a state $s$ yields an execution $\lambda o s$ that can be explained by (i.e., its a legal execution in) domain $\mathcal{D}$ but not by any model higher than $\mathcal{D}$ (including $\mathcal{D}'$). When this happens, state $s$ is a triggering state for $\mathcal{D}$, that is state where the executor may have to start operating under domain model $\mathcal{D}$ when using policy $\mathcal{C}_{\mathcal{D}}$.

Next, for a controller $\mathcal{C}$ to be a solution for an MTP $\mathcal{M}$, it must achieve the associated goal of a domain in $\mathcal{M}$ *from all the triggering states of the domain in question.*

**Definition B.5.** *An MTC $\mathcal{C}$ is a* **solution controller** *for an MTP $\mathcal{M} = \langle\langle\Omega, \leqslant\rangle, s_I, G\rangle$* iff *for every domain $\mathcal{D} \in \Omega$, the projected policy $\mathcal{C}_{\mathcal{D}}$ is a solution plan for planning problem $\langle\mathcal{D}, s, G_{\mathcal{D}}\rangle$, for every state $s \in Init(\mathcal{D}, \mathcal{C})$.*

Note that, unlike standard planning, this definition requires each policy to work from more than one initial state. However, it is not the case that all policies in $\mathcal{C}$ need to work from the initial state $s_I$. Such a requirement would be too demanding for capturing the intended operational framework as described above, this is because most policies, if not all but $\mathcal{C}_{\hat{\mathcal{D}}}$, will ever be used at state $s_I$ (unless the system comes back to such state after some degradation).

## B.4 SOLVING MULTI-TIER PLANNING PROBLEMS

Informally, an MTP is a collection of similar planning problems and a solution amounts to solution policies for each problem that can be "connected", if necessary, at degradation time. A naive approach thus would repetitively compute solution policies for each planning problem, making sure they "connect." We show here we can solve the whole problem in a principled manner and in one shot. Concretely, we build a single Dual FOND planning task $\mathcal{P}_{\mathcal{M}}$ from a given MTP $\mathcal{M}$ such that a strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$ amount to an MTC solution for $\mathcal{M}$. To argue for technique's generality, we first identify a meaningful fragment of MTDs.

**Definition B.6.** *A planning domain $\mathcal{D}_2 = \langle F_2, A_2\rangle$ is an* **oneof-refinement** *of a domain $\mathcal{D}_1 = \langle F_1, A_1\rangle$ iff $A_1 = A_2$ and for every $\langle o, Pre^2_o, Eff^2_o\rangle \in A_2$, there is a $\mathcal{D}_1$-operator $\langle o, Pre^1_o, Eff^1_o\rangle \in A_1$ such that $Pre^1_o = Pre^2_o$ and $Eff^2_o \subseteq Eff^1_o$;*

That is, $\mathcal{D}_2$ is like $\mathcal{D}_1$ but may contain fewer non-deterministic effects on some operators. It turns out that, in the context of Dual FOND planning, oneof-refinements capture all possible refinements in a multi-tier planning task—any MTD is equivalent to a oneof-refinement type.

**Theorem B.1.** *Let $\langle\Omega, \leqslant\rangle$ be an MTD and $\mathcal{D}_1, \mathcal{D}_2 \in \Omega$. Then, $\mathcal{D}_1 < \mathcal{D}_2$ (i.e., planning domain $\mathcal{D}_2$ is a refinement of domain $\mathcal{D}_1$)* **iff** *there exists a planning domain $\mathcal{D}'_2$ such that:*

1. *$Ex(\mathcal{D}_2, s) = Ex(\mathcal{D}'_2, s)$, for all $s \in S$ (that is, $\mathcal{D}'_2$ are equivalent planning domains); and*

2. *$\mathcal{D}_2$ is an oneof-refinement of $\mathcal{D}_1$.*

This states that the only meaningful difference between ordered domains in $\Omega$ comes, only, in the refined domain ($\mathcal{D}_2$) having *less* (in terms of set inclusion) non-deterministic effects in some operators.

### *Compilation to Dual FOND planning*

Let $\mathcal{M} = \langle\langle\Omega, \leqslant\rangle, s_I, G\rangle$ be a a multi-tier planning problem such that $\mathcal{D} \leqslant \mathcal{D}'$ if and only if $\mathcal{D}'$ is a oneof-refinement of $\mathcal{D}$. Due to Theorem B.1, restricting $\leqslant$ to a oneof-refinement relation does not affect generality. From now on, for technical legibility and without loss of generality, we assume domains in $\Omega$ are STRIP-like with no conditional effects.

In this section, we shall construct a single dual-FOND planning problem $\mathcal{P}_\mathcal{M} = \langle\mathcal{D}_\mathcal{M}, s_\mathcal{M}, G_\mathcal{M}\rangle$ that will fully capture problem $\mathcal{M}$. For compactness, we use $\mathit{Eff}_o^\mathcal{D}$ to denote the effects of operator o in planning domain $\mathcal{D}$.

Let us start by explaining the general strategy being encoded into $\mathcal{P}_\mathcal{M}$. Roughly speaking, the planning problem $\mathcal{P}_\mathcal{M}$ will model a dynamic system running as per multi-tier specification $\mathcal{M}$. As such, at any time, the system is operating relative to some model $\mathcal{D}$ in $\Omega$ (initially, the most ambitious $\hat{\mathcal{D}}$), trying to achieve $\mathcal{D}$'s goal via an appropriate plan, and degrading to an adequate (lower) model when action outcomes' do not align with model $\mathcal{D}$. To achieve this, the encoding will model an *iterative two-phase* process in which an *acting* phase is, sometimes if necessary, followed by an *alignment & degradation* phase. A special variable *act* is used to distinguish both phases. As expected, during an *acting phase*, an operator representing some domain action is executed. This step involves the execution of a non-deterministic action with fair semantics, and the optional subsequent execution of an unfair version of the action. In the latter case, the system will then evolve to an *alignment phase*, in which the encoding verifies whether the outcomes seen correspond to the assumed current model $\mathcal{D}$; and if not, the behavior is "degraded" to an appropriate (lower-level) model that is able to explain the observed outcome.

It turns out that one of the key challenges is to encode a proper and scalable alignment phase in a planning domain (i.e., in PDDL): *how can we encode that a given effect could be explained by some model in $\Omega$ (but not by another model)?* In some sense, doing so would amount to reducing meta-level reasoning to the object (PDDL) level. We show that, via a clever encoding, that reduction is indeed possible. The technical difficulty is depicted in the following example.

**Example B.2.** *Consider the case in which the robot is operating in the highest domain model $\mathcal{D}_3$ and decides to execute action* `walk`*. Upon execution, the robot senses variable* `scratch` *true—the robot is now damaged. In the standard (intuitive) configuration, in which the robot starts non-damaged, the robot should* degrade *its operational model to domain $\mathcal{D}_2$, as that is the highest model explaining the damage. However, if the robot happens to start damaged (i.e., scratched) already, then domain model $\mathcal{D}_3$ still explains the transition, and no degradation should occur. Here,* `walk`*'s effects under $\mathcal{D}_3$ and $\mathcal{D}_2$ are indistinguishable.*

This example shows that just observing a proposition (`scratch`) in a transition that does not appear in an effect (`walk`'s effect under $\mathcal{D}_3$) does *not* directly imply the effect may not explain the transition. Can we then characterize, succinctly, under which conditions a set of observed propositions E is explained by some operator o in a domain model $\mathcal{D}$? It turns out we can.

**Definition B.7** (Effect Explicability). *Let* $\mathsf{E}$ *be a set of literals (e.g., effects that have just seen to be true after an action execution). The conditions for operator* $\mathsf{o}$ *in domain* $\mathcal{D}$ *to explain* $\mathsf{E}$*, denoted* $\mathit{Explains}[\mathsf{o}, \mathcal{D}, \mathsf{E}]$*, is defined as follows (recall* $\Delta$ *is the set symmetric difference operation):*

$$\mathit{Explains}[\mathsf{o}, \mathcal{D}, \mathsf{E}] = \bigvee_{\mathsf{E}' \in \mathit{Eff}_{\mathsf{o}}^{\mathcal{D}}} \bigwedge_{\mathsf{l} \in \mathsf{E} \Delta \mathsf{E}'} \mathsf{l}.$$

That is, some effect $\mathsf{E}'$ of $\mathsf{o}$ in model $\mathcal{D}$ yields the same result as effect $\mathsf{E}$, if all the literals that one effect has an the other does not *were already true* (at the outset of $\mathsf{o}$'s execution). In our Example B.2, if we take $\mathsf{E}$ to be the second effect of `walk` in $\mathcal{D}_2$ (i.e., $\mathsf{E} = \{$`(not (at ?o)),(at ?d),(scratch)`$\}$) we have $\mathit{Explains}[$`walk`$, \mathcal{D}_3, \mathsf{E}] =$ `scratch`, as the literal `scratch` is the only one in the effects' symmetric difference.

Observe that if $\mathsf{E}$ and $\mathsf{E}'$ are inconsistent, the formula will contain the conjunction of a proposition and its negation, thus reducing to false. In fact, the following result guarantees the intended meaning of the above definition.

**Lemma 1.** *Let* $\mathsf{s}, \mathsf{s}' \in 2^{V \cup \overline{V}}$ *be two domain states. Let* $\mathsf{E} \in 2^{V \cup \overline{V}} \supseteq \mathsf{s}' \setminus \mathsf{s}$ *be a set of literals including at least all new literals in* $\mathsf{s}'$ *w.r.t. to* $\mathsf{s}$*. Then, state* $\mathsf{s}'$ *is a possible successor when operator* $\mathsf{o}$ *is executed in state* $\mathsf{s}'$ *under model* $\mathcal{D}$ *if and only if* $\mathsf{s} \models \mathit{Explains}[\mathsf{o}, \mathcal{D}, \mathsf{E}]$.

We are now ready to provide the encoding of $\mathcal{M}$ into a dual-FOND planning problem $\mathcal{P}_{\mathcal{M}} = \langle \mathcal{D}_{\mathcal{M}}, \mathsf{s}_{\mathcal{M}}, \mathsf{G}_{\mathcal{M}} \rangle$.

DOMAIN VARIABLES. The *set of propositional variables* $V^{+}$ of $\mathcal{D}_{\mathcal{M}}$ is obtained by extending the set of variables $V$ in $\mathcal{M}$'s domains with the following additional variables:

- $\varepsilon_{\mathcal{D}}$, for each domain $\mathcal{D} \in \Omega$, that will be used to signal that model $\mathcal{D}$ is a/the highest model explaining the effect of the last executed action;
- $\ell_{\mathcal{D}}$, for each domain $\mathcal{D} \in \Omega$, that will be used to track the most "ambitious" compatible model so far;
- *act*, use to denote the system is in the *acting phase* (otherwise, it is in the *alignment phase*);
- $\mathsf{u}_{\mathsf{o}}$, for each operator $\mathsf{o} \in \hat{\mathcal{D}}$, that will be used to ensure the execution of a unfair action; and
- *end*, used to denote the goal achievement of the current model of execution.

INITIAL STATE & GOAL CONDITION. The *initial state* of $\mathcal{P}_{\mathcal{M}}$ is:

$$\mathsf{s}_{\mathcal{M}} = \mathsf{s}_{\mathsf{I}} \wedge [\ell_{\hat{\mathcal{D}}} \wedge \bigwedge_{\mathcal{D} \in \Omega^{-}} (\neg \varepsilon_{\mathcal{D}} \wedge \neg \ell_{\mathcal{D}}) \wedge \mathit{act} \wedge \neg \mathit{end}].$$

This encodes the initial state $\mathsf{s}_{\mathsf{I}}$ of the MTP $\mathcal{M}$ and the fact that the system starts in the $\Omega$'s greatest, most ambitious, domain model $\hat{\mathcal{D}}$ (proposition $\ell_{\hat{\mathcal{D}}}$

and all other $\ell_x$'s are false) and in the action phase. In addition, all effect level signaling variables $\varepsilon_x$ are initialized to false (no action has been executed), as well as the goal variable *end*.

Finally, the goal condition of $\mathcal{P}_\mathcal{M}$ is simply $G_\mathcal{M} = end$. We will see below which actions make variable *end* true.

DOMAIN OPERATORS. The planning domain $\mathcal{D}_\mathcal{M}$ will include *two types* of operators, one for modeling the actual domain actions and one for implementing the alignment check (and potential degradation) process. Let us start with the former.

So, for each (domain) operator $o$ in domain $\mathcal{D} \in \Omega$, we include a operator $\langle o_\mathcal{D}, Pre, Eff \rangle$ in $\mathcal{D}_\mathcal{M}$, where:

- $Pre = Pre_o^\mathcal{D} \wedge \ell_\mathcal{D} \wedge act \wedge \bigwedge_{o \in \hat{\mathcal{D}}} \neg u_o$, that is, action $o_\mathcal{D}$ is executable when $o$ itself is executable in $\mathcal{D}$, and the system is currently operating under model $\mathcal{D}$ and is the fair-acting phase (*act* is true and all $u_x$ are false); and

- $Eff = Eff_o^\mathcal{D} \cup \{u_o\}$, that is, when operator $o_\mathcal{D}$ is executed, either one of original effects of $o$ in $\mathcal{D}$ ensues or a distinguished predicate $o_u$ is made true.

When one of the effects of $o$ in domain $\mathcal{D}$ happens, it just resembles the dynamics of domain $\mathcal{D}$. However, if the effect that ensues is $u_o$, the system evolves into a "unfair-acting" phase ($act \wedge u_o$), explained after the following example.

**Example B.3.** *The resulting walk action for the domain level $\mathcal{D}_2$ in the compilation would look as follows in PDDL:*

```
(:action walk_d2
  :parameters (?o - cell ?d - cell)
  :precondition (and (at ?o) (adj ?o ?d) (not (broken))
        (d2) (act) (not (u_walk)) (not (u_run)))
  :effect (oneof
   (and (not (at ?o)) (at ?d))
   (and (not (at ?o)) (at ?d) (scratch))
   (u_walk)
   ))
```

Next, when the the system evolves to the unfair-acting phase (e.g., due to effect `u_walk` happening in the example above), the only executable action will be a second version of domain operator $o$, which in turn will include *all effects* of $o$ *across all domains* in $\Omega$, together with additional book-keeping variables $\varepsilon_x$ to support the next alignment, and potential degradation, reasoning phase. More concretely, for each domain operator $o$ in $\hat{\mathcal{D}}$, $\mathcal{D}_\mathcal{M}$ includes a rather powerful operator $\langle o_{unfair}, Pre, Eff \rangle$, where:

- $Pre = act \wedge u_o \wedge Pre_o$, that is, the system is in the unfair-acting phase for operator $o$; and

- *Eff* is a set of nondeterministic effects, each being a collection (i.e., conjunction) of conditional effects built as follows. For every effect $E$ of operator $o$ that is mentioned in a domain $\mathcal{D}$ but not in any lower one (i.e.,

$E \in \mathit{Eff}_o^{\mathcal{D}} \setminus \bigcup_{\mathcal{D}':\mathcal{D}'<\mathcal{D}} \mathit{Eff}_o^{\mathcal{D}'}$), *Eff* contains, as one of its non-deterministic effects, the following complex effect:

$$\bigwedge_{\mathcal{D}':\mathcal{D}'\geqslant\mathcal{D}} (C^{E}_{\mathcal{D}'} \Rightarrow E \wedge \neg \mathit{act} \wedge \neg u_o \wedge \varepsilon_{\mathcal{D}'}),$$

where $C^{E}_{\mathcal{D}'} = \mathit{Explains}[o, \mathcal{D}', E] \wedge \bigwedge_{\mathcal{D}'':\mathcal{D}''>\mathcal{D}'} \neg \mathit{Explains}[o, \mathcal{D}'', E]$.

Intuitively, the operator $o_{\mathit{unfair}}$ contains each possible effects $E$ of $o$ (from any domain in $\mathcal{M}$) as a non-deterministic option. In turn, the set of conditional effects for a particular effect $E$ will not only make the effect $E$ itself ensue, but will also set a "marker" proposition $\varepsilon_{\mathcal{D}}$ signaling the highest domains explaining the effect in question. To realize that, condition $C^{E}_{\mathcal{D}'}$ above states that the original effect $E$ is explained by (some effect of) operator $o$ at domain model $\mathcal{D}'$ but not by any other model higher than $\mathcal{D}'$. When that is the case, proposition $\varepsilon_{\mathcal{D}'}$ is set to true, recording the fact that $\mathcal{D}'$ is the *highest* model explaining such effect. Observe that by the way all conditions are designed, they ought to be mutually exclusive, so only one $\varepsilon_x$ will be made true. In addition, *act* is set to false so as to force the reasoner into the *alignment* phase, to be explained shortly. (We note that the effects of level $\mathcal{D}$ itself are accounted when $\mathcal{D}' = \mathcal{D}$.)

Importantly, while $o_{\mathcal{D}}$ operator will be treated *fair*, action $o_{\mathit{unfair}}$ will be treated as *unfair*—this is where dual FOND semantics (Geffner & Geffner, 2018) come into play. Also, as the following example shows, significant syntactic simplifications can be achieved in $o_{\mathit{unfair}}$ by analyzing conditions in conditional effects and precondition of the action.

**Example B.4.** *Let us see complete Example B.3 by showing the unfair version of the* `walk` *action. After syntactic simplification w.r.t. conditions and the action precondition, we obtain the simpler, more readable, equivalent action:*

```
(:action walk_unfair
  :parameters (?o — cell ?d — cell)
  :precondition (and (act) (u_walk)
                     (at ?o) (adj ?o ?d) (not (broken)))
  :effect (and (not (act)) (not (u_walk))
    (oneof
      (when (true)
          (and (not (at ?o)) (at ?d) (e3)) )
      (and (when (not (scratch))
              (and (not (at ?o)) (at ?d) (scratch) (e2)) )
           (when (scratch)
              (and (not (at ?o)) (at ?d) (e3)) ))
      (when (true) (and (scratch) (e1) )) )))
```

As we discussed before, this unfair action contemplates the effects present in all the domain models. The intended meaning of this is that whenever an action executes, it may fail and we may observe effects of any domain level. However, we do not want the planner to rely on these possible failures, so we contemplate them as unfair actions.

ALIGNMENT & DEGRADATION OPERATORS. When the unfair version of a domain operator has been executed, an effect could ensue that might not be explained by the current domain under which the reasoner is operating under (encoded via propositions $\ell_x$). If so, the system ought to gracefully degrade to a lower level model that is able to explain the last system evolution. We encode this reasoning, and potential degradation, in the so-called *alignment* phase (*act* is false).

In the best case, the state observed after the execution of an action corresponds to one of the expected ones w.r.t. the current planning domain model the executor is operating under. Technically, the reasoner continues operating under current model $\mathcal{D}$ (proposition $\ell_{\mathcal{D}}$ is true), provided domain $\mathcal{D}$ has been able to explain the evolution of the last executed action: proposition $\varepsilon_{\mathcal{D}'}$ has been set to true for some domain $\mathcal{D}'$ that is either $\mathcal{D}$ itself or a higher one in the hierarchy (recall effects in higher level domains are subsets of ). So, in such case, the planner (and executor) is able to execute special action $\langle \text{CONTINUE}_{\mathcal{D}}, \mathit{Pre}, \mathit{Eff} \rangle$ to keep planning/executing under the current model and goal:

- $\mathit{Pre} = (\neg act \wedge \ell_{\mathcal{D}} \wedge \bigvee_{\mathcal{D}' \geqslant \mathcal{D}} \varepsilon_{\mathcal{D}'})$, that is, the action can be executed during the alignment phase when the current domain or one of its refinements accounts for the last effect outcome.

- $\mathit{Eff} = (act \wedge \bigwedge_{\mathcal{D} \in \Omega^-} \neg \varepsilon_{\mathcal{D}})$, that is, effect signals are all reset and the system goes back to the action phase.

If, on the other hand, the state observed does *not* conform to the current operating model (i.e., proposition $\varepsilon_{\mathcal{D}}$ is false), then the system must *degrade* to a lower tier where the environment model would fit the observation, and adjust the objective to the corresponding (often less ambitious) goal. Needless to say, we expect a "smart" reasoner/executor to degrade as little as possible, by retaining as many assumptions on the environment as possible and only dropping those that have been observed to be wrong. This will allow the agent to aim for the highest, most valuable, goal so far.

Technically, when $\mathcal{D}, \mathcal{D}' \in \Omega$ such that $\mathcal{D}' < \mathcal{D}$, we include an operator $\langle \text{DEGRADE}_{\mathcal{D}\mathcal{D}'}, \mathit{Pre}, \mathit{Eff} \rangle$ in $\mathcal{P}_{\mathcal{M}}$, where:

- $\mathit{Pre} = \neg act \wedge \ell_{\mathcal{D}} \wedge \displaystyle\bigvee_{\{\mathcal{D}^* : \mathcal{D}^* \geqslant \mathcal{D}', \neg(\mathcal{D} \geqslant \mathcal{D}^* \geqslant \mathcal{D}'), \mathcal{D}^* \ngeqslant \mathcal{D}\}} \varepsilon_{\mathcal{D}^*}$; and

- $\mathit{Eff} = \neg \ell_{\mathcal{D}} \wedge \ell_{\mathcal{D}'} \wedge \bigwedge_{x \in \Omega} (\neg \varepsilon_x \wedge act)$.

That is, the controller can degrade from current operating domain $\mathcal{D}$ to domain $\mathcal{D}'$ if the last effect seen was explained by lower domain $\mathcal{D}'$ or any other domain higher than $\mathcal{D}'$ that is unrelated to $\mathcal{D}$ (so as to handle MTPs with a non-linear structure). The effect results in the controller being degraded to level $\mathcal{D}'$ (proposition $\ell_{\mathcal{D}'}$ becomes true), all booking explicability effect prepositions $\varepsilon_x$ being reset, and the reasoner progressing to the acting phase.

Note that, effectively, the dynamics of level variables $\ell_x$ are *outside the control of the reasoner*, as these depend only on which non-deterministic effects have occurred and how (i.e., how variables $\varepsilon_x$ have been set).

GOAL OPERATORS. The only part remaining is the overall goal of the multi-tier problem. Intuitively this should be "achieve the highest level goal", which under a conservative degradation process, it reduces to "achieve the goal of the current operating model." We therefore include goal actions $\langle \textsc{checkgoal}_{\mathcal{D}}, (\mathcal{G}_{\mathcal{D}} \wedge \ell_{\mathcal{D}}), \mathit{end}\rangle$, one per domain $\mathcal{D} \in \Omega$.

This completes the encoding of a multi-tier planning problem $\mathcal{M}$ into a single non-deterministic planning domain $\mathcal{P}_{\mathcal{M}}$. We now prove its correctness w.r.t. Definition B.5. First, any solution policy for the planning task amounts, as is, to a solution to the corresponding multi-tier planning problem.

**Theorem B.2.** *If $\pi$ is a strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$, then controller $\mathcal{C}^{\pi}(\mathcal{D})$ is an MTC solution for $\mathcal{M}$, where:*

$$\mathcal{C}^{\pi}_{\mathcal{D}}(s) = \pi(s \wedge \ell_{\mathcal{D}} \wedge \mathit{act} \wedge \bigwedge_{o \in \hat{\mathcal{D}}} \neg u_o), \textit{for all } s \in S.$$

*Proof.* Consider $s_i \in S$ and $\mathcal{D} \in \Omega$ such that $s_i \in \mathit{Init}(\mathcal{D}, \mathcal{C}^{\pi})$, and an infinite and fair execution $\lambda \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}, s_i)$. We show that goal $\mathcal{G}(\mathcal{D})$ holds true somewhere along $\lambda$ as follows:

1. We transform $\lambda$ into an execution $\hat{\lambda} \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}_{\mathcal{M}}, s_i^+)$, with $s_i^+ = s_i \cup \{\mathit{act}, \ell_{\mathcal{D}}\}$, by adding propositions $\mathit{act}$ and $\ell_{\mathcal{D}}$ to every state in $\lambda$ and replacing every domain operator $o$ with its $o_{\mathcal{D}}$ version.

2. If $so_{\mathcal{D}}$ appears infinitively often in $\hat{\lambda}$, we replace every second appearance of the form $so_{\mathcal{D}}s'$ by two-action steps $so_{\mathcal{D}}(s \cup \{u_{o_{\mathcal{D}}}\})o_{\mathit{unfair}}(s' \cup \{\mathit{act}, \varepsilon_{\mathcal{D}'}\})$ such that $\mathcal{D}' \geqslant \mathcal{D}$ is the highest domain in $\Omega$ that contains the effect of $o$ that supports the transition $so_{\mathcal{D}}s'$—we know there is one because $\hat{\lambda}$ is a legal execution in $\mathcal{D}_{\mathcal{M}}$ from state $s_i^+$. By doing this changes in $\hat{\lambda}$ we are guarantee that the execution is fair, while still preserving the fact that every domain action in it has the effects as per domain $\mathcal{D}$. So, execution $\hat{\lambda}$ mirrors the original $\lambda$ for domain $\mathcal{D}$ but over the extended language of $\mathcal{D}_{\mathcal{M}}$.

3. Because $s_i \in \mathit{Init}(\mathcal{D}, \mathcal{C}^{\pi})$, there exists a finite execution $\lambda_i \in \mathit{Ex}_{\pi}(\mathcal{D}_{\mathcal{M}}, s_I)$ (i.e., execution in $\mathcal{P}_{\mathcal{M}}$ via policy $\pi$) that ends in state $s_i \cup \{\ell_{\mathcal{D}}, \mathit{act}\}$. This means that $\lambda_i\hat{\lambda} \in \mathit{Ex}_{\mathcal{C}^{\pi}}(\mathcal{D}_{\mathcal{M}}, s_I)$, and since $\lambda_i\hat{\lambda}$ is fair (w.r.t. the fair actions) and $\hat{\lambda}$ has $\lambda_{\mathcal{D}}$ always true, it follows that $\hat{\lambda}$ has to reach the $\mathcal{P}_{\mathcal{M}}$'s goal by executing operator $\textsc{checkgoal}_{\mathcal{D}}$. Then, its precondition $\mathcal{G}_{\mathcal{D}}$ holds true at some point in $\hat{\lambda}$ and therefore in $\lambda$ too.

□

That is, the MTC controller $\mathcal{C}$ under domain $\mathcal{D}$ and in state $s$, what the strong-cyclic solution for $\mathcal{P}_{\mathcal{M}}$ prescribes when $\ell_{\mathcal{D}}$ is true and the reasoning cycle is in the acting phase.

In addition, any possible MTC solution will be represented by some strong-cyclic policy of $\mathcal{P}_{\mathcal{M}}$ (i.e., completeness).

**Theorem B.3.** *If $\mathcal{C}$ is an MTC solution for $\mathcal{M}$, then there exists a strong-cyclic solution $\pi$ for $\mathcal{P}_{\mathcal{M}}$ such that $\mathcal{C}^{\pi}(\mathcal{D}) = \mathcal{C}(\mathcal{D})$, for every domain $\mathcal{D}$ in $\mathcal{M}$ (where $\mathcal{C}^{\pi}(\mathcal{D})$ is as in B.2).*

*Proof.* Policy $\pi$ follows the domain actions prescribed by $\mathcal{C}_{\mathcal{M}}$ exactly, augmented with the booking auxiliary actions as needed. A similar argument, based on execution traces, as in Theorem B.2 can be built. □

We close by noting that the size of $\mathcal{P}_{\mathcal{M}}$ is increased by a linear number of bookkeeping propositional variables, and a quadratic number (w.r.t. the number of domains in $\Omega$) of extra actions. So, while the multi-tier planning framework appears to be more involved than the standard (non-deterministic) planning, it can be suitably reduced to the latter, with an encoding that is, arguably, fairly natural and comparable in size. Importantly, though, the solution proposed relies on the fact that we can specify *both* fair and unfair actions in the same planning model. This is a feature that will prove a challenge when actually realizing the technique in current planning technology, as we shall see next.

## B.5 VALIDATION

In this section, we demonstrate that MTPs can indeed be solved today with existing planning technology, but argue that additional effort in Dual FOND is necessary. The first obstacle is the availability of FOND planning technology supporting both fair and unfair assumptions. To the best of our knowledge, the only off-the-shelf planner to do so is Geffner and Geffner's (2018) FOND-SAT system. By leveraging on SAT solvers, their system yields an elegant declarative technique for FOND planning that features the possibility of combining fair and unfair actions. So, we report on using FOND-SAT over the encoding for our non-running example. Notwithstanding, the experiments reported are intended to demonstrate the existence of systems to solve MTPs and to provide a baseline for future work, rather than for providing a performance evaluation.

**Reproducibility.** The experiments were run in an i7-4510 CPU with 8GB of RAM. The code used to perform the compilation from the multi-tier specification to Dual-FOND is available online.[1]

Listing B.5 shows a fragment, in a readable plan-like format, of the outcome when FOND-SAT is ran on our encoding for the non-running example. Due to space issues, the full controller is shown in Appendix C. First of all, the plan cautiously avoids the run action altogether, as it may get the robot broken and precludes the achievement of all tier goals. After performing the walk *fair*-version action in (line 2) corresponding to the highest model $\mathcal{D}3$, the plan checks its effects (line 3). If proposition `u_walk` remains false (lines 4-9), the effect in model $\mathcal{D}3$ has occurred—the robot has done a successful move. If another walk (line 4) succeeds as well (lines 5-7), the robot achieves the top level $\mathcal{D}3'$ goal (line 6). Note that, in such a run, no alignment action is included: the walk unfair version has never been performed and hence only effects of $\mathcal{D}3$ has ensued.

[1] https://github.com/ssardina-planning/pypddl-translator

```
1   (:plan [
2     (walk_d3 c2 c1)
3     (if ((not (u_walk))) [
4       (walk_d3 c1 c0)
5       (if ((not (u_walk))) [
6         (check_goal_d3)
7       ])
8       ...
9     ]
10    (else) [
11      (walk_unfair)
12      (case (eff_e3_walk) [
13        (walk_e3_explained_by_d3)
14        (continue_d3)
15        ...
16      ]
17      (case (eff_e2_walk) [
18        (walk_e2_explained_by_d2)
19        (degrade_d3_d2)
20        ...
21      ]
22      (case (eff_e1_walk) [
23        (walk_e1_explained_by_d1)
24        (degrade_d3_d1)
25        ...
26      ]
27    ]  ])
28
```

Listing B.5: A fragment of the policy found by FOND-SAT.

If, instead, the first walking action (line 2) yields the special effect `u_walk`, the plan jumps to line 11. There, the only action available is the *unfair* version of walking (line 11), which has *all* the effects, as non-deterministic options, of the walking action across all domains $\mathcal{D}3$, $\mathcal{D}2$, and $\mathcal{D}1$. As FOND-SAT does not handle conditional effects, we simulate each conditional effect for the effect chosen by a set of `walk_eE_explained_by_dx` whose precondition is $C^{E}_{\mathcal{D}_x}$, together with the original precondition of the operator (`walk` in this case). Finally, if the effect chosen could be explained by the current operating domain (line 13, explained by domain $\mathcal{D}3$), the system executes a continue operation at the current level, enabling the next domain action. On the other hand, when the effect is explained by a lower domain than the one operating under (lines 18 and 23), degradation to the corresponding domain is carried out (line 19 and 24).

It is easy to see how this plan also represents a multi-tier controller which only outputs the fair version of the operators, and all the other auxiliar actions and propositions are the controller internal memory.

Now, what would happen if the robot starts scratched as discussed in Example B.2? It turns out the problem becomes *unsolvable*. The reason is that any observed scratch after movement does *not* need to be explained by a different model than $\mathcal{D}3$ (e.g., the walk effect of $\mathcal{D}2$ that scratches the robot), as the scratch is explained already by being true originally. Thus, when walking always advances the agent, it would never degrade its behavior, remain operating in $\mathcal{D}3$ without ever achieving $\mathcal{D}3$'s goal. If, however, we drop the non-scratched requirement from $\mathcal{D}3$, the problem would be solvable again, though with a slightly different policy. The robot would just try to achieve the top goal, degrading only to $\mathcal{D}1$ if it does not move after a walk action. Since, as discussed, $\mathcal{D}_2$'s scratch effect would be explained by $\mathcal{D}_3$ itself, line 18 would become `walk_e2_explained_by_l3` and line 19 would become `continue_l3`. The policy for this example is shown in Appendix C.

While the above demonstrates the possibility to solve MTPs using (the only) existing planning technology, running our, arguably simple, example takes more than 600 seconds to produce the 29 states first controller shown in the Appendix C, when using the off-the-shelf version of the planner. This clearly indicates the need for more and better dual-FOND implementations or the development of specialized optimizations for MTPs.

For example, we can use domain knowledge to tell the planner the number of controller states we want to start with (by default this number is 2), thus avoiding numerous calls to the SAT solver. One can approximate this number by taking into account (1) the number of domains; and (2) running the top level domain, which will provide a lower bound of the number of states required to solve the MTP. Another optimization involves the modification of the planner. As we are only allowing degradation and not upgrades, one can modify the SAT encoding to specify a number of controllers to be used per domain level, without allowing transitions from lower to upper levels. In preliminary tests we did, we experienced a speed-up of approx 60% by combining these optimizations.

## B.6 RELATED WORK

The work presented in this chapter is mostly related to existing work that aim to better handle execution failures and exceptions as well work that aim to provide richer goal specifications.

Fault Tolerant Planning (FTP) explores how to build plans that can cope with operations' failures. Notably, the work of Domshlak (2013) considers the FTP problem via reductions to planning. There, k-admissible plans are defined as those that guarantee to achieve the goal even if an operation happens to fail up to k times. Like ours, the approach reduces to (classical) planning via an intelligent compilation inspired in that of Bonet and Geffner (2011). In other words, the problem is reduced to bounded liveness. We do not impose such a restriction and our adaptive framework is actually orthogonal to theirs. In fact, it would be possible to accommodate k-admissible plans within our hierarchy, so that degradation is trigger only after some number of observed failures.

In Robust Planning (RP), the usual approach is to search for the best plan assuming the worst possible environment, e.g., (Buffet & Aberdeen, 2005). Typically, RP looks for subclasses of MDPs for which policies are guaranteed to eventually reach a goal state. Policies are normally computed as solutions to Stochastic Shortest Path problems. Our approach, instead, is based on a *qualitative* formalization of action and change, with no specification of probabilities, which can sometimes be not trivial, unfeasible, or simply uneconomical to obtain. More importantly, having a single problem description forces the engineer to represent a goal that may never be achieved in the actual environment. Indeed, the engineer may find it difficult, or impossible, to model degraded goals depending on the actual environment's dynamics; it would require the goals to somehow predicate on the probabilities associated to the assumed degraded behavior. We in turn provide an accessible formalism for modelling different levels of assumptions and goals. Also, our technique is not rooted in dynamic programming or optimization algorithms, but in planning ones.

In terms of representation formalisms, there has been efforts to provide more powerful goal specifications, such as EaGle language (Lago et al., 2002), Agent Planning Programs (Giacomo et al., 2016), and Hierarchical Goal Networks (HGN) (Shivashankar et al., 2012), but always assuming operation on a single model of the environment aiming for a single goal.

Our work also keeps some relationship with deliberative control architectures such as T-REX (McGann et al., 2008), and other similar planning-execution-monitoring architectures like Propice-Plan (Despouys & Ingrand, 1999), CPEF (Myers, 1999), or ARTUE (Klenk et al., 2013). Our approach differs in that those systems usually *replan* when goals (or perceived state) change, whereas our proposal aims to generate a policy that takes into account all the scenarios *prior* to the system's execution (as per model). As a result, we can provide guarantees of goal achievability and program termination (although it s more demanding computationally). Also, our framework puts more emphasis on formal semantics (based on PDDL and transition systems), modeling, and plan synthesis, whereas the mentioned agent architectures primarly focus on implemented platforms and execution.

As stated, our work is inspired by work in the area of Software Engineering (D'Ippolito et al., 2014). At a general level, our account is rooted in Knowledge Representation, and specifically, automated planning, which allows us to leverage on advanced representation formalisms (e.g., PDDL) as well as computational techniques for such representations. Still, as ours, their approach provides support for a tiered architecture with multiple behavioural models and goals. Unlike ours, though, their account is limited to a lineal hierarchy of models, so independent assumptions, as in our example, cannot be represented. More importantly, D'Ippolito et al. require solution (sub-)controllers of lower tiers to *simulate* those in upper tiers, and thus it would not handle our simple no-running example. Finally, being rooted in knowledge representation, we are able to exploit planning technology.

## B.7 SUMMARY

In this work, we proposed a novel multi-tier framework that integrates planning and acting with the aim of providing adaptive behavior to intelligent agents. Under such framework, the knowledge engineer has the opportunity to consider multiple levels of assumptions (domain models) and associated goals. This augment the autonomy of agents, since now they can reason with a hierarchy of domains and goals, rather than using a fixed model. The problem amounts to synthesize a meta-controller that, while running, is able to gracefully degrade its "level of service" (i.e., the goals to achieve) when the assumptions on the environment are not met.

We developed a compilation technique to construct such adaptive meta-controllers via dual-FOND planning. We note that plain FOND planning, where every action is assumed fair, would not work, as the agent may decide to keep trying an action to obtain one of the “failing" effects to achieve an “easier" lower-level goal (this artifact was already noted by Camacho and McIlraith (2016)).

# C MULTI-TIER PLANNING RESULTS

This Appendix contains: (1) the labeled PDDLs that serves as input to the multi-tier compilation presented in Appendix B; (2) the resulting compiled Dual-FOND PDDLs; and (3) the controllers obtained after solving the compiled tasks.

## C.1 LABELED PDDL MODELS

PDDL domain definition of the no-running example.

```
(define (domain no_running_1)
    (:requirements :typing)
    (:types Cell)
    (:constants c0 c1 c2 - Cell)
    (:predicates
        (at ?c - Cell)
        (adj ?o - Cell ?d - Cell)
        (broken)
        (scratch)
    )
    (:action walk
        :parameters (?o - Cell ?d - Cell)
        :precondition (and
            (at ?o)
            (adj ?o ?d)
            (not (broken)))
        :effect (oneof
            (d3
                (and (not (at ?o)) (at ?d))
            )
            (d2
                (and (not (at ?o)) (at ?d) (scratch))
            )
            (d1
                (and (scratch))
            )
        )
    )
    (:action run
        :parameters ()
        :precondition (and
            (at c2)
            (not (broken)))
        :effect (oneof
            (d3
                (and (not (at c2)) (at c0))
            )
            (d2
                (and (not (at c2)) (at c0) (scratch))
            )
            (d1
                (and (broken))
            )
        )
    )
)
```

PDDL problem definition of the no-running example.

```
(define (problem p1)
    (:domain no_running_1)
    (:init
        (at c2)
        (adj  c2 c1)
        (adj  c1 c0)
        (adj c0 c1)
        (adj c1 c2)
    )
    (:goal (oneof
        (d3 (and (at c0) (not (scratch)) (not (broken))))
        (d2 (and (at c0) (not (broken))))
        (d1 (and (at c2) (not (broken))))
        )
    )
)
```

## C.2 DUAL-FOND COMPILATION

Dual-FOND domain obtained through the MTP compilation for the no-running example.

```
(define (domain no_running_1)
(:requirements :typing)
(:types cell)
(:constants c0 c1 c2 - cell)
(:predicates
    (at ?c - cell) (adj ?o - cell ?d - cell) (broken) (scratch) (end) (act)
    (l_d3) (e_d3) (eff_d3_walk) (eff_d3_run) (l_d2) (e_d2) (eff_d2_walk)
    (eff_d2_run) (l_d1) (e_d1) (eff_d1_walk) (eff_d1_run) (u_walk) (u_run)
)
(:action continue_d3
    :parameters ()
    :precondition (and  (not (act)) (l_d3) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)) (or (e_d3)))
    :effect (and (act) (not (e_d3)) (not (e_d2)) (not (e_d1)))
)
(:action continue_d2
    :parameters ()
    :precondition (and  (not (act)) (l_d2) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)) (or (e_d3) (e_d2)))
    :effect (and (act) (not (e_d3)) (not (e_d2)) (not (e_d1)))
)
(:action continue_d1
    :parameters ()
    :precondition (and  (not (act)) (l_d1) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)) (or (e_d3) (e_d2) (e_d1)))
    :effect (and (act) (not (e_d3)) (not (e_d2)) (not (e_d1)))
)
(:action degrade_d3_d2
    :parameters ()
    :precondition (and  (not (act)) (l_d3) (e_d2) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)))
    :effect (and (act) (l_d2) (not (l_d3)) (not (e_d3)) (not (e_d2))
    (not (e_d1)))
)
```

```
(:action degrade_d3_d1
    :parameters ()
    :precondition (and (not (act)) (l_d3) (e_d1) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)))
    :effect (and (act) (l_d1) (not (l_d3)) (not (e_d3)) (not (e_d2))
    (not (e_d1)))
)
(:action degrade_d2_d1
    :parameters ()
    :precondition (and (not (act)) (l_d2) (e_d1) (not (eff_d3_walk))
    (not (eff_d3_run)) (not (eff_d2_walk)) (not (eff_d2_run))
    (not (eff_d1_walk)) (not (eff_d1_run)))
    :effect (and (act) (l_d1) (not (l_d2)) (not (e_d3)) (not (e_d2))
    (not (e_d1)))
)
(:action walk_unfair_
    :parameters ()
    :precondition (and (act) (u_walk))
    :effect (oneof (and (eff_d3_walk) (not (act))) (and (eff_d2_walk)
    (not (act))) (and (eff_d1_walk) (not (act))))
)
(:action walk_d3
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (l_d3) (act)
    (not (u_walk)) (not (u_run)))
    :effect (oneof (and (not (at ?o)) (at ?d)) (u_walk))
)
(:action walk_d2
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (l_d2) (act)
    (not (u_walk)) (not (u_run)))
    :effect (oneof (and (not (at ?o)) (at ?d)) (and (not (at ?o)) (at ?d)
    (scratch)) (u_walk))
)
(:action walk_d1
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (l_d1) (act)
    (not (u_walk)) (not (u_run)))
    :effect (oneof (and (not (at ?o)) (at ?d)) (and (not (at ?o)) (at ?d)
    (scratch)) (scratch))
)
(:action walk_eff_d3_explained_by_d3
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (eff_d3_walk))
    :effect (and (not (at ?o)) (at ?d) (e_d3) (not (eff_d3_walk))
    (not (act)) (not (u_walk)))
)
(:action walk_eff_d2_explained_by_d3
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (eff_d2_walk)
    (scratch))
    :effect (and (not (at ?o)) (at ?d) (scratch) (e_d3) (not (eff_d2_walk))
    (not (act)) (not (u_walk)))
)
(:action walk_eff_d2_explained_by_d2
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (eff_d2_walk)
    (or (not (scratch))))
    :effect (and (not (at ?o)) (at ?d) (scratch) (e_d2) (not (eff_d2_walk))
    (not (act)) (not (u_walk)))
)
(:action walk_eff_d1_explained_by_d3
    :parameters (?o - cell ?d - cell)
    :precondition (and (at ?o) (adj ?o ?d) (not (broken)) (eff_d1_walk)
    (scratch) (not (at ?o)) (at ?d))
```

```
        :effect (and (scratch) (e_d3) (not (eff_d1_walk)) (not (act))
        (not (u_walk)))
    )
    (:action walk_eff_d1_explained_by_d2
        :parameters (?o - cell ?d - cell)
        :precondition (and  (at ?o) (adj ?o ?d) (not (broken)) (eff_d1_walk)
        (not (at ?o)) (at ?d) (or (not (scratch)) (at ?o) (not (at ?d))))
        :effect (and (scratch) (e_d2) (not (eff_d1_walk)) (not (act))
        (not (u_walk)))
    )
    (:action walk_eff_d1_explained_by_d1
        :parameters (?o - cell ?d - cell)
        :precondition (and  (at ?o) (adj ?o ?d) (not (broken)) (eff_d1_walk)
        (or (not (scratch)) (at ?o) (not (at ?d))) (or (at ?o) (not (at ?d))))
        :effect (and (scratch) (e_d1) (not (eff_d1_walk)) (not (act))
        (not (u_walk)))
    )
    (:action run_unfair_
        :parameters ()
        :precondition (and  (act) (u_run))
        :effect (oneof (and (eff_d3_run) (not (act))) (and (eff_d2_run)
        (not (act))) (and (eff_d1_run) (not (act))))
    )
    (:action run_d3
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (l_d3) (act) (not (u_walk))
        (not (u_run)))
        :effect (oneof (and (not (at c2)) (at c0)) (u_run))
    )
    (:action run_d2
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (l_d2) (act) (not (u_walk))
        (not (u_run)))
        :effect (oneof (and (not (at c2)) (at c0)) (and (not (at c2)) (at c0)
        (scratch)) (u_run))
    )
    (:action run_d1
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (l_d1) (act) (not (u_walk))
        (not (u_run)))
        :effect (oneof (and (not (at c2)) (at c0)) (and (not (at c2)) (at c0)
        (scratch)) (broken))
    )
    (:action run_eff_d3_explained_by_d3
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (eff_d3_run))
        :effect (and (not (at c2)) (at c0) (e_d3) (not (eff_d3_run)) (not (act))
        (not (u_run)))
    )
    (:action run_eff_d2_explained_by_d3
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (eff_d2_run) (scratch))
        :effect (and (not (at c2)) (at c0) (scratch) (e_d3) (not (eff_d2_run))
        (not (act)) (not (u_run)))
    )
    (:action run_eff_d2_explained_by_d2
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (eff_d2_run)
        (or (not (scratch))))
        :effect (and (not (at c2)) (at c0) (scratch) (e_d2) (not (eff_d2_run))
        (not (act)) (not (u_run)))
    )
    (:action run_eff_d1_explained_by_d3
        :parameters ()
        :precondition (and  (at c2) (not (broken)) (eff_d1_run) (broken)
        (not (at c2)) (at c0))
    :effect (and (broken) (e_d3) (not (eff_d1_run)) (not (act))
    (not (u_run)))
)
(:action run_eff_d1_explained_by_d2
    :parameters ()
    :precondition (and  (at c2) (not (broken)) (eff_d1_run) (broken)
    (not (at c2)) (at c0) (scratch) (or (not (broken)) (at c2)
    (not (at c0))))
    :effect (and (broken) (e_d2) (not (eff_d1_run)) (not (act))
    (not (u_run)))
)
(:action run_eff_d1_explained_by_d1
    :parameters ()
    :precondition (and  (at c2) (not (broken)) (eff_d1_run)
    (or (not (broken)) (at c2) (not (at c0)))
    (or (not (broken)) (at c2) (not (at c0)) (not (scratch))))
    :effect (and (broken) (e_d1) (not (eff_d1_run)) (not (act))
    (not (u_run)))
)
(:action check_goal_d3
    :parameters ()
    :precondition (and  (at c0) (not (scratch)) (not (broken)) (l_d3) (act))
    :effect (end)
)
(:action check_goal_d2
    :parameters ()
    :precondition (and  (at c0) (not (broken)) (l_d2) (act))
    :effect (end)
)
(:action check_goal_d1
    :parameters ()
    :precondition (and  (at c2) (not (broken)) (l_d1) (act))
    :effect (end)
)
)
```

Dual-FOND problem obtained through the MTP compilation for the no-running example.

```
(define (problem p1)
 (:domain no_running_1)
 (:init
  (at c2)
  (adj c2 c1)
  (adj c1 c0)
  (adj c0 c1)
  (adj c1 c2)
  (act)
  (l_d3))
 (:goal (and
       (end))
    )
)
```

## C.3 SOLUTION CONTROLLERS

Graphical representation of the policy obtained by running Geffner and Geffner's (2018) in the above mentioned Dual-FOND planning task.

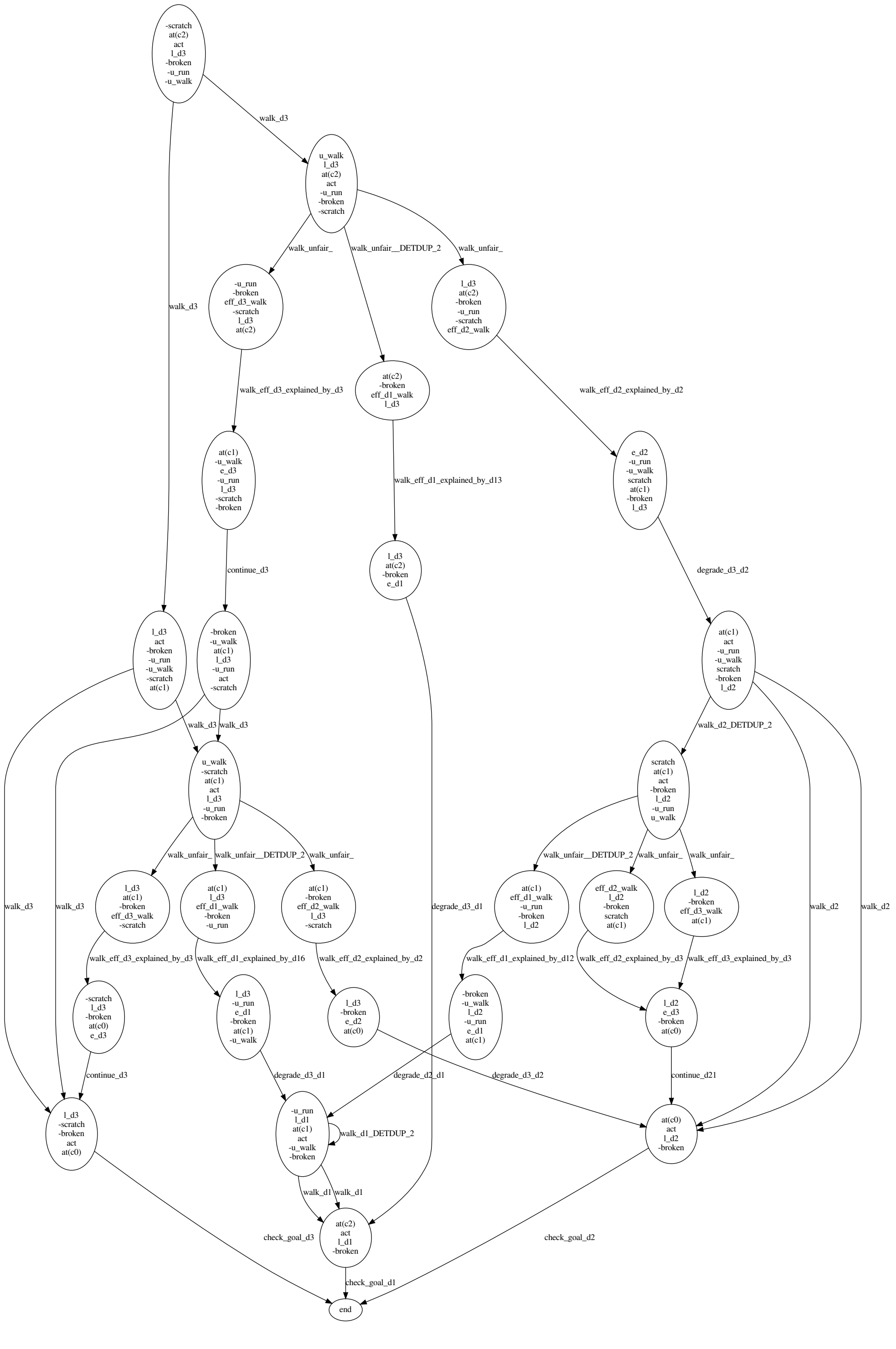

-scratch at(c2) act l_d3 -broken -u_run -u_walk
walk_d3
u_walk l_d3 at(c2) act -u_run -broken -scratch
walk_d3
walk_unfair_
walk_unfair__DETDUP_2
walk_unfair_
-u_run -broken eff_d3_walk -scratch l_d3 at(c2)
l_d3 at(c2) -broken -u_run -scratch eff_d2_walk
at(c2) -broken eff_d1_walk l_d3
walk_eff_d3_explained_by_d3
walk_eff_d2_explained_by_d2
at(c1) -u_walk e_d3 -u_run l_d3 -scratch -broken
walk_eff_d1_explained_by_d13
e_d2 -u_run -u_walk scratch at(c1) -broken l_d3
continue_d3
l_d3 at(c2) -broken e_d1
degrade_d3_d2
l_d3 act -broken -u_run -u_walk -scratch at(c1)
-broken -u_walk at(c1) l_d3 -u_run act -scratch
at(c1) act -u_run -u_walk scratch -broken l_d2
walk_d3
walk_d3
walk_d2_DETDUP_2
u_walk -scratch at(c1) act l_d3 -u_run -broken
scratch at(c1) act -broken l_d2 -u_run u_walk
walk_unfair_
walk_unfair__DETDUP_2
walk_unfair_
walk_unfair__DETDUP_2
walk_unfair_
walk_unfair_
walk_d3
walk_d3
l_d3 at(c1) -broken eff_d3_walk -scratch
at(c1) l_d3 eff_d1_walk -broken -u_run
at(c1) -broken eff_d2_walk l_d3 -scratch
degrade_d3_d1
at(c1) eff_d1_walk -u_run -broken l_d2
eff_d2_walk l_d2 -broken scratch at(c1)
l_d2 -broken eff_d3_walk at(c1)
walk_d2
walk_d2
walk_eff_d3_explained_by_d3
walk_eff_d1_explained_by_d16
walk_eff_d2_explained_by_d2
walk_eff_d1_explained_by_d12
walk_eff_d2_explained_by_d3
walk_eff_d3_explained_by_d3
-scratch l_d3 -broken at(c0) e_d3
l_d3 -u_run e_d1 -broken at(c1) -u_walk
l_d3 -broken e_d2 at(c0)
-broken -u_walk l_d2 -u_run e_d1 at(c1)
l_d2 e_d3 -broken at(c0)
continue_d3
degrade_d3_d1
degrade_d2_d1
degrade_d3_d2
continue_d21
l_d3 -scratch -broken act at(c0)
-u_run l_d1 at(c1) act -u_walk -broken
walk_d1_DETDUP_2
at(c0) act l_d2 -broken
walk_d1
walk_d1
check_goal_d3
at(c2) act l_d1 -broken
check_goal_d2
check_goal_d1
end

Figure C.1: Graphical representation of the policy obtained for the non-running scenario.

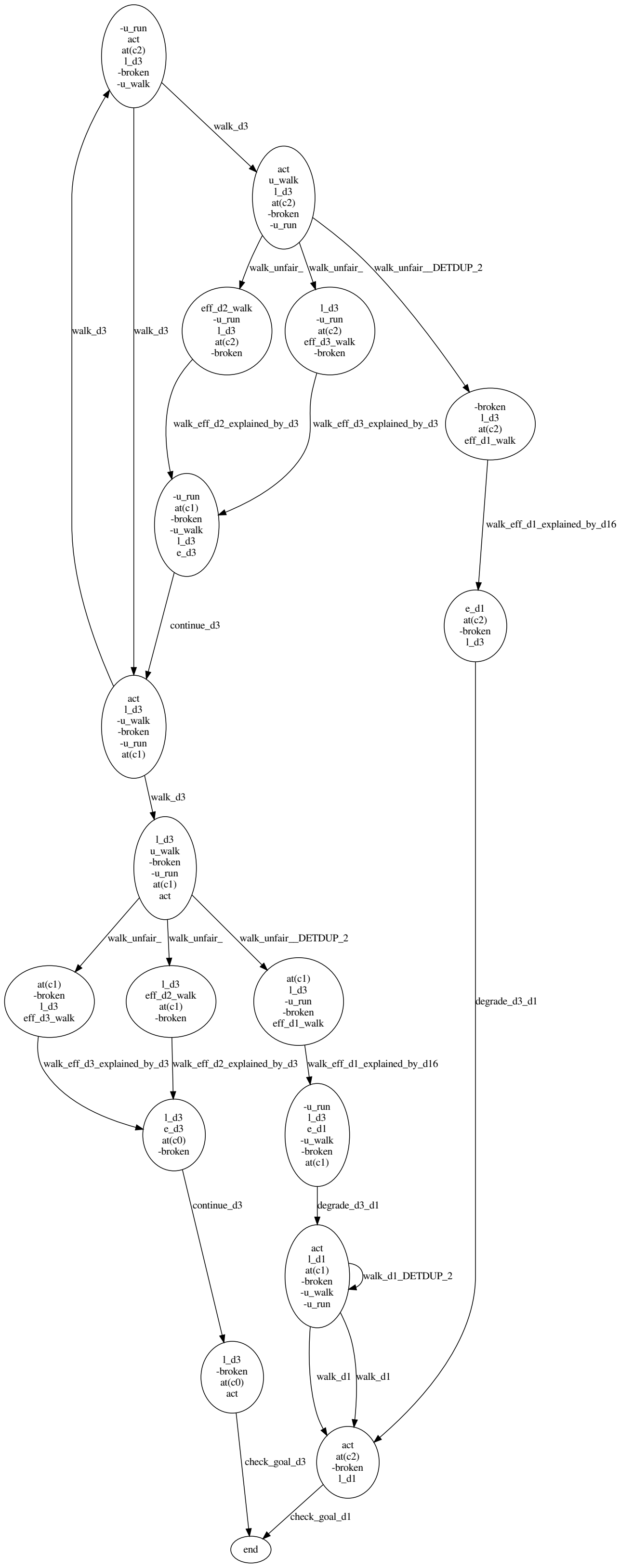


Figure C.2: In turn, this is the controller for the variant of the scenario where (1) the robot is initially scratched; and (2) the non-scratched requirement from $\mathcal{D}_3$'s goal is dropped.

# D

# COUNTERPLANNING DOMAINS

This Appendix contains the domains used in Chapters 5 and 6.

## D.1 POLICE CONTROL

**SEEK**

```
(define (domain terrorist)
(:requirements :typing :action-costs)
(:types
cell station - objects
terrorist police - movable
)
(:predicates
(at ?x1 - movable ?x2 - cell)
(in ?x1 - cell ?x2 - station)
(connected ?x1 - cell ?x2 - cell)
(free ?x1 - cell)
(phone-booth ?x1 - cell)
(call-made)
(phone-available)
)
(:functions
(total-cost)
)

(:action move
:parameters (?x1 - terrorist ?x2 - cell ?x3 - cell)
:precondition (and
(at ?x1 ?x2)
(free ?x3)
(connected ?x2 ?x3)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(free ?x2)
(not (free ?x3))
(increase (total-cost) 1)
)
)

(:action make-call
:parameters (?x1 - terrorist ?x2 - cell)
:precondition (and
(at ?x1 ?x2)
(phone-available)
(phone-booth ?x2)
)
:effect (and
(call-made)
(not (phone-available))
(increase (total-cost) 1)
)
)
)
```

**PREV**

```
(define (domain police)
(:requirements :typing :action—costs)
(:types
cell station — objects
terrorist police — movable
)
(:predicates
(at ?x1 — movable ?x2 — cell)
(in ?x1 — cell ?x2 — station)
(connected ?x1 — cell ?x2 — cell)
(free ?x1 — cell)
(phone—booth ?x1 — cell)
(call—made)
(phone—available)
(office—at ?x1 — cell)
)
(:functions
(total—cost)
)

(:action move
:parameters (?x1 — police ?x2 — cell ?x3 — cell)
:precondition (and
(at ?x1 ?x2)
(free ?x3)
(connected ?x2 ?x3)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(free ?x2)
(not (free ?x3))
(increase (total—cost) 1)
)
)

(:action tap—phone—booths
:parameters (?x1 — police ?x2 — cell)
:precondition (and
(at ?x1 ?x2)
(office—at ?x2)
)
:effect (and
(not (phone—available))
(increase (total—cost) 1)
)
)
)
```

## D.2 PAINTED BLOCKS-WORDS

**SEEK**

```
(define (domain blocks)
(:requirements :strips)
(:types
block room — object
)
(:predicates
(on ?x1 — block ?x2 — block)
(ontable ?x1 — block)
(clear ?x1 — block)
(handempty)
```

```
(holding ?x1 - block)
(painted ?x1 - block)
(have-painting ?x1 - block)
(at-painting ?x1 - block ?x2 - room)
(connected ?x1 - room ?x2 - room)
(at-preventing ?x1 - room)
(blocks-room ?x1 - room)
)

(:functions
(total-cost)
)

(:action pick-up
:parameters (?x1 - block)
:precondition (and
(clear ?x1)
(ontable ?x1)
(handempty)
)
:effect (and
(not (ontable ?x1))
(not (clear ?x1))
(not (handempty))
(holding ?x1)
(increase (total-cost) 1)
)
)

(:action put-down
:parameters (?x1 - block)
:precondition (and
(holding ?x1)
)
:effect (and
(not (holding ?x1))
(clear ?x1)
(handempty)
(ontable ?x1)
(increase (total-cost) 1)
)
)

(:action stack
:parameters (?x1 - block ?x2 - block)
:precondition (and
(holding ?x1)
(clear ?x2)
)
:effect (and
(not (holding ?x1))
(not (clear ?x2))
(clear ?x1)
(handempty)
(on ?x1 ?x2)
(increase (total-cost) 1)
)
)

(:action unstack
:parameters (?x1 - block ?x2 - block)
:precondition (and
(on ?x1 ?x2)
(clear ?x1)
(handempty)
)
```

```
:effect (and
(holding ?x1)
(clear ?x2)
(not (clear ?x1))
(not (handempty))
(not (on ?x1 ?x2))
(increase (total-cost) 1)
)
)
)
```

PREV

```
(define (domain counter-blocks)
(:requirements :strips)
(:types
block room - object
)
(:predicates
(on ?x1 - block ?x2 - block)
(ontable ?x1 - block)
(clear ?x1 - block)
(handempty)
(holding ?x1 - block)
(have-painting ?x1 - block)
(at-painting ?x1 - block ?x2 - room)
(at-preventing ?x1 - room)
(connected ?x1 - room ?x2 - room)
(blocks-room ?x1 - room)
)

(:functions
(total-cost)
)

(:action paint
:parameters (?x1 - block ?x2 - room)
:precondition (and
(have-painting ?x1)
(at-preventing ?x2)
(blocks-room ?x2)
)
:effect (and
(not (clear ?x1))
(increase (total-cost) 1)
)
)

(:action pick-paint
:parameters (?x1 - block ?x2 - room)
:precondition (and
(at-painting ?x1 ?x2)
(at-preventing ?x2)
)
:effect (and
(not (at-painting ?x1 ?x2))
(have-painting ?x1)
(increase (total-cost) 1)
)
)

(:action move
:parameters (?x1 - room ?x2 - room)
:precondition (and
(at-preventing ?x1)
(connected ?x1 ?x2)
)
:effect (and
(not (at-preventing ?x1))
(at-preventing ?x2)
(increase (total-cost) 1)
)
)
)
```

## D.3 COUNTER LOGISTICS

### SEEK

```
(define (domain logistics)
(:requirements :strips :typing)
(:types
city place physobj - object
package bomb vehicle - physobj
truck airplane enemytruck - vehicle
airport location - place
)

(:predicates
(in-city ?x1 - place ?x2 - city)
(at ?x1 - physobj ?x2 - place)
(in ?x1 - package ?x2 - vehicle)
(in-bomb ?x1 - bomb ?x2 - vehicle)
)

(:functions
(total-cost)
)

(:action LOAD-TRUCK
:parameters (?x1 - package ?x2 - truck ?x3 - place)
:precondition (and
(at ?x2 ?x3)
(at ?x1 ?x3)
)
:effect (and
(not (at ?x1 ?x3))
(in ?x1 ?x2)
(increase (total-cost) 1)
)
)

(:action LOAD-AIRPLANE
:parameters (?x1 - package ?x2 - airplane ?x3 - place)
:precondition (and
(at ?x1 ?x3)
(at ?x2 ?x3)
)
:effect (and
(not (at ?x1 ?x3))
(in ?x1 ?x2)
(increase (total-cost) 1)
)
)

(:action UNLOAD-TRUCK
:parameters (?x1 - package ?x2 - truck ?x3 - place)
:precondition (and
(at ?x2 ?x3)
(in ?x1 ?x2)
)
:effect (and
(not (in ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action UNLOAD-AIRPLANE
:parameters (?x1 - package ?x2 - airplane ?x3 - place)
:precondition (and
(in ?x1 ?x2)
(at ?x2 ?x3)
)
:effect (and
(not (in ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action DRIVE-TRUCK
:parameters (?x1 - truck ?x2 - place ?x3 - place ?x4 - city)
:precondition (and
(not (= ?x2 ?x3))
(at ?x1 ?x2)
(in-city ?x2 ?x4)
(in-city ?x3 ?x4)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action FLY-AIRPLANE
:parameters (?x1 - airplane ?x2 - airport ?x3 - airport)
:precondition (and
(not (= ?x2 ?x3))
(at ?x1 ?x2)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)
)
```

**PREV**

```
(define (domain counter-logistics)
(:requirements :strips :typing)
(:types
city place physobj - object
package bomb vehicle - physobj
truck airplane enemytruck - vehicle
airport location - place
)

(:predicates
(in-city ?x1 - place ?x2 - city)
(at ?x1 - physobj ?x2 - place)
(in ?x1 - package ?x2 - vehicle)
(in-bomb ?x1 - bomb ?x2 - vehicle)
)
```

```
(:functions
(total-cost)
)

(:action LOAD-ENEMY-TRUCK
:parameters (?x1 - bomb ?x2 - enemytruck ?x3 - place)
:precondition (and
(at ?x2 ?x3)
(at ?x1 ?x3)
)
:effect (and
(not (at ?x1 ?x3))
(in-bomb ?x1 ?x2)
(increase (total-cost) 1)
)
)

(:action LOAD-ENEMY-AIRPLANE
:parameters (?x1 - bomb ?x2 - enemytruck ?x3 - place)
:precondition (and
(at ?x1 ?x3)
(at ?x2 ?x3)
)
:effect (and
(not (at ?x1 ?x3))
(in-bomb ?x1 ?x2)
(increase (total-cost) 1)
)
)

(:action UNLOAD-ENEMY-TRUCK
:parameters (?x1 - bomb ?x2 - enemytruck ?x3 - place)
:precondition (and
(at ?x2 ?x3)
(in-bomb ?x1 ?x2)
)
:effect (and
(not (in-bomb ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action UNLOAD-ENEMY-AIRPLANE
:parameters (?x1 - bomb ?x2 - enemytruck ?x3 - place)
:precondition (and
(in-bomb ?x1 ?x2)
(at ?x2 ?x3)
)
:effect (and
(not (in-bomb ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action DRIVE-ENEMY-TRUCK
:parameters (?x1 - enemytruck ?x2 - place ?x3 - place ?x4 - city)
:precondition (and
(not (= ?x2 ?x3))
(at ?x1 ?x2)
(in-city ?x2 ?x4)
(in-city ?x3 ?x4)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action FLY-ENEMY-AIRPLANE
:parameters (?x1 - enemytruck ?x2 - airport ?x3 - airport)
:precondition (and
(not (= ?x2 ?x3))
(at ?x1 ?x2)
)
:effect (and
(not (at ?x1 ?x2))
(at ?x1 ?x3)
(increase (total-cost) 1)
)
)

(:action BREAK-UP-TRUCK
:parameters (?x1 - truck ?x2 - place ?x3 - bomb)
:precondition (and
(at ?x3 ?x2)
(at ?x1 ?x2)
)
:effect (and
(not (at ?x1 ?x2))
(increase (total-cost) 1)
)
)

(:action BREAK-UP-PLANE
:parameters (?x1 - airplane ?x2 - place ?x3 - bomb)
:precondition (and
(at ?x3 ?x2)
(at ?x1 ?x2)
)
:effect (and
(not (at ?x1 ?x2))
(increase (total-cost) 1)
)
)
)
```

## D.4 ROVERS & MARTIANS

### SEEK

```
(define (domain Rover)
(:requirements :typing)
(:types
rover waypoint store camera mode lander objective
)

(:predicates
(at ?x - rover ?y - waypoint)
(at_lander ?x - lander ?y - waypoint)
(can_traverse ?r - rover ?x - waypoint ?y - waypoint)
(equipped_for_soil_analysis ?r - rover)
(equipped_for_rock_analysis ?r - rover)
(equipped_for_imaging ?r - rover)
(empty ?s - store)
(have_rock_analysis ?r - rover ?w - waypoint)
(have_soil_analysis ?r - rover ?w - waypoint)
(full ?s - store)
```

```
(calibrated ?c - camera ?r - rover)
(supports ?c - camera ?m - mode)
(available ?r - rover)
(visible ?w - waypoint ?p - waypoint)
(have_image ?r - rover ?o - objective ?m - mode)
(communicated_soil_data ?w - waypoint)
(communicated_rock_data ?w - waypoint)
(communicated_image_data ?o - objective ?m - mode)
(at_soil_sample ?w - waypoint)
(at_rock_sample ?w - waypoint)
(visible_from ?o - objective ?w - waypoint)
(store_of ?s - store ?r - rover)
(calibration_target ?i - camera ?o - objective)
(on_board ?i - camera ?r - rover)
(channel_free ?l - lander)
(at-martian ?w - waypoint)
)

(:functions
(total-cost)
)


(:action navigate
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and
(can_traverse ?x ?y ?z)
(available ?x)
(at ?x ?y)
(visible ?y ?z)
)
:effect (and
(not (at ?x ?y))
(at ?x ?z)
(increase (total-cost) 1)
)
)

(:action sample_soil
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and
(at ?x ?p)
(at_soil_sample ?p)
(equipped_for_soil_analysis ?x)
(store_of ?s ?x)
(empty ?s)
)
:effect (and
(not (empty ?s))
(full ?s)
(have_soil_analysis ?x ?p)
(not (at_soil_sample ?p))
(increase (total-cost) 1)
)
)

(:action sample_rock
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and
(at ?x ?p)
(at_rock_sample ?p)
(equipped_for_rock_analysis ?x)
(store_of ?s ?x)
(empty ?s)
)
:effect (and
(not (empty ?s))
(full ?s)
(have_rock_analysis ?x ?p)
(not (at_rock_sample ?p))
(increase (total-cost) 1)
)
)

(:action drop
:parameters (?x - rover ?y - store)
:precondition (and
(store_of ?y ?x)
(full ?y)
)
:effect (and
(not (full ?y))
(empty ?y)
(increase (total-cost) 1)
)
)

(:action calibrate
:parameters (?r - rover ?i - camera ?t - objective ?w - waypoint)
:precondition (and
(equipped_for_imaging ?r)
(calibration_target ?i ?t)
(at ?r ?w)
(visible_from ?t ?w)
(on_board ?i ?r)
)
:effect (and
(calibrated ?i ?r)
(increase (total-cost) 1)
)
)




(:action take_image
:parameters (?r - rover ?p - waypoint ?o - objective ?i - camera ?m - mode)
:precondition (and
(calibrated ?i ?r)
(on_board ?i ?r)
(equipped_for_imaging ?r)
(supports ?i ?m)
(visible_from ?o ?p)
(at ?r ?p)
)
:effect (and
(have_image ?r ?o ?m)
(not (calibrated ?i ?r))
(increase (total-cost) 1)
)
)


(:action communicate_soil_data
:parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
:precondition (and
(at ?r ?x)
(at_lander ?l ?y)
(have_soil_analysis ?r ?p)
(visible ?x ?y)
(available ?r)
(channel_free ?l)
```

```
)
:effect (and
(not (available ?r))
(not (channel_free ?l))
(channel_free ?l)
(communicated_soil_data ?p)
(available ?r)
(increase (total-cost) 1)
)
)

(:action communicate_rock_data
:parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y - waypoint)
:precondition (and
(at ?r ?x)
(at_lander ?l ?y)
(have_rock_analysis ?r ?p)
(visible ?x ?y)
(available ?r)
(channel_free ?l)
)
 :effect (and
(not (available ?r))
(not (channel_free ?l))
(channel_free ?l)
(communicated_rock_data ?p)
(available ?r)
(increase (total-cost) 1)
)
)


(:action communicate_image_data
:parameters (?r - rover ?l - lander ?o - objective ?m - mode ?x - waypoint ?y - waypoint)
:precondition (and
(at ?r ?x)
(at_lander ?l ?y)
(have_image ?r ?o ?m)
(visible ?x ?y)
(available ?r)
(channel_free ?l)
)
 :effect (and
(not (available ?r))
(not (channel_free ?l))
(channel_free ?l)
(communicated_image_data ?o ?m)
(available ?r)
(increase (total-cost) 1)
)
)
)
```

**PREV**

```
(define (domain Rover)
(:requirements :typing)
(:types
rover waypoint store camera mode lander objective
)

(:predicates
(at ?x - rover ?y - waypoint)
(at_lander ?x - lander ?y - waypoint)
(can_traverse ?r - rover ?x - waypoint ?y - waypoint)
(equipped_for_soil_analysis ?r - rover)
(equipped_for_rock_analysis ?r - rover)
```

```
(equipped_for_imaging ?r - rover)
(empty ?s - store)
(have_rock_analysis ?r - rover ?w - waypoint)
(have_soil_analysis ?r - rover ?w - waypoint)
(full ?s - store)
(calibrated ?c - camera ?r - rover)
(supports ?c - camera ?m - mode)
(available ?r - rover)
(visible ?w - waypoint ?p - waypoint)
(have_image ?r - rover ?o - objective ?m - mode)
(communicated_soil_data ?w - waypoint)
(communicated_rock_data ?w - waypoint)
(communicated_image_data ?o - objective ?m - mode)
(at_soil_sample ?w - waypoint)
(at_rock_sample ?w - waypoint)
(visible_from ?o - objective ?w - waypoint)
(store_of ?s - store ?r - rover)
(calibration_target ?i - camera ?o - objective)
(on_board ?i - camera ?r - rover)
(channel_free ?l - lander)
(at-martian ?w - waypoint)
)

(:functions
(total-cost)
)


(:action navigate
:parameters (?y - waypoint ?z - waypoint)
:precondition (and
(at-martian ?y)
(visible ?y ?z)
)
:effect (and
(not (at-martian ?y))
(at-martian ?z)
(increase (total-cost) 1)
)
)

(:action break-channel
:parameters (?l - lander ?p - waypoint)
:precondition (and
(at ?l ?p)
(channel_free ?l)
(at-martian ?p)
)
:effect (and
(not (channel_free ?l))
(increase (total-cost) 1)
)
)

(:action sample_rock
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and
(at ?x ?p)
(at_rock_sample ?p)
(equipped_for_rock_analysis ?x)
(store_of ?s ?x)
(empty ?s)
)
:effect (and
(not (empty ?s))
(full ?s)
(have_rock_analysis ?x ?p)
(not (at_rock_sample ?p))
(increase (total-cost) 1)
)
)

(:action steal_sample
:parameters (?x - waypoint ?y - rover ?z - store)
:precondition (and
(at-martian ?x)
(at ?y ?x)
(full ?z)
)
:effect (and
(not (full ?z))
(increase (total-cost) 1)
)
)
)
```

## D.5 STARCRAFT

### SEEK

```
(define (domain starcraft-seeking)
(:requirements :typing)
(:types
square location - object
)
(:predicates
(connected ?x1 - square ?x2 - square)
(at ?x1 - square)
(at-enemy ?x1 - square)
(at-target ?x1 - square)
(empty ?x1)
(at-vespene ?x1 - square)
(at-mineral ?x1 - square)
(at-refinery ?x1 - square)
(at-barracks ?x1 - square)
(at-command-center ?x1 - square)
(at-refinery-enemy ?x1 - square)
(at-barracks-enemy ?x1 - square)
(at-command-center-enemy ?x1 - square)
(refinery-built)
(barracks-built)
(command-center-built)
(refinery-built-enemy)
(barracks-built-enemy)
(command-center-built-enemy)
(carrying-mineral)
(carrying-vespene)
(carrying-mineral-enemy)
(carrying-vespene-enemy)
(have-mineral)
(have-vespene)
(have-mineral-enemy)
(have-vespene-enemy)
(alive-enemy)
(alive)
(under-attack-command-center)
(under-attack-command-center-enemy)
(under-attack-refinery)
(under-attack-refinery-enemy)
(under-attack-barracks)
(under-attack-barracks-enemy)
(marine-trained)
(marine-trained-enemy)
)

(:action build-barracks-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(have-mineral-enemy)
(empty ?x1)
(at-enemy ?x2)
(connected ?x1 ?x2)
)
:effect (and
(not (empty ?x1))
(at-barracks-enemy ?x1)
(barracks-built-enemy)
(increase (total-cost) 128)
)
)

(:action build-refinery-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(have-mineral-enemy)
(at-vespene ?x1)
(at-enemy ?x2)
(connected ?x1 ?x2)
)
:effect (and
(at-refinery-enemy ?x1)
(not (at-vespene ?x1))
(refinery-built-enemy)
(increase (total-cost) 60)
)
)

(:action train-marine-enemy
:parameters ()
:precondition (and
(have-vespene-enemy)
(barracks-built-enemy)
(not (marine-trained-enemy))
)
:effect (and
(marine-trained-enemy)
)
)

(:action gather-vespene-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-refinery ?x2)
(not (carrying-vespene-enemy))
(not (carrying-mineral-enemy))
)
:effect (and
(carrying-vespene-enemy)
(increase (total-cost) 4)
)
)
```

```
(:action gather-mineral-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-mineral ?x2)
(not (carrying-vespene-enemy))
(not (carrying-mineral-enemy))
)
:effect (and
(carrying-mineral-enemy)
(increase (total-cost) 9)
)
)

(:action store-vespene-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-command-center ?x2)
(connected ?x1 ?x2)
(carrying-vespene-enemy)
)
:effect (and
(have-vespene-enemy)
(not (carrying-vespene-enemy))
(increase (total-cost) 1)
)
)

(:action store-mineral-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-command-center ?x2)
(connected ?x1 ?x2)
(carrying-mineral-enemy)
)
:effect (and
(have-mineral-enemy)
(not (carrying-mineral-enemy))
(increase (total-cost) 1)
)
)

(:action attack-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at ?x2)
(connected ?x1 ?x2)

):effect (and
(not (alive))
)
)

(:action attack-barracks-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-barracks ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-barracks)
(not (barracks-built))
(increase (total-cost) 1)
)
)

(:action attack-refinery-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-refinery ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-refinery)
(not (refinery-built))
(increase (total-cost) 1)
)
)

(:action attack-command-center-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(at-command-center ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-command-center)
(not (command-center-built))
(increase (total-cost) 1)
)
)

(:action move-enemy
:parameters (?x1 - square ?x2 - square)
:precondition (and
(alive-enemy)
(at-enemy ?x1)
(connected ?x1 ?x2)
(empty ?x2)
)
:effect (and
(not (at-enemy ?x1))
(at-enemy ?x2)
(not (empty ?x2))
(empty ?x1)
(increase (total-cost) 1)
)
)
)
```

**PREV**

```
(define (domain starcraft-preventing)
(:requirements :typing)
(:types
square location - object
)
(:predicates
(connected ?x1 - square ?x2 - square)
(at ?x1 - square)
```

```
(at-enemy ?x1 - square)
(at-target ?x1 - square)
(empty ?x1)
(at-vespene ?x1 - square)
(at-mineral ?x1 - square)
(at-refinery ?x1 - square)
(at-barracks ?x1 - square)
(at-command-center ?x1 - square)
(at-refinery-enemy ?x1 - square)
(at-barracks-enemy ?x1 - square)
(at-command-center-enemy ?x1 - square)
(refinery-built)
(barracks-built)
(command-center-built)
(refinery-built-enemy)
(barracks-built-enemy)
(command-center-built-enemy)
(carrying-mineral)
(carrying-vespene)
(carrying-mineral-enemy)
(carrying-vespene-enemy)
(have-mineral)
(have-vespene)
(have-mineral-enemy)
(have-vespene-enemy)
(alive-enemy)
(alive)
(under-attack-command-center)
(under-attack-command-center-enemy)
(under-attack-refinery)
(under-attack-refinery-enemy)
(under-attack-barracks)
(under-attack-barracks-enemy)
(marine-trained)
(marine-trained-enemy)
)

(:action attack
:parameters (?x1 - square)
:precondition (and
(at ?x1)
)
:effect (and
(not (alive-enemy))
(increase (total-cost) 12)
)
)

(:action attack-barracks
:parameters (?x1 - square ?x2 - square)
:precondition (and
(at ?x1)
(at-barracks-enemy ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-barracks-enemy)
(not (barracks-built-enemy))
(increase (total-cost) 516)
)
)

(:action attack-refinery
:parameters (?x1 - square ?x2 - square)
:precondition (and
(at ?x1)
(at-refinery-enemy ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-refinery-enemy)
(not (refinery-built-enemy))
(increase (total-cost) 401)
)
)

(:action attack-command-center
:parameters (?x1 - square ?x2 - square)
:precondition (and
(at ?x1)
(at-command-center-enemy ?x2)
(connected ?x1 ?x2)
)
:effect (and
(under-attack-command-center-enemy)
(not (command-center-built-enemy))
(increase (total-cost) 810)
)
)

(:action move
:parameters (?x1 - square ?x2 - square)
:precondition (and
(at ?x1)
(connected ?x1 ?x2)
(empty ?x2)
)
:effect (and
(not (at ?x1))
(at ?x2)
(not (empty ?x2))
(empty ?x1)
(increase (total-cost) 1)
)
)
)
```

# BIBLIOGRAPHY

Aha, D. W. (2018). Goal Reasoning: Foundations, Emerging Applications, and Prospects. *AI Magazine*, *39*(2), 3–24. https://www.aaai.org/ojs/index.php/aimagazine/article/view/2800 (cit. on pp. 14, 49, 111)

Aha, D. W., Molineaux, M., & Ponsen, M. J. V. (2005). Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game, In *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Proceedings*, Springer. https://doi.org/10.1007/11536406_4. (Cit. on p. 111)

Ai-Chang, M., Bresina, J. L., Charest, L., Chase, A., jung Hsu, J. C., Jónsson, A. K., Kanefsky, B., Morris, P. H., Rajan, K., Yglesias, J., Chafin, B. G., Dias, W. C., & Maldague, P. F. (2004). MAPGEN: Mixed-Initiative Planning and Scheduling for the Mars Exploration Rover Mission. *IEEE Intell. Syst.*, *19*(1), 8–12. https://doi.org/10.1109/MIS.2004.1265878 (cit. on p. 53)

Albrecht, D. W., Zukerman, I., Nicholson, A. E., & Bud, A. (1997). Towards a Bayesian Model for Keyhole Plan Recognition in Large Domains, In *User Modeling*. Springer. (Cit. on p. 17).

Albrecht, S. V., & Stone, P. (2018). Autonomous Agents Modelling other Agents: A Comprehensive Survey and Open Problems. *Artif. Intell.*, *258*, 66–95. https://doi.org/10.1016/j.artint.2018.01.002 (cit. on p. 17)

Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting Regression in Planning, In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, IJCAI/AAAI. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6877. (Cit. on p. 10)

Baier, J. A., Bacchus, F., & McIlraith, S. A. (2009). A Heuristic Search Approach to Planning with Temporally Extended Preferences. *Artif. Intell.*, *173*(5-6), 593–618. https://doi.org/10.1016/j.artint.2008.11.011 (cit. on p. 51)

Blockeel, H., & Raedt, L. D. (1998). Top-Down Induction of First-Order Logical Decision Trees. *Artif. Intell.*, *101*(1-2), 285–297. https://doi.org/10.1016/S0004-3702(98)00034-4 (cit. on pp. 61, 70)

Boddy, M. S., Gohde, J., Haigh, T., & Harp, S. A. (2005). Course of Action Generation for Cyber Security Using Classical Planning, In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, AAAI. http://www.aaai.org/Library/ICAPS/2005/icaps05-002.php. (Cit. on p. 75)

Bonet, B., & Geffner, H. (2001). Planning as Heuristic Search. *Artif. Intell.*, *129*(1-2), 5–33. https://doi.org/10.1016/S0004-3702(01)00108-4 (cit. on p. 10)

Bonet, B., & Geffner, H. (2011). Planning under Partial Observability by Classical Replanning: Theory and Experiments, In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona,* *Catalonia, Spain, July 16-22, 2011*, IJCAI/AAAI. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-324. (Cit. on pp. 15, 148)

Bonet, B., & Geffner, H. (2015). Policies that Generalize: Solving Many Planning Problems with the Same Policy, In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, AAAI Press. http://ijcai.org/Abstract/15/396. (Cit. on p. 132)

Borck, H., Karneeb, J., Alford, R., & Aha, D. W. (2015). Case-Based Behavior Recognition in Beyond Visual Range Air Combat, In *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida, USA, May 18-20, 2015*, AAAI Press. http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS15/paper/view/10412. (Cit. on p. 75)

Borrajo, D. (2013). Multi-agent Planning by Plan Reuse, In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, IFAAMAS. http://dl.acm.org/citation.cfm?id=2485111. (Cit. on p. 12)

Borrajo, D., & Fernández, S. (2019). Efficient Approaches for Multi-agent Planning. *Knowl. Inf. Syst.*, *58*(2), 425–479. https://doi.org/10.1007/s10115-018-1202-1 (cit. on p. 12)

Boutilier, C., & Brafman, R. I. (2001). Partial-Order Planning with Concurrent Interacting Actions. *J. Artif. Intell. Res.*, *14*, 105–136. https://doi.org/10.1613/jair.740 (cit. on p. 13)

Bowling, M. H., Browning, B., & Veloso, M. M. (2004). Plays as Effective Multiagent Plans Enabling Opponent-Adaptive Play Selection, In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, AAAI. http://www.aaai.org/Library/ICAPS/2004/icaps04-044.php. (Cit. on p. 75)

Bowling, M. H., Jensen, R. M., & Veloso, M. M. (2003). A Formalization of Equilibria for Multiagent Planning, In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, Morgan Kaufmann. http://ijcai.org/Proceedings/03/Papers/238.pdf. (Cit. on pp. 11, 13)

Brafman, R. I., & Domshlak, C. (2008). From One to Many: Planning for Loosely Coupled Multi-Agent Systems, In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*. http://www.aaai.org/Library/ICAPS/2008/icaps08-004.php. (Cit. on pp. 12, 13)

Brown, G. G., Carlyle, W. M., Salmeron, J., & Wood, K. (2005). Analyzing the Vulnerability of Critical Infrastructure to Attack and Planning Defenses, In *Emerging Theory, Methods, and Applications*. INFORMS. (Cit. on p. 110).

Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, *35*(8), 677–691. https://doi.org/10.1109/TC.1986.1676819 (cit. on p. 10)

Buffet, O., & Aberdeen, D. (2005). Robust Planning with (L)RTDP, In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, Professional Book Center. http://ijcai.org/Proceedings/05/Papers/1605.pdf. (Cit. on p. 148)

Burns, E., Benton, J., Ruml, W., Yoon, S. W., & Do, M. B. (2012). Anticipatory On-Line Planning, In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. (Cit. on pp. 24, 50, 53, 57, 70).

Buro, M. (2003). Real-Time Strategy Games: A New AI Research Challenge, In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, Morgan Kaufmann. http://ijcai.org/Proceedings/03/Papers/265.pdf. (Cit. on p. 100)

Bylander, T. (1994). The Computational Complexity of Propositional STRIPS planning. *Artificial Intelligence*, *69*(1-2), 165–204 (cit. on pp. 8, 26).

Camacho, A., & McIlraith, S. A. (2016). Strong-Cyclic Planning when Fairness is Not a Valid Assumption, In *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning co-located with 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York City, USA, July 10, 2016*, CEUR-WS.org. http://ceur-ws.org/Vol-1648/paper11.pdf. (Cit. on pp. 135, 150)

Carbonell, J. G. (1978). POLITICS: Automated Ideological Reasoning. *Cognitive Science*, *2*(1), 27–51 (cit. on p. 110).

Carbonell, J. G. (1981). Counterplanning: A Strategy-Based Model of Adversary Planning in Real-World Situations. *Artif. Intell.*, *16*(3), 295–329. https://doi.org/10.1016/0004-3702(81)90003-5 (cit. on pp. 75, 110)

Cashmore, M., Collins, A., Krarup, B., Krivic, S., Magazzeni, D., & Smith, D. E. (2019). Towards Explainable AI Planning as a Service, 1908.05059. http://arxiv.org/abs/1908.05059 (cit. on p. 16)

Cashmore, M., Fox, M., Long, D., Magazzeni, D., & Ridder, B. (2018). Opportunistic Planning in Autonomous Underwater Missions. *IEEE Trans Autom. Sci. Eng.*, *15*(2), 519–530. https://doi.org/10.1109/TASE.2016.2636662 (cit. on p. 16)

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtós, N., & Carreras, M. (2015). ROSPlan: Planning in the Robot Operating System, In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, AAAI Press. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10619. (Cit. on p. 7)

Cenamor, I., de la Rosa, T., Núñez, S., & Borrajo, D. (2017). Planning for Tourism Routes using Social Networks. *Expert Syst. Appl.*, *69*, 1–9. https://doi.org/10.1016/j.eswa.2016.10.030 (cit. on p. 53)

Chakraborti, T., Kulkarni, A., Sreedharan, S., Smith, D. E., & Kambhampati, S. (2019). Explicability? Legibility? Predictability? Transparency? Privacy? Security? The Emerging Landscape of Interpretable Agent Behavior, In

*Proceedings of the International Conference on Automated Planning and Scheduling*. (Cit. on p. 52).

Chakraborti, T., Sreedharan, S., & Kambhampati, S. (2020). The Emerging Landscape of Explainable AI Planning and Decision Making, 2002.11697. https://arxiv.org/abs/2002.11697 (cit. on p. 16)

Charniak, E., & Goldman, R. P. (1991). *Probabilistic Abduction for Plan Recognition*. Brown University, Department of Computer Science. (Cit. on p. 17).

Chen, J., Sturtevant, N. R., Doyle, W. J., & Ruml, W. (2019). Revisiting Suboptimal Search, In *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*. https://aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18324. (Cit. on p. 36)

Choi, D. K. (2010). *Coordinated Execution and Goal Management in a Reactive Cognitive Architecture* (Doctoral dissertation). Stanford University. (Cit. on p. 16).

Churchill, D., Saffidine, A., & Buro, M. (2012). Fast Heuristic Search for RTS Game Combat Scenarios, In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, USA, October 8-12, 2012*, The AAAI Press. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/view/5469. (Cit. on p. 111)

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artif. Intell.*, *147*(1-2), 35–84. https://doi.org/10.1016/S0004-3702(02)00374-0 (cit. on pp. 12, 134, 135)

Coddington, A. M., Fox, M., Gough, J., Long, D., & Serina, I. (2005). MADbot: A Motivated and Goal Directed Robot, In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, AAAI Press / The MIT Press. http://www.aaai.org/Library/AAAI/2005/isd05-006.php. (Cit. on pp. 53, 55, 69)

Cox, M. T. (2007). Perpetual Self-aware Cognitive Agents. *AI magazine*, *28*(1), 32 (cit. on pp. 4, 14, 16, 49, 53, 69, 111).

Cox, M. T. (2013). Goal-driven Autonomy and Question-based Problem Recognition, In *Second Annual Conference on Advances in Cognitive Systems 2013, Poster Collection*. Citeseer. (Cit. on p. 54).

Cox, M. T., Alavi, Z., Dannenhauer, D., Eyorokon, V., Muñoz-Avila, H., & Perlis, D. (2016). MIDCA: A Metacognitive, Integrated Dual-Cycle Architecture for Self-Regulated Autonomy, In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12292. (Cit. on pp. 19, 69)

Crosby, M., Jonsson, A., & Rovatsos, M. (2014). A Single-Agent Approach to Multiagent Planning, In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Pres-*

*tigious Applications of Intelligent Systems (PAIS 2014)*, IOS Press. https://doi.org/10.3233/978-1-61499-419-0-237. (Cit. on p. 13)

Dannenhauer, D., Floyd, M. W., Magazzeni, D., & Aha, D. W. (2018). Explaining Rebel Behavior in Goal Reasoning Agents, In *Proceedings of ICAPS-18 Workshop on Explainable Planning*. (Cit. on p. 16).

de la Asunción, M., Castillo, L. A., Fernández-Olivares, J., García-Pérez, O., Muñoz, A. G., & Palao, F. (2005). SIADEX: An Interactive Knowledge-based Planner for Decision Support in Forest Fire Fighting. *AI Commun.*, *18*(4), 257–268. http://content.iospress.com/articles/ai-communications/aic348 (cit. on p. 53)

Despouys, O., & Ingrand, F. F. (1999). Propice-Plan: Toward a Unified Framework for Planning and Execution, In *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, Springer. https://doi.org/10.1007/10720246_22. (Cit. on p. 149)

D'Ippolito, N., Braberman, V. A., Kramer, J., Magee, J., Sykes, D., & Uchitel, S. (2014). Hope for the Best, Prepare for the Worst: Multi-tier Control for Adaptive Systems, In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, ACM. https://doi.org/10.1145/2568225.2568264. (Cit. on pp. 131, 136, 137, 149)

Do, M. B., Benton, J., van den Briel, M., & Kambhampati, S. (2007). Planning with Goal Utility Dependencies, In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. http://ijcai.org/Proceedings/07/Papers/302.pdf. (Cit. on p. 16)

Domshlak, C. (2013). Fault Tolerant Planning: Complexity and Compilation, In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*, AAAI. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/5977. (Cit. on p. 148)

Domshlak, C., & Mirkis, V. (2015). Deterministic Oversubscription Planning as Heuristic Search: Abstractions and Reformulations. *J. Artif. Intell. Res.*, *52*, 97–169. https://doi.org/10.1613/jair.4443 (cit. on pp. 16, 26, 50)

Duff, S., Harland, J., & Thangarajah, J. (2006). On Proactivity and Maintenance Goals, In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, ACM. https://doi.org/10.1145/1160633.1160817. (Cit. on p. 56)

E-Martín, Y., R.-Moreno, M. D., & Smith, D. E. (2015). A Fast Goal Recognition Technique Based on Interaction Estimates, In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, AAAI Press. http://ijcai.org/Abstract/15/113. (Cit. on pp. 17, 50)

Edelkamp, S., & Helmert, M. (2001). MIPS: The Model-Checking Integrated Planning System. *AI Magazine*, *22*(3), 67–72. http://www.aaai.org/ojs/index.php/aimagazine/article/view/1574 (cit. on p. 10)

Evans, J. S. B., & Stanovich, K. E. (2013). Dual-process Theories of Higher Cognition: Advancing the Ddebate. *Perspectives on psychological science*, *8*(3), 223–241 (cit. on p. 7).

Fox, M., Gerevini, A., Long, D., & Serina, I. (2006). Plan Stability: Replanning versus Plan Repair, In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, AAAI. http://www.aaai.org/Library/ICAPS/2006/icaps06-022.php. (Cit. on pp. 15, 131)

Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, *20*, 61–124. https://doi.org/10.1613/jair.1129 (cit. on p. 13)

Fuentetaja, R., Borrajo, D., & de la Rosa, T. (2018). Anticipation of Goals in Automated Planning. *AI Commun.*, *31*(2), 117–135. https://doi.org/10.3233/AIC-180753 (cit. on pp. 24, 50, 57, 70)

Furelos-Blanco, D., & Jonsson, A. (2019). Solving Multiagent Planning Problems with Concurrent Conditional Effects, In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press. https://doi.org/10.1609/aaai.v33i01.33017594. (Cit. on pp. 12, 13)

García, J., Flórez, J. E., de Reyna, Á. T. A., Borrajo, D., López, C. L., Olaya, A. G., & Sáenz, J. (2013). Combining Linear Programming and Automated Planning to Solve Intermodal transportation problems. *Eur. J. Oper. Res.*, *227*(1), 216–226. https://doi.org/10.1016/j.ejor.2012.12.018 (cit. on pp. 7, 53)

Geffner, H. (2018). Model-free, Model-based, and General Intelligence, In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, ijcai.org. https://doi.org/10.24963/ijcai.2018/2. (Cit. on p. 7)

Geffner, T., & Geffner, H. (2018). Compact Policies for Fully Observable Non-Deterministic Planning as SAT, In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, AAAI Press. https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17732. (Cit. on pp. 135, 136, 143, 146, 155)

Geib, C. W., & Goldman, R. P. (2009). A Probabilistic Plan Recognition Algorithm Based on Plan Tree Grammars. *Artif. Intell.*, *173*(11), 1101–1132. https://doi.org/10.1016/j.artint.2009.01.003 (cit. on p. 17)

Gerevini, A., & Serina, I. (2000). Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques, In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, AAAI. http://www.aaai.org/Library/AIPS/2000/aips00-012.php. (Cit. on p. 15)

Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated Planning - Theory and Practice*. Elsevier. (Cit. on pp. 3, 7).

Ghallab, M., Nau, D. S., & Traverso, P. (2014). The Actor's View of Automated Planning and Acting: A Position Paper. *Artif. Intell.*, *208*, 1–17. https://doi.org/10.1016/j.artint.2013.11.002 (cit. on p. 3)

Ghallab, M., Nau, D. S., & Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB. (Cit. on p. 3)

Giacomo, G. D., Gerevini, A. E., Patrizi, F., Saetti, A., & Sardiña, S. (2016). Agent Planning Programs. *Artif. Intell.*, *231*, 64–106. https://doi.org/10.1016/j.artint.2015.10.001 (cit. on p. 149)

González, J. C., Veloso, M., Fernández, F., & García-Olaya, Á. (2015). Task Monitoring and Rescheduling for Opportunity and Failure Management (cit. on p. 16).

Gopalakrishnan, S, Avila, H. M., & Kuter, U. (2016). Word2htn: Learning Task Hierarchies using Statistical Semantics and Goal Reasoning, In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*. (Cit. on p. 70).

Gottfredson, L. S. (1997). Mainstream Science on Intelligence: An Editorial with 52 Signatories, History, and Bibliography. Citeseer. (Cit. on p. 3).

Guzmán, C., Alcázar, V., Prior, D., Onaindia, E., Borrajo, D., Fdez-Olivares, J., & Quintero, E. (2012). PELEA: A Domain-independent Architecture for Planning, Execution and Learning, In *Proceedings of ICAPS*. (Cit. on p. 57).

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Systems Science and Cybernetics*, *4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136 (cit. on p. 10)

Haslum, P., & Geffner, H. (2000). Admissible Heuristics for Optimal Planning, In *Proceedings of the fifth international conference on artificial intelligence planning systems, breckenridge, co, usa, april 14-17, 2000*, AAAI. http://www.aaai.org/Library/AIPS/2000/aips00-015.php. (Cit. on p. 90)

Haslum, P., Lipovetzky, N., Magazzeni, D., & Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00900ED2V01Y201902AIM042. (Cit. on p. 9)

Helmert, M. (2006). The Fast Downward Planning System. *J. Artif. Intell. Res.*, *26*, 191–246. https://doi.org/10.1613/jair.1705 (cit. on pp. 35, 90)

Helmert, M., & Domshlak, C. (2009). Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/735. (Cit. on pp. 11, 35, 90, 106)

Helmert, M., & Lasinger, H. (2010). The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem, In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, AAAI. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1423. (Cit. on p. 7)

Hoffmann, Porteous, J., & Sebastia, L. (2004). Ordered Landmarks in Planning. *J. Artif. Intell. Res.*, *22*, 215–278. https://doi.org/10.1613/jair.1492 (cit. on pp. 10, 51, 76)

Hoffmann, J. (2015). Simulated Penetration Testing: From Dijkstra to Turing Test++, In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, AAAI Press. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10495. (Cit. on p. 75)

Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.*, *14*, 253–302. https://doi.org/10.1613/jair.855 (cit. on p. 35)

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2011). Integrated Learning for Goal-Driven Autonomy, In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, IJCAI/AAAI. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-408. (Cit. on p. 70)

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2013). Case-based Goal-driven Coordination of Multiple Learning Agents, In *International Conference on Case-Based Reasoning*. Springer. (Cit. on pp. 53, 111).

Jarvis, P., Lunt, T. F., & Myers, K. L. (2004). Identifying Terrorist Activity with AI Plan Recognition Technology, In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, AAAI Press / The MIT Press. (Cit. on p. 75).

Jeantet, G., & Spanjaard, O. (2008). Rank-Dependent Probability Weighting in Sequential Decision Problems under Uncertainty, In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*. http://www.aaai.org/Library/ICAPS/2008/icaps08-019.php. (Cit. on p. 51)

Jensen, R. M., & Veloso, M. M. (2000). OBDD-based Universal Planning for Synchronized Agents in Non-Deterministic Domains. *J. Artif. Intell. Res.*, *13*, 189–226. https://doi.org/10.1613/jair.649 (cit. on p. 111)

Jensen, R. M., Veloso, M. M., & Bowling, M. H. (2001). OBDD-based Optimistic and Strong Cyclic Adversarial Planning, In *Sixth European Conference on Planning*. (Cit. on p. 12).

Kabanza, F., Bellefeuille, P., Bisson, F., Benaskeur, A. R., & Irandoust, H. (2010). Opponent Behaviour Recognition for Real-Time Strategy Games, In *Plan, Activity, and Intent Recognition, Papers from the 2010 AAAI Workshop, Atlanta, Georgia, USA, July 12, 2010*, AAAI. http://aaai.org/ocs/index.php/WS/AAAIW10/paper/view/2024. (Cit. on pp. 75, 111)

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan. (Cit. on p. 7).

Kaminka, G. A., Vered, M., & Agmon, N. (2018). Plan Recognition in Continuous Domains, In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, AAAI Press. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17372. (Cit. on p. 17)

Katz, M., Sohrabi, S., & Udrea, O. (2020). Top-Quality Planning: Finding Practically Useful Sets of Best Plans, In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press. https://aaai.org/ojs/index.php/AAAI/article/view/6544. (Cit. on p. 90)

Kautz, H. A., & Allen, J. F. (1986). Generalized Plan Recognition, In *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, Morgan Kaufmann. http://www.aaai.org/Library/AAAI/1986/aaai86-006.php. (Cit. on p. 17)

Keren, S., Gal, A., & Karpas, E. (2016). Privacy Preserving Plans in Partially Observable Environments, In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. http://www.ijcai.org/Abstract/16/449. (Cit. on pp. 24, 50)

Keren, S., Gal, A., & Karpas, E. (2020). Goal Recognition Design Survey, In *Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI 2020*, ijcai.org. https://doi.org/10.24963/ijcai.2020/675. (Cit. on p. 126)

Klenk, M., Molineaux, M., & Aha, D. W. (2013). Goal-Driven Autonomy for Responding to Unexpected Events in Strategy Simulations. *Comput. Intell.*, *29*(2), 187–206. https://doi.org/10.1111/j.1467-8640.2012.00445.x (cit. on pp. 15, 16, 19, 53, 69, 149)

Koenig, S., & Simmons, R. G. (1994). How to Make Reactive Planners Risk-sensitive, In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*. (Cit. on pp. 30, 51).

Kolobov, A., Mausam, & Weld, D. S. (2009). ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning, In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. http://ijcai.org/Proceedings/09/Papers/291.pdf. (Cit. on p. 134)

Kulkarni, A., Srivastava, S., & Kambhampati, S. (2019). A Unified Framework for Planning in Adversarial and Cooperative Environments, In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019 Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press. https://doi.org/10.1609/aaai.v33i01.33012479. (Cit. on pp. 24, 25, 30, 50, 52)

Kurup, U., Lebiere, C., Stentz, A., & Hebert, M. (2012). Using Expectations to Drive Cognitive Behavior, In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5106. (Cit. on p. 16)

Kvarnström, J., & Doherty, P. (2010). Automated Planning for Collaborative UAV Systems, In *11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010, Singapore, 7-10 December 2010, Proceed-*

*ings*, IEEE. https://doi.org/10.1109/ICARCV.2010.5707969. (Cit. on p. 25)

Lago, U. D., Pistore, M., & Traverso, P. (2002). Planning with a Language for Extended Goals, In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, AAAI Press / The MIT Press. http://www.aaai.org/Library/AAAI/2002/aaai02-068.php. (Cit. on p. 149)

Lipovetzky, N., Ramirez, M., Muise, C., & Geffner, H. (2014). Width and Inference Based Planners: SIW, BFS (f), and PROBE. *International Planning Competition* (cit. on p. 106).

Lu, Q., George, B., & Shekhar, S. (2005). Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results, In *Advances in Spatial and Temporal Databases, 9th International Symposium, SSTD 2005, Angra dos Reis, Brazil, August 22-24, 2005, Proceedings*. https://doi.org/10.1007/11535331_17. (Cit. on p. 51)

Luis, N., Pereira, T. R., Fernández, S., Moreira, A., Borrajo, D., & Veloso, M. (2020). Using Pre-Computed Knowledge for Goal Allocation in Multi-Agent Planning. *J. Intell. Robotic Syst.*, *98*(1), 165–190. https://doi.org/10.1007/s10846-019-01022-0 (cit. on p. 12)

Maliah, S., Brafman, R. I., & Shani, G. (2017). Increased Privacy with Reduced Communication in Multi-Agent Planning, In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, AAAI Press. https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15739. (Cit. on p. 12)

Masters, P., & Sardiña, S. (2017). Deceptive Path-Planning, In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, ijcai.org. https://doi.org/10.24963/ijcai.2017/610. (Cit. on pp. 30, 50)

Masters, P., & Sardiña, S. (2019). Cost-Based Goal Recognition in Navigational Domains. *J. Artif. Intell. Res.*, *64*, 197–242. https://doi.org/10.1613/jair.1.11343 (cit. on p. 17)

Maynord, M., Cox, M. T., Paisner, M., & Perlis, D. (2013). Data-Driven Goal Generation for Integrated Cognitive Systems, In *2013 AAAI Fall Symposia, Arlington, Virginia, USA, November 15-17, 2013*, AAAI Press. http://www.aaai.org/ocs/index.php/FSS/FSS13/paper/view/7618. (Cit. on pp. 69, 70)

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational. (Cit. on p. 9).

McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., & McEwen, R. S. (2008). A Deliberative Architecture for AUV Control, In *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23,* *2008, Pasadena, California, USA*, IEEE. https://doi.org/10.1109/ROBOT.2008.4543343. (Cit. on p. 149)

Molineaux, M., & Aha, D. W. (2014). Learning Unknown Event Models, In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8161. (Cit. on p. 70)

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-Driven Autonomy in a Navy Strategy Simulation, In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1928. (Cit. on pp. 4, 14, 49, 111)

Morgenstern, O., & Von Neumann, J. (1953). *Theory of Games and Economic Behavior*. Princeton university press. (Cit. on p. 51).

Muise, C. J., Belle, V., & McIlraith, S. A. (2014). Computing Contingent Plans via Fully Observable Non-Deterministic Planning, In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8656. (Cit. on p. 136)

Muise, C. J., Felli, P., Miller, T., Pearce, A. R., & Sonenberg, L. (2016). Planning for a Single Agent in a Multi-Agent Environment Using FOND, In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, IJCAI/AAAI Press. http://www.ijcai.org/Abstract/16/454. (Cit. on p. 13)

Muñoz-Avila, H., Dannenhauer, D., & Reifsnyder, N. (2019). Is Everything Going According to Plan? Expectations in Goal Reasoning Agents, In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press. https://doi.org/10.1609/aaai.v33i01.33019823. (Cit. on p. 16)

Muñoz-Avila, H., Jaidee, U., Aha, D. W., & Carter, E. (2010). Goal-driven Autonomy with Case-based Reasoning, In *International Conference on Case-Based Reasoning*. Springer. (Cit. on p. 16).

Myers, K. L. (1999). CPEF: A Continuous Planning and Execution Framework. *AI Magazine*, *20*(4), 63–69. http://www.aaai.org/ojs/index.php/aimagazine/article/view/1480 (cit. on p. 149)

Nebel, B., & Koehler, J. (1995). Plan Reuse Versus Plan Generation: A Theoretical and Empirical Analysis. *Artif. Intell.*, *76*(1-2), 427–454. https://doi.org/10.1016/0004-3702(94)00082-C (cit. on p. 15)

Newell, A., Simon, H. A. Et al. (1972). *Human Problem Solving* (Vol. 104). Prentice-Hall Englewood Cliffs, NJ. (Cit. on p. 14).

Norman, T. J., & Long, D. (1995). Alarms: An Implementation of Motivated Agency, In *International Workshop on Agent Theories, Architectures, and Languages*. Springer. (Cit. on p. 14).

Olsen, M. G., Granmo, O., & Radianti, J. (2015). Escape Planning in Realistic Fire Scenarios with Ant Colony Optimisation. *Appl. Intell.*, *42*(1), 24–35. https://doi.org/10.1007/s10489-014-0538-9 (cit. on p. 51)

Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Trans. Comput. Intell. AI Games*, *5*(4), 293–311. https://doi.org/10.1109/TCIAIG.2013.2286295 (cit. on pp. 75, 111, 112)

Paredes, A., & Ruml, W. (2017). Goal Reasoning as Multilevel Planning, In *ICAPS-2017 Workshop on Integrated Execution of Planning and Acting*. (Cit. on p. 20).

Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordóñez, F., & Kraus, S. (2008). Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games, In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 2*, IFAAMAS. https://dl.acm.org/citation.cfm?id=1402348. (Cit. on p. 110)

Pattison, D., & Long, D. (2010). Domain Independent Goal Recognition, In *STAIRS 2010 - Proceedings of the Fifth Starting AI Researchers' Symposium, Lisbon, Portugal, 16-20 August, 2010*, IOS Press. https://doi.org/10.3233/978-1-60750-676-8-238. (Cit. on p. 17)

Pentney, W., Popescu, A., Wang, S., Kautz, H. A., & Philipose, M. (2006). Sensor-Based Understanding of Daily Life via Large-Scale Use of Common Sense, In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, AAAI Press. http://www.aaai.org/Library/AAAI/2006/aaai06-143.php. (Cit. on p. 17)

Pereira, R. F., Oren, N., & Meneguzzi, F. (2020). Landmark-based Approaches for Goal Recognition as Planning. *Artif. Intell.*, *279*. https://doi.org/10.1016/j.artint.2019.103217 (cit. on pp. 17, 50)

Perny, P., Spanjaard, O., & Storme, L. (2007). State Space Search for Risk-Averse Agents, In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. http://ijcai.org/Proceedings/07/Papers/379.pdf. (Cit. on p. 51)

Pita, J., Jain, M., Marecki, J., Ordóñez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., & Kraus, S. (2008). Deployed ARMOR Protection: The Application of a Game Theoretic Model for Security at the Los Angeles International Airport, In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Industry and Applications Track Proceedings*, IFAAMAS. https://dl.acm.org/citation.cfm?id=1402819. (Cit. on pp. 75, 110)

Pokahr, A, Braubach, W, & Lamersdorf, W. (2003). Implementing a BDI Infrastructure for JADE Agents. 2003. *Exp in Search of Innovation*, *3*(3) (cit. on p. 56).

Porteous, J., Sebastia, L., & Hoffmann. (2001). On the Extraction, Ordering, and Usage of Landmarks in Planning, In *6th European Conference on Planning*. (Cit. on p. 11).

Powell, J., Molineaux, M., & Aha, D. W. (2011). Active and Interactive Discovery of Goal Selection Knowledge, In *Twenty-Fourth International FLAIRS Conference*. (Cit. on pp. 16, 70).

Pozanco, A., E-Martín, Y., Fernández, S., & Borrajo, D. (2018a). Counterplanning using Goal Recognition and Landmarks, In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. https://doi.org/10.24963/ijcai.2018/668. (Cit. on pp. 25, 51)

Pozanco, A., E-Martín, Y., Fernández, S., & Borrajo, D. (2019). Finding Centroids and Minimum Covering States in Planning, In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*. https://aaai.org/ojs/index.php/ICAPS/article/view/3497. (Cit. on p. 114)

Pozanco, A., Fernández, S., & Borrajo, D. (2018b). Learning-driven Goal Generation. *AI Commun.*, *31*(2), 137–150. https://doi.org/10.3233/AIC-180754 (cit. on p. 50)

Pozanco, A., Fernández, S., & Borrajo, D. (2016). On Learning Planning Goals for Traffic Control, In *Proceedings of 4th Workshop on Goal Reasoning (IJCAI'16)*. workshop-ijcai16-learning.pdf. (Cit. on p. 56)

Preuss, M., Kozakowski, D., Hagelbäck, J., & Trautmann, H. (2013). Reactive Strategy Choice in StarCraft by means of Fuzzy Control, In *2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013*, IEEE. https://doi.org/10.1109/CIG.2013.6633627. (Cit. on p. 111)

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley. https://doi.org/10.1002/9780470316887. (Cit. on p. 134)

Pynadath, D. V., & Marsella, S. (2005). PsychSim: Modeling Theory of Mind with Decision-Theoretic Agents, In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, Professional Book Center. http://ijcai.org/Proceedings/05/Papers/1559.pdf. (Cit. on p. 17)

Quiggin, J. (2012). *Generalized Expected Utility Theory: The Rank-dependent Model*. Springer Science & Business Media. (Cit. on p. 51).

Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Mach. Learn.*, *5*, 239–266. https://doi.org/10.1007/BF00117105 (cit. on p. 70)

Ramírez, M., & Geffner, H. (2009). Plan Recognition as Planning, In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. http://ijcai.org/Proceedings/09/Papers/296.pdf. (Cit. on pp. 17, 23, 26, 37, 50, 76, 78, 85, 89, 90, 100)

Ramírez, M., & Geffner, H. (2010). Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners, In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1821. (Cit. on pp. 17–19, 76, 89, 90, 100)

Rao, A. S., & Georgeff, M. P. (1995). BDI Agents: From Theory to Practice, In *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, The MIT Press. (Cit. on p. 14).

Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks Revisited, In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, AAAI Press. http://www.aaai.org/Library/AAAI/2008/aaai08-155.php. (Cit. on p. 11)

Richter, S., & Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, *39*, 127–177. https://doi.org/10.1613/jair.2972 (cit. on p. 90)

Rintanen, J. (2004a). Complexity of Planning with Partial Observability, In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, AAAI. http://www.aaai.org/Library/ICAPS/2004/icaps04-041.php. (Cit. on p. 136)

Rintanen, J. (2004b). Evaluation Strategies for Planning as Satisfiability, In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, IOS Press. (Cit. on p. 10).

Roy, P., Bouzouane, A., Giroux, S., & Bouchard, B. (2011). Possibilistic Activity Recognition in Smart Homes for Cognitively Impaired People. *Applied Artificial Intelligence*, *25*(10), 883–926. https://doi.org/10.1080/08839514.2011.617248 (cit. on p. 17)

Ruml, W., Do, M. B., Zhou, R., & Fromherz, M. P. J. (2011). On-line Planning and Scheduling: An Application to Controlling Modular Printers. *J. Artif. Intell. Res.*, *40*, 415–468. https://doi.org/10.1613/jair.3184 (cit. on p. 53)

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall. http://www.worldcat.org/oclc/314283679. (Cit. on p. 3)

Sailer, F., Buro, M., & Lanctot, M. (2007). Adversarial Planning Through Strategy Simulation, In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, CIG 2007, Honolulu, Hawaii, USA, 1-5 April, 2007*, IEEE. https://doi.org/10.1109/CIG.2007.368082. (Cit. on p. 111)

Schermerhorn, P. W., Benton, J., Scheutz, M., Talamadupula, K., & Kambhampati, S. (2009). Finding and Exploiting Goal Opportunities in Real-time During Plan Execution, In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 11-15, 2009, St. Louis, MO, USA*, IEEE. https://doi.org/10.1109/IROS.2009.5354119. (Cit. on p. 16)

Shivashankar, V., Kuter, U., Nau, D. S., & Alford, R. (2012). A Hierarchical Goal-based Formalism and Algorithm for Single-agent Planning, In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, IFAAMAS. http://dl.acm.org/citation.cfm?id=2343837. (Cit. on p. 149)

Smith, D. E. (2004). Choosing Objectives in Over-Subscription Planning, In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*. http://www.aaai.org/Library/ICAPS/2004/icaps04-046.php. (Cit. on pp. 16, 26, 50, 58)

Speicher, P., Steinmetz, M., Backes, M., Hoffmann, J., & Künnemann, R. (2018). Stackelberg Planning: Towards Effective Leader-Follower State Space Search, In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17209. (Cit. on pp. 25, 110)

Stackelberg, H. v. (1952). Theory of the Market Economy (cit. on p. 110).

Stanescu, M., Barriga, N. A., & Buro, M. (2014). Hierarchical Adversarial Search Applied to Real-Time Strategy Games, In *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014, October 3-7, 2014, North Carolina State University, Raleigh, NC, USA*, AAAI. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/view/8993. (Cit. on p. 111)

Stanescu, M., & Certický, M. (2016). Predicting Opponent's Production in Real-Time Strategy Games With Answer Set Programming. *IEEE Trans. Comput. Intell. AI Games*, *8*(1), 89–94. https://doi.org/10.1109/TCIAIG.2014.2365414 (cit. on p. 111)

Sukthankar, G., Geib, C., Bui, H. H., Pynadath, D., & Goldman, R. P. (2014). *Plan, Activity, and Intent Recognition: Theory and Practice*. Newnes. (Cit. on p. 17).

Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P. W., & Scheutz, M. (2010). Planning for Human-robot Teaming in Open Worlds. *ACM Trans. Intell. Syst. Technol.*, *1*(2), 14:1–14:24. https://doi.org/10.1145/1869397.1869403 (cit. on p. 53)

Tambe, M. (2012). *Security and Game Theory - Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/communications-information-theory-and-security/security-and-game-theory-algorithms-deployed-systems-lessons-learned?format=AR. (Cit. on p. 75)

Tavares, A. R., Azpurua, H., Santos, A., & Chaimowicz, L. (2016). Rock, Paper, StarCraft: Strategy Selection in Real-Time Strategy Games, In *Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2016, October 8-12, 2016, burlingame, california, USA*, AAAI Press. http://aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/13998. (Cit. on p. 111)

Torralba, Á., López, C. L., & Borrajo, D. (2018). Symbolic Perimeter Abstraction Heuristics for Cost-optimal Planning. *Artif. Intell.*, *259*, 1–31. https://doi.org/10.1016/j.artint.2018.02.002 (cit. on p. 10)

Torreño, A., Onaindia, E., Komenda, A., & Stolba, M. (2018). Cooperative Multi-Agent Planning: A Survey. *ACM Comput. Surv.*, *50*(6), 84:1–84:32. https://doi.org/10.1145/3128584 (cit. on p. 12)

Tozicka, J., Stolba, M., & Komenda, A. (2017). The Limits of Strong Privacy Preserving Multi-Agent Planning, In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, AAAI Press. https://

aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15754. (Cit. on p. 12)

Uriarte, A., & Ontañón, S. (2014). Game-Tree Search over High-Level Game States in RTS Games, In *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014, October 3-7, 2014, North Carolina State University, Raleigh, NC, USA*, AAAI. http://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/view/8960. (Cit. on p. 111)

van Riemsdijk, M. B., Dastani, M., & Winikoff, M. (2008). Goals in Agent Systems: A Unifying Framework, In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 2*, IFAAMAS. https://dl.acm.org/citation.cfm?id=1402323. (Cit. on p. 56)

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). *Breadth of Approaches to Goal Reasoning: A Research Survey* (tech. rep.). Naval Research Lab Washington DC. (Cit. on pp. 15, 49, 53, 111).

Weber, B. G., & Mateas, M. (2009). A Data Mining Approach to Strategy Prediction, In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy, 7-10 September, 2009*, IEEE. https://doi.org/10.1109/CIG.2009.5286483. (Cit. on p. 111)

Weber, B. G., Mateas, M., & Jhala, A. (2010). Applying Goal-Driven Autonomy to StarCraft, In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, October 11-13, 2010, Stanford, California, USA*, The AAAI Press. http://aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/view/2142. (Cit. on p. 111)

Weber, B. G., Mateas, M., & Jhala, A. (2012). Learning from Demonstration for Goal-Driven Autonomy, In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5090. (Cit. on pp. 16, 70)

Wender, S., & Watson, I. D. (2014). Combining Case-Based Reasoning and Reinforcement Learning for Unit Navigation in Real-Time Strategy Game AI, In *Case-Based Reasoning Research and Development - 22nd International Conference, ICCBR 2014, Cork, Ireland, September 29, 2014 - October 1, 2014. Proceedings*, Springer. https://doi.org/10.1007/978-3-319-11209-1_36. (Cit. on p. 111)

Widmer, G., & Kubat, M. (1996). Learning in the Presence of Concept Drift and Hidden Contexts. *Mach. Learn.*, *23*(1), 69–101. https://doi.org/10.1007/BF00116900 (cit. on p. 67)

Wiest, J., Höffken, M., Kressel, U., & Dietmayer, K. (2012). Probabilistic Trajectory Prediction with Gaussian Mixture Models, In *2012 IEEE Intelligent Vehicles Symposium, IV 2012, Alcal de Henares, Madrid, Spain, June 3-7, 2012*, IEEE. https://doi.org/10.1109/IVS.2012.6232277. (Cit. on p. 17)

Willmott, S., Richardson, J., Bundy, A., & Levine, J. (2001). Applying Adversarial Planning Techniques to Go. *Theor. Comput. Sci.*, *252*(1-2), 45–82. https://doi.org/10.1016/S0304-3975(00)00076-1 (cit. on p. 75)

Wilson, M. A., Molineaux, M., & Aha, D. W. (2013). Domain-Independent Heuristics for Goal Formulation, In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida, USA, May 22-24, 2013*, AAAI Press. http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS13/paper/view/5948. (Cit. on p. 16)

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems, Second Edition*. Wiley. (Cit. on p. 11).

Wu, J., Osuntogun, A., Choudhury, T., Philipose, M., & Rehg, J. M. (2007). A Scalable Approach to Activity Recognition based on Object Use, In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, IEEE Computer Society. https://doi.org/10.1109/ICCV.2007.4408865. (Cit. on p. 17)

Yoon, S. W., Fern, A., & Givan, R. (2007). FF-Replan: A Baseline for Probabilistic Planning, In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, AAAI. http://www.aaai.org/Library/ICAPS/2007/icaps07-045.php. (Cit. on p. 134)