Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau

Anytime Optimal MDP Planning with Trial-based Heuristic Tree Search

Thomas Keller

2015

Dean:
Prof. Dr. Georg Lausen, *University of Freiburg, Germany*

PhD advisor and first reviewer:
Prof. Dr. Bernhard Nebel, *University of Freiburg, Germany*

Second reviewer:
Prof. Dr. Wolfram Burgard, *University of Freiburg, Germany*

Examination committee:
Prof. Dr. Bernd Becker (co-chair), *University of Freiburg, Germany*
Prof. Dr. Wolfram Burgard, *University of Freiburg, Germany*
Prof. Dr. Bernhard Nebel, *University of Freiburg, Germany*
Prof. Dr. Matthias Teschner (chair), *University of Freiburg, Germany*

Date of disputation:
July 7, 2015

To Catherine & Jonathan.

Abstract

Planning and acting in a dynamic environment is a challenging task for an autonomous agent, especially in the presence of uncertain and exogenous effects, a large number of states, and a long-term planning horizon. In this thesis, we approach the problem by considering algorithms that interleave planning for the current state and execution of the taken decision. The main challenge of the agent is to use its tight deliberation time wisely.

One solution are determinizations, which simplify the Markov Decision Process that describes the uncertain environment to a deterministic planning problem. We introduce an all-outcomes determinization where, unlike in comparable methods, the number of deterministic actions is not exponentially but polynomially bounded in the number of parallel probabilistic effects. We discuss three algorithms that base their decision solely on the solution to a determinization, and show that they have fundamental limitations that prevent optimal behavior even if provided with unlimited resources.

The main contribution of this thesis, the Trial-based Heuristic Tree Search (THTS) framework, allows the description of algorithms in terms of only six ingredients that can be mixed and matched at will. We present a selection of ingredients and analyze theoretically which combinations yield asymptotically optimal behavior. Our implementation of the THTS framework, the probabilistic planner PROST, not only allows to evaluate all anytime optimal algorithms empirically on the benchmarks of the International Probabilistic Planning Competition (IPPC), but furthermore emphasizes the potential of THTS by being the back to back winner of the competition in 2011 and 2014.

In the final chapter, we introduce the MDP-Evaluation Stopping Problem, the optimization problem faced by participants of IPPC 2014. We show how it can be constructed formally, discuss three special cases that are solvable in practice, and present approximate algorithms that are based on techniques that are derived from the solutions for the special cases. Finally, we show theoretically and empirically that all proposed algorithms improve significantly over the application of the state-of-the-art approach.

Zusammenfassung

Planen und Handeln in einer dynamischen Umgebung ist eine große Herausforderung für einen autonomen Agenten, insbesondere unter Unsicherheit, vielen Zuständen sowie einem langfristigen Planungshorizont. Wir gehen das Problem in dieser Thesis mit Algorithmen an, die abwechselnd für den momentanen Zustand planen und die resultierende Aktion ausführen. Die wichtigste Herausforderung eines Agenten liegt darin, die begrenzte Zeit zur Entscheidungsfindung sinnvoll zu nutzen. Determinisierungen kompilieren den die unsichere Umgebung modellierenden Markov'schen Entscheidungsprozess in ein deterministisches Planungsproblem. Wir präsentieren eine Determinisierung, in welcher alle möglichen Ausgänge erhalten bleiben und die Anzahl der deterministischen Aktionen erstmals nicht exponentiell sondern polynomiell in der Anzahl paralleler probabilistischer Effekte begrenzt ist.

Wir stellen drei Algorithmen vor die ihre Entscheidung ausschließlich auf Basis einer Determinisierung treffen. Allerdings haben diese fundamentale Schwächen die dazu führen dass sie selbst mit unbegrenzten Ressourcen nicht optimal sind. Der Hauptbeitrag dieser Thesis, das Trial-based Heuristic Tree Search (THTS) Framework, erlaubt die Beschreibung von Algorithmen durch sechs Zutaten welche beliebig gemischt werden können. Wir präsentieren eine Auswahl von Zutaten und analysieren theoretisch welche zu asymptotisch optimalen Rezepten kombiniert werden können. Unsere Implementierung des THTS Frameworks, der probabilistische Planer PROST, erlaubt nicht nur die Evaluierung aller optimaler Algorithmen auf den Benchmarks des Internationalen Probabilistischen Planungswettbewerbs (IPPC), sondern zeigt auch die Stärken von THTS durch den wiederholten Gewinn des IPPC 2011 und 2014.

Im letzten Kapitel beschreiben wir das Optimierungsproblem aller Teilnehmer des IPPC 2014, das MDP-Evaluation Stopping Problem. Wir zeigen wie es formal konstruiert werden kann, diskutieren drei Spezialfälle die auch in der Praxis gelöst werden können und präsentieren darauf basierende, näherungsweise Verfahren. Schließlich zeigen wir theoretisch und empirisch dass unsere Algorithmen eine deutliche Verbesserung zum naiven Ansatz darstellen.

Acknowledgments

In the time it takes to write a thesis there is a large number of people that contribute to a successful outcome in one way or another, and it is my pleasure to thank all those wonderful people. First of all, I would like to thank my advisor Bernhard Nebel for offering me the possibility to be part of his research group. I am grateful that he gave me the freedom to pursue my own ideas while pushing me in the right moments to get the work done. Plenty of amazing opportunities have arisen due to his efforts. Most notably, they allowed me to travel to conferences and work meetings all around the world, which led to valuable input of researchers I met on those trips. He managed to establish a relaxed and productive atmosphere in his research group, and it has always been a pleasure to be part of the group.

I also thank all other members from the research group. Roswitha Hilden and Petra Geiger have been the two persons one could turn to with any problem, and the day has yet to come that one remains unsolved. Uli Jakob did not only keep the computer infrastructure run smoothly (his support with the grid has been crucial for the empirical part of this thesis), he has also made sure that I never lost my spirit by providing me with numerous headsets and by maintaining the coffee machine. I thank all three of them for everything they did for me.

I do not want to miss the opportunity to thank my colleagues Alexander Kleiner, Christian Becker-Asano, Christian Dornhege, Florian Geißer, Gabi Röger, Johannes Aldinger, Johannes Löhr, Michael Brenner, Moritz Göbelbecker, Stefan Wölfl, and Tim Schulte for many scientific and non-scientific discussions that made the research group such a great place to work. There are also some former colleagues that I collaborated especially closely with. Sebastian Kupferschmid supervised my Studienarbeit, which was a crucial point during my studies as it sparked my enthusiasm for AI, and I still benefit from his advice on good coding habits. Robert Mattmüller always made me remember that, while writing comprehensible papers is an important aim, formal correctness must not suffer from it.

I was fortunate enough to have Malte Helmert as a colleague. He willingly shared his knowledge and experience with me in numerous discussions that were key to many of my publications. Having Malte as a mentor has simplified my research significantly, and I owe him my deepest gratitude. And finally, I would like to single out Patrick Eyerich. I remember countless late nights working side by side with him, trying to make some robot clean up a table or pick up a cup, our planning system produce reasonable policies, or finish up a paper for a deadline that was approaching way faster than what we anticipated. I no doubt have greatly benefited from our collaboration and have grown both professionally and personally. Most importantly, it has always been a great pleasure to work with Patrick, and I am truly grateful for that.

I would also like to thank Florian Geißer, Patrick Eyerich, and Robert Mattmüller for proofreading this thesis very carefully. Their comments have been of extremely high value.

Luckily, life is not always about AI, and I would like to thank all the people who enrich my life in those moments where I am not (primarily) a computer scientist. Spending time with my friends, especially with Alexander Kirschner, Andreas Rau, Daniel Kurreck, Florian Mutschelknaus, Holle Bergmann, Jasper Kittel, Regina Kurreck, Tilman Schieber, and Toffer Risch, has often given me much needed distraction, joy, and happiness.

The last couple of months prior to finishing this thesis have been special in several ways, but most importantly due to the birth of my son Jonathan. Becoming a father has influenced me like nothing before, and it has made my life so much more joyful. However, it also meant additional duties that complicated the process of writing significantly. Without the incredible support of my parents-in-law, Wiebke and Franz, it would not have been possible to finish this thesis. The time I spent with my brother-in-law Philip and his girlfriend Sabine has always been a refreshing experience that allowed me to motivate myself for whatever challenge lay ahead.

My father Wolfgang, my sister Sabrina with her husband Oli and her daughter Emma, and my aunt Hermine and her family have always provided me with unconditional support and love, and I am grateful to have such wonderful persons in my life. I have saved the last word of acknowledgment for my wife Catherine. Due to the sacrifices you were willing to accept, especially in the last couple of months, your merits concerning this thesis cannot be put in words. Thank you for your love, support, patience, and encouragement, and for sharing the best moments of life with me!

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- oder Beratungsdiensten (Promotionsberaterinnen oder Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Freiburg, April 2015 (Thomas Keller)

Contents

CHAPTER 1

Introduction

One of the defining properties of intelligence of autonomous agents is their ability to plan which action to execute in which situation in order to reach a desired situation. The corresponding field of research, automated planning, is one of the fundamental and most widely studied problems in the AI literature. However, most planning formulations make strong assumptions regarding the agent's control over the world. In particular, all effects of executed actions are considered to be deterministic, and there are no exogenous influences that alter the world state independently of the agent's behavior. In applications where computer programs are applied in a controlled environment or where embodied agents are carefully separated from sources of uncertainty, it is not necessary to take unexpected events into account in the decision-making process and such a formulation therefore suffices.

However, this is in stark contrast to the current progress in the development of autonomous agents. Household robots are about to clean up kitchens or do the laundry, self-driving cars bring passengers to their desired destination, autonomous aircrafts deliver urgent parcels, and smart homes interact with the energy market or order groceries for their inhabitants. All of these "next-generation" applications that involve intelligent, autonomous agents have in common that the agents operate in the real world. In order to make sure that the technological progress is a safe progress for us – the people who share their environment with autonomous agents – it is necessary that the agents are able to consider the possibility that unexpected events occur.

However, planning in a complex, uncertain environment is a challenging task. Early attempts take all contingencies of all actions into account and compute a complete policy that describes the best action in each possible situation. Even though such a policy leads to optimal behavior upon execution, it is intractable in practice, as the curse of dimensionality leads to a prohibitively large number of situations an agent can face. Online algorithms, on the other hand, incorporate a concept that allows their computation in practice: they spend some time to plan for the situation the agent faces at a given moment

and execute the taken decision thereafter. By repeating this process in each subsequent state, an agent that is equipped with an online algorithm is able to restrict the policy computation only to states that are actually encountered during execution. Moreover, an agent that plans online can adapt to altered circumstances easily.

Determinization-based replanning approaches are a popular strategy to compute online policies. They are based on the idea that all non-determinism is removed from the representation of the agent's environment, and a planner from the more thoroughly studied field of classical planning is used to derive a plan in the determinization. Most of these replanning algorithms use a determinization that preserves all possible outcomes by replacing each action with a set of actions (one for each outcome). The state-of-the-art method to compute such an all-outcomes determinization suffers from the problem that the number of actions can get out of hand easily in the presence of parallel probabilistic effects. We describe a method that allows the computation of all-outcomes determinizations in polynomial time and space in Chapter 3. There are not only different techniques to compute determinizations, but also a wide variety of algorithms that base their decisions exclusively on computing and solving determinizations and applying the solution to the original problem. In Chapter 4, we apply three such algorithms to a path planning problem with stochastic information about the roadmap. The problem resembles the task that is faced by a robot who wishes to navigate in unknown terrain with access only to a map that is derived from a (blurry) satellite image. An empirical evaluation shows significant improvement over the state-of-the-art, which is based on the assumption that all parts of the roadmap are traversable unless the agent knows differently.

In the main part of this thesis, which spans the Chapters 5 through 8, we introduce the Trial-based Heuristic Tree Search framework. It is a framework for online algorithms that allows the specification of a large number of algorithms for planning under uncertainty, including some that are well-known and many novel ones. Algorithms in the Trial-based Heuristic Tree Search framework are not based on a determinization (they use them for heuristic guidance, though), but it suffices nonetheless to consider only a fraction of the reachable states in the planning process to derive a near-optimal decision after short deliberation time. We show that many algorithms in the framework are even anytime optimal, which means that the decision making process can be interrupted at any time, but converges towards optimal behavior in the limit. Combined, this allows for agents that adapt their deliberation time dynamically and in dependence of the urgency of the task at hand. Our development focuses on two aspects of algorithms in the Trial-based Heuristic Tree Search framework: the used action selection, which is the most important tool that defines which states are relevant for the current decision; and the backup function, which describes how information that is collected in the planning process is combined to take an informed decision.

Our implementation of the Trial-based Heuristic Tree Search framework has been published as the probabilistic planning system PROST. It is publically available at `http://www.prost.informatik.uni-freiburg.de`. PROST has participated at the International Probabilistic Planning Competitions in 2011 and 2014 with two different Trial-based Heuristic Tree Search algorithms, and has demonstrated the potential of the framework by winning both competitions. The final part of this thesis is on the MDP-Evaluation Stopping Problem, a variant of an optimal stopping problem that arose at the competition in 2014. We show that it can be modelled as a meta-MDP and demonstrate that it is possible to evaluate an MDP solver such that its expected reward is higher than the expected reward of an optimal solver if the MDP-Evaluation Stopping Problem is taken into consideration.

1.1 Contributions

The key contributions of this thesis in the field of planning and acting under uncertainty are as follows:

- Chapter 3 describes a method that computes an all-outcomes determinization of a finite-horizon, factored MDP. In contrast to the state-of-the-art, our approach is polynomial in time and space and can also be applied to problems that contain parallel probabilistic effects.
- Chapter 4 applies well-known algorithms for MDP planning to a path planning problem with stochastic information about the roadmap and empirically shows significant improvement over the state-of-the-art.
- Chapter 5 proposes the Trial-based Heuristic Tree Search framework, which not only subsumes most well-known algorithms for MDP planning but also allows to derive novel algorithms by mixing and matching ingredients such that the whole is more than the sum of its parts.
- Chapter 6 presents a selection of ingredients for the Trial-based Heuristic Tree Search framework with a focus on action selection and backup functions. Most ingredients are novel or have never been applied to MDP planning before.
- Chapter 7 provides a theoretical analysis of the Trial-based Heuristic Tree Search framework and determines 115 anytime algorithms with asymptotically optimal behavior that can be described with the presented ingredients.
- Chapter 8 evaluates the anytime optimal algorithms on the benchmark suite of the International Probabilistic Planning Competitions 2011 and 2014. Our implementation, the probabilistic planning system PROST,

impressively demonstrates the potential of the Trial-based Heuristic Tree Search framework by winning both competitions.

- Chapter 9 introduces the MDP-Evaluation Stopping Problem as an optimization problem that resembles both MDP planning and optimal stopping problems. An evaluation of the algorithms that are derived from an analysis of practically tractable special cases shows significant improvement over the state-of-the-art.

1.2 Relevant Publications

This thesis is partially based on the following publications:

- Patrick Eyerich, Thomas Keller and Malte Helmert. *High-Quality Policies for the Canadian Traveler's Problem*. In Proceedings of the 24th AAAI Conference on Artificial Intelligence (**AAAI 2010**).
- Thomas Keller and Patrick Eyerich. *A Polynomial All Outcome Determinization for Probabilistic Planning*. In Proceedings of the 21st International Conference on Automated Planning and Scheduling (**ICAPS 2011**).
- Thomas Keller and Patrick Eyerich. PROST: *Probabilistic Planning Based on UCT*. In Proceedings of the 22nd International Conference on Automated Planning and Scheduling (**ICAPS 2012**).
- Thomas Keller and Malte Helmert. *Trial-based Heuristic Tree Search for Finite Horizon MDPs*. In Proceedings of the 23rd International Conference on Automated Planning and Scheduling (**ICAPS 2013**).
- Thomas Keller and Florian Geißer. *Better Be Lucky Than Good: Exceeding Expectations in MDP Evaluation*. In Proceedings of the 29th AAAI Conference on Artificial Intelligence (**AAAI 2015**).

The following papers have been published during the author's doctoral process but are not covered by this thesis:

- Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner and Bernhard Nebel. *Semantic Attachments for Domain-Independent Planning Systems*. In Proceedings of the 19th International Conference on Automated Planning and Scheduling (**ICAPS 2009**).
- Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner and Bernhard Nebel. *Coming Up with Good Excuses: What To Do When No Plan Can be Found*. In Proceedings of the 20th International Conference on Automated Planning and Scheduling (**ICAPS 2010**).

- Thomas Keller, Patrick Eyerich and Bernhard Nebel. *Task Planning for an Autonomous Service Robot*. In Proceedings on the 33rd Annual German Conference on Artificial Intelligence (**KI 2010**).
- Danijel Skocaj, Matej Kristan, Alen Vrecko, Marko Mahnic, Miroslav Janicek, Geert-Jan M. Kruijff, Marc Hanheide, Nick Hawes, Thomas Keller, Michael Zillich and Kai Zhou. *A System for Interactive Learning in Dialogue With a Tutor*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS 2011**).
- Andreas Hertle, Christian Dornhege, Thomas Keller and Bernhard Nebel. *Planning with Semantic Attachments: An Object-Oriented View*. In Proceedings of the European Conference on Artificial Intelligence (**ECAI 2012**).
- Johannes Löhr, Patrick Eyerich, Thomas Keller and Bernhard Nebel. *A Planning Based Framework for Controlling Hybrid Systems*. In Proceedings of the 22nd International Conference on Automated Planning and Scheduling (**ICAPS 2012**).
- Florian Geißer, Thomas Keller and Robert Mattmüller. *Past, Present, and Future: An Optimal Online Algorithm for Single-Player GDL-II Games*. In Proceedings of the 21st European Conference on Artificial Intelligence (**ECAI 2014**).
- Andreas Hertle, Christian Dornhege, Thomas Keller, Robert Mattmüller, Manuela Ortlieb and Bernhard Nebel. *An Experimental Comparison of Classical, FOND and Probabilistic Planning*. In Proceedings of the 37th German Conference on Artificial Intelligence (**KI 2014**).
- Tim Schulte and Thomas Keller. *Balancing Exploration and Exploitation in Classical Planning*. In Proceedings of the 7th Annual Symposium on Combinatorial Search (**SoCS 2014**).
- Florian Geißer, Thomas Keller and Robert Mattmüller. *Delete Relaxations for Planning with State-Dependent Action Costs*. In Proceedings of the 24th International Joint Conference on Artificial Intelligence (**IJCAI 2015**).

1.3 Awards

The following awards have been received during the author's doctoral process for work that is part of this thesis:

- July 2010: **Best Poster Presentation Award** for the paper *High-Quality Policies for the Canadian Traveler's Problem (Extended Abstract)* at **SOCS 2010** in Atlanta, USA (with Patrick Eyerich and Malte Helmert).

- April 2011: **Winner of the Boolean MDP Track** for the probabilistic planning system PROST at the 4th International Probabilistic Planning Competition at **ICAPS 2011** in Freiburg, Germany (with Patrick Eyerich).
- May 2013: **Best Student Paper Award** for the paper *Trial-based Heuristic Tree Search for Finite Horizon MDPs* at **ICAPS 2013** in Rome, Italy (with Malte Helmert).
- May 2014: **Winner of the Boolean MDP Track** for the probabilistic planning system PROST at the 5th International Probabilistic Planning Competition at **ICAPS 2014** in Portsmouth, USA (with Florian Geißer).

CHAPTER 2

Markov Decision Processes

In order to discuss problems that arise when an agent aims to plan and act under uncertainty, we first have to provide a definition of the theoretical framework that is considered in this thesis. The following Section 2.1 introduces *Markov Decision Processes* (MDPs), discusses different models of optimal behavior in MDPs and establishes the one that is used over the course of this thesis. Section 2.2 shows how the MDPs we are interested in can be described compactly by using a factored representation, and how a finite horizon induces an MDP that is acyclic. The concluding Section 2.3 defines what an optimal solution of a finite-horizon, factored MDP looks like, and introduces a simple (yet in practice intractable) baseline procedure for its computation.

2.1 Preliminaries

Sequential planning and acting under uncertainty is a problem that is faced by an agent that interacts with a dynamic environment. In each step of the interaction, the agent receives the state from the environment (we only consider the case of fully observable states here), and the agent decides which action to execute next. The execution of an action changes the state of the environment, and the agent obtains a reward for the performed state transition. The agent's behavior should pursue the objective of maximizing the sum over the obtained reward values. The interaction between an agent and an environment is depicted schematically in Figure 2.1.

Planning and acting under uncertainty is an important research area both in the closely related Reinforcement Learning (RL) and Planning communities, but the focus in the considered scenarios diverges significantly. Both scenarios differ mostly in the a-priori model of the environment that can be accessed by the agent: a *learning* agent has no initial knowledge on how its actions alter the environment and when it can expect a positive reward signal, and it has to improve its behavior by learning from interaction with the environment. A

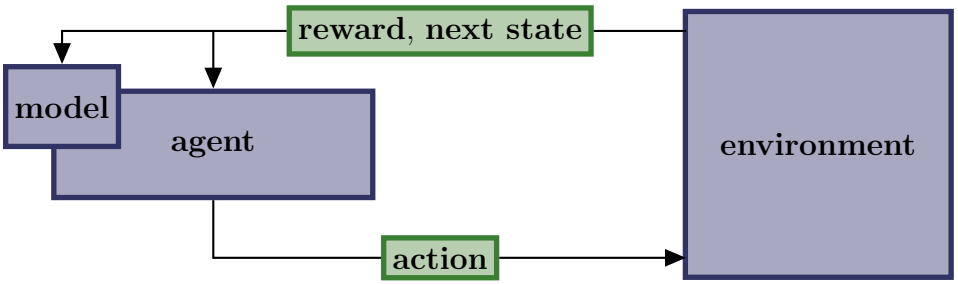

Figure 2.1: Interaction between agent and environment.

planning agent, on the other hand, has a slightly simpler task as it is provided with an initial intuition (a *generative model*) or a clear picture (a *declarative model*) of the influence of his actions on the environment. Even though the focus of this thesis is the planning scenario, we are interested in both due to the influence of popular learning methods on algorithms for probabilistic planning.

The framework that is considered in this thesis is based on the formal concept that is known as a Markov Decision Process (MDP). Literature on MDPs goes as far back as the seminal work of Bellman (1957), and MDPs have been the topic of well-known textbooks (e.g., Puterman, 1994; Bertsekas, 1995) and plenty of research ever since.

Definition 1 (Markov Decision Process). *A* **Markov Decision Process** *is a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$*, where* $\mathcal{S}$ *is a finite set of* **states***;* $\mathcal{A}$ *is a finite set of* **actions***;* $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the* **transition function***, a probability distribution that gives the probability* $\mathbb{P}_{\mathcal{T}}[s' \mid s, a]$ *that applying* $a \in \mathcal{A}$ *in* $s \in \mathcal{S}$ *leads to* $s' \in \mathcal{S}$*;* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [\mathcal{R}^-, \mathcal{R}^+]$ *is the* **reward function** *with* **lower** *and* **upper bound** $\mathcal{R}^-, \mathcal{R}^+ \in \mathbb{R}$*; and* $s_0 \in \mathcal{S}$ *is the* **initial state**.

The transition function of an MDP induces the *successor set* of a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$ as $\text{succ}(s, a) := \{s' \in \mathcal{S} \mid \mathbb{P}_{\mathcal{T}}[s' \mid s, a] > 0\}$. If $\text{succ}(s, a) \neq \varnothing$, we call a *applicable* in s, and we denote the set of applicable actions in s with $\mathcal{A}(s)$. If $\mathcal{A}(s) = \varnothing$, we say that s is a *dead-end*. We call each $s' \in \text{succ}(s, a)$ an *outcome* of a in s, and say that a is *deterministic in* s iff there is exactly one outcome in $\text{succ}(s, a)$, and *probabilistic in* s iff a is applicable in s and not deterministic. If an action a is deterministic in all $s \in \mathcal{S}$ where a is applicable, we call it deterministic, and if all $a \in \mathcal{A}$ are deterministic we say that the MDP is deterministic. Sampling an outcome $s' \in \text{succ}(s, a)$ of a state s and an action a that is applicable in s according to the transition function $\mathcal{T}$ is denoted with $s' \sim \text{succ}(s, a)$, and when we know that a is deterministic in s we write $s' = \text{succ}(s, a)$.

Let us also specify some notational conventions that are used throughout this thesis. When we refer to all state-action pairs in an MDP, we mean all

pairs (s, a) for $s \in \mathcal{S}$ and $a \in \mathcal{A}$ such that a is applicable in s. When we specify a discrete probability distribution D over a set X (like, for instance, a transition function), we give it as $D = \{(p_1 : x_1), \ldots, (p_n : x_n)\}$, and we mean the probability distribution that assigns a probability of p_i with $0 \leq p_i \leq 1$ to $x_i \in X$ for all $i \in \{1, \ldots, n\}$, and a probability of 0 to all $x \in X \setminus \{x_1, \ldots, x_n\}$. When we specify a tuple without mentioning its elements explicitly, we assume implicitly that all its elements share the superscript that is also used in the tuple specification, e. g., an MDP $\mathcal{M}'$ consists implicitly of a set of states $\mathcal{S}'$, a set of actions $\mathcal{A}'$, etc. If no superscript is used, the corresponding elements go without superscript as well. We also regard mathematical objects that share the same superscript as associated with each other if there is a one-to-one relationship between objects of their kind. For instance, the set of policies (see Definition 2) of an MDP $\mathcal{M}^{\mathrm{d}}$ is (implicitly defined as) Π^{d}.

Example 1. *Figure 2.2 depicts an example MDP. As in all MDP figures in this thesis, states are depicted as (blue) rectangles and actions as (green) circles. Actions have exactly one incoming edge that is labeled with the reward of applying the action in the connected state, and one or more outgoing edges that are connected with possible outcomes and labeled with the probability that the corresponding outcome takes place (rewards of* 0 *and probabilities of* 1 *are omitted). The MDP* $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$ *that is shown in Figure 2.2 is described by*

- *a set of states* $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$*;*
- *a set of actions* $\mathcal{A} = \{a_1, a_2, a_3\}$*;*
- *and a transition function* $\mathcal{T}$ *and a reward function* $\mathcal{R}$ *that are such that*
  - $\mathcal{T}(s_0, a_1) = \{(\frac{1}{3} : s_0), (\frac{2}{3} : s_3)\}$ $\quad \mathcal{R}(s_0, a_1) = -1$
  - $\mathcal{T}(s_0, a_2) = \{(\frac{1}{2} : s_1), (\frac{1}{2} : s_2)\}$ $\quad \mathcal{R}(s_0, a_2) = +1$
  - $\mathcal{T}(s_1, a_1) = \{(\frac{1}{3} : s_1), (\frac{2}{3} : s_3)\}$ $\quad \mathcal{R}(s_1, a_1) = +6$
  - $\mathcal{T}(s_1, a_3) = \{(1 : s_0)\}$ $\quad \mathcal{R}(s_1, a_3) = +4$
  - $\mathcal{T}(s_2, a_1) = \{(\frac{1}{3} : s_2), (\frac{2}{3} : s_3)\}$ $\quad \mathcal{R}(s_2, a_1) = -2$
  - $\mathcal{T}(s_2, a_3) = \{(1 : s_0)\}$ $\quad \mathcal{R}(s_2, a_3) = +2$
  - $\mathcal{T}(s_3, a_1) = \{(1 : s_3)\}$ $\quad \mathcal{R}(s_3, a_1) = +1.$

The behavior of an agent is described in terms of a *policy*. Puterman (1994) defines a policy as a function that maps states and the whole interaction history of the agent to a probability distribution over applicable actions, which is sufficient for most scenarios involving MDPs one can come up with. In this thesis, we prefer a more restricted and simpler definition, where we require policies to be *deterministic* and *stationary*. A deterministic policy does not select an action according to a probability distribution but always picks the same action given the same input. And a stationary policy does not use

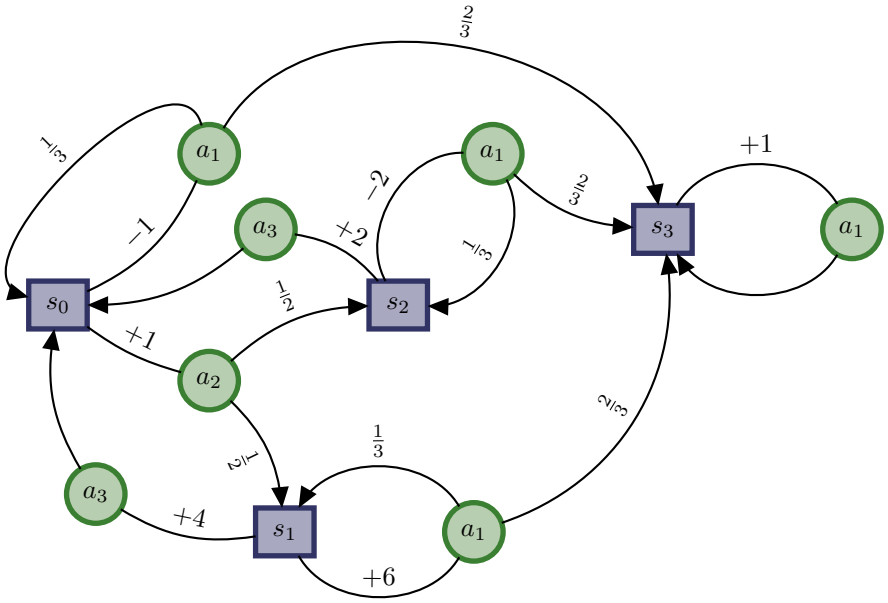

Figure 2.2: An example MDP with 4 states and 3 actions.

the agent's history to make a decision, but always selects the same action in the same state independently from the path that leads to the state.

Definition 2 (Policy). *Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$, a* **policy** *is a mapping $\pi : \mathcal{S} \to \mathcal{A} \cup \{\bot\}$ such that $\pi(s) \in \mathcal{A}(s) \cup \{\bot\}$ for all $s \in \mathcal{S}$. Let $\mathcal{S}^\pi(s)$ be the set of* **reachable states** *from s under π, which is defined recursively as the smallest set satisfying the rules $s \in \mathcal{S}^\pi(s)$ and $\text{succ}(s', \pi(s')) \subseteq \mathcal{S}^\pi(s)$ for all $s' \in \mathcal{S}^\pi(s)$ where $\pi(s') \neq \bot$. We call π* **executable in** *s iff $\pi(s') \neq \bot$ for all $s' \in \mathcal{S}^\pi(s)$, and* **executable** *iff π is executable in s_0. A policy π where $\pi(s) \neq \bot$ for all $s \in \mathcal{S}$ is called* **complete***. We denote the set of all policies in an MDP with* Π.

We allow a policy to be undefined on some states, and call it *executable* if it is guaranteed that no state can be reached during its execution where it is undefined. Since complete policies are defined for all states, they are also executable in all states.

Example 2. *Consider the following state-action mappings for our example MDP:*

- *π_1 selects an applicable action uniformly at random in each state.*
- *π_2 is such that $\pi_2(s_0) = a_2$, $\pi_2(s_1) = a_3$, and $\pi_2(s_2) = a_3$.*
- *π_3 is such that $\pi_3(s) = a_1$ for all $s \in \mathcal{S}$.*
- *π_4 is such that $\pi_4(s_0) = a_2$.*

Apart from π_1, which is not deterministic, all of these mappings are policies in the sense of Definition 2. Policy π_2 is not complete but executable since $s_3 \notin \mathcal{S}^{\pi_2}(s_0)$ and $\pi_2(s) \neq \bot$ for all other states s, and π_3 is complete and hence also executable. Finally, π_4 is neither complete nor executable since $\pi_4(s_1) = \bot$ and $s_1 \in \mathcal{S}^{\pi_4}(s_0)$.

The interaction between an agent and the environment is formalized as a *run* of finite or infinite length. It is derived by applying the policy that describes the agent's behavior, starting from the current state of the environment. In each state that is encountered, a finite reward value according to the MDP's reward function is obtained. A run yields an accumulated reward that corresponds to the sum of the obtained reward values.

Definition 3 (Agent-Environment Interaction). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0, \rangle$ be an MDP and π be a policy that is executable in s_0. A* **run** *of length $n \in \mathbb{N} \cup \{\infty\}$ under π is a sequence of states and actions $\phi^\pi = (s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$ such that $a_t = \pi(s_t)$ and $s_{t+1} \sim \text{succ}(s_t, a_t)$ for all $t \in \{0, \ldots, n-1\}$. The* **accumulated reward** *of a run ϕ^π of finite length n is $\mathcal{R}(\phi^\pi) := \sum_{t=0}^{n-1} \mathcal{R}(s_t, a_t)$.*

Example 3. *Consider the following runs in our example MDP:*

- $\phi^{\pi_2} = (s_0, a_2, s_1, a_3, s_0)$
- $\phi_1^{\pi_3} = (s_0, a_1, s_0, a_1, s_0, a_1, s_0, a_1, s_0)$
- $\phi_2^{\pi_3} = (s_0, a_1, s_0, a_1, s_3, a_1, s_3, a_1, s_3, \ldots)$

ϕ^{π_2} is a run of length 2 under π_2 with accumulated reward $\mathcal{R}(\phi^{\pi_2}) = +5$, and $\phi_1^{\pi_3}$ and $\phi_2^{\pi_3}$ are two different runs under π_3. The length of $\phi_1^{\pi_3}$ is 4 and its accumulated reward is $\mathcal{R}(\phi_1^{\pi_3}) = -4$, and the accumulated reward of $\phi_2^{\pi_3}$ is not defined since it is of infinite length.

Initially, we have described that an agent should aim for a behavior that pursues the objective of maximizing the sum over the obtained reward values. Before we can turn the description into a well-defined *model of optimal behavior* (Kaelbling et al., 1996), which describes the criterion that specifies an *optimal policy*, there are some decisions to be made. In particular, we have to decide how an agent takes future rewards into account in the decisions it is about to make now. An important influence on this decision is the considered *horizon*. It describes a criterion for the termination – or, if infinite, non-termination – of a policy's execution, and hence the number of action applications that are considered to determine the quality of a policy. We compare four different models of optimal behavior in the following. For illustration, we show their assessment of the policies π_2 and π_3 from Example 2. The results are summarized in Table 2.1. We think it is intuitive to state that the quality of policy π_3 should be higher than the quality of policy π_2, since π_3 continually traverses the two cycles s_0 – s_1 – s_0 and s_0 – s_2 – s_0 which yield a reward of $+5$ and $+3$ over the course of two steps, respectively. Policy π_3, on the other

Optimality Model	π_2	π_3
Undiscounted Infinite Horizon	∞	∞
Discounted Infinite Horizon	≈ 10	≈ 2.27
Average Reward	2	1
Finite Horizon	40	≈ 17

Table 2.1: Summary of the expected quality evaluations of the policies π_2 and π_3 of our example MDP in the discussed models of optimal behavior. A discount factor of $\gamma = 0.8$ (for the values in the second column) and a finite horizon of $\mathcal{H} = 20$ (for the values in the last column) are assumed.

hand, receives a reward of -1 in each step as long as it remains in s_0, and even after it proceeds to s_3 it only collects a reward of $+1$ in each step.

Let us first consider the case of an infinite horizon, and let us assume that we aim to execute the policy that maximizes the *expected accumulated reward*

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \mathcal{R}(s_t, a_t)\right]$$

that can be achieved in a run $\phi^\pi = (s_0, a_0, s_1, a_1, s_2, \dots)$. However, if there is a positive reward cycle (e. g., a set of states where all paths lead back to the same state, and where the sum of obtained rewards is greater than zero in all paths) in the MDP, all policies that have a non-zero chance that the agent enters that cycle and remains in it with probability 1 will be considered equally good as the expected sum over the obtained reward values of those policies is infinite. In our example MDP, this is the case both for π_2 (which ends up in s_0 every other step regardless of action outcomes) and for π_3 (which ends up in s_3 eventually, where only the deterministic action a_1 that leads back to s_3 is applicable). Therefore, both π_2 and π_3 have an infinite expected accumulated reward and are considered equally good in terms of the proposed model of optimal behavior.

Unfortunately, this contradicts our initial intuition. Well-defined models of optimal behavior therefore ensure a finite optimization quantity. Surprisingly, the model that is considered most often in the context of MDPs considers an infinite horizon nonetheless. Instead of limiting the horizon to a finite value, it is such that the influence of decisions decreases the further they are in the future. To obtain this, a *discount factor* γ with $0 < \gamma < 1$ is used to discount rewards such that they have less influence the further they are obtained in the future. The optimization criterion is then the maximization of the *expected discounted reward*

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathcal{R}(s_t, a_t)\right]$$

of a run ϕ^π of infinite length. If rewards are discounted with a discount factor of, for instance, $\gamma = 0.8$, the quality of our example policy π_2 converges to a value that is close to 10, while π_3 only achieves a quality of roughly 2.27 – a result where π_2 is superior to π_3, just as one would expect.

While the relative order of the quality of policies with an expected discounted reward is as desired, the quality values themselves are fairly unintuitive. An alternative way that also achieves a finite optimization criterion in an infinite horizon setting is the maximization of the *average reward*

$$\lim_{n\to\infty} \mathbb{E}\left[\frac{1}{n}\cdot\sum_{t=0}^{n-1}\mathcal{R}(s_t,a_t)\right]$$

that can be achieved by the agent in a run ϕ^π. The average reward of policy π_2 in our example MDP is 2 because half of all typical runs are described by the sequence $(s_0, a_2, s_2, a_3, s_0)$ with an average reward of $\frac{3}{2}$, and the other half can be expected to correspond to the sequence $(s_0, a_2, s_1, a_3, s_0)$ with an average reward of $\frac{5}{2}$. The average reward of policy π_3 is dominated by the endless application of a_1 in s_3, since applying a_1 in s_0 will yield to the outcome s_3 after a finite number of steps, and from then on the reward will always be 1 (and the average reward hence converges towards 1).

Unfortunately, the *evaluation* of policies that are achieved by maximizing the expected discounted reward or the average reward is difficult. The computation of both quality values is intractable in non-trivial MDPs, and the *simulation* of a run is obviously impossible if an infinite horizon is considered. We therefore prefer an approach where a *finite horizon* $\mathcal{H}$ specifies the number of future decisions that are considered in the current decision, i. e., where a policy is considered to be optimal iff it maximizes the *finite expected accumulated reward* (or simply *expected reward* in the following)

$$\mathbb{E}\left[\sum_{t=0}^{\mathcal{H}-1}\mathcal{R}(s_t,a_t)\right]$$

of a run (ϕ^π) of length $\mathcal{H}$. This allows us to approximate the expected reward of a policy π given a (sufficiently large) sequence of runs $(\phi_1^\pi, \ldots, \phi_n^\pi)$ under π of length $\mathcal{H}$ as the average of all accumulated rewards, since

$$\lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n}\mathcal{R}(\phi_k^\pi) = \mathbb{E}\left[\mathcal{R}(\phi^\pi)\right],$$

due to the law of the large numbers.

This is the simplest and – in our opinion – also a very elegant model of optimal behavior that both ensures a well-defined optimization criterion and allows the simulation of runs to approximate the quality of a policy. If we assume a finite horizon of $\mathcal{H} = 20$, the expected reward of applying policy

π_2 is 40 for the same reason that the average reward in each step is 2, while it is approximately 17 for π_3. Note that there is an alternative where the specification of a set of *goal states* that must be reached allows the usage of undiscounted accumulated rewards as well. However, in order to obtain a well-defined optimality criterion for such a Stochastic Shortest Path problem (Bertsekas and Tsitsiklis, 1991), the set of goal states must be absorbing (a state s is absorbing iff $\text{succ}(s, a) = s$ for all $a \in \mathcal{A}(s)$) and the reward function such that $\mathcal{R}(s, a) = 0$ for all goal states s and applicable actions a. Additionally, the Stochastic Shortest Path problem has to be acyclic or restricted to *costs*, i. e., have a reward function where $\mathcal{R}(s, a) \leq 0$ for all state-action pairs. Also, the existence of a policy that reaches a goal state with certainty in a finite number of steps must be guaranteed to exclude infinite expected costs. The finite-horizon setting can be used to model Stochastic Shortest Path problems by assigning a reward of 0 to the application of an action in a goal state, and of the applied operator's cost to all other state-action pairs (Condon, 1992). Of course, a reasonable bound on the horizon $\mathcal{H}$ which is such that the goal can be reached within $\mathcal{H}$ steps with certainty must be known.

2.2 Compact Description

Since learning is complex already in small MDPs, a lot of early research in RL considers an MDP syntax that corresponds to the flat representation that is also used in Example 1 where states and transitions between states are listed explicitly (e.g., Bellman, 1957; Howard, 1960). An alternative is the description of MDPs with large state spaces in a compact manner as *Factored MDPs* (Boutilier et al., 2000). Factored MDPs incorporate an idea that has been central to most research in the planning community already when STRIPS (Fikes and Nilsson, 1971) was designed as a modeling language for deterministic planning problems. By lifting the domain to a compact description, it is possible to describe large state spaces syntactically with exponentially smaller space requirements. The transition function can be described compactly with actions that allow a systematic computation of the probability distribution over successor states. In combination with the finite horizon that is required by our model of optimal behavior, these are used to induce an MDP from a *finite-horizon, factored MDP*.

2.2.1 Variables and States

Helmert (2008) describes a formalism that is based on SAS$^+$ (Bäckström and Nebel, 1995), where *finite-domain variables* replace a representation that is restricted to Boolean variables. Mattmüller (2013) adapts the formalism to a nondeterministic environment, which we extend in the following to fit the probabilistic setting.

Definition 4 (Finite-domain Variable). *A* **finite-domain state variable** *(or simply* **variable***) is a mathematical object v, associated with a finite set of values, the* **domain** $\mathcal{D}_v$ *of v. We call the set $\mathcal{D}_v^+ := \mathcal{D}_v \cup \{\bot\}$ the* **extended domain** *of v, where $\bot \notin \mathcal{D}_v$ is the* **undefined** *value.*

Since we only consider fully observable environments in this thesis, states can be described as valuations of variables over a given set of finite-domain variables.

Definition 5 (State). *A* **partial variable assignment** *over a finite set of variables $\mathcal{V}$ is a mapping $s : \mathcal{V} \to \bigcup_{v \in \mathcal{V}} \mathcal{D}_v^+$ such that $s[v] \in \mathcal{D}_v^+$ for all $v \in \mathcal{V}$. The* **scope** *of s is the set of variables where s does not have the undefined value $\bot$; it is denoted as* $\text{vars}(s) := \{v \in \mathcal{V} \mid s[v] \neq \bot\}$. *A partial variable assignment s is called a* **state** *iff* $\text{vars}(s) = \mathcal{V}$.

Often, we use variables with a domain that consists of the first n natural numbers $\mathcal{D}_v = \{0, \ldots, n\} \subset \mathbb{N}$. Even though arithmetic operations on variable values are not considered in principle, we use addition and subtraction for abbreviation purposes in cases where it is obvious that the result of the operation is such that it remains in the variable's domain. For *binary* variables with domain $\mathcal{D}_v = \{\texttt{true}, \texttt{false}\}$, we furthermore simplify the notation of variable assignments and write v rather than $v = \texttt{true}$ and $\neg v$ instead of $v = \texttt{false}$.

The finite horizon that is considered in our model of optimal behavior leads to the unfortunate property that optimal behavior not only depends on the state but also on the *depth* of a state s (the number of action applications that were necessary to reach s from s_0) or, alternatively, on the *steps-to-go* (the horizon $\mathcal{H}$ reduced by the depth). For instance, in the MDP from Example 1, it is preferable to execute action a_3 in s_1 if the horizon is large since the positive reward cycle between s_0 and s_1 or s_2 promises a better long-term reward than applying a_1 in s_3. However, if there are only one or two steps-to-go, it is better to go with the reward of $+6$ that is gained when a_1 is applied in s_1 than with the immediate reward of $+4$ that is gained when action a_3 is applied instead.

Not only policies must take the steps-to-go into account, but many definitions and algorithms that can be found in this thesis have to consider it *in addition* to a state, often withdrawing attention from what is truly important. We take an alternative approach where we require a *steps-to-go variable* h as part of the set of variables that is used to describe the set of states of an MDP compactly. Its domain is $\mathcal{D}_h = \{0, \ldots, \mathcal{H}\}$, it equals $\mathcal{H}$ in the initial state and its value decreases by one in each action application. We call a state s where $s[h] = 0$ a *terminal state*. Furthermore, it is sometimes more convenient to describe a state s in terms of its depth, which we denote with $s[d]$ and which we define in terms of the steps-to-go variable as $s[d] := \mathcal{H} - s[h]$.

Example 4. *One advantage of factored MDPs is that they allow the specification of complex environments in a compact manner. Let us therefore use a more*

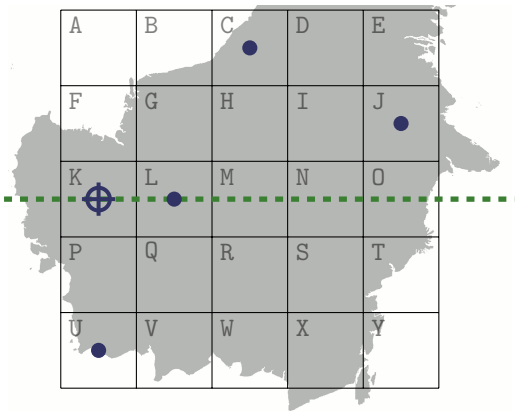

Figure 2.3: Initial state of the EARTHOBSERVATION example instance.

elaborate example domain for this part of the thesis. In the EARTHOBSERVATION *domain, which has been part of our experimental evaluation of different planning formalisms (Hertle et al., 2014), there is a satellite that surrounds Earth on a circular, non-stationary orbit that is not inclined and with a constant orbit height. Its ground track is equal to the equator, which is traversed with a constant ground speed from west to east (the ground speed covers both the satellite's relative movement and the rotation of earth). The agent's task is to take pictures of predefined observation patches within a region of interest with an on-board camera.*

The camera is nadir-pointed in its base orientation, i.e., such that it focuses on the projection of the satellite position on the ground track. To take images of patches that are north or south of the equator, the satellite can move its camera orthogonally to the direction of flight. Moving the camera north or south therefore means to slew it such that the instrument's line of sight moves north or south. The camera cannot be slewed in or against the satellite's direction of movement, and its focal point – the intersection of its line of sight and the surface of earth – will hence automatically move eastwards with constant speed due to the (uncontrollable, external) movement of the satellite from west to east. Since the satellite surrounds earth, the landscape is wrapped around a cylindrical projection of the earth surface, i. e., we end up at the western end of the map after leaving it in the east.

An example state with a region of interest that covers the largest part of the island of Borneo is depicted in Figure 2.3. (We chose Borneo for its fairly identifiable coastal line and the fact that it is separated by the equator in two more or less equally sized halves.) To simplify the task, we represent the area of interest as a grid. The satellite's ground track is depicted as a green, dashed line, and the grid cell that is marked with the blue crosslines is the initial focal point of the on-board camera. The small blue circles mark grid cells that contain an observation patch. We refer to each grid cell by the letter in the upper left corner

of Figure 2.3 – e. g., the grid cell that is the initial focal point of the camera, is referred to as grid cell K*, and the set of all grid cells is* $\mathcal{Z} = \{\mathtt{A}, \ldots, \mathtt{Y}\}$.

The set of variables $\mathcal{V}$ *that is used to model a state in an instance of the* EARTHOBSERVATION *domain contains*

- *a single variable* `cam-pos` *that describes the current focal point of the camera with domain* $\mathcal{D}_{\texttt{cam-pos}} = \mathcal{Z}$*;*
- *for each grid cell* $z \in \mathcal{Z}$ *a binary variable* `target-`z *that describes if the respective grid cell is a remaining observation patch (e. g.,* $s[\texttt{target-}C]$ *is true iff grid cell* C *is a remaining observation patch in state* s*); and*
- *with a planning horizon of* $\mathcal{H} = 20$*, the steps-to-go variable* $h \in \mathcal{V}$ *is such that* $\mathcal{D}_h = \{0, \ldots, 20\}$.

The state s_0 *that is depicted in Figure 2.3 is described with the variables in* $\mathcal{V}$ *as*

$$\begin{aligned} s_0 = &\{\texttt{cam-pos} = \mathtt{K}, \texttt{target-}C, \texttt{target-}J, \texttt{target-}L, \texttt{target-}U\} \cup \\ &\{\neg\texttt{target-}z \mid z \in \mathcal{Z} \setminus \{\mathtt{C}, \mathtt{J}, \mathtt{L}, \mathtt{U}\}\} \cup \{h = 20\}. \end{aligned}$$

2.2.2 Actions

Variables not only allow the compact description of states, but also of transitions between states that are given indirectly by *actions* that consist of a *precondition* and an *effect*. The formalism that is presented in the following is inspired from PPDDL (Younes and Littman, 2004), the probabilistic dialect of PDDL (McDermott et al., 1998) that has been the input language of the IPPC prior to 2011. However, we do not provide a mapping of PPDDL keywords to the structure of effects, but we believe that the mapping is trivial. Also, it should be mentioned that PPDDL allows the description of a large number of actions with a schematic description that contains variables that are instantiated with provided constants (objects) to obtain actions like those that are considered here. Since all algorithms in this thesis are defined in terms of instantiated actions, we provide definitions for instantiated formulas only. Apart from the partial observability, we follow the definition of Rintanen (2003) for effects of actions:

Definition 6 (Action). *An* **action** *over a set of variables* $\mathcal{V}$ *is a tuple* $a = \langle \text{pre}, \text{eff} \rangle$*, where* pre *is a Boolean formula over* $\mathcal{V}$ *and* eff *is an* **effect** *that is recursively defined by the following rules:*

- $\top$ *is the* **empty** *effect;*
- $v \leftarrow x$ *for* $v \in \mathcal{V}, x \in \mathcal{D}_v$ *is an* **atomic** *effect;*
- $e \wedge e'$ *is a* **conjunctive** *effect if* e *and* e' *are effects;*

- $c \rhd e$ *is a* **conditional** *effect if* c *is a Boolean formula over* $\mathcal{V}$ *and* e *is an effect; and*

- $(p_1 : e_1) \mid \cdots \mid (p_n : e_n)$ *is a* **probabilistic** *effect if* $e_1, \ldots, e_n$ *are effects,* $0 < p_1, \ldots, p_n \leq 1$*, and* $\sum_{i=1}^{n} p_i = 1$*.*

We use the same abbreviations for effects on binary variables that were introduced for variable assignments earlier and write v and $\neg v$ instead of $v \leftarrow \texttt{true}$ and $v \leftarrow \texttt{false}$. The precondition of an action $a = \langle \text{pre}, \text{eff} \rangle$ is used to encode the applicability of a. We say that an action a is applicable in a state s if $s \models \text{pre}$. The only constraint we impose on preconditions is that the resulting MDP must be *dead-end free*, i. e., it may not be the case that a state can be reached where no action is applicable. As the detection of dead-ends is hard and not the focus of this thesis, we assume in the remainder of this work that a problem specification is well-defined in the sense that it leads to a dead-end free MDP. We furthermore assume that the Boolean formula that encodes a precondition is such that the computational cost of its evaluation in a state is negligible. It is then possible to compute the set of applicable actions $\mathcal{A}(s)$ in a state s efficiently (as long as $|\mathcal{A}|$ is manageable, which we assume). An effect that does not contain a probabilistic effect is *deterministic*. When an action assigns the value that variable $v' \in \mathcal{V}$ assumes in the current state to another variable $v \in \mathcal{V}$ (with $\mathcal{D}_{v'} \subseteq \mathcal{D}_v$), we use the abbreviation $v \leftarrow s[v'] := \bigwedge_{x \in \mathcal{D}_{v'}} (v' = x) \rhd v \leftarrow x$.

Example 5. *Our description of the* EARTHOBSERVATION *domain already mentions all kinds of actions the satellite can take: it must be able to take images of observation patches, and it has the ability to slew the camera to the north or south. The uncertainty in the* EARTHOBSERVATION *domain stems from a failure probability for all of these maneuvers. In the case that a failure happens, the successor state does not reflect the desired outcome of the taken action and only the exogenous event that moves the satellite's focal point to the east takes place. In the formal description, we use the following three functions to describe connections between grid cells:*

- $\texttt{east} : \mathcal{Z} \to \mathcal{Z}$ *is such that for all* $z \in \mathcal{Z}$*,* $\texttt{east}(z)$ *is the cell that is the neighbor cell to the east of* z *(e. g.,* $\texttt{east}(\texttt{H}) = \texttt{I}$*)*

- $\texttt{south} : \mathcal{Z} \to \mathcal{Z} \cup \{\bot\}$ *is such that* $\texttt{south}(z) = \bot$ *for* $z \in \{\texttt{U}, \ldots, \texttt{Y}\}$*, and the cell to the south-east of* z *otherwise (e. g.,* $\texttt{south}(\texttt{M}) = \texttt{S}$*)*

- $\texttt{north} : \mathcal{Z} \to \mathcal{Z} \cup \{\bot\}$ *is such that* $\texttt{north}(z) = \bot$ *for* $z \in \{\texttt{A}, \ldots, \texttt{E}\}$*, and the cell to the north-east of* z *otherwise (e. g.,* $\texttt{north}(\texttt{J}) = \texttt{A}$*)*

We distinguish between four types of actions:

- *For each $z \in \mathcal{Z}$, there is an action* noop-z *that lets the satellite be idle for one step (this has no effect apart from the automatic movement of the focal point from west to east). There is no restriction on the applicability of an action* noop-z *apart from the fact that the current focal point of the camera must be z.*

- *For each $z \in \mathcal{Z}$, there are actions* slew-north-z *and* slew-south-z *that slew the camera to the north or south if the execution is successful, and they have the same effect as the corresponding* noop-z *otherwise. The actions are applicable if the current focal point of the camera is* z *and if there is a grid cell in the corresponding direction of the current focal point.*

- *For each $z \in \mathcal{Z}$, there is an action* take-image-z *that takes a picture if execution is successful, and it is idle in the failure case.*

The preconditions of these actions are, for all $z \in \mathcal{Z}$, formalized as follows:

$$\begin{aligned}
\text{pre}(\texttt{noop-}z) &:= \texttt{cam-pos} = z \\
\text{pre}(\texttt{slew-north-}z) &:= \texttt{cam-pos} = z \wedge \neg(\texttt{north}(z) = \bot) \\
\text{pre}(\texttt{slew-north-}z) &:= \texttt{cam-pos} = z \wedge \neg(\texttt{south}(z) = \bot) \\
\text{pre}(\texttt{take-image-}z) &:= \texttt{cam-pos} = z \wedge \texttt{target-}z
\end{aligned}$$

As each action requires that the focal point of the camera is a certain grid cell, all preconditions are such that for all states there is at most one action of each type that is applicable (namely the one that is indexed with the focal point of the camera). This allows us to omit the index in the following whenever the camera's current focal point is clear from the context. The corresponding effects for all $z \in \mathcal{Z}$ are modeled as:

$$\begin{aligned}
\text{eff}(\texttt{noop-}z) &:= \texttt{cam-pos} \leftarrow \texttt{east}(z) \\
\text{eff}(\texttt{slew-north-}z) &:= (0.9 : \texttt{cam-pos} \leftarrow \texttt{north}(z)) \mid (0.1 : \texttt{cam-pos} \leftarrow \texttt{east}(z)) \\
\text{eff}(\texttt{slew-south-}z) &:= (0.9 : \texttt{cam-pos} \leftarrow \texttt{south}(z)) \mid (0.1 : \texttt{cam-pos} \leftarrow \texttt{east}(z)) \\
\text{eff}(\texttt{take-image-}z) &:= (0.8 : (\texttt{cam-pos} \leftarrow \texttt{east}(z) \wedge \neg\texttt{target-}z)) \mid \\
&\qquad (0.2 : \texttt{cam-pos} \leftarrow \texttt{east}(z))
\end{aligned}$$

An additional effect of all actions that is not listed here is the decrease of the steps-to-go variable by one. All effects eff(a) *that are given above are hence assumed to be* eff$(a) \wedge h \leftarrow (s[h] - 1)$.

Rintanen (2003) also defines several normal forms for actions, two of which are used in Chapter 3 and hence briefly repeated here. We say that an effect e is in *unary conditionality normal form* if e' is atomic for all $c \rhd e'$ in e, and it is in *unary non-determinism normal form* if e is of the form $(p_1 : e_1) \mid \cdots \mid (p_n : e_n)$ with deterministic $e_1, \ldots, e_n$. For every effect e there is an equivalent

one in unary conditionality normal form. It can be constructed by replacing the left-hand-side of the equivalences on effects

$$c \rhd (e_1 \wedge \cdots \wedge e_n) \equiv (c \rhd e_1) \wedge \cdots \wedge (c \rhd e_n) \tag{2.1}$$
$$c \rhd (c' \rhd e) \equiv (c \wedge c') \rhd e \tag{2.2}$$
$$c \rhd ((p_1 : e_1) \mid \cdots \mid (p_n : e_n)) \equiv (p_1 : c \rhd e_1) \mid \cdots \mid (p_n : c \rhd e_n) \tag{2.3}$$

with the right-hand-side until e is in unary conditionality normal form, a transformation that is possible in polynomial time and space according to Rintanen (2003). An effect e can be translated into unary non-determinism normal form by first translating it to unary conditionality normal form, and then by replacing the left-hand-side of the equivalences on effect formulas

$$e \wedge ((p_1 : e_1) \mid \cdots \mid (p_n : e_n)) \equiv (p_1 : e \wedge e_1) \mid \cdots \mid (p_n : e \wedge e_n) \tag{2.4}$$
$$\begin{aligned} &\big(p_1 : \big[\big(p'_1 : e'_1\big) \mid \cdots \mid \big(p'_j : e'_j\big)\big]\big) \mid (p_2 : e_2) \mid \cdots \mid (p_n : e_n) \equiv \\ &\qquad \big(p_1 \cdot p'_1 : e'_1\big) \mid \cdots \mid \big(p_1 \cdot p'_j : e'_j\big) \mid (p_2 : e_2) \mid \ldots \mid (p_i : e_i) \end{aligned} \tag{2.5}$$
$$\begin{aligned} &\Big((p_1 : e_1) \mid \cdots \mid (p_n : e_n) \Big) \wedge \Big(\big(p'_1 : e'_1\big) \mid \cdots \mid \big(p'_k : e'_k\big) \Big) \equiv \\ &\qquad \big(p_1 \cdot p'_1 : e_1 \wedge e'_1\big) \mid \cdots \mid \big(p_n \cdot p'_k : e_n \wedge e'_k\big) \end{aligned} \tag{2.6}$$

with the right-hand-side until e is in unary non-determinism normal form. However, the compilation of an effect formula to unary non-determinism normal form can increase the size of the resulting effect formula exponentially due to the application of Equivalence 2.6.

Before we continue with a compact representation of the reward function, we show how the transition function $\mathcal{T}$ of an MDP $\mathcal{M}$ can be derived from a set of actions. Let the effect set $[e]_s$ of an effect e in state s over $\mathcal{V}$ be a set of pairs $\langle p, w \rangle$, where p is a probability $0 < p \leq 1$ and w is a partial variable assignment. The effect set $[e]_s$ is obtained recursively as

$$[\top]_s = \langle 1.0, \varnothing \rangle, \tag{2.7}$$
$$[v \leftarrow x]_s = \langle 1.0, \{v \leftarrow x\} \rangle, \tag{2.8}$$
$$[e \wedge e]_s = \bigcup_{\langle p,w \rangle \in [e]_s} \bigcup_{\langle p',w' \rangle \in [e']_s} \{\langle p \cdot p', w \cup w' \rangle\}, \tag{2.9}$$
$$[c \rhd e]_s = \begin{cases} [e]_s & \text{if } s \models c \\ \langle 1.0, \varnothing \rangle & \text{otherwise, and} \end{cases} \tag{2.10}$$
$$[(p_1 : e_1) \mid \cdots \mid (p_n : e_n)]_s = \bigcup_{i=1}^{n} \{\langle p_i \cdot p, w \rangle \mid \langle p, w \rangle \in [e_i]_s\}. \tag{2.11}$$

We assume that the union operators of Equations 2.9 and 2.11 are such that pairs with the same partial variable assignment are combined by adding their probability, i. e., each pair of pairs $\langle p, w \rangle$ and $\langle p', w \rangle$ is merged to a single pair

$\langle p+p', w\rangle$. Moreover, we assume that the union that merges partial variable assignments w and w' in 2.9 is unambiguous for all actions i. e., there is no variable $v \in \mathcal{V}$ where $v{\leftarrow}x \in w$ and $v{\leftarrow}x' \in w'$ for $x, x' \in \mathcal{D}_v$. Let $s' = \text{app}([e]_s, s)$ be the state that is such that $s'(v) = [e]_s(v)$ for all $v \in \text{vars}([e]_s)$ and $s'(v) = s(v)$ for all $v \notin \text{vars}([e]_s)$. Then, a set of actions $\mathcal{A}$ over a set of variables $\mathcal{V}$ induces a transition function $\mathcal{T}$ that is such that for all $s, s' \in \mathcal{S}$ and $a = \langle \text{pre}, \text{eff}\rangle \in \mathcal{A}$

$$\mathcal{T}(s, a, s') = \begin{cases} \sum_{\{\langle p,w\rangle \in [\text{eff}]_s \mid \text{app}([\text{eff}]_s, s) = s'\}} p & \text{if } s \models \text{pre} \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

Please observe that this implies that it is *not* possible to compute $\text{succ}(s, a)$ of a state-action pair (s, a) efficiently as the effect set $[e]_s$ can be exponential in the number of state variables (i. e., each state can be an outcome of an action execution). On a side note (and since the IPPCs 2011 and 2014 play an important role in our empirical evaluation), it is an interesting fact that it is possible to compute $\text{succ}(s, a)$ efficiently with the subset of RDDL that was used at IPPC 2011 and 2014. The subset is less expressive than the formalism that is considered here due to the independence assumption of parallel probabilistic effects in RDDL. An input language that is able to encode the same set of transition functions can be derived if interm-fluents are allowed in addition. However, all of our algorithms are designed such that an efficient computation of $\text{succ}(s, a)$ is not required.

2.2.3 Reward Function

We also use variables for the compact description of an MDP's reward function by specification of a set of *factored reward functions*. In a finite-horizon, factored MDP, there is a factored reward function $\mathcal{R}_a(s)$ for each of its actions $a \in \mathcal{A}$ that is an arithmetic function over the set of variables $v \in \mathcal{V}$ and $\mathbb{R}$. Arithmetic operations that include variables are made possible by the use of *Iverson brackets* (Graham et al., 1994), where $\mathbb{I}\big[s[v] = x\big]$ evaluates to 1 if $v \in \mathcal{V}$ assumes the value $x \in \mathcal{D}_v$ in s and to 0 otherwise. However, factored reward functions are not important for this thesis, and we hence do not specify their structure any further. It is only important that a factored reward function can be evaluated efficiently for all state-action pairs. In most planning formalisms, including STRIPS, SAS$^+$, or the formalism that is given by PPDDL, each action induces a constant cost that is independent from the state in which the action is applied. Rewards are typically considered in the context of MDPs. They are more general than costs and rely on both the current state and the applied action[1].

[1] In parts of the literature, rewards additionally depend on the applied actions' outcome. We opt for the simpler version that is considered in the benchmarks of IPPC 2011 and 2014.

Example 6. *Let the factored reward functions be such that each cell that is marked as an observation target in a state s yields a reward of -1. The application of a* noop *action does not incur any further cost and is hence*

$$\mathcal{R}_{\texttt{noop-}z}(s) = 0 - \sum_{y \in \mathcal{Z}} \mathbb{I}\big[s[\texttt{target-}y]\big].$$

The application of all other actions $a \in \mathcal{A} \backslash \{\texttt{noop-}z \mid z \in \mathcal{Z}\}$ incurs an additional cost of 1:

$$\mathcal{R}_a(s) = (-1) - \sum_{z \in \mathcal{Z}} \mathbb{I}\big[s[\texttt{target-}z]\big].$$

2.2.4 Finite-horizon, Factored MDP

Now that all components have been defined, we can compound them to our definition of finite-horizon, factored MDPs.

Definition 7 (Finite-horizon, Factored MDP). *A* **finite-horizon, factored MDP** *is a tuple $\langle \mathcal{V}, \mathcal{A}, \{\mathcal{R}_a \mid a \in \mathcal{A}\}, s_0 \rangle$, where $\mathcal{V}$ is a set of variables that includes the* **steps-to-go** *variable h; $\mathcal{A}$ is a set of actions; $\{\mathcal{R}_a \mid a \in \mathcal{A}\}$ is a set that contains a factored reward function $\mathcal{R}_a$ for each $a \in \mathcal{A}$; and s_0 is the initial state over $\mathcal{V}$. The* **finite-horizon** *is given implicitly by $s_0[h]$ and denoted with $\mathcal{H}$.*

A finite-horizon, factored MDP $\langle \mathcal{V}, \mathcal{A}, \{\mathcal{R}_a \mid a \in \mathcal{A}\}, s_0 \rangle$ induces an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$ with a set of states $\mathcal{S}$ that contains all states that can be described by the variables in $\mathcal{V}$, with $\mathcal{R}(s, a) = \mathcal{R}_a(s)$ for all state-action pairs (s, a), and with a transition function as described by Equation 2.12. The line between a finite-horizon, factored MDP and the MDP it induces is often blurry in this work, and drawing a clearer line does not provide additional value. If the factored reward functions do not play a role, we specify a finite-horizon, factored MDP by a tuple $\langle \mathcal{V}, \mathcal{A}, \mathcal{R}, s_0 \rangle$ where the reward function $\mathcal{R}$ is given directly. Moreover, when we specify an MDP indirectly via a finite-horizon, factored MDP $\mathcal{M}$, we do not distinguish between the induced MDP and the finite-horizon, factored MDP, but regard both as the same mathematical object $\mathcal{M}$ with assigned set of states, reward function and transition function on the one hand and set of variables, factored reward functions and actions that consist of preconditions and effects on the other.

Example 7. *Figure 2.4 shows a part of the MDP that is induced by the description of the* EARTHOBSERVATION *via a finite-horizon, factored MDP that has been given over the course of this section. All states that are reachable from the initial state within two action applications are depicted. States show the focal point of the camera and the remaining observation patches in the grid (which are identical in all states but the one state that is second from the left in the bottom row). The steps-to-go variable is equal for all states within the dotted lines that partition the search space. The state at the top of Figure 2.4 is the same initial state s_0 that is also depicted in detail in Figure 2.3.*

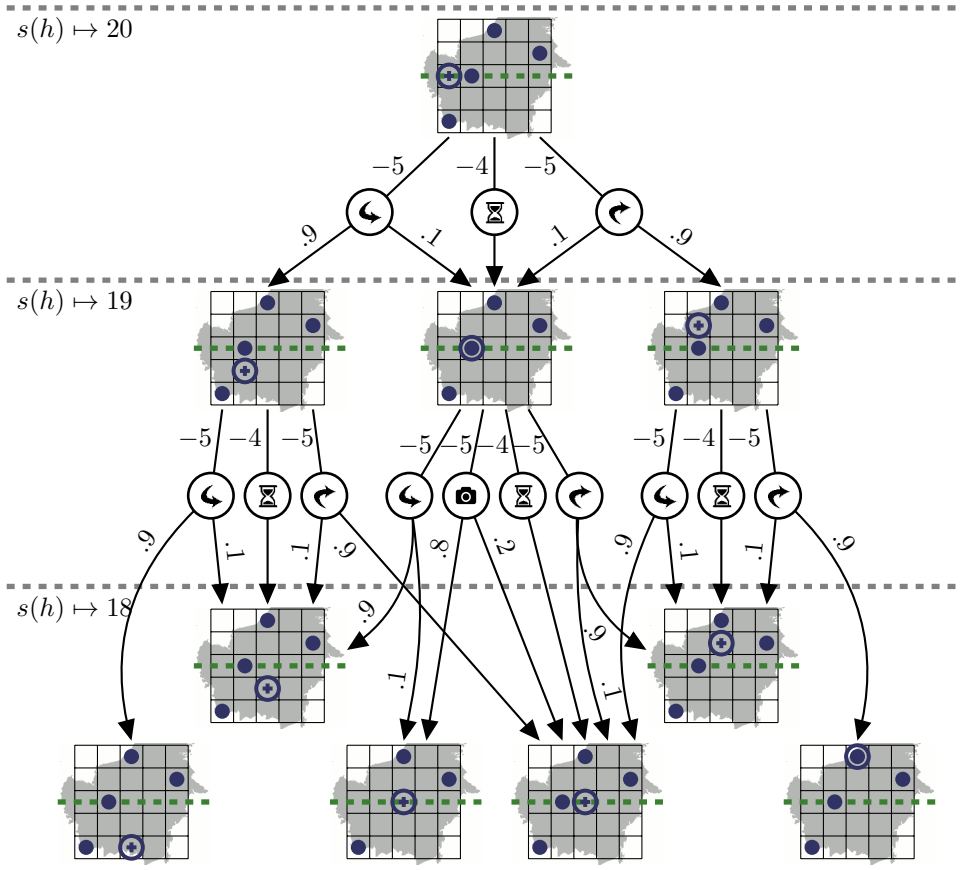

Figure 2.4: The part of the EARTHOBSERVATION example instance that can be reached with at most two action applications.

There are three applicable actions in $\mathcal{A}(s_0)$: from left to right, the icons represent the actions `slew-south` *(an arrow that points to the south-east),* `noop` *(an hourglass), and* `slew-north` *(an arrow that points to the north-east). The fourth type of action of our example domain,* `take-image`*, is depicted with a camera symbol as in the center of Figure 2.4.*

A property of finite-horizon, factored MDPs that is essential for this thesis is the fact that the induced MDP is *acyclic*. This is due to fact that the steps-to-go variable decreases by one in each step regardless of the applied action, which makes it impossible to encounter the same state twice in a run (recall that the steps-to-go variable is treated like any other variable here, and states are only equal if the values of *all* variables, including the steps-to-go variable, are equal). It is also important that the factored description allows the compact definition of a large number of states and action outcomes, while actions have to be specified explicitly and are hence expected to be comparably few.

There is related work where the action space is large (e.g., Raghavan et al., 2012) or even infinite (e.g., Antos et al., 2008), but all algorithms in this thesis are built on the assumption that the number of actions is such that their definition by enumeration is possible.

Before we discuss solutions of finite-horizon, factored MDPs in the next section, we require a notion of the *equivalence* of two finite-horizon, factored MDPs. We define it with the help of *trajectories*, which are similar to runs (in fact, each run induces a trajectory), but we do not use them in the context of the interaction between an agent and the environment. They differ from a run in the fact that they can start in any state, that they have to be of finite length, and that the outcome of an action is not necessarily sampled according to the underlying probability distribution. (It is not important how the outcome s' of a state-action pair (s, a) is generated in a trajectory, it only matters that $s' \in \text{succ}(s, a)$.)

Definition 8 (Trajectory). *Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{A}, \mathcal{R}, s_0 \rangle$ be a finite-horizon, factored MDP. A finite sequence of states and actions $\theta = (s_1, a_1, \ldots, s_{n-1}, a_{n-1}, s_n)$ is a* **trajectory** *of length $(n-1)$ in $\mathcal{M}$ iff $a_t \in \mathcal{A}(s_t)$ and $s_{t+1} \in \text{succ}(s_t, a_t)$ for all $t \in \{1, \ldots, n-1\}$. A trajectory $\theta = (s, a, s')$ of length 1 is called a* **transition**. *The* **accumulated reward** *of θ is $\mathcal{R}(\theta) := \sum_{t=1}^{n-1} \mathcal{R}(s_t, a_t)$, and the* **combined probability** *of θ is $\mathbb{P}[\theta] := \prod_{t=1}^{n-1} \mathbb{P}_{\mathcal{T}}[s_{t+1} \mid s_t, a_t]$. We denote the set of all trajectories in $\mathcal{M}$ with Θ, and define the set of all minimal trajectories between states of a set $\mathcal{S}' \subseteq \mathcal{S}$ as $\Theta_{\mathcal{S}'} := \{(s, a, s_1, \ldots, s_n, a_n, s') \in \Theta \mid s, s' \in \mathcal{S}' \text{ and } s_1, \ldots, s_n \notin \mathcal{S}'\}$.*

It is usually sufficient to define the equivalence of two finite-horizon, factored MDPs by requiring a bijective mapping between state-action pairs such that all corresponding action applications yield the same reward and occur with the same probability. Since we have to be able to compare MDPs where the application of a single action in one MDP is matched to the application of a sequence of actions in the other (i.e. there is no one-to-one correspondence between transitions), we require a more general definition of equivalence.

Definition 9 (Equivalence). *Let the set of* **decision states** *of a finite-horizon, factored MDP $\mathcal{M}$ be $\mathcal{S}_{\text{dec}} = \{s_0\} \cup \{s \in \mathcal{S} \mid |\mathcal{A}(s)| > 1\} \cup \{s \in \mathcal{S} \mid s[h] = 0\}$, and let $\mathcal{A}_{\text{dec}} = \bigcup_{s \in \mathcal{S}_{\text{dec}}} \mathcal{A}(s)$ be the set of* **decision actions**. *Two finite-horizon, factored MDPs $\mathcal{M}$ and $\mathcal{M}'$ are* **equivalent** *($\mathcal{M} \equiv \mathcal{M}'$) iff there are bijective mappings $\sigma : \mathcal{S}_{\text{dec}} \to \mathcal{S}'_{\text{dec}}$, $\alpha : \mathcal{A}_{\text{dec}} \to \mathcal{A}'_{\text{dec}}$, and $\tau : \Theta_{\mathcal{S}_{\text{dec}}} \to \Theta'_{\mathcal{S}'_{\text{dec}}}$ such that there is a trajectory $\theta' = (\sigma(s_1), \alpha(a), s''_1, a''_1, \ldots, s''_n, a''_n, \sigma(s_2)) \in \Theta'_{\mathcal{S}'_{\text{dec}}}$ for each trajectory $\theta = (s_1, a, s'_1, a'_1, \ldots, s'_n, a'_n, s_2) \in \Theta_{\mathcal{S}_{\text{dec}}}$ with $\tau(\theta) = \theta'$, $\mathcal{R}(\theta) = \mathcal{R}'(\theta')$, and $\mathbb{P}[\theta] = \mathbb{P}[\theta']$.*

Our equivalence relation is, as usual, symmetric and transitive. Moreover, it is such that two complete policies π and π' in $\mathcal{M}$ and $\mathcal{M}'$, respectively, yield the same expected reward if $\mathcal{M} \equiv \mathcal{M}'$ and $\alpha(\pi(s)) = \pi(\sigma(s))$ for all decision states $s \in \mathcal{S}_{\text{dec}}$.

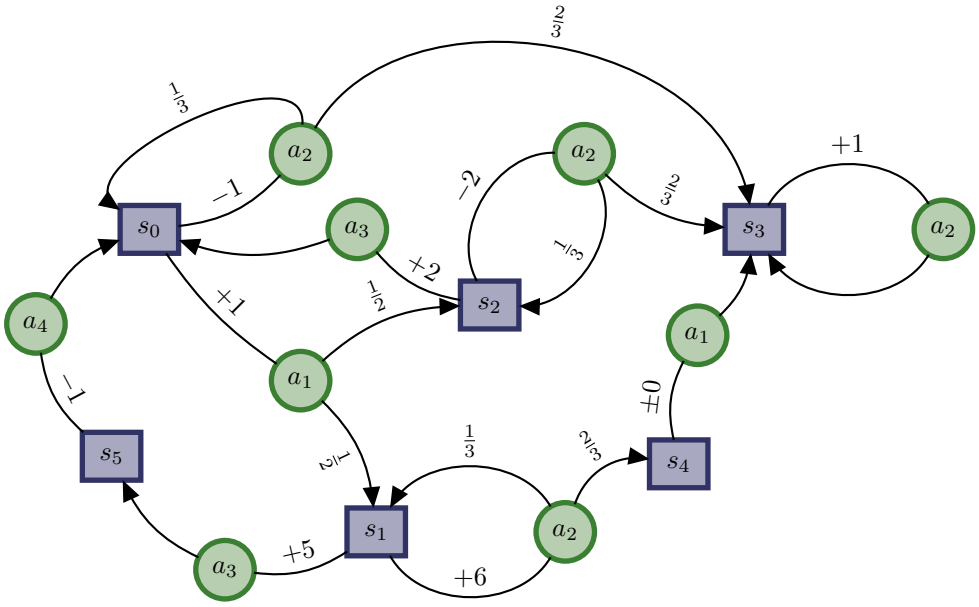

Figure 2.5: An MDP that is equivalent to the MDP in Figure 2.2.

Example 8. *Let $\mathcal{M}$ be the example MDP from Figure 2.2 and $\mathcal{M}'$ be the MDP that is depicted in Figure 2.5. Even though they are no finite-horizon, factored MDPs, they can be used to illustrate the key concepts of our equivalence relation. First observe that, if we ignore the two trajectories $\theta_1 = (s_1, a_3, s_0)$ and $\theta_2 = (s_1, a_1, s_3)$ in $\mathcal{M}$ and $\theta'_1 = (s_1, a_3, s_5, a_4, s_0)$ and $\theta'_2 = (s_1, a_2, s_4, a_1, s_3)$ for the moment, both "remaining" MDPs are equivalent since they are identical apart from swapping the names of the actions a_1 and a_2. Now consider the functions $\sigma(s) = s$ for $s \in \mathcal{S}$, $\alpha(a_1) = a_2$, $\alpha(a_2) = a_1$, and $\alpha(a_3) = a_3$, and τ such that*

- $\tau((s, a, s')) = (\sigma(s), \alpha(a), \sigma(s'))$ *for all* $(s, a, s') \in \Theta_{\mathcal{S}} \setminus \{\theta_1, \theta_2\}$,
- $\tau(\theta_1) = \theta'_1$, $\theta'_1 = (\sigma(s_1), \alpha(a_3), s_5, a_4, \sigma(s_0))$, *and*
- $\tau(\theta_2) = \theta'_2$, $\theta'_2 = (\sigma(s_1), \alpha(a_2), s_4, a_1, \sigma(s_3))$.

With σ, α, and τ, we can see that $\mathcal{M} \equiv \mathcal{M}'$ according to our equivalence definition since

- *σ, α and τ are bijections as s_4 and s_5 are no decision states and a_4 not in the set of decision actions;*
- $\mathcal{R}((s, a, s')) = \mathcal{R}(\sigma(s), \alpha(a), \sigma(s'))$ *for all* $(s, a, s') \in \Theta_{\mathcal{S}} \setminus \{\theta_1, \theta_2\}$;
- $\mathcal{R}(\theta_1) = 4 = 5 - 1 = \mathcal{R}'(\theta'_1)$ *and* $\mathbb{P}[\theta_1] = 1 = 1 \cdot 1 = \mathbb{P}[\theta'_1]$; *and*
- $\mathcal{R}(\theta_2) = 6 = 6 + 0 = \mathcal{R}'(\theta'_2)$ *and* $\mathbb{P}[\theta_2] = \frac{2}{3} = \frac{2}{3} \cdot 1 = \mathbb{P}[\theta'_2]$.

2.3 Solutions

We have selected a model of optimal behavior where the quality of a given policy π is described in terms of the expected accumulated reward, but we have not yet discussed how to compute $\mathbb{E}[\phi^\pi]$, the reward that can be expected in a run ϕ^π under π in an MDP $\mathcal{M}$. The answer is surprisingly simple: in any state $s \in \mathcal{S}$, we can expect the immediate reward $\mathcal{R}(s, \pi(s))$ (since we are about to apply the action $\pi(s)$ in s) plus the sum of the rewards that can be expected in the possible successor states $s' \in \text{succ}(s, \pi(s))$, each weighted with the probability $\mathbb{P}_\mathcal{T}[s' \mid s, \pi(s)]$ that we end up in that specific successor state s'. A formalization of this intuition applied to all states s and all state-action pairs (s, a) leads to the sets of *state-value functions* $V_\pi(s)$ and *action-value* (or *Q-value*) *functions* $Q_\pi(s)$ of an MDP. The expected reward of ϕ^π in an MDP $\mathcal{M}$ is the unique solution of the state-value function in the initial state s_0, i. e.$\mathbb{E}[\phi^\pi] := V^\pi(s_0)$.

Definition 10 (Value functions). *Let $\mathcal{M}$ be an MDP, $s \in \mathcal{S}$ be a state and π be a policy that is executable in s. The* **state-value** *$V_\pi(s)$ of s under π is defined as*

$$V_\pi(s) := \begin{cases} 0 & \text{if } s \text{ is terminal} \\ Q_\pi(s, \pi(s)) & \text{otherwise,} \end{cases}$$

where the **action-value** *$Q_\pi(s, a)$ under π is defined as*

$$Q_\pi(s, a) := \mathcal{R}(s, a) + \sum_{s' \in \text{succ}(s,a)} \left(\mathbb{P}_\mathcal{T}\left[s' \mid s, a\right] \cdot V_\pi(s')\right)$$

for all state-action pairs (s, a).

Iterative policy evaluation (Sutton and Barto, 1998) is a baseline algorithm to compute the unique solution of $V^\pi(s_0)$. It treats the state-value functions of Definition 10 as assignment operators, and applies them iteratively to update the state-values of all states until all values have converged. However, iterative policy evaluation needs to iterate only due to the fact that it can also be used to optimizes the discounted reward over an infinite horizon in a cyclic MDP. The computation of the expected reward is eased significantly in our formalism by the fact that we consider acyclic MDPs only. This allows us to order all state-value functions such that we start with the computation of V_π for terminal states and continue with solving those state-value functions that only depend on state-value functions that have already been solved earlier in the process. This is equivalent to ordering the state-value functions such that states with fewer steps-to-go are served earlier. We can thereby compute all state-value functions in a single iteration with the *acyclic policy evaluation* algorithm that is depicted in Algorithm 1.

However, we are usually not interested in the expected reward of a given policy π, but we are interested in deriving an *optimal policy* $\pi^\star$ (as usual, the

Algorithm 1: Acyclic policy evaluation

```
acyclic_policy_evaluation(M, π):
    for s ∈ S with s[h] = 0 do V_π(s) ← 0;
    for t = 1, ..., s_0[h] do
        for s ∈ S with s[h] = t do
            V_π(s) ← R(s, π(s)) + Σ_{s' ∈ succ(s,π(s))} (P_T[s' | s, a] · V_π(s'))
```

star hints at optimality in this thesis). Let $\arg\max_{x \in X} \phi(x)$ be the subset of X that contains all elements that are maximal among all elements of X with respect to ϕ, i. e.,

$$\arg\max_{x \in X} \phi(x) := \{x \in X \mid \forall x' \in X : \phi(x) \geq \phi(x')\}.$$

We say that a policy $\pi^\star$ is optimal if it is among those policies with the highest expected reward, i. e., if

$$\pi^\star \in \arg\max_{\pi \in \Pi} V_\pi(s_0).$$

If we talk about *the* optimal policy $\pi^\star$, we mean any optimal policy. It could, for instance, be drawn uniformly at random (in the following, we denote the process of drawing an element x from a set X with $x \sim \mathcal{U}(X)$, e.g. in this case $\pi^\star \sim \mathcal{U}\left(\arg\max_{\pi \in \Pi} V_\pi(s_0)\right)$). In combination with acyclic policy evaluation, this already provides us with a first naïve algorithm for the computation of the optimal policy: we can use acyclic policy evaluation to compute $V_\pi(s_0)$ for all $\pi \in \Pi$, and select any of the policies π with the highest state-value of s_0 under π. However, the number of policies in an MDP is in $O\left(|\mathcal{A}|^{|\mathcal{S}|}\right)$ and hence exponential in a number that is itself exponential in the number of state variables. This is clearly intractable for all but the smallest MDPs. An alternative computation of the optimal policy is described by the *Bellman optimality equation* (Bellman, 1957; Bertsekas, 1995), an equation that is related to the state- and action value functions.

Definition 11 (Optimal Policy). *Let the* **Bellman optimality equation** *for a state s be the set of equations that describes $V_\star(s)$, where*

$$\begin{aligned} V_\star(s) &:= \begin{cases} 0 & \text{if } s \text{ is terminal} \\ \max_{a \in \mathcal{A}(s)} Q_\star(s,a) & \text{otherwise,} \end{cases} \\ Q_\star(s,a) &:= \mathcal{R}(s,a) + \sum_{s' \in \text{succ}(s,a)} \left(\mathbb{P}_\mathcal{T}\left[s' \mid s, a\right] \cdot V_\star(s')\right). \end{aligned}$$

A policy $\pi^\star$ is an **optimal policy** *if $\pi^\star(s) \in \arg\max_{a \in \mathcal{A}(s)} Q_\star(s,a)$ for all $s \in \mathcal{S}$.*

Algorithm 2: Acyclic value computation

acyclic_value_computation$(\langle \mathcal{V}, \mathcal{A}, \mathcal{R}, s_0 \rangle)$**:**
  for $s \in \mathcal{S}$ with $s[h] = 0$ **do** $V_\star(s) \leftarrow 0$;
  for $t = 1, \ldots, s_0[h]$ **do**
    for $s \in \mathcal{S}$ with $s[h] = t$ **do**
      for $a \in \mathcal{A}(s)$ **do**
        $Q_\star(s, a) \leftarrow \mathcal{R}(s, a) + \sum_{s' \in \text{succ}(s,a)} \left(\mathbb{P}_\mathcal{T}\left[s' \mid s, a\right] \cdot V_\star(s) \right)$
      $V_\star(s) \leftarrow \max_{a \in \mathcal{A}(s)} Q_\star(s, a)$

The state-value $V_\star(s)$ of a state s under a policy that is a solution to the Bellman optimality equation $\pi^\star$ is also called the state-value of s, and $Q_\star(s, a)$ the action-value (or Q-value) of a in s. Bellman (1957) showed that an optimal, deterministic and stationary policy exists for each MDP, and there is hence a $\pi^\star \in \arg\max_{\pi \in \Pi} V_\pi(s_0)$ that satisfies the Bellman optimality equation and $V_\star(s_0) = \max_{\pi \in \Pi} V_\pi(s_0)$. Note that it is not the case that all policies in $\arg\max_{\pi \in \Pi} V_\pi(s_0)$ satisfy the Bellman optimality equation, as the behavior in states that cannot be reached from s_0 is irrelevant for the value of $V_\pi(s_0)$. However, all policies in $\arg\max_{\pi \in \Pi} V_\pi(s_0)$ satisfy the Bellman optimality equation for all *reachable* states and state-action pairs, which suffices for our needs. We hence use the state-value of a state $V_\star(s)$ and $Q_\star(s, a)$ as abbreviations of $V_{\pi^\star}(s)$ and $Q_{\pi^\star}(s, a)$, respectively.

Similar to the relation between the value functions of a policy and the computation of expected rewards, we can use the Bellman optimality equation to compute the optimal policy in MDPs by regarding the equations of Definition 11 as assignment operators (an operation that we call a *Full Bellman backup* later in this thesis) and iterate until all state- and action-values have converged. This algorithm is called *value iteration* (Bellman, 1957), and it is one of the most fundamental algorithms for MDPs. However, it is again possible to exploit the acyclic structure of the search space of finite horizon MDPs, thereby deriving the non-iterative *acyclic value computation* algorithm (e.g., Dai et al., 2011) that is depicted in Algorithm 2. Unlike value iteration, it computes the optimal policy in a single sweep over the state space. Therefore, not only value iteration but also other fundamental base algorithms like policy iteration (Howard, 1960; Puterman, 1994) and linear programming (e.g., Derman, 1970; Kallenberg, 1994) that are typically used to solve MDPs are not relevant for our needs since their advantages are irrelevant in acyclic search spaces.

The fact that acyclic value computation has advantages in our formalism over the approaches that also solve cyclic MDPs does not make it an algorithm that is tractable in practice, though: it computes a number of state- and action-values that is equal to the number of states and state-action pairs, respectively, for a total of $(|\mathcal{S}| \cdot (|\mathcal{A}| + 1))$ operations. Unfortunately, it is not

possible to design efficient algorithms that compute an executable policy of high quality as the number of reachable states is exponential in the horizon. We are therefore interested in algorithms that compute decision *online* and interleave planning for a single current state with execution of the computed decision. A special case of online algorithms are anytime algorithms, which can additionally be interrupted at any time and return a decision quickly. Our anytime algorithms achieve this by basing their decisions on state- and action-value *estimates* that are improved over time in an *iterative process*. We denote the state-value estimate of a state s after the k-th iteration with $\hat{V}^k(s)$, and the action-value estimate of a state-action pair (s, a) with $\hat{Q}^k(s, a)$. Please observe that the superscript of estimates denotes the number of iterations that have been performed by an anytime optimal algorithm, while a superscript of state- and action-values (under a policy) always refers to the superscript of the corresponding MDP. In both cases, indices refer to the policy π that is used to compute the values or estimates. Finally, an important subclass of anytime algorithms are anytime *optimal* algorithms, which additionally have in common that their decisions are optimal if the algorithm is provided with unrestricted resources (run time and memory).

CHAPTER 3

Determinization

In the previous chapter, we have presented acyclic value computation, a simple procedure that allows the computation of optimal policies in an MDP. However, solving MDPs with algorithms that require the entire MDP explicitly in memory is intractable in the majority of practical applications. Most successful approaches therefore simplify the given MDP, solve the simplification, and then either combine execution of the derived policy with replanning or derive a heuristic from the computed policy that guides a search procedure in the original search space such that considering only a fraction of all states is sufficient for good behavior. A popular simplification strategy removes all uncertainty from an MDP. The general idea of so-called *determinizations* is discussed in Section 3.1, and three determinizations – the *all-outcomes* determinization, the *most-likely* determinization and the *weather* – are presented. Section 3.2 introduces a method that allows the computation of an all-outcomes determinization for finite-horizon, factored MDPs, where, unlike in previously described all-outcomes determinizations, the number of deterministic actions is not exponentially but polynomially bounded in the number of parallel probabilistic effects.

3.1 Preliminaries

The usage of determinizations has always been a popular strategy in MDP planning, but the success of FF-Replan (Yoon et al., 2007) in the stochastic setting of IPPC 2004 was still surprising due to the simplicity of the approach that neglects probabilistic information entirely: FF-Replan converts the MDP to a classical planning problem, computes a *plan* (i. e., a trajectory from the current state to a goal) with the heuristic search planner FF (Hoffmann and Nebel, 2001) and executes actions until an outcome occurs that is not accounted for in the plan. In this case, the FF planner is used to find a new plan from the current state and start execution anew until a goal is reached.

The results of the two subsequent IPPCs were similar: the top performer in 2006 was FPG (Buffet and Aberdeen, 2007), a planning systems that uses FF-Replan to learn reasonable behavior, and RFF-(BG/PG) (Teichteil-Königsbuch et al., 2010) won IPPC 2008 with an approach where an initial policy is computed with FF and step-by-step expanded with results of additional plans that are found with FF. All of these planners have in common that a compilation of the MDP to a classical planning problem is required, which is called a *determinization* in this thesis.

Definition 12 (Determinization). *A tuple $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$ with deterministic MDP $\mathcal{M}^{\mathrm{d}}$ and surjective mapping $\delta : \mathcal{A}^{\mathrm{d}} \to \mathcal{A}$ is a* **determinization** *of an MDP $\mathcal{M}$ iff*

- $\mathcal{S}^{\mathrm{d}} = \mathcal{S}$
- $s_0^{\mathrm{d}} = s_0$
- *for each state-action pair (s, a) in $\mathcal{M}$ there is a state-action pair (s^d, a^d) in $\mathcal{M}^{\mathrm{d}}$ with $\delta(a^d) = a$; and*
- $\mathcal{R}^{\mathrm{d}}(s, a^{\mathrm{d}}) = \mathcal{R}(s, \delta(a^{\mathrm{d}}))$ *for all state-action pairs (s, a^{d}).*

We call $\mathcal{M}^{\mathrm{d}}$ the **determinization**[1] *of $\mathcal{M}$.*

In other words, the determinization of an MDP $\mathcal{M}$ is a deterministic MDP $\mathcal{M}^{\mathrm{d}}$ with the same set of states and at least one action a^{d} for each $a \in \mathcal{A}$. However, there can be any number of actions in $\mathcal{A}^{\mathrm{d}}$ that map to the same action in $\mathcal{A}$. In the following, we call an action $a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}}$ a determinization of $a \in \mathcal{A}$ iff $\delta(a^{\mathrm{d}}) = a$.

The example we use in this chapter is the CANADIAN TRAVELER'S PROBLEM (CTP), a path planning problem with imperfect information about the roadmap (Papadimitriou and Yannakakis, 1991). It has drawn considerable attention from researchers in AI search (Nikolova and Karger, 2008; Bnaya et al., 2009; Dey et al., 2014) and is closely related to navigation tasks in uncertain terrain that are considered in the robotics literature (e.g., Koenig and Likhachev, 2002; Ferguson et al., 2004). One practical application is outdoor navigation of autonomous robots with the help of a rough map based on a satellite image or a map constructed from previous scans of the environment (Likhachev and Stentz, 2006). The policies that are discussed in this chapter have been applied to the CTP by Eyerich et al. (2010), but they are presented in a generalized, domain-independent form that is applicable to all finite-horizon, factored MDPs.

Namesake of the CANADIAN TRAVELER'S PROBLEM is a scenario with the objective to drive a truck from an initial location to a goal location on a road network that is given as an undirected weighted graph. This is complicated

[1] Should the difference matter, we make sure that it is clear if we refer to the whole tuple or only to the deterministic MDP with the term "determinization".

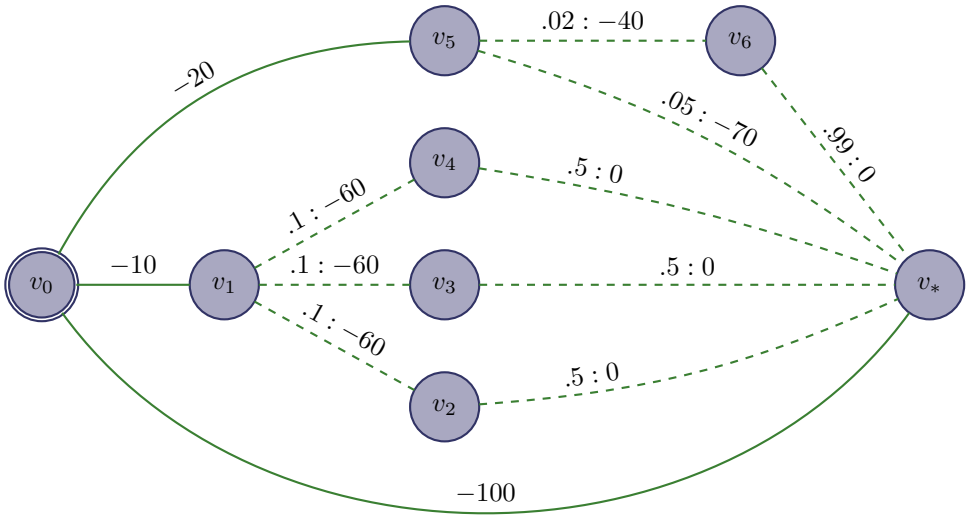

Figure 3.1: CTP example instance with truck location v_0 and goal location $v_\star$.

by the fact that roads may be blocked due to snow. The traversability of a road can only be observed from the two incident locations. Usually, the CTP is modeled as a partially observable MDP (POMDP). However, the weather remains static during the agent's traversal of the graph, so once a road has been observed, its status is known with certainty. CTP instances are therefore *deterministic POMDPs* where the only source of uncertainty is incomplete information about the initial state (Littman, 1996; Bonet, 2009). Deterministic POMDPs are less complex than general POMDPs in that they always have a finite set of reachable belief states, and they can be modeled as a stochastic MDP (of exponential size), which is the representation we consider here. Therefore, an instance of the CTP is a 6-tuple $\langle V, E, p, c, v_0, v_\star \rangle$, where

- $\langle V, E \rangle$ is a connected, undirected graph (the roadmap) with vertex set V (the locations) and edge set E (the roads),
- $p : E \to [0, 1)$ defines the blocking probabilities of roads,
- $c : E \to \mathbb{N}_0$ defines the travel costs of roads, and
- $v_0, v_\star \in V$ are the initial and goal locations.

Example 9. *Figure 3.1 shows a small example instance of the* CTP *with eight locations $v_0, \ldots, v_6, v_\star$. The initial location is v_0, and the goal is to find a path with low expected cost to the goal location at $v_\star$ by using only on roads that are traversable. In all figures that depict a roadmap of the* CTP*, we depict unknown roads as dashed and labeled with their blocking probability and travel cost. Traversable roads are solid and labeled with their travel cost.*

The simplest mapping of a CTP instance to an MDP yields a Stochastic Shortest Path Problem where goal states are all states where the agent is located in the goal location. However, it can also be modeled as a finite-horizon, factored MDP if *macro actions* are considered that bring the agent from its current location to a previously unvisited location on the border of the known subgraph. The macro actions are based on the observation that all *reasonable policies* (Eyerich et al., 2010) are such that the agent never moves arbitrary within the known subgraph. Instead, it selects a location in each decision steps that

1. is reachable on roads that are known to be traversable, and
2. has adjacent roads with unknown status,

and moves to such a location on the shortest path within the known subgraph. With an encoding where macro actions are considered, it is then guaranteed that the goal is reached within at most $|V|-1$ macro action applications which allows to derive an appropriate finite horizon. However, the only parts of the finite-horizon, factored MDP $\langle \mathcal{V}, \mathcal{A}, \mathcal{R}, s_0 \rangle$ that encodes a CTP instance $\langle V, E, p, c, v_0, v_\star \rangle$ that are relevant in this chapter are:

- the set of variables $\mathcal{V}$ contains the variable $\texttt{loc}$ with $\mathcal{D}_{\texttt{loc}} = V$ for the truck's current location, and the variables stat_{xy} for each road $(v_x, v_y) \in E$ with domain $\mathcal{D}_{\text{stat}_{xy}} = \{u, t, b\}$ that describes if the road is traversable (t), blocked (b), or if the status is unknown (u);
- the set of actions $\mathcal{A}$ contains actions $\texttt{move}_{xy}$ that change $\texttt{loc}$ from v_x to v_y if the agent's current location is v_x, v_y has not been visited before and if there is a path from v_x to v_y that involves only roads that are known to be traversable. The transition probabilities of all state-action pairs are determined according to the blocking probabilities of all roads with unknown status that involve v_y. (See Figure 3.2 for an example.)

A state in the CTP domain can hence be represented by the agent's current location on the roadmap and a partition of the roads into three disjoint sets: the roads known to be traversable, the roads known to be blocked, and the roads that have not been observed yet. In a CTP instance with roadmap $\langle V, E \rangle$, there are therefore $|V| \cdot 3^{|E|}$ states, which clearly shows that the CTP is a challenging domain.

Example 10. *Instead of showing the techniques that are discussed in this Chapter on the whole example* CTP *instance of Figure 3.1, we restrict to the action where the truck moves from v_0 to v_1. Apart from the (deterministic) effect that the current location of the truck changes to v_1, the status of the three adjacent roads (v_1, v_2), (v_1, v_3), and (v_1, v_4) is determined. The corresponding part of the finite-horizon, factored MDP, which is depicted in Figure 3.2, shows that the*

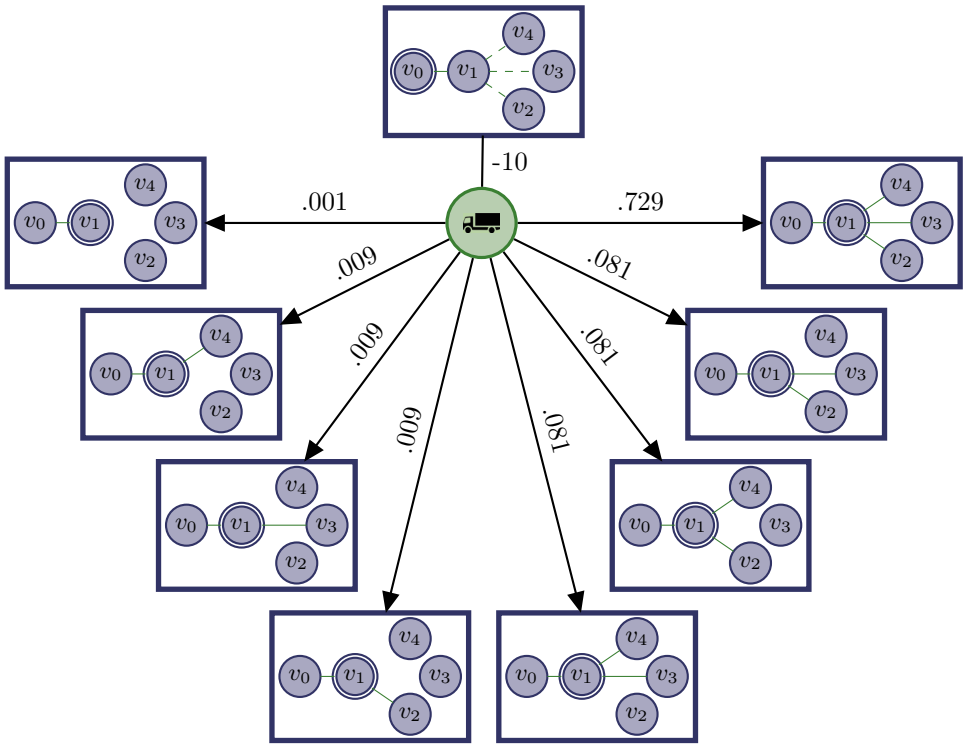

Figure 3.2: The part of the MDP that is induced by the CTP example instance that is depicted in Figure 3.1 where the truck moves from v_0 to v_1 (only the relevant part of the roadmap is depicted in the states).

application of $\texttt{move}_{01}$ *in* s_0 *can lead to eight states that differ only in the status of the three roads adjacent to* v_1.

We consider three determinizations in this thesis: the all-outcomes determinization, where each outcome of each applicable action is maintained in the determinization, and two single-outcome determinizations where only a single outcome of each applicable action persists in the determinization.

All-outcomes Determinization The all-outcomes determinization is the determinization that is used most often in practice. If combined with a suitable search algorithm, it allows an optimistic interpretation of the environment's uncertainty (see Section 4.1 for details). As the name implies, each (stochastic) outcome in the original MDP is converted to a (deterministic) action in the determinization.

Definition 13 (**All-Outcomes Determinization**). *The* **all-outcomes determinization** *of an MDP* $\mathcal{M}$ *is a determinization* $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$ *such that for all* $s, s' \in \mathcal{S}$

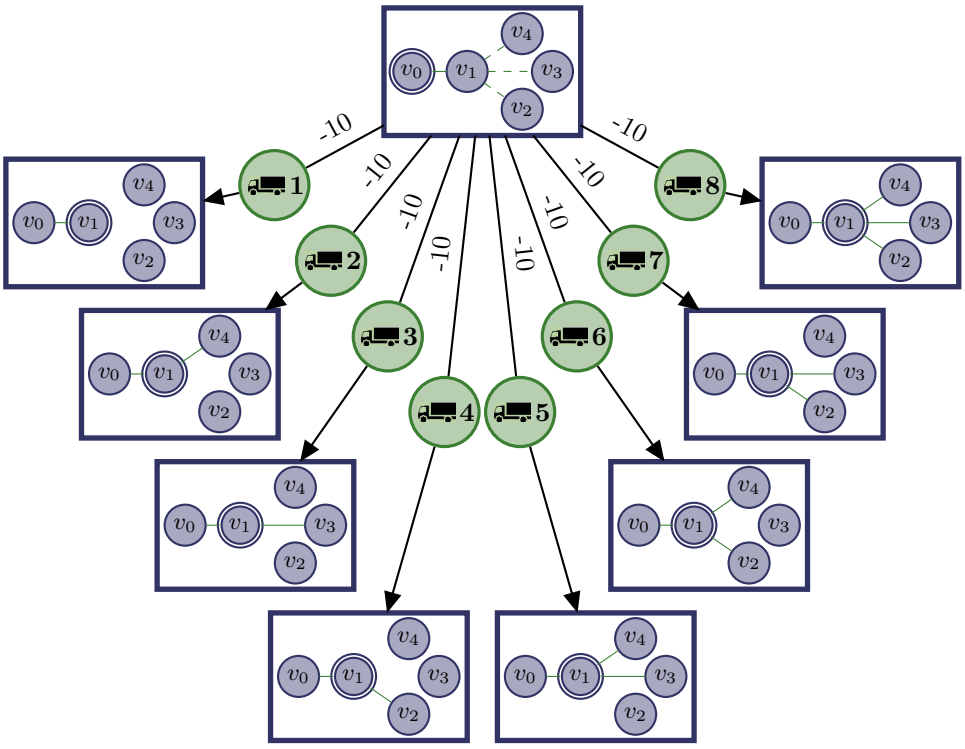

Figure 3.3: All-outcomes determinization of the $\texttt{move}_{01}$ action with eight deterministic actions (each depicted by a truck symbol with unique index).

and $a \in \mathcal{A}$ *with* $\mathcal{T}(s, a, s') > 0$ *there is an* $a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}}$ *with* $\delta(a^{\mathrm{d}}) = a$ *and* $\mathcal{T}^{\mathrm{d}}(s, a^{\mathrm{d}}, s') = 1$.

The all-outcomes determinization of a given MDP can be created by replacing each stochastic outcome of each action in the original MDP with a deterministic action that connects the same two states in the determinization. The all-outcomes determinization of our example action is depicted in Figure 3.3 (different indices next to the truck symbol indicate different actions).

Single-outcome Determinization The difference between the all-outcomes determinization and a single-outcome determinization is, as the name implies, that only a single outcome is maintained for each state-action pair in the latter.

Definition 14 (Single-outcome Determinization). *Let* $\mathcal{M}$ *be an MDP and* $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$ *be a determinization of* $\mathcal{M}$. $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$ *is a* **single-outcome determinization** *of* $\mathcal{M}$ *iff* δ *is bijective.*

An advantage of single-outcome determinizations is that they are (in practice) often easier to solve as the branching factor is typically smaller. We also exploit this fact in the heuristic of the PROST planner that is used for our

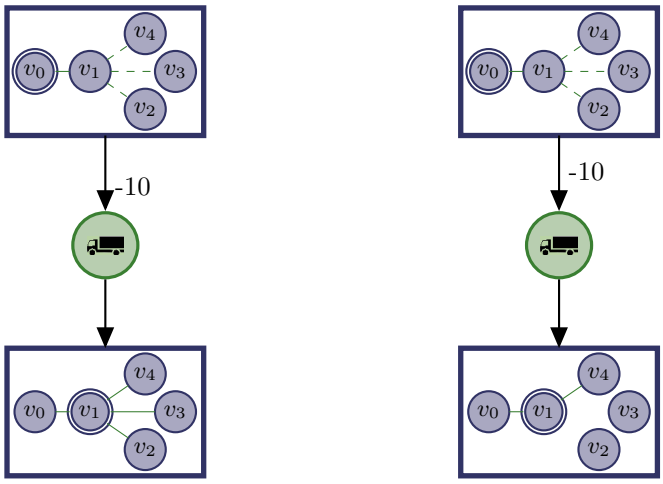

Figure 3.4: Two single-outcome determinizations of the $\texttt{move}_{01}$ action. The left one is the most-likely determinization. Furthermore, it is a weather that is selected with probability .729, while the right one is a weather that is selected with probability .009.

empirical evaluation in Chapter 8. It is based on the popular most-likely determinization, a single-outcome determinization where the outcome that occurs with the highest probability persists. (For the sake of simplicity, we assume there is only one outcome with highest probability. In practice, we select one of the outcomes that are tied for highest probability uniformly at random.)

Definition 15 (Most-likely Determinization). *The* **most-likely determinization** *of an MDP $\mathcal{M}$ is a single-outcome determinization $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$ such that*

$$\mathcal{T}^{\mathrm{d}}(s, a^{\mathrm{d}}, s') = 1 \Rightarrow \mathcal{T}(s, \delta(a^{\mathrm{d}}), s') \geq \mathcal{T}(s, \delta(a^{\mathrm{d}}), s'')$$

for all $s, s', s'' \in \mathcal{S}$, $a \in \mathcal{A}$, and $a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}}$.

The outcome that manifests with highest probability in our example state transition is the one where all roads are traversable. It is therefore the left partial MDP of Figure 3.4 that depicts the most-likely determinization of our example action.

Another single-outcome determinization that is considered in this thesis is what we call a *weather* $\mathcal{M}^{\mathrm{w}}$ of an MDP $\mathcal{M}$. Even though the term "weather" stems from our work on the CTP domain originally, we also use it for general MDPs as it conveys the intuition nicely. A weather is not defined in terms of a specific deterministic MDP – in fact, any single-outcome determinization is a weather – but it is the *procedure* that creates a single-outcome determinization that defines a weather, namely the procedure that selects one outcome for each state-action pair according to the outcome's probability. As this thesis

considers only acyclic MDPs, each weather reflects the circumstances an agent faces during a run with the probability that the weather is selected (this is not true in cyclic MDPs where the same state-action pair can be executed more than once and with different outcomes). We denote the set of all weathers in an MDP $\mathcal{M}$ with $\mathcal{W}$.

3.2 Syntactical All-outcomes Determinizations

There are several applications where it is not sufficient to have a compilation schema that allows the construction of a determinization by altering the graph that is an MDP directly. Successful probabilistic planning systems of the IPPC like the aforementioned FF-Replan (Yoon et al., 2007), FF-Hindsight (Yoon et al., 2008, 2010), or RFF-(BG/PG) (Teichteil-Königsbuch et al., 2010) use the classical FF planner of Hoffmann and Nebel (2001) as an *external* tool, which must be provided with a determinization of the MDP *on the syntactical level*. In the case of the FF planner, an input in PDDL, the deterministic version of PPDDL, is required. In this section, we discuss two different methods – one that is typically used in practice, and one that has been described in an earlier article of us (Keller and Eyerich, 2011) – to compute a syntactical all-outcomes determinization of an MDP.

Determinization via Unary Non-determinism Normal Form

To our knowledge, there is only one all-outcomes determinization that is used in practice, which is based on a compilation to unary non-determinism normal form (Rintanen, 2003) to create a determinization. Recall from the previous chapter that an MDP is in unary non-determinism normal form if the effect of each action $a = \langle \text{pre}, \text{eff} \rangle \in \mathcal{A}$ has the form $\text{eff} = (p_1 : e_1) \mid \cdots \mid (p_n : e_n)$ with deterministic effects e_i. In this form, the all-outcomes determinization can simply be derived by creating deterministic actions $a_1^{\mathsf{d}}, \ldots, a_n^{\mathsf{d}}$ for each action $a \in \mathcal{A}$ such that $a_i^{\mathsf{d}} = \langle \text{pre}, e_i \rangle$ for all $i \in \{1, \ldots, n\}$.

Even though the compilation via unary non-determinism normal form is, to our knowledge, the only practically relevant method to create a (syntactical) all-outcomes determinization, it can be prohibitively inefficient: if there are independent probabilistic effects in the effect of an action *before* its translation to unary non-determinism normal form, the translation leads to a probabilistic effect that contains a number of elements that is exponential in the number of independent probabilistic effects (Rintanen, 2003). Consequentially, the procedure generates exponentially many actions in the all-outcomes determinization as well.

Example 11. *The effect of the* $\texttt{move}_{01}$ *action of Figure 3.2 can be modeled by*

$$\begin{aligned}
&\big[\mathsf{stat}_{12} \mapsto u \rhd \big((0.9 : \mathsf{stat}_{12}{\leftarrow}t) \mid (0.1 : \mathsf{stat}_{12}{\leftarrow}b)\big)\big] \wedge \\
&\big[\mathsf{stat}_{13} \mapsto u \rhd \big((0.9 : \mathsf{stat}_{13}{\leftarrow}t) \mid (0.1 : \mathsf{stat}_{13}{\leftarrow}b)\big)\big] \wedge \\
&\big[\mathsf{stat}_{14} \mapsto u \rhd \big((0.9 : \mathsf{stat}_{14}{\leftarrow}t) \mid (0.1 : \mathsf{stat}_{14}{\leftarrow}b)\big)\big] \wedge \\
&\texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}h{-}1.
\end{aligned}$$

The three independent probabilistic effects of $\texttt{move}_{01}$ *lead to eight – i. e., exponentially many – conjunctive elements when translated to unary non-determinism normal form:*

$$\begin{aligned}
\big[&\big(0.729 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}t) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}t) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}t)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.081 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}t) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}t) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}b)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.081 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}t) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}b) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}t)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.081 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}b) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}t) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}t)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.009 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}t) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}b) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}b)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.009 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}b) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}t) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}b)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.009 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}b) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}b) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}t)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1 \mid \\
&\big(0.001 : (\mathsf{stat}_{12} \mapsto u \rhd \mathsf{stat}_{12}{\leftarrow}b) \wedge (\mathsf{stat}_{13} \mapsto u \rhd \mathsf{stat}_{13}{\leftarrow}b) \wedge \\
&\quad (\mathsf{stat}_{14} \mapsto u \rhd \mathsf{stat}_{14}{\leftarrow}b)\big) \wedge \texttt{loc}{\leftarrow}v_1 \wedge h{\leftarrow}s[h]{-}1\big].
\end{aligned}$$

Note that an action in unary non-determinism normal form has one probabilistic statement for each possible outcome, which allows a straightforward procedure for the creation of the all-outcomes determinization.

Determinization via Forked Normal Form

The main idea of the all-outcomes determinization via forked normal form (which is introduced later in this section) is to split a single action into several *subactions* (formally, subactions do not differ from actions, but we use the term to indicate an action that has been created by splitting another action). The task compilation is such that their sequential application is enforced, an idea that is inspired from the work on (different) compilations of Nebel (2000). It takes four subsequent steps, the first three of which transform the finite-horizon, factored MDP in equivalence-preserving ways, and only the last step

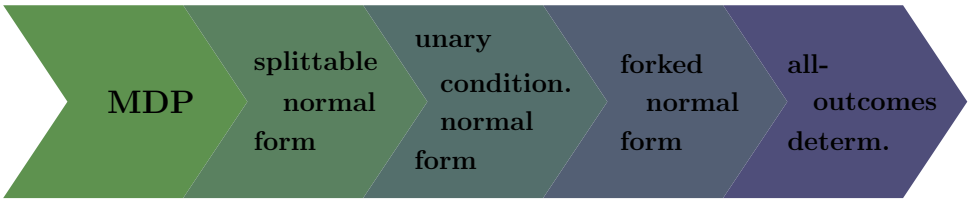

Figure 3.5: Process flowchart that shows the creation of an all-outcomes determinization via forked normal form. The first three steps are equivalence-preserving transformations of the finite-horizon, factored MDP, and the final step creates the all-outcomes determinization.

creates the determinization. The process is depicted in Figure 3.5. In the first step, all actions are converted to *splittable normal form*, which makes sure that subactions can be applied sequentially such that the sequence leads to the same states with the same transition probabilities even in the presence of conditional effects. The result in splittable normal form is then converted to unary conditionality normal form, a normal form that has been introduced in the previous chapter and is hence not discussed here. The *forked normal form* incorporates the idea of splitting all actions such that independent probabilistic effects of all actions are applied sequentially one at a time. Finally, an all-outcomes determinization that is based on forked normal form can be derived such that actions are split into several deterministic ones that are applied sequentially to simulate one outcome.

Splittable Normal Form We have seen that a determinization via unary non-determinism normal form leads to exponentially many actions in the presence of conjunctions of probabilistic effects. The main idea of the compilation via forked normal form is to split the conjunction such that only a single action per conjunctive element is created, and to enforce the sequential application of the subactions. The advantage is that, due to the independence of the parallel probabilistic effects, each can be compiled to a single subaction, and the exponential blowup can hence be avoided. We start by showing the general idea with an example. Splitting of action effects in this way leads to *intermediate* states in the corresponding MDP. In intermediate states, only a part of the original action's effects have been applied (by application of some of the created subactions). To control the application of a sequence of actions $a_1, \ldots, a_n$ in the correct order, we introduce a variable v_{aux} and extend all preconditions of actions in a first step such that $v_{\text{aux}} \mapsto 0$ is required in addition to all other preconditions. In intermediate states, we make sure that only a single action is applicable by requiring different, unique values for v_{aux} to enforce sequential application in an equivalence-preserving way.

Example 12. *Consider a finite-horizon, factored MDP $\mathcal{M}'$ where the subactions*

$$\texttt{move}_{01}^1 = \langle \text{pre}(\texttt{move}_{01}) \wedge v_{\text{aux}} \mapsto 0, \texttt{loc} \leftarrow v_1 \wedge h \leftarrow s[h] - 1 \wedge v_{\text{aux}} \leftarrow 1 \rangle$$
$$\texttt{move}_{01}^2 = \langle v_{\text{aux}} \mapsto 1, v_{\text{aux}} \leftarrow 2 \wedge \text{stat}_{12} \mapsto u \rhd \big((0.9 : \text{stat}_{12} \leftarrow t) \mid (0.1 : \text{stat}_{12} \leftarrow b)\big) \rangle$$
$$\texttt{move}_{01}^3 = \langle v_{\text{aux}} \mapsto 2, v_{\text{aux}} \leftarrow 3 \wedge \text{stat}_{13} \mapsto u \rhd \big((0.9 : \text{stat}_{13} \leftarrow t) \mid (0.1 : \text{stat}_{13} \leftarrow b)\big) \rangle$$
$$\texttt{move}_{01}^4 = \langle v_{\text{aux}} \mapsto 3, v_{\text{aux}} \leftarrow 0 \wedge \text{stat}_{14} \mapsto u \rhd \big((0.9 : \text{stat}_{14} \leftarrow t) \mid (0.1 : \text{stat}_{14} \leftarrow b)\big) \rangle$$

replace the $\texttt{move}_{01}$ *action in the MDP $\mathcal{M}$ from Example 11:*

- $\mathcal{V}' = \mathcal{V} \cup \{v_{\text{aux}}\}$*;*
- $\mathcal{A}' = \{a' \mid a \in \mathcal{A} \setminus \{\texttt{move}_{01}\}\} \cup \{\texttt{move}_{01}^1, \ldots, \texttt{move}_{01}^4\}$*, where* $a' = \langle \text{pre}(a) \wedge v_{\text{aux}} \mapsto 0, \text{eff}(a) \rangle$ *for each* $a \in \mathcal{A}$*;*
- $\mathcal{R}'_{a'}(s) = \mathcal{R}_a(s)$ *for all* $a \in \mathcal{A} \setminus \{\texttt{move}_{01}\}$*,* $\mathcal{R}'_{\texttt{move}_{01}^1}(s) = \mathcal{R}_{\texttt{move}_{01}}(s)$*, and* $\mathcal{R}'_{\texttt{move}_{01}^i}(s) = 0$ *for* $i = 2, 3, 4$*; and*
- $s'_0 = s_0 \cup \{v_{\text{aux}} \mapsto 0\}$.

The MDPs that are induced by $\mathcal{M}$ and $\mathcal{M}'$ are equivalent under the functions $\sigma(s) = s \cup \{v_{\text{aux}} \mapsto 0\}$; $\alpha(a) = a'$ for all $a \in \mathcal{A} \setminus \{\texttt{move}_{01}\}$ and $\alpha(\texttt{move}_{01}) = \texttt{move}_{01}^1$, both of which are bijections between the respective sets of decision states and decision actions; and τ such that

$$\tau(s, \texttt{move}_{01}, s') = (\sigma(s), \texttt{move}_{01}^1, s'_1, \texttt{move}_{01}^2, s'_2, \texttt{move}_{01}^3, s'_3, \texttt{move}_{01}^4, \sigma(s'))$$

and $\tau(s, a, s') = \tau(\sigma(s), \alpha(a), \sigma(s'))$ otherwise. It is also easy to see that two MDPs are equivalent if they only differ in the fact that all transitions that correspond to what is depicted in Figure 3.2 are replaced with transitions as shown in Figure 3.6 as the root and the leaves of both partial MDPs are equivalent and as all intermediate states are no decision states.

Figure 3.6 is also well-suited to show the main idea of our determinization: in an all-outcomes determinization of $\mathcal{M}'$, the number of created actions grows by two per additional parallel probabilistic effects and does not double as it does if $\mathcal{M}$ is determinized. In the example, the number of created deterministic actions is seven, which is not that different from the eight that were created by the all-outcomes determinization that is derived from a compilation to unary non-determinism normal form, but a fourth parallel probabilistic effect (i. e., a fourth unknown road) would lead to 16 action in unary non-determinism normal form and to nine with the proposed method. In general, n roads lead to 2^n actions in unary non-determinism normal form, but only to $2 \cdot n + 1$ with our method.

However, it cannot be guaranteed that splitting an action results in an equivalent MDP *in the presence of conditional effects* as it is possible that one subaction has an effect on a variable that is part of the effect condition of

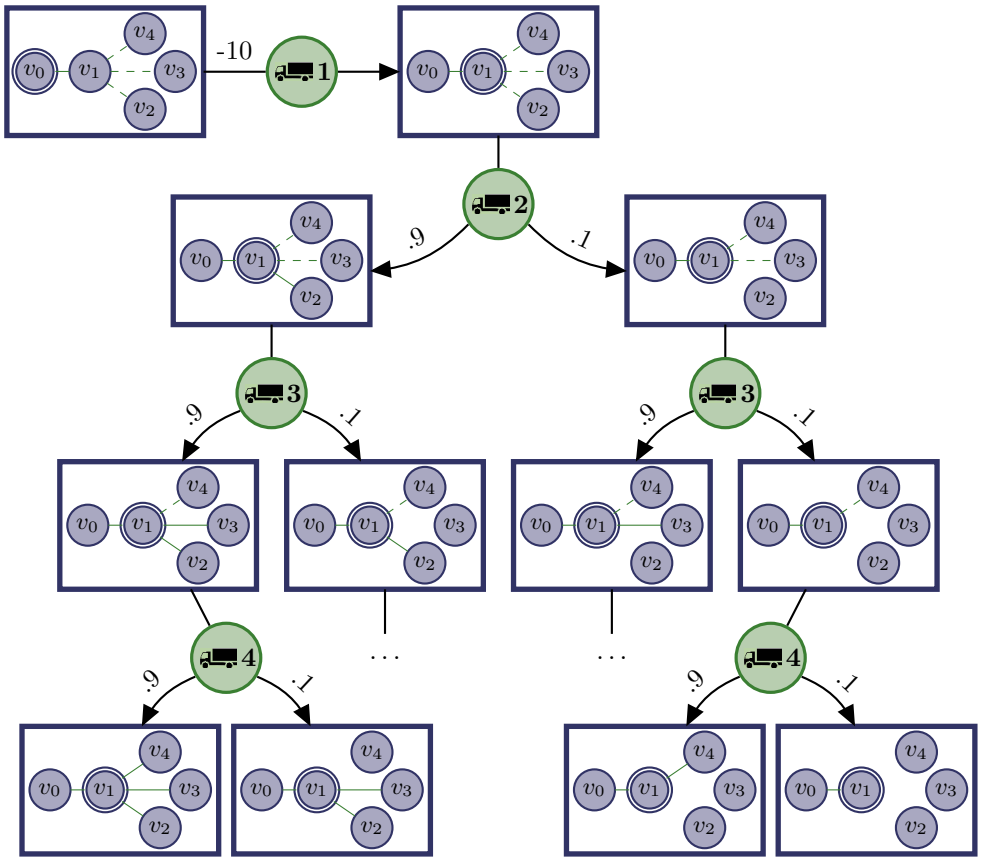

Figure 3.6: The CTP example action $\texttt{move}_{01}$ after splitting it to four subactions $\texttt{move}_{01}^1, \ldots, \texttt{move}_{01}^4$.

another subaction that is applied later in the sequence. Consider, for instance, an action with effect $a \wedge (\neg a) \rhd b$ (with two Boolean variables a and b). If the action is split such that the subaction with effect a is applied first, the second subaction will never have an effect. However, it is possible to compile all actions in an MDP such that splitting is possible even with conditional effects. The idea is to introduce a new variable for each variable that is only used to memorize the value of all variables while a sequence of subactions is applied.

Definition 16 (Splittable Normal Form). *Let $\mathcal{M}$ be a finite-horizon, factored MDP and* idx $: \mathcal{A} \to \mathbb{N}^+$ *be a function that maps each action to a unique value. We say that an MDP $\mathcal{M}^\rhd$ is the* **splittable normal form** *of $\mathcal{M}$ if it is such that*

- $\mathcal{V}^\rhd = \mathcal{V} \cup \{v^\rhd \mid v \in \mathcal{V} \setminus \{h\}\} \cup \{v_{\text{aux}}\}$;
- $\mathcal{A}^\rhd = \underbrace{\{a^\vdash \mid a \in \mathcal{A}\}}_{=\mathcal{A}^\vdash} \cup \underbrace{\{a^\dashv \mid a \in \mathcal{A}\}}_{=\mathcal{A}^\dashv}$, *where for all* $a \in \mathcal{A}$

  - $\text{pre}(a^{\vdash}) = \text{pre}(a) \cup \{v \mapsto 0 \mid v \in \mathcal{V}^{\triangleright} \setminus \mathcal{V}\}$
  - $\text{eff}(a^{\vdash}) = v_{\text{aux}} \leftarrow \text{idx}(a) \wedge h \leftarrow s[h] - 1 \wedge \bigwedge_{v \in \mathcal{V}} v^{\triangleright} \leftarrow s[v]$
  - $\text{pre}(a^{\dashv}) = \{v_{\text{aux}} \mapsto \text{idx}(a)\}$
  - $\text{eff}(a^{\dashv}) = \text{eff}(a)[v/v^{\triangleright}] \wedge \bigwedge_{v \in \mathcal{V}^{\triangleright} \setminus \mathcal{V}} v \leftarrow 0$

- $\mathcal{R}^{\triangleright} = \mathcal{R}^{\vdash} \cup \mathcal{R}^{\dashv}$*, where* $\mathcal{R}^{\vdash}_{a^{\vdash}}(s^{\triangleright}) = \mathcal{R}(s, a)$ *for* $s^{\triangleright} = s \cup \{v \mapsto 0 \mid v \in \mathcal{V}^{\triangleright} \setminus \mathcal{V}\})$ *and* $\mathcal{R}^{\vdash}_{a^{\vdash}}(s^{\triangleright})$ *otherwise, and* $\mathcal{R}^{\dashv}_{a}(s^{\triangleright}) = 0$*;*
- $s_0^{\triangleright} = s_0 \cup \{v \mapsto 0 \mid v \in \mathcal{V}^{\triangleright} \setminus \mathcal{V}\}$*,*

where $\text{eff}(a)[v/v^{\triangleright}]$ *denotes the effect that is equal to* $\text{eff}(a)$ *apart from the effect on the steps-to-go variable* h *(which is already considered in the corresponding* $\text{eff}(a^{\vdash})$*), and except that all occurrences of all* $v \in \mathcal{V}$ *in conditions of conditional effects are replaced by their corresponding* $v^{\triangleright}$*.*

The idea of the splittable normal form uses the idea of splitting actions in several subactions itself by creating a *head subaction* (which is denoted with $a^{\vdash}$) and a *body subaction* ($a^{\dashv}$). The head subaction takes care of all the technical details that are required to split the body subaction (later in this section) safely even in the presence of conditional effects: it assigns the value of each v to the corresponding $v^{\triangleright}$; it assigns a unique value to v_{aux} that ensures that only the corresponding body subaction is applicable; and it yields the reward of the original action and decreases the steps-to-go variable, so we can ignore both when we discuss how the body action can be compiled to forked normal form later in this section.

Theorem 1. *Each finite-horizon, factored MDP can be normalized in polynomial time and space to an equivalent finite-horizon, factored MDP in splittable normal form.*

Proof: It is not hard to see that the size of $\mathcal{M}^{\triangleright}$ is polynomial in the size of $\mathcal{M}$ since the number of variables and actions doubles at most, and the size of the description of preconditions and effects increases by no more than $O(|\mathcal{V}|)$. Additionally, the transformation is possible in polynomial time since idx is a simple counter function and the replacement of all $v \in \mathcal{V}$ with their respective $v^{\triangleright}(v)$ is possible in constant time. For the proof of equivalence, we consider the mappings

$$\sigma(s) = s \cup \{v \mapsto 0 \mid v \in \mathcal{V}^{\triangleright} \cup \{v_{\text{aux}}\}\}$$
$$\alpha(a) = a^{\vdash}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ in the following, which are bijections between the respective sets of decision states and decision actions as states where a $v \in \mathcal{V}^{\triangleright} \cup \{v_{\text{aux}}\}$ does not map to 0 are no decision states and as all $a \in \mathcal{A}^{\dashv}$ are applicable only in non-decision states.

It remains to show that there is a bijection τ between transitions in $\mathcal{M}$ and trajectories in $\mathcal{M}^{\triangleright}$ such that accumulated rewards and combined probabilities match and that all intermediate states in the trajectories are no decision states. Let us therefore consider the function τ to be such that it maps each transition $\theta = (s_1, a, s_2)$ in $\mathcal{M}$ to a sequence $\theta^{\triangleright} = (\sigma(s_1), \alpha(a), s', a^{\dashv}, s_2)$ in $\mathcal{M}^{\triangleright}$. Each $\theta^{\triangleright}$ is a valid trajectory in $\mathcal{M}^{\triangleright}$, since

- $\alpha(a) = a^{\vdash}$ is applicable in $\sigma(s_1)$ iff the following two conditions hold: 1., $\text{pre}(a) \subseteq \sigma(s_1)$, which holds since $\text{pre}(a) \subseteq s_1$ (otherwise, a is not applicable in s_1 and θ no transition in $\mathcal{M}$); and 2., $\{v \mapsto 0 \mid v \in \mathcal{V}^{\triangleright} \setminus \mathcal{V}\} \subseteq \sigma(s_1)$, which holds due to the definition of σ;
- $a^{\dashv}$ is applicable in s' as the action's only precondition $v_{\text{aux}} \mapsto \text{idx}(a)$ is a deterministic effect of $a^{\vdash}$; and
- $\sigma(s_2) \in \text{succ}(s', a^{\dashv})$ since $s_2 \in \text{succ}(s_1, a)$ and since $\text{eff}(a)[v/v^{\triangleright}]$ is identical to $\text{eff}(a)$ except for the swapped variables in effect conditions, which do not alter action application since $s'[v] = s'[v^{\triangleright}(v)]$ for all $v \in \mathcal{V}$ due to the effect of $a^{\vdash}$.

The intermediate state s' is no image of a state $s \in \mathcal{S}$ with respect to σ because v_{aux} does not map to 0 in s' due to the effect of $a^{\vdash}$ and the definition of $\text{idx}(a)$. Since no action apart from $a^{\dashv}$ is applicable in s' due to the uniqueness assumption in the definition of $\text{idx}(a)$, we can conclude that the proposed function is a bijection. Finally, $\mathcal{R}(\theta) = \mathcal{R}^{\triangleright}(\theta^{\triangleright})$ and $\mathbb{P}[\theta] = \mathbb{P}[\theta^{\triangleright}]$ since

$$\mathcal{R}(\theta) = \mathcal{R}_a(s_1) = \underbrace{\mathcal{R}^{\vdash}_{a^{\vdash}}(\sigma(s_1))}_{=\mathcal{R}_a(s_1)} + \underbrace{\mathcal{R}^{\dashv}_{a^{\dashv}}(s')}_{=0} = \mathcal{R}^{\triangleright}(\theta^{\triangleright}))$$

$$\mathbb{P}[\theta] = \mathbb{P}_{\mathcal{T}}[s_2 \mid s_1, a] = \underbrace{\mathbb{P}_{\mathcal{T}^{\triangleright}}[s' \mid \sigma(s_1), a^{\vdash}]}_{=1} \cdot \underbrace{\mathbb{P}_{\mathcal{T}^{\triangleright}}[\sigma(s_2) \mid s', a^{\dashv}]}_{=\mathbb{P}_{\mathcal{T}}[s_2 \mid s_1, a]} = \mathbb{P}[\theta^{\triangleright}].$$

■

We call a finite-horizon, factored MDP that has been created with this procedure a finite-horizon, factored MDP in splittable normal form because conjunctive effects of all actions in the MDP can be split safely even in the presence of conditional effects.

Forked Normal Form So far, we have shown how to compile the finite-horizon, factored MDP we aim to determinize to splittable normal form, and we have even sketched a schema how effects can be split. However, our splitting schema only works in the absence of nested probabilistic effects, a form that can in general only be obtained by a compilation that incurs exponential blowup. The process flowchart of Figure 3.5 shows that our procedure continues with a conversion of all body subactions to unary conditionality normal

form (which has been introduced in Section 2.2). In unary conditionality normal form, each effect e can be written as

$$e = \underbrace{\bigwedge_i (c_i \rhd e_i)}_{E_d} \wedge \underbrace{\bigwedge_j \big((p_{j1} : e_{j1}) \mid \cdots \mid (p_{jn_j} : e_{jn_j}) \big)}_{E_p}, \tag{3.1}$$

with a deterministic part E_d that is a conjunction of conditional effects $c_i \rhd e_i$ with atomic effects e' (and possibly empty conditions c_i); and a probabilistic part E_p that is a conjunction of probabilistic effects where each effect e_{jk} is an effect of the form of Equation 3.1 itself. In the following, we also write $e = \langle E_d, E_p \rangle$ for an effect e in unary conditionality normal form, where E_d and E_p are as given by Equation 3.1. Let us now introduce the forked normal form, a normal form for actions that is used to split effects in a way that allows the computation of an efficient all-outcomes determinization.

Definition 17 (Forked Normal Form). *An effect e is in* **forked normal form** *iff it is of the form*

$$\left(\bigwedge_i e_i \right) \wedge \big((p_1 : e'_1) \mid \cdots \mid (p_n : e'_n) \big),$$

where all e_i and $e'_1, \ldots, e'_n$ are conditional effects $c \rhd e''$ with atomic effect e''. An action is in forked normal form iff its effect is in forked normal form, and a finite-horizon, factored MDP is in forked normal form iff all its actions are in forked normal form.

While there are effects that cannot be transformed into a *single* effect in forked normal form, it is interesting nevertheless as there is a polynomial time and space algorithm that transforms a finite-horizon, factored MDP $\mathcal{M}$ into an equivalent finite-horizon, factored MDP $\mathcal{M}^\Psi$ where all actions are in forked normal form and where $|\mathcal{A}^\Psi|$ is polynomial in $|\mathcal{A}|$. The algorithm is given in Algorithm 3. It creates an *FNF-DAG* for each action, which is a directed acyclic graph $\mathcal{F} = \langle V_\mathcal{F}, E_\mathcal{F}, v_0 \rangle$ with vertices $v = \langle E_{\text{det}}, \text{idx} \rangle$ with assigned deterministic effect $E_{\text{det}} = \bigwedge_j e_j$ and an unique index idx; with edges $e = \langle v, v', p \rangle$ from vertex v to v' that are labeled with a probability $p \in [0; 1]$; and with root node $v_0 \in V_\mathcal{F}$. The FNF-DAG is created by recursively calling the procedure `build_fnf_dag` for each effect $e = \langle E_d, E_p \rangle$ that is in the form of Equation 3.1 until e is deterministic. Each created FNF-DAG that reflects a parallel probabilistic effect is connected with the `append_fnf_dags` function to all leaves of the current FNF-DAG, such that each path from the root to a leaf in the created structure corresponds to one possible set of effects that take place when the action is applied.

Let us have a look at an example to showcase Algorithm 3. As our running example for this chapter, the instance of the CANADIAN TRAVELER'S PROBLEM, does not contain nested effects, we use an abstract example instead.

Algorithm 3: Compilation of actions to forked normal form

translate_to_forked_normal_form($\mathcal{A}^{\dashv}$)**:**
  $\mathcal{A}^{\Psi} \leftarrow \{\}$
  for $a^{\dashv} \in \mathcal{A}^{\dashv}$ **do**
    $\mathcal{A}^{\Psi}_{a^{\dashv}} \leftarrow$ translate_to_forked_normal_form($a^{\dashv}$)
    $\mathcal{A}^{\Psi} \leftarrow \mathcal{A}^{\Psi} \cup \mathcal{A}^{\Psi}_{a^{\dashv}}$
  return $\mathcal{A}^{\Psi}$

translate_to_forked_normal_form($a^{\dashv}$)**:**
  $\mathcal{A}_{a^{\dashv}} \leftarrow \{\}$
  $\langle V_{\mathcal{F}}, E_{\mathcal{F}}, v_0 \rangle \leftarrow$ build_fnf_dag(eff($a^{\dashv}$))
  for $v \in V_{\mathcal{F}}$ **do**
    if v is a leaf **then** $e^{\Psi} \leftarrow E_{\text{det}}(v) \wedge v_{\text{aux}}{\leftarrow}0$*;*
    else $e^{\Psi} \leftarrow E_{\text{det}}(v) \wedge \big|_{\langle v,v',p\rangle \in E_{\mathcal{F}}} (p : v_{\text{aux}}{\leftarrow}\text{idx}(v'))$*;*
    $\mathcal{A}_{a^{\dashv}} \leftarrow \mathcal{A}_{a^{\dashv}} \cup \{\langle \{v_{\text{aux}} \mapsto \text{idx}(v)\}, e^{\Psi} \rangle\}$
  return $\mathcal{A}_{a^{\dashv}}$

build_fnf_dag($\langle E_d, E_p \rangle$)**:**
  $v \leftarrow \langle E_d,$ generate_index()$\rangle, \mathcal{F} \leftarrow \langle \{v\}, \{\}, v \rangle$
  for $e_p \in E_p$ **do**
    $\mathcal{F}_{e_p} \leftarrow \{\}$
    for $(p : e') \in e_p$ **do**
      $\mathcal{F}_{e_p} \leftarrow \mathcal{F}_{e_p} \cup \{\langle$build_fnf_dag($e'$)$, p \rangle\}$
    $\mathcal{F} \leftarrow$ append_fnf_dags($\mathcal{F}, \mathcal{F}_{e_p}$)
  return $\mathcal{F}$

append_fnf_dags($\langle V_{\mathcal{F}}, E_{\mathcal{F}}, v_0 \rangle, \{\langle \mathcal{F}_1, p_1 \rangle, \ldots, \langle \mathcal{F}_n, p_n \rangle\}$)
  Let $L_{\mathcal{F}}$ be the set of leaves of $\langle V_{\mathcal{F}}, E_{\mathcal{F}}, v_0 \rangle$
  for $\langle \langle V', E', v'_0 \rangle, p' \rangle \in \{\langle \mathcal{F}_1, p_1 \rangle, \ldots, \langle \mathcal{F}_n, p_n \rangle\}$ **do**
    $V_{\mathcal{F}} \leftarrow V_{\mathcal{F}} \cup V'$
    $E_{\mathcal{F}} \leftarrow E_{\mathcal{F}} \cup E'$
    for $v \in L_{\mathcal{F}}$ **do**
      $E_{\mathcal{F}} \leftarrow E_{\mathcal{F}} \cup \langle (v, v'_0), p' \rangle$
  return $\langle V_{\mathcal{F}}, E_{\mathcal{F}}, v_0 \rangle$

Example 13. *Consider the action a with* eff(a) $= e$ *such that*

$$
\begin{aligned}
e &= (q{\leftarrow}2) \wedge \big((0.5 : e_1) \mid (0.5 : e_2)\big) \wedge \big((0.5 : e_3) \mid (0.5 : e_4)\big), \text{where} \\
e_1 &= (s{\leftarrow}0) \wedge \big((0.3 : t{\leftarrow}1) \mid (0.7 : u{\leftarrow}3)\big) \wedge \big((0.6 : v{\leftarrow}0) \mid (0.4 : w{\leftarrow}4)\big), \\
e_2 &= (x{\leftarrow}1), \\
e_3 &= (y{\leftarrow}2) \wedge (0.4 : z{\leftarrow}0 \mid 0.6 : z{\leftarrow}2), \text{ and} \\
e_4 &= (z{\leftarrow}1).
\end{aligned}
$$

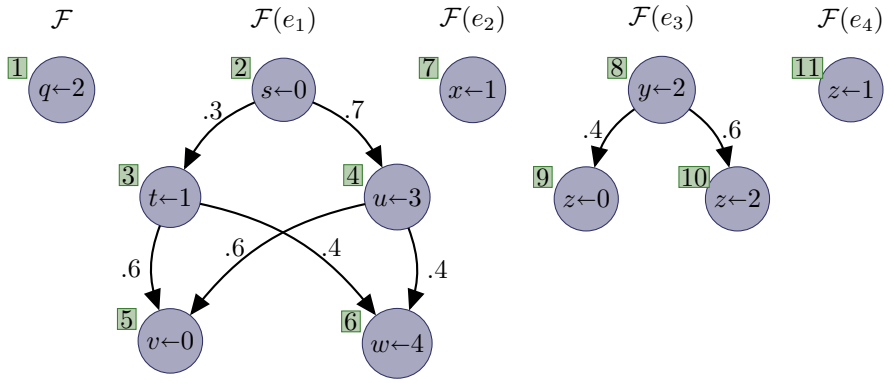

Figure 3.7: Intermediate FNF-DAGs that are created by Algorithm 3 for the action of Example 13. Numbers to the left of each node give the values of v_{aux} in preconditions, and labels give transition probabilities.

Let the k-th generated index be k. Algorithm 3 starts by creating an FNF-DAG that only contains a single vertex with effect $q{\leftarrow}2$ in line 16, which is depicted as $\mathcal{F}$ on the left of Figure 3.7. In the first iteration of the loop in lines 17–21, the FNF-DAGs for e_1 and e_2 are created as illustrated in Figure 3.7 and appended to $\mathcal{F}$ by connecting all leaves of $\mathcal{F}$ with the roots of $\mathcal{F}(e_1)$ and $\mathcal{F}(e_2)$. In the second iteration, the FNF-DAGs $\mathcal{F}(e_3)$ and $\mathcal{F}(e_4)$ are created and appended to all leaves of the result of the first step. The final result is depicted in Figure 3.8, where the small graph shows how the partial FNF-DAGs $\mathcal{F}, \mathcal{F}(e_1), \ldots, \mathcal{F}(e_4)$ are connected to obtain the complete FNF-DAG in the right part of the figure. The actions $a_1^\psi, \ldots, a_{11}^\psi$ in forked normal form that are derived from the FNF-DAG in Figure 3.8 are:

$$
\begin{aligned}
a_1^\psi &= \langle \{v_{\text{aux}} \mapsto 1\}, q{\leftarrow}2 \wedge (0.5 : v_{\text{aux}}{\leftarrow}2 \mid 0.5 : v_{\text{aux}}{\leftarrow}7) \rangle \\
a_2^\psi &= \langle \{v_{\text{aux}} \mapsto 2\}, s{\leftarrow}0 \wedge (0.3 : v_{\text{aux}}{\leftarrow}3 \mid 0.7 : v_{\text{aux}}{\leftarrow}4) \rangle \\
a_3^\psi &= \langle \{v_{\text{aux}} \mapsto 3\}, t{\leftarrow}1 \wedge (0.6 : v_{\text{aux}}{\leftarrow}5 \mid 0.4 : v_{\text{aux}}{\leftarrow}6) \rangle \\
a_4^\psi &= \langle \{v_{\text{aux}} \mapsto 4\}, u{\leftarrow}3 \wedge (0.6 : v_{\text{aux}}{\leftarrow}5 \mid 0.4 : v_{\text{aux}}{\leftarrow}6) \rangle \\
a_5^\psi &= \langle \{v_{\text{aux}} \mapsto 5\}, v{\leftarrow}0 \wedge (0.5 : v_{\text{aux}}{\leftarrow}8 \mid 0.5 : v_{\text{aux}}{\leftarrow}11) \rangle \\
a_6^\psi &= \langle \{v_{\text{aux}} \mapsto 6\}, w{\leftarrow}4 \wedge (0.5 : v_{\text{aux}}{\leftarrow}8 \mid 0.5 : v_{\text{aux}}{\leftarrow}11) \rangle \\
a_7^\psi &= \langle \{v_{\text{aux}} \mapsto 7\}, x{\leftarrow}1 \wedge (0.5 : v_{\text{aux}}{\leftarrow}8 \mid 0.5 : v_{\text{aux}}{\leftarrow}11) \rangle \\
a_8^\psi &= \langle \{v_{\text{aux}} \mapsto 8\}, y{\leftarrow}2 \wedge (0.4 : v_{\text{aux}}{\leftarrow}9 \mid 0.6 : v_{\text{aux}}{\leftarrow}10) \rangle \\
a_9^\psi &= \langle \{v_{\text{aux}} \mapsto 9\}, z{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}0 \rangle \\
a_{10}^\psi &= \langle \{v_{\text{aux}} \mapsto 10\}, z{\leftarrow}2 \wedge v_{\text{aux}}{\leftarrow}0 \rangle \\
a_{11}^\psi &= \langle \{v_{\text{aux}} \mapsto 11\}, z{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}0 \rangle
\end{aligned}
$$

Now that we have seen an example compilation of an action to forked

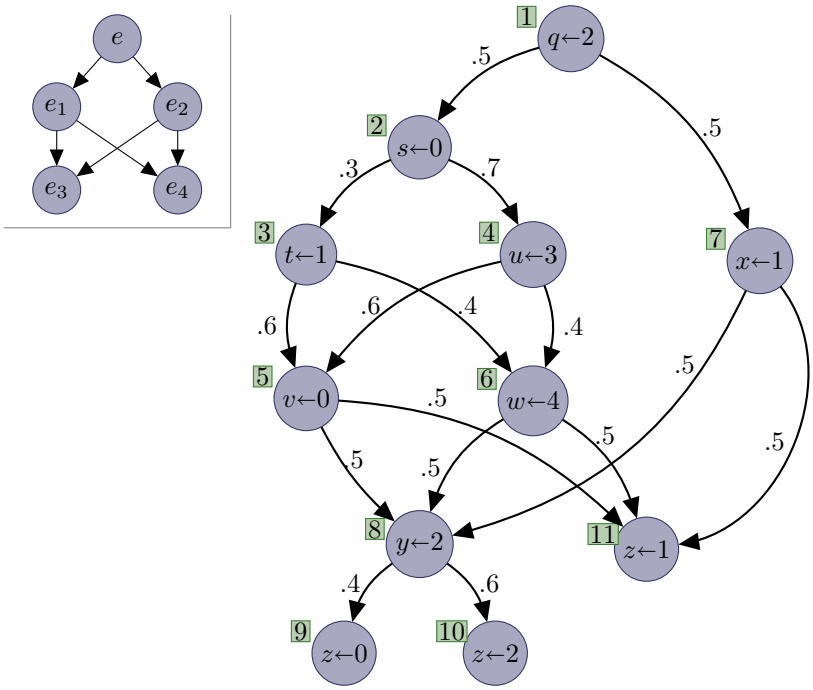

Figure 3.8: The final FNF-DAG that is created by Algorithm 3 for the action of Example 13 by connecting the components of Figure 3.7 as indicated by the small graph.

normal form with the help of an FNF-DAG, we show that the procedure is applicable in polynomial time and space to all actions.

Theorem 2. *Each finite-horizon, factored MDP can be normalized in polynomial time and space to an equivalent finite-horizon, factored MDP in forked normal form.*

Proof: We have already covered the first two steps of the normalization, where the initial finite-horizon, factored MDP $\mathcal{M}$ is translated to a finite-horizon, factored MDP $\mathcal{M}^{\triangleright} = \langle \mathcal{V}^{\triangleright}, \mathcal{A}^{\vdash} \cup \mathcal{A}^{\dashv}, \mathcal{R}^{\triangleright}, s_0^{\triangleright} \rangle$ in splittable normal form, and where all actions $a^{\dashv} \in \mathcal{A}^{\dashv}$ are in unary conditionality normal form. We have also discussed that both procedures take polynomial time and space. It remains to show how to translate $\mathcal{M}^{\triangleright}$ to an equivalent finite-horizon, factored MDP $\mathcal{M}^{\Psi}$ where $\mathcal{V}^{\Psi} = \mathcal{V}^{\triangleright}$; $\mathcal{A}^{\Psi} = \mathcal{A}^{\vdash} \cup \{\mathcal{A}^{\Psi}_{a^{\dashv}} \mid a \in \mathcal{A}^{\dashv}\}$; and $s_0^{\Psi} = s_0^{\triangleright}$. The only non-trivial part of the compilation is the generation of a set of actions $\mathcal{A}^{\Psi}_{a^{\dashv}}$ for each $a^{\dashv} \in \mathcal{A}^{\dashv}$, which is given by Algorithm 3. There is no need to translate $\mathcal{A}^{\vdash}$ since all $a^{\vdash} \in \mathcal{A}^{\vdash}$ are already in forked normal form according to Definition 16.

It is easy to see that most parts of Algorithm 3 can be computed in polynomial time and space. Exceptions are the collection of the leaves of an FNF-DAG

(in line 24) and the recursive call that builds the FNF-DAG (in line 20). Here, the important argument is that the size of the FNF-DAG of an action is bound by the number of effects of the form of Equation 3.1 in the action's effect, which is linear in the input size of the MDP. As each recursion step creates exactly one vertex and as the collection of leaves depends on the size of the FNF-DAG, both can be computed in polynomial time.

Equivalence of $\mathcal{M}^{\Psi}$ and $\mathcal{M}$ can be shown by using the same mappings σ and α as in the proof of Theorem 1. The trajectories that replace the transitions of $\mathcal{M}$ are all trajectories that are induced by the created actions in forked normal form (prepended by the corresponding $a^{\vdash}$). Due to the uniqueness of v_{aux}, all are such that there is only one applicable action in each intermediate state, and that there is no $s \in \mathcal{S}$ where $\sigma(s) = s'$ because $v_{\text{aux}} \mapsto 0 \notin s'$ for all intermediate states s'. Accumulated rewards of transitions and trajectories match for the same reason they match in Theorem 1 (rewards are taken care of by $a^{\vdash}$). Finally, effects and combined probabilities of mapped transitions and trajectories match because each pair of probabilistic effect and probability in an action's effect is appended to the FNF-DAG such that each trajectory has to sample the outcome on that variable according to its probability. ■

Determinization The computation of an all-outcomes determinization $\mathcal{M}^{\text{d}}$ of a given finite-horizon, factored MDP $\mathcal{M}$ is simple once $\mathcal{M}$ has been by converted to a finite-horizon, factored MDP $\mathcal{M}^{\Psi}$ in forked normal form. Since all actions $a^{\vdash} \in \mathcal{A}^{\triangleright}$ are already deterministic, only the actions $a^{\Psi} \in \mathcal{A}^{\Psi}$ must be determinized. They have in common that $\text{pre}(a^{\Psi}) = \{v_{\text{aux}} \mapsto k\}$ for some unique $k > 0$ and

$$\text{eff}(a^{\Psi}) = \underbrace{\bigwedge_{i} e_i}_{E_d} \wedge \underbrace{\big((p_1 : v_{\text{aux}} \leftarrow x_1) \mid \cdots \mid (p_n : v_{\text{aux}} \leftarrow x_n) \big)}_{E_{\text{aux}}}$$

with $x_1, \ldots, x_n \geq 0$ (a value of 0 is assigned to v_{aux} only iff a^{Ψ} is deterministic, though). As in the all-outcomes determinization that is derived by compiling the MDP to unary non-determinism normal form, we replace the probabilistic transition to a successor state with a representation that is such that the path can be selected. All we have to do is to replace the probabilistic effect E_{aux} over v_{aux} with a deterministic one, and we have to assign new values to the variable v_{aux} that determines which actions can be applied at what point in the sequence.

Example 14. *The all-outcomes determinization of the action from Example 13*

Locations/Actions	Roads	Det. Actions	⌀Effects
20/98	4.9 ± 3.7	1122 ↑11.4	3.7
		6008 ↑61.3	21.9
50/278	5.56 ± 4.8	3550 ↑12.8	3.7
		29744 ↑107.0	25.3
100/568	5.68 ± 3.7	7348 ↑12.9	3.7
		55160 ↑97.0	23.7

Table 3.1: Statistics on all-outcomes determinizations from a compilation to unary non-determinism normal form (white) and forked normal form (gray) on three instances of the CTP with different number of locations, actions and average number of adjacent roads per location. We state the number of generated deterministic actions (including the blowup factor relative to the number of original actions, which is indicated by ↑) and the average number of effects of the generated actions.

is given as a set of actions that contains exactly the following actions::

$$a_1^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 1, q{\leftarrow}2 \wedge v_{\text{aux}}{\leftarrow}2 \rangle \qquad a_2^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 2, s{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}3 \rangle$$
$$a_3^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 3, t{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}4 \rangle \qquad a_4^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 3, u{\leftarrow}3 \wedge v_{\text{aux}}{\leftarrow}4 \rangle$$
$$a_5^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 4, v{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}5 \rangle \qquad a_6^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 4, w{\leftarrow}4 \wedge v_{\text{aux}}{\leftarrow}5 \rangle$$
$$a_7^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 2, x{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}5 \rangle \qquad a_8^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 5, y{\leftarrow}2 \wedge v_{\text{aux}}{\leftarrow}6 \rangle$$
$$a_9^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 6, z{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}0 \rangle \qquad a_{10}^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 6, z{\leftarrow}2 \wedge v_{\text{aux}}{\leftarrow}0 \rangle$$
$$a_{11}^{\mathrm{d}} = \langle v_{\text{aux}} \mapsto 5, z{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}0 \rangle$$

Experimental Evaluation One of the main criticism of the International Probabilistic Planning Competitions before 2011 was their lack of probabilistically interesting problems (Little and Thiébaux, 2007) and hence also their lack of parallel probabilistic effects. Even though this changed with IPPC 2011, there are no standard benchmarks that can be used to evaluate our method empirically as the input language of IPPC 2011 and 2014, RDDL (Sanner, 2010), differs significantly from PPDDL such that a determinization via forked normal form cannot be computed. We therefore use the CTP instances that were created for our work on policies for the CTP domain (Eyerich et al., 2010), where independent probabilistic effects play an important role.

Table 3.1 compares the two all-outcomes determinizations that were presented in this section. For three (hand-selected but typical) instances of the

CTP with an increasing number of locations and ground actions, we denote the number of generated deterministic actions in the all-outcomes determinization, the blowup factor relative to the number of probabilistic actions (indicated by ↑ in the table), and the average number of conditional or atomic effects in each of the generated actions.

It can be seen that the all-outcomes determinization that is based on forked normal form generates significantly less actions than the determinization via unary non-determinism normal form. The number of actions in the determinization that is derived by compilation to forked normal form is approximately 12 times the number of actions in the probabilistic MDP, while it is between roughly 60 and 110 times that number for the compilation via unary non-determinism normal form. The large difference for the latter is due to its exponential dependency on the number of independent probabilistic effects: the more of those, the larger the blowup factor, which also explains why the blowup factor in the problem with 50 locations (standard deviation of 4.8) is larger than in the problem with 100 locations (standard deviation of 3.7). In contrast to that, an all-outcomes determinization from an MDP in forked normal form depends only linearly on the average number of independent probabilistic effects and incurs a similar blowup factor over problems of all sizes. As a side effect, the deterministic actions that are created by the determinization with forked normal form are also *much smaller* than those that are created from an MDP in unary non-determinism normal form – the former contain an average number of 3.7 atomic effects, independently from the problem size, while the latter are a conjunctive effect with up to 25.3 elements.

We finish our discussion of determinization techniques with this encouraging result and turn our attention to MDP algorithms that entirely base their decisions on determinzations that can be computed as efficiently as the ones that have been presented over the course of this chapter.

CHAPTER 4

Suboptimal Policies

Now that we have the preliminaries and basic concepts of planning under uncertainty behind us, we are ready to start our discussion of different algorithms for finite-horizon, factored MDPs. In this chapter, we present three methods that are, despite being suboptimal, among the state-of-the-art in a variety of applications. The first one is the *optimistic policy*, which is presented in Section 4.1. It computes an all-outcomes determinization in each step, and executes the action that corresponds to the optimal decision in the determinization. *Hindsight optimization*, which is introduced in Section 4.2, is a popular algorithm that takes at least some uncertainty into account: it repeatedly generates a weather of the MDP, and computes action value estimates by averaging over action values in the determinization. In Section 4.3, we present *optimistic rollout*, an algorithm that overcomes the *assumption of clairvoyance* that causes hindsight optimization to fail under certain circumstances by simulating *trials* where it follows the optimistic policy. The theoretical evaluation in Section 4.4 shows that neither of the presented methods achieves optimal behavior even with unlimited resources. However, the empirical data on the CANADIAN TRAVELER'S PROBLEM that is presented in Section 4.5 nonetheless emphasizes that there is a (positive) correlation between the performance of each algorithm and the amount of uncertainty it takes into account.

4.1 Optimism

The first algorithm for planning and acting under uncertainty that is considered in this thesis – apart from the practically intractable acyclic value computation that was presented in Section 2.3 – is the *optimistic policy* (OMT). Recall that all algorithms that are considered in this thesis are online algorithms that do not compute a complete policy but alternatingly plan a decision for the current state and execute the corresponding action. The optimistic policy does so

by compiling the input MDP $\mathcal{M}$ to an all-outcomes determinization $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle$. The simplified task $\mathcal{M}^{\mathrm{d}}$ is then solved, often by invoking an external, classical planning system like FF (Hoffmann and Nebel, 2001) or Fast Downward (Helmert, 2006). The action-value of an action $a \in \mathcal{A}(s_0)$ under the optimistic policy $Q_{\mathrm{OMT}}(s_0, a)$ corresponds to the maximum over all action-values $Q^{\mathrm{d}}_{\star}(s_0, a^{\mathrm{d}})$ of all deterministic actions $a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}}$ that are a determinization of a:

$$Q_{\mathrm{OMT}}(s_0, a) = \max_{a^{\mathrm{d}} \in \{a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}} | \delta(a^{\mathrm{d}}) = a\}} Q^{\mathrm{d}}_{\star}(s_0, a^{\mathrm{d}}).$$

Optimism is a popular approach for many MDP applications, and is especially popular in the area of robotic motion planning in uncertain environments, where many papers focus on efficient implementations of the optimistic policy (e.g., Stentz, 1994; Koenig and Likhachev, 2002). In fact, a large percentage of the work on path planning under uncertainty considers the optimistic policy to the exclusion of everything else, so it is a natural baseline for all other algorithms that are presented over the course of this thesis. One of the reasons for the popularity of optimism in path planning problems under uncertainty is that the optimistic policy can be computed with a simpler and much more efficient method than by computing and solving an all-outcomes determinization of the MDP that is induced by the problem instance: it suffices to alter the roadmap graph according to what is called the *free space assumption* in the robotics literature: as long as we have not observed that a road is blocked, we assume that it is traversable. We call the (deterministic) roadmap that is created by replacing all roads with unknown status with traversable roads the *optimistic roadmap*. The action-value estimate $Q_{\mathrm{OMT}}(s, a)$ of an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ can then be computed as the distance from the agent's location in s to the goal state in the optimistic roadmap.

Finding shortest paths in the optimistic roadmap is a standard shortest path problem without uncertainty, and Q_{OMT} can hence be computed efficiently. A sophisticated implementation might use algorithms like D* Lite (Koenig and Likhachev, 2002) to speed up the distance computations by exploiting that, over the course of a run, an agent solves a sequence of *similar* path planning problems, allowing reuse of information. Since the focus of the evaluation of our algorithms in Section 4.5 is on the *quality* of the different policies, which is not affected by how the optimistic policy is computed, our implementation simply uses Dijkstra's algorithm.

Example 15. *We have introduced an example* CTP *instance in the previous chapter in Figure 3.1. The optimistic roadmap of that instance, which is depicted in Figure 4.1, differs from the original roadmap only in the fact that all roads with unknown status (which are depicted as dashed edges) are replaced by traversable roads (which are depicted as solid edges). The action-value estimates of the three applicable actions that move the agent to v_1, v_5, or $v_\star$ that are computed by the optimistic policy correspond to the cost of the shortest path in the optimistic road map. They are $Q_{\mathrm{OMT}}(s_0, \mathtt{move}_{05}) = -60$ (on the path via v_6),*

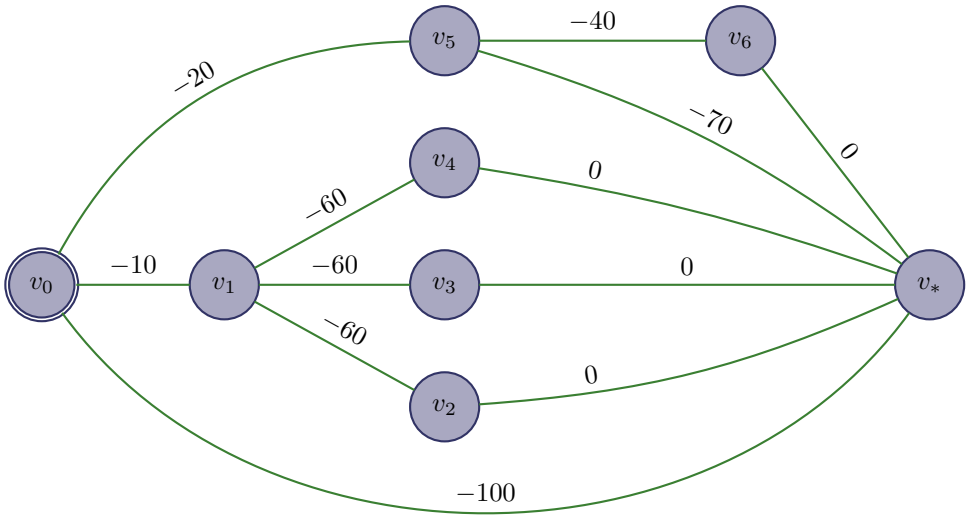

Figure 4.1: The optimistic roadmap of the CTP example instance that was introduced in Figure 3.1 in Chapter 3.

$Q_{\text{OMT}}(s_0, \texttt{move}_{01}) = -70$ *(on any of the three paths over* v_2*,* v_3*, or* v_4*), and* $Q_{\text{OMT}}(s_0, \texttt{move}_{0\star}) = -100$*. The first step that is taken by the optimistic policy in our example instance is hence* $\texttt{move}_{05}$*. While this first step is conform with the optimal decision, it will try to follow the path via* v_6 *even though the road between* v_6 *and* $v_\star$ *is blocked with high probability.*

The term "optimistic policy" derives from a property of optimal policies in an all-outcomes determinization: since there is a deterministic action for each outcome of a probabilistic action, an algorithm that solves $\mathcal{M}^{\text{d}}$ has full control over the outcomes of actions and can hence always choose the outcome that takes place. According to Definition 11, an optimal solver computes a policy that is such that $V^{\text{d}}_\star(s_0)$ is maximal. It will hence always select deterministic actions where only the effects take place that reflect the best outcome of the probabilistic actions in $\mathcal{M}$ – or, in other words, it is *optimistic* that all random events will turn out in its favor. It should come as no surprise that the optimistic policy can be led astray easily in all but the simplest applications, and Example 15 shows that the optimistic assumption can indeed be inherently poor.

Despite the liability of the optimistic policy to adhere to a plan that has an unrealistically low probability of success, it is a popular algorithm in many applications. Little and Thiébaux (2007) argue that *determinization-based replanning* algorithms like the optimistic policy are popular in applications that have uncertain outcomes but which are not probabilistically interesting. They argue that a problem falls in this category if, e. g., non-deterministic actions can be repeated until they succeed or because there are no alternative action

Algorithm 4: Optimistic policy

optimistic_policy$(\mathcal{M})$**:**
    $\langle \mathcal{M}^{\mathrm{d}}, \delta \rangle =$ compute_all_outcomes_determinization$(\mathcal{M})$
    return deterministic_policy$(\mathcal{M}, \langle \mathcal{M}^{\mathrm{d}}, \delta \rangle, \pi^\star)$
deterministic_policy$(\mathcal{M}, \langle \mathcal{M}^{\mathrm{d}}, \delta \rangle, \pi)$**:**
    for $a \in \mathcal{A}(s_0)$ **do**
        $Q_{\text{DetPol}}(s_0, a) \leftarrow \max_{a^{\mathrm{d}} \in \{a^{\mathrm{d}} \in \mathcal{A}^{\mathrm{d}} \mid \delta(a^{\mathrm{d}}) = a\}} Q^{\mathrm{d}}_\pi(s_0, a^{\mathrm{d}})$
    $a_{\text{OMT}} \sim \mathcal{U}\left(\arg\max_{a \in \mathcal{A}(s_0)} Q_{\text{DetPol}}(s_0, a)\right)$
    return a_{OMT}

sequences that lead to a goal state. It is not hard to see that the optimistic policy is a reasonable algorithm for such a task, since there is usually no better policy than one that is built on the assumption that all action outcomes will be as desired. However, we will see from our empirical results that both the benchmarks of the last two installments of the IPPC and the CTP are domains where reasoning over uncertainty pays off.

We have seen in Example 15 that computing Q_{OMT} in the CTP reduces to the comparably simple problem of finding a shortest path in the roadmap. However, this can not be generalized to arbitrary MDPs, where it is usually not only intractable to solve the MDP itself, but also impossible to compute an optimal solution to the MDPs' all-outcomes determinization in practice. There are two possibilities to adjust the problem at hand that make it more likely that the resulting optimization problem is solvable in practice. The first is to drop the requirement that the determinization is solved optimally. This is, for instance, the strategy of the FF-Replan version of 2008 (Yoon et al., 2007), which uses the satisficing (and hence not necessarily optimal) FF planner to determine approximate action-values in the all-outcomes determinization. Other than that, FF-Replan is identical to our description of the optimistic policy, i. e., action-value estimates for all actions are derived from the corresponding action-values in determinization. The second option is the usage of a determinization that can be solved faster (in practice, not in terms of complexity) than an all-outcomes determinization. The FF-Replan version of 2006 won the IPPC with a combination of both strategies, i. e., it used the FF planner to compute approximate action-values in the determinization, which was a most-likely determinization rather than the all-outcomes determinization that was used two years later (the branching factor, which is given by the number of applicable actions, is usually significantly smaller in the most-likely determinization than in the all-outcomes determinization, and it can hence be solved faster in practice).

While the FF-Replan version of 2008 still computes a policy that can be regarded as an optimistic policy (just a suboptimal version of it), this is no

Algorithm 5: Hindsight optimization

hindsight_optimization$(\mathcal{M}, n)$**:**
  for $a \in \mathcal{A}(s_0)$ **do** $\hat{Q}^0_{\text{HOP}}(s_0, a) \leftarrow 0$;
  for $k = 1, \ldots, n$ **do**
    $\langle \mathcal{M}^{\text{w}}, \delta \rangle = \text{sample\_weather}(\mathcal{M})$
    for $a \in \mathcal{A}(s_0)$ **do**
      $\hat{Q}^k_{\text{HOP}}(s_0, a) \leftarrow \hat{Q}^{k-1}_{\text{HOP}}(s_0, a) + \frac{Q^{\text{w}}_\star(s^{\text{w}}_0, \delta(a)) - \hat{Q}^{k-1}_{\text{HOP}}(s_0, a)}{k}$
  $a_{\text{HOP}} \sim \mathcal{U}\left(\arg\max_{a \in \mathcal{A}(s_0)} \hat{Q}^n_{\text{HOP}}(s_0, a)\right)$
  return a_{HOP}

longer the case for the FF-Replan version of 2006 where outcomes are selected according to their probability and not according to outcome preferences. The approaches are related nevertheless and can be described in terms of the common algorithmic framework that is given as the *deterministic policy* family in Algorithm 4. Different members of the family are distinguished by two parameters: first, a determinization $\langle \mathcal{M}^{\text{d}}, \delta \rangle$ of the input MDP $\mathcal{M}$; and second, the solver π that is used to compute action-value estimates for all applicable actions in $\mathcal{M}^{\text{d}}$. Choosing an all-outcomes determinization and an optimal solver for both parameters leads to the optimistic policy. FF-Replan 2008 is derived as the determinization-based replanning algorithm that uses an all-outcomes determinization and the FF-Planner, and FF-Replan 2006 by choosing the most-likely determinization and the FF-planner as respective parameters. Even though we are not aware of other members in the family of determinization-based replanning algorithms that are used in practice, many other combinations are possible. We restrict our analysis in the following to the most popular member of the family, the optimistic policy, though.

4.2 Hindsight Optimization

The optimistic policy is indeed exceedingly optimistic: if we look at it from a different angle, all of its action-value estimates are such that they correspond to the maximal possible reward in the *best possible weather*. An alternative approach that is less optimistic but still allows the reduction of the action-value estimation to solving (a series of) deterministic MDPs is *hindsight optimization* (HOP). The hindsight optimization approach samples and solves a series of n weathers of the MDP, and computes its action-value estimates $\hat{Q}^n_{\text{HOP}}$ as the average of the action-values in the sampled weathers. The number of weathers that is used for the computation of the action-value estimates of hindsight optimization is a parameter of the algorithm. More trials require more time, but tend to produce more stable action-value estimates. The algorithm (with an iterative computation of the average) is given as Algorithm 5.

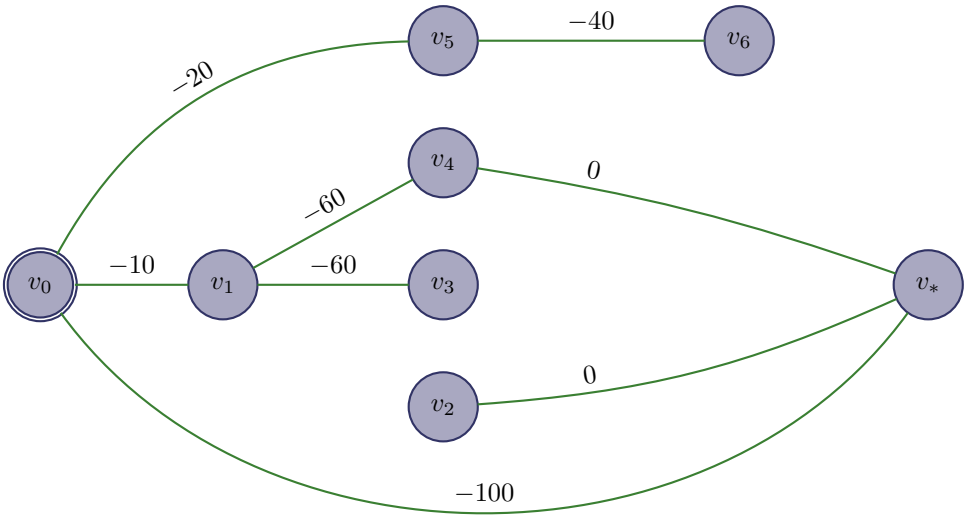

Figure 4.2: Roadmap of an example weather of the CTP example instance that was introduced in Chapter 3.

An alternative and fairly descriptive name for hindsight optimization is *averaging over clairvoyance* (Russell and Norvig, 1995). For each weather we consider, we assume that the agent is "clairvoyant", i. e., knows ahead of time the result of all random events. Since we do not know the actual weather, we *average* over several weathers (that are obtained through stochastic sampling, see Chapter 3 for details). Hindsight optimization has attracted considerable interest in the stochastic planning community, where it was used for network scheduling problems (Chong et al., 2000), for task and motion planning of robots (e.g., Godoy et al., 2014; Kiesel and Ruml, 2014), or as the basis of the highly efficient, domain-independent planning system FF-Hindsight (Yoon et al., 2008, 2010). To our knowledge, policies based on hindsight optimization for the CTP have not been considered prior to our work (Eyerich et al., 2010). However, Bnaya et al. (2008) independently suggested essentially the same idea for the *sensing decisions* in an algorithm for the CTP with remote sensing. (Their movement decisions are based on the optimistic policy, though.) Hindsight optimization has furthermore been successfully used for dealing with hidden information in card games, including the single-player game Klondike Solitaire (Bjarnason et al., 2009) and the two-party games Bridge (Ginsberg, 1999) and Skat (Buro et al., 2009). Like the determinization-based replanning framework of the previous section, the algorithms that are used in these applications actually span a whole family of related algorithms that share the computation of action-value estimates and differ only in the solver that is used to compute a solution of the sampled weather. Replacing the optimal deterministic solver that computes $\pi^\star$ with a solver that is provided by a parameter extends Algorithm 5 from hindsight

optimization to the respective family of algorithms. It would, for instance, include the popular FF-Hindsight planner (Yoon et al., 2008, 2010).

Example 16. *As in the computation of the optimistic policy for the* CTP*, we can use domain-specific knowledge and simplify the computation of a policy in a weather of the* CTP *by computing a shortest path in a roadmap instead of solving a (determinized) MDP: it corresponds to the cost of the shortest path between the location of the current state and the goal location. Please note that it is possible that a weather is such that there is no path from the current location to the goal location. In our work on the* CTP *(Eyerich et al., 2010), we excluded this case by ensuring with the* good weather assumption *that there is always a traversable path to the goal location. In this thesis, where we consider finite-horizon, factored MDPs instead of Stochastic Shortest Path Problems, a* `give-up` *action can be used instead that is applicable only if the current location and the goal location are not connected and which incurs a high, constant cost. Since both assumptions lead to well-defined state- and action-values, and since our running example is such that the road $v_0 - v_\star$ is always traversable, we ignore this subtlety in the remainder of our discussion.*

An example weather $\mathcal{M}^{\mathrm{w}}$ of the CTP *example instance is depicted in Figure 4.2. The action-values in the weather are $Q^{\mathrm{w}}_\star(s^{\mathrm{w}}_0, \delta(\mathtt{move}_{05})) = -110$ due to the path $v_0 - v_5 - v_0 - v_1 - v_4 - v_\star$, $Q^{\mathrm{w}}_\star(s^{\mathrm{w}}_0, \delta(\mathtt{move}_{01})) = -70$ on the direct path over v_4, and $Q^{\mathrm{w}}_\star(s^{\mathrm{w}}_0, \delta(\mathtt{move}_{0\star})) = -100$. Hindsight optimization repeatedly samples weathers, computes the shortest paths in the corresponding roadmap graph and maintains an average value $\hat{Q}^k_{\mathrm{HOP}}$ for all applicable actions. After n trials have been performed in this way, the action with the highest action-value estimate $\hat{Q}^n_{\mathrm{HOP}}$ is executed.*

Despite the success of hindsight optimization in a large number of applications, the approach has well-known theoretical weaknesses: it often converges to a suboptimal policy as the number of trials approaches infinity. Frank and Basin (2001) give an example of this for the game of Bridge, and Russell and Norvig (1995) describe a very simple MDP where hindsight optimization fails. Instead of repeating the example of Russell and Norvig, we discuss the algorithms of this chapter in the context of the CTP domain in the theoretical evaluation in Section 4.4, and give an example of the suboptimality of the hindsight optimization policy for the CTP.

4.3 Optimistic Rollout

The assumption of clairvoyance is the Achilles heel of the hindsight optimization approach. Our next algorithm, *optimistic rollout* (ORO), addresses this issue by performing a sequence of iterations called *trials*[1]. A trial differs from

[1]The name "optimistic rollout" comes from our initial work on the topic (Eyerich et al., 2010) where trials were called *rollouts*.

Algorithm 6: Optimistic rollout

optimistic_rollout$(\mathcal{M}, n)$**:**
  for $a \in \mathcal{A}(s_0)$ **do** $\hat{Q}^0_{\text{ORO}}(s_0, a) \leftarrow 0$;
  for $k = 1, \ldots, n$ **do**
    $\langle \mathcal{M}^{\text{w}}, \delta \rangle = \text{sample\_weather}(\mathcal{M})$
    for $a \in \mathcal{A}(s_0)$ **do**
      $r \leftarrow \text{perform\_trial}(\mathcal{M}^{\text{w}}, \delta(a))$
      $\hat{Q}^k_{\text{ORO}}(s_0, a) \leftarrow \hat{Q}^{k-1}_{\text{ORO}}(s_0, a) + \frac{r - \hat{Q}^{k-1}_{\text{ORO}}(s_0, a)}{k}$
  $a_{\text{ORO}} \sim \mathcal{U}\left(\arg\max_{a \in \mathcal{A}(s_0)} \hat{Q}^n_{\text{ORO}}(s_0, a)\right)$
  return a_{ORO}
perform_trial$(\mathcal{M}^{\text{w}}, a_0)$**:**
  $r \leftarrow 0$
  $s = \text{succ}^{\text{w}}(s^{\text{w}}_0, a_0)$
  while s is not terminal **do**
    $r \leftarrow r + \mathcal{R}^{\text{w}}(s, \pi_{\text{OMT}}(s))$
    $s = \text{succ}^{\text{w}}(s, \pi_{\text{OMT}}(s))$
  return r

a run only in the fact that the former is simulated while the latter is executed. Optimistic rollout is similar to hindsight optimization in that it computes the action-value estimates $\hat{Q}^n_{\text{ORO}}$ by performing a sequence of n trials (where n is a parameter of the algorithm), each in a sampled weather, and by averaging over the different action-value estimates that were achieved in the trials. The difference between the two algorithms is hidden in the computation of the action-value estimates: rather than using the clairvoyant goal distance in a weather $\mathcal{M}^{\text{w}}$, optimistic rollout *simulates the execution of the optimistic policy* on $\mathcal{M}^{\text{w}}$ and uses the resulting accumulated reward of the trial to update the action-value estimate $\hat{Q}^n_{\text{ORO}}$. Optimistic rollout achieves this behavior by pretending not to know the sampled weather in its policy computation, i. e., by following the shortest path in the optimistic roadmap that is consistent with the information that has been collected in the current rollout. The sampled weather is used only to determine outcomes of simulated actions.

Example 17. *In the* CTP*, each trial of the optimistic rollout algorithm is such that the agent follows the shortest path in the optimistic graph that is in accordance with the information about the weather it has collected in the current trial. When it reaches a road which is blocked in* $\mathcal{M}^{\text{w}}$*, it recomputes the optimistic distances based on the new information and follows a new path, iterating in this fashion until it reaches the goal location. The accumulated reward that was collected in the trial is then used to update the action-value estimate of the corresponding action.*

In a trial in the weather that is depicted in Figure 4.2, optimistic rollout has to decide in its computation of the action-value estimate of the action that moves the agent to v_1 if it tries the path over v_3 or v_4, since both paths to $v_\star$ yield the same reward and appear equally good for optimistic rollout since it does not have the information that the path over v_3 is blocked. Assuming that it decides to move on to v_3, the path it takes is $v_0 - v_1 - v_3 - v_1 - v_4 - v_\star$ with a total reward of $\hat{Q}^{\mathrm{W}}(s_0^{\mathrm{W}}, \delta(\mathtt{move}_{01})) = -190$.

Clearly, optimistic rollout is (again) only one representative of a family of algorithms, as *any* policy could be used in place of the optimistic policy. In the CTP, the optimistic policy offers a good trade-off between speed and quality. However, we will see in the next section that optimistic rollout has fundamental weaknesses that prevent its convergence to the optimal policy even if the number of trials is unlimited. It is nevertheless an important step on the way to anytime optimal algorithms for finite-horizon, factored MDPs as its core idea – to perform a series of trials that simulate the execution of a policy – is also crucial for algorithms in the anytime optimal Trial-based Heuristic Tree Search framework that is presented in Chapter 5.

4.4 Theoretical Evaluation

We have presented three algorithms for MDPs and briefly demonstrated their behavior in an instance of the CANADIAN TRAVELER'S PROBLEM. In this section, we analyze their strengths and weaknesses, and determine how accurately their action-value estimates approximate the true action-values. We begin with a basic result:

Theorem 3. *As the number of weathers that are considered in hindsight optimization approaches ∞, the action-value estimates of hindsight optimization converge in probability, and as the number of trials that are performed by the optimistic rollout algorithm approaches ∞, the action-value estimates of optimistic rollout converge in probability.*

Proof: The result of each individual action-value computation in the hindsight optimization and optimistic rollout algorithms given a weather is an independent and identically-distributed bounded random variable, so the strong law of large numbers applies. (Boundedness of the obtained reward in each sample follows from the finite horizon of the problem.) ■

Convergence in probability of the action-value estimates implies convergence of the induced policies with probability one for those states that have a unique action that maximizes the reward in the limit. If the maximizing action is not unique, the policy in the limit will randomly choose one of the

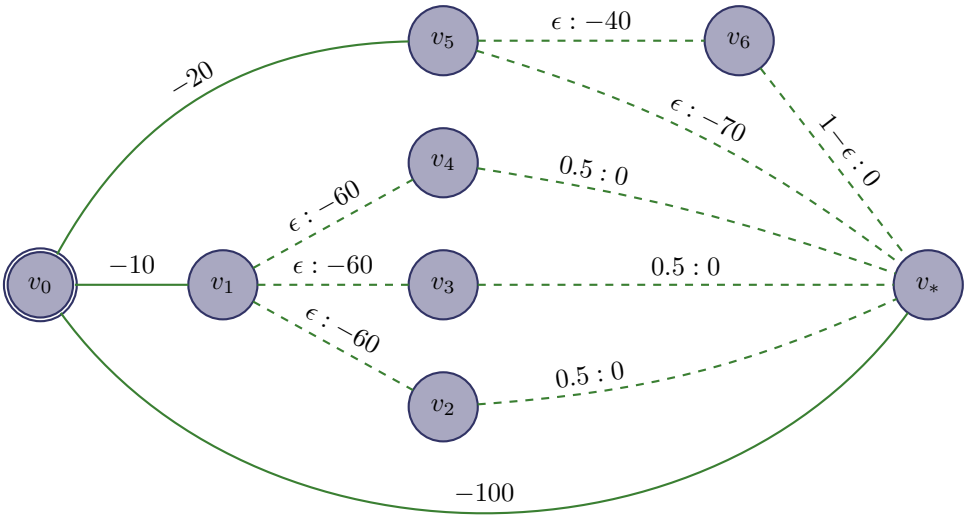

Figure 4.3: Modified CTP example instance with pitfalls for the optimistic policy, hindsight optimization, and optimistic rollout. Please observe that the instance is slightly different from the instance of Figure 3.1.

maximizers, and the resulting policy will still be optimal with respect to the Bellman optimality equation of Definition 11.

In the rest of this section, we denote the action-value estimates to which the estimates of hindsight optimization and optimistic rollout converge as the number of considered weathers or performed trials approaches infinity with $\hat{Q}^{\infty}_{\text{HOP}}$ and $\hat{Q}^{\infty}_{\text{ORO}}$, and consider the policies in the limit rather than policies based on a finite number of weathers or trials. Theorem 3 ensures that these notions are well-defined. We perform the theoretical evaluation of the proposed algorithms on the CTP, but all insights can easily be generalized to arbitrary MDPs. To motivate the ideas underlying our main result, we use a slight variation of the example instance that has been used in this and the previous chapter. It is depicted in Figure 4.3, and such that the used probabilities are slightly different (small probabilities ≤ 0.1 become ϵ, a value that is arbitrary close to zero, and the large probability on the road that connects v_6 and $v_\star$ becomes $(1-\epsilon)$, which is arbitrary close to one). This allows us to limit attention to runs where all roads with blocking probability ϵ are traversable and the road with blocking probability $(1-\epsilon)$ is blocked. It shows why the optimistic policy, hindsight optimization and optimistic rollout are approximate, non-optimal algorithms by illustrating the different pitfalls the algorithms fall prey to.

We have already hinted at the reason why the optimistic policy is suboptimal in Example 15: it always follows the cheapest path that is traversable with non-zero probability, no matter how close to zero it is. It is therefore led astray by the very unlikely path that reaches $v_\star$ via v_6, and follows the path

v_0–v_5–v_6–v_5–$v_\star$ for an accumulated reward of -170.

Hindsight optimization successfully identifies the trap the optimistic policy falls prey to, but chooses wrongly because there is a high probability of a rewarding goal path via v_1 and any of the locations v_2, v_3, or v_4, but it is not clear *which* of these three locations to enter. The action-value estimate that is assigned to the $\texttt{move}_{0\star}$ by hindsight optimization is $\hat{Q}^{\infty}_{\text{HOP}}(s_0, \texttt{move}_{0\star}) = -100$, an action-value estimate close to $\hat{Q}^{\infty}_{\text{HOP}}(s_0, \texttt{move}_{05}) \approx -90$ to $\texttt{move}_{05}$ (due to path v_0–v_5–$v_\star$), while the estimated value of $\texttt{move}_{01}$ is $\hat{Q}^{\infty}_{\text{HOP}}(s_0, \texttt{move}_{01}) \approx -73.75$ (which is $-(10 + \frac{7}{8} \cdot 60 + \frac{1}{8} \cdot 90)$). It therefore moves to v_1 first. The overly optimistic approximation of $\texttt{move}_{01}$ comes from the assumption of clairvoyance, which allows hindsight optimization to "see" which of the three paths over v_1 is actually traversable and always select the correct one. At v_1, it realizes the suboptimality of its choice and reaches the goal ultimately via path v_0–v_1–v_0–v_5–$v_\star$ with an accumulated reward of -110.

Optimistic rollout successfully detects the trap that leads hindsight optimization astray and assigns an action-value estimate to the $\texttt{move}_{01}$ action that is roughly $\hat{Q}^{\infty}_{\text{ORO}}(s_0, \texttt{move}_{01}) \approx -180$ (which is $-(\frac{1}{2} \cdot 70 + \frac{1}{4} \cdot 190 + \frac{1}{8} \cdot 310 + \frac{1}{8} \cdot 470)$. However, it is fooled by the fact that the optimistic policy acts suboptimally in v_5 (since the optimistic policy proposes $\texttt{move}_{56}$ in each trial over v_5), giving rise to a poor action-value estimate for $\texttt{move}_{05}$ (approximately -170). It therefore follows the direct path v_0–$v_\star$ with a reward of -100. The optimal policy, which follows the path v_0–v_5–$v_\star$ with an accumulated reward of -90, is hence not found by any of the approximate algorithms that are presented in this chapter. However, we revisit this experiment in Chapter 8, where we also consider algorithms that converge towards optimal behavior with increasing deliberation time.

The behavior in the example can be generalized to a result that holds for all instances in the CTP.

Theorem 4. *For all* CTP *instances* $\mathcal{M}$ *and state-action pairs* (s, a)*:*

$$Q_{\text{OMT}}(s, a) \geq \hat{Q}^{\infty}_{\text{HOP}}(s, a) \geq Q_\star(s, a) \geq \hat{Q}^{\infty}_{\text{ORO}}(s, a).$$

Moreover, there are instances where all inequalities are strict and the ratio between any two different action-value estimates is arbitrarily large.

Proof: $Q_\star(s, a) \geq \hat{Q}^{\infty}_{\text{ORO}}(s, a)$ holds because each optimistic rollout trial corresponds to an *actual* run of the CTP instance under the optimistic policy, which can (in expectation) not result in a higher reward than the true action-value $Q_\star(s, a)$. To prove $Q_{\text{OMT}}(s, a) \geq \hat{Q}^{\infty}_{\text{HOP}}(s, a) \geq Q_\star(s, a)$, let $\mathcal{M}$ be the given CTP instance, and let Π be the set of all policies in $\mathcal{M}$. We can show that for

the initial state s_0:

$$V_{\text{OMT}}(s_0) = \max_{\pi \in \Pi} \max_{\mathcal{M}^{\text{w}}} V_\pi^{\text{w}}(s_0^{\text{w}})$$
$$\hat{V}^{\infty}_{\text{HOP}}(s_0) = \mathbb{E}[\max_{\pi \in \Pi} V_\pi(s_0)]$$
$$V_\star(s_0) = \max_{\pi \in \Pi} \mathbb{E}[V_\pi(s_0)],$$

where expected values are with respect to the random choice of a weather $\mathcal{M}^{\text{w}}$. In words, the estimate of the optimistic policy is such that is assumes that the weather is always the best possible, and it always follows the policy with the highest reward (in that weather). Weathers are considered by hindsight optimization with correct probabilities, but its clairvoyance allows it to always select the policy with the highest reward in that weather. The result for the action-value estimate follows from this since $V_\star(s) := \max_{a \in \mathcal{A}(s)} Q_\star(s, a)$, and the results for s_0 follows by simple arithmetic and readily generalizes to all states. Lastly, to show arbitrary separation between Q_{OMT}, $\hat{Q}^{\infty}_{\text{HOP}}$, $Q_\star$ and $\hat{Q}^{\infty}_{\text{ORO}}$, we use augmented versions of the pitfalls for the respective algorithms as exemplified in Figure 4.3. ■

4.5 Empirical Evaluation

We have evaluated the algorithms that are discussed in this chapter on instances of the CTP. Our experiments were performed on Delaunay graphs, following the example of Bnaya et al. (2009). For each algorithm and benchmark graph, we performed 1000 runs to estimate the expected rewards of the policies with sufficient accuracy.

Main experiment. In our main experiment, we generated ten medium-sized problem instances of the CTP with 50 locations and 133–139 roads and ten large problem instances with 100 locations and 281–287 roads. (Additional results on ten small problem instances with 20 locations and 49–52 roads can be found in the work of Eyerich et al. (2010). As they do not provide further insight they are omitted here.) All problem instances were generated from random Delaunay graphs. Bnaya et al. argue that Delaunay graphs are reasonable models for small-sized road networks. All roads can potentially be blocked, with blocking probabilities chosen uniformly and independently for each road from the range $[0, 1)$. Travel costs were generated independently by assigning a reward of $[-50, 1]$ that was drawn uniformly at random. The initial location and the goal location were selected such that they are at "opposite ends" of the graph.

We evaluated all algorithms on these 30 benchmarks with $n = 10000$ weathers or trials for each applicable action for the hindsight optimization

	50 locations			100 locations		
	OMT	**HOP**	**ORO**	**OMT**	**HOP**	**ORO**
1	-255.5±10	-250.6±9	**-214.3**±7	-370.9±11	**-319.3**±9	-326.8±9
2	-467.1±11	**-375.4**±7	-406.1±8	-160.6±8	-154.5±7	**-153.2**±7
3	-281.5±9	-294.5±7	**-268.5**±7	-550.2±18	-488.1±15	**-451.3**±14
4	-289.8±9	-263.9±7	**-241.6**±7	-420.1±10	**-329.8**±7	-348.7±8
5	-285.5±10	-239.5±8	**-229.5**±7	-397.0±16	-452.4±18	**-348.1**±13
6	-251.3±10	-253.2±9	**-238.3**±9	-455.0±12	-487.9±11	**-399.9**±10
7	-242.2±9	-221.9±7	**-209.3**±7	-431.4±15	-403.9±14	**-370.1**±12
8	-355.1±11	-302.2±9	**-300.4**±8	-335.6±12	-322.0±12	**-295.7**±11
9	-327.4±13	-281.8±11	**-238.1**±9	-327.5±14	-366.1±15	**-273.8**±11
10	-281.6±8	-271.2±7	**-249.0**±6	-381.5±11	-388.4±11	**-347.1**±9
$\varnothing\,\mathcal{R}$	-303.7±3	-275.4±3	**-259.5**±3	-383.0±5	-371.3±4	**-331.5**±4
$\varnothing\,T_{\text{run}}$	**0.00 s**	6.36 s	28.02 s	**0.00 s**	20.33 s	124.03 s
$\varnothing\,T_{\text{dec}}$	**0.00 s**	0.42 s	2.39 s	**0.00 s**	0.88 s	7.71 s

Table 4.1: Average rewards with 95% confidence intervals for 1000 runs on 10 roadmaps with 50 locations (left) and 10 roadmaps with 100 locations (right) for the optimistic policy (OMT), hindsight optimization (HOP), and optimistic rollout (ORO). The three last rows show average reward ($\varnothing\,\mathcal{R}$), average runtime per run ($\varnothing\,T_{\text{run}}$), and average runtime per decision ($\varnothing\,T_{\text{dec}}$).

and optimistic rollout algorithms. Tables 4.1 shows the results of the experiment. Optimistic rollout clearly dominates hindsight optimization and the optimistic policy, always providing the best policy except for three instances where hindsight optimization performs better. Both algorithms outperform the optimistic policy (which is still among the state-of-the-art in many applications) significantly, clearly demonstrating the benefit of taking uncertainty into account. While our theoretical results already hint at the fact that hindsight optimization outperforms the optimistic policy, the relationship between hindsight optimization and optimistic rollout has not been answered so far. The experimental evaluation clearly shows that the bias that is incurred by the under-approximation of the action-values of optimistic rollout is not as severe as the over-approximation of hindsight optimization in most instances.

While we want to emphasize policy quality and not runtime, Table 4.1 also provides some runtime results. They show that in our implementation, the runtime of solving the all-outcomes determinization that is needed in the optimistic policy is barely measurable even though we use Dijkstra's algorithm for its computation and not a faster variant for shortest-path computation like D* LITE (Koenig and Likhachev, 2002). Of course, the optimistic rollout strategy also benefits from the fast computation since it invokes the method that computes the optimistic policy as a subroutine. However, optimistic rollout

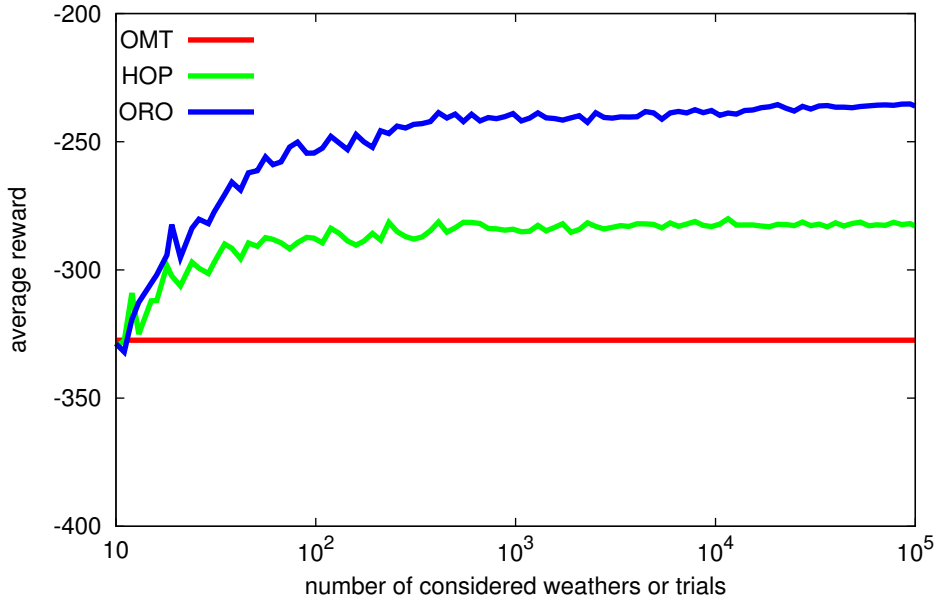

Figure 4.4: Average reward as a function of considered weathers or trials for benchmark instance 50-9.

nevertheless takes about six times longer on graphs of medium size and about eight times in the large instances.

Trials and Scalability. To analyze the speed of convergence and scalability of hindsight optimization and optimistic rollout, we performed additional experiments on individual benchmarks where we varied the number of trials or considered weathers in the range 10–100000. Figure 4.4 shows the outcome for (the prototypical) benchmark graph 50-9. We see that both hindsight optimization and optimistic rollout improve over the optimistic policy more or less directly from the beginning where the trials require very little computation time even for optimistic rollout. Hindsight optimization begins to level off after about 500 considered weathers, while optimistic rollout continues to improve until it uses approximately 10000 trials (even though it is barely visible in the provided figure, there is still a significant improvement between 1000 and 10000 weathers or trials).

Remote Sensing. In our main experiment, we did not compare to other algorithms than those discussed in this chapter because there is practically no work in the literature that describes different policies for the CTP with a focus on policy quality. The major exception to this is the paper by Bnaya et al. (2009), which introduces and compares four algorithms for a slightly different problem, the CTP with remote sensing. In this CTP variant, agents

	OMT	HOP	ORO	EXP	VOI
$p=0.1$ $\varnothing C$	-155.2±0.3	-156.8±0.3	-155.0±0.3	**-154.9**±0.3	**-154.9**±0.3
$\varnothing T_{\text{run}}$	0.00 s	5.12 s	12.31 s	0.05 s	15.88 s
$\varnothing T_{\text{dec}}$	0.00 s	0.54 s	1.38 s	0.00 s	0.86 s
$p=0.3$ $\varnothing C$	-216.9±1.0	-234.5±1.1	**-212.3**±0.9	-219.2±0.9	-218.9±0.9
$\varnothing T_{\text{run}}$	0.00 s	5.03 s	13.66 s	0.06 s	26.77 s
$\varnothing T_{\text{dec}}$	0.00 s	0.35 s	1.22 s	0.00 s	1.08 s
$p=0.5$ $\varnothing C$	-298.9±1.5	-302.6±1.5	**-281.7**±1.4	-304.3±1.4	-309.3±1.5
$\varnothing T_{\text{run}}$	0.00 s	3.23 s	6.25 s	0.07 s	38.60 s
$\varnothing T_{\text{dec}}$	0.00 s	0.19 s	0.47 s	0.00 s	1.21 s
$p=0.6$ $\varnothing C$	-302.4±1.5	-293.8±1.4	**-286.0**±1.4	-311.4±1.4	-316.4±1.5
$\varnothing T_{\text{run}}$	0.00 s	2.55 s	4.11 s	0.07 s	39.19 s
$\varnothing T_{\text{dec}}$	0.00 s	0.17 s	0.30 s	0.00 s	1.24 s

Table 4.2: Results for the CTP with sensing benchmarks of Bnaya et al. (2009) with fixed sensing cost of 5. The notation is as in Table 4.1. Our algorithms (left half) do not make use of the sensing capabilities, while the others do. Each block summarizes the results for 30 roadmaps where all roads are blocked with the same probability p.

may sense the status of roads from a distance, at a cost, and the objective is to minimize the total of travel cost and sensing cost. The policies *Exp* and *VOI* are the two best performing algorithms that are suggested by Bnaya et al.. Both implement different strategies for sensing decisions, but use the optimistic policy for movement decisions.

In our last experiment, we compare the optimistic policy, hindsight optimization, and optimistic rollout approaches to the two best performing algorithms by Bnaya et al. on the CTP with remote sensing. For our approaches, we treat the CTP with remote sensing instances as regular CTP instances. Thus, we compare policies that attempt to make use of sensing capabilities intelligently to ones that *never perform remote sensing*. We evaluate on 120 problem instances, a subset of the 200 instances used in the original experiments by Bnaya et al.. (The remaining 80 instances could unfortunately not be made available by the authors.) All instances have 50 locations and were created from Delaunay graphs. Different from our main experiments, in these roadmaps all roads have the same blocking probability p, with 30 instances each for $p = 0.1$, $p = 0.3$, $p = 0.5$, and $p = 0.6$. Similar to our stochastic algorithms, the VOI algorithm introduced by Bnaya et al. has a sampling size parameter, which we set to 50 to keep the runtime of the approach reasonable. (It should be noted, though, that the implementation of Bnaya et al. is not optimized with respect to runtime, and no conclusions should be drawn from the comparatively large runtime of VOI.) The cost for sensing actions

was set to 5 in these experiments, one of three settings considered in the work of Bnaya et al.

The experimental results in Table 4.2 show that our policies are competitive with the best policies of Bnaya et al. and outperform them on the instances with larger blocking probabilities even though they never perform any sensing. In fact, in this experiment the algorithms that make use of sensing capabilities do not improve over the optimistic policy on average, which differs from the results that were reported by Bnaya et al. We do not know why this is the case and cannot rule out a bug. If there is no bug, our results are likely more accurate since they consider a much larger number of runs per algorithm (120000 instead of 200).

Apparently, the idea to simulate a policy in a random weather leads to the best results over all experiments. On the other side, a practical consequence of Theorem 4 is that there are MDPs on which the hindsight optimization and optimistic rollout policies behave suboptimally no matter how many computing resources are available, i. e., they have fundamental limitations rather than just practical limitations caused by limited sampling. The *Trial-based Heuristic Tree Search* framework that is discussed in the following chapters picks up the idea of simulating a policy repeatedly, but it improves the simulated policy while doing so and thereby allows for algorithms that do not share these fundamental weaknesses.

CHAPTER 5

Trial-based Heuristic Tree Search

So far, we have discussed algorithms for finite-horizon, factored MDPs that either need the whole state space in memory to even get started (Chapter 2) or have fundamental limitations that prevent optimal behavior even with unlimited resources (Chapter 4). We now turn our attention to algorithms that are optimal in the limit, yet also operate under tight time and memory constraints. The *Trial-based Heuristic Tree Search* (THTS) framework that is presented in Section 5.2 is an algorithmic schema that allows the specification of algorithms that compute reasonable policies with limited deliberation time and converge towards optimal decisions if the right mix of *ingredients* is used. We lay the ground for THTS by discussing four related optimization problems, their solutions, and popular algorithms in Section 5.1. Many of the discussed algorithms find their way into the framework, either as an ingredient or because there is an equivalent THTS version.

5.1 Preliminaries

There is not only numerous related work in planning and acting under uncertainty, there is a wide variety of different optimization problems in the research area as well. Even though their theoretical properties differ significantly, most are more or less closely related, and comparable methods are used to find optimal or approximate solutions. We discuss four different optimization problems in this part that influence the algorithms in the Trial-based Heuristic Tree Search framework that is presented in the subsequent section. The optimization problems differ only in the *a-priori model of the environment* that is revealed to the agent before its interaction with the environment starts. In all scenarios, the agent can fully observe its current state (including the information if it is a terminal state) and it knows in every state which options are available (i. e., it can generate the set of applicable actions $\mathcal{A}(s)$ for all $s \in \mathcal{S}$). Given an MDP $\mathcal{M}$, we say that an agent

- has *no a-priori model* of $\mathcal{M}$ iff it is not provided with additional information;

- can access a *generative model* of $\mathcal{M}$ iff it is provided with a randomized algorithm that, on input of a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, outputs $\mathcal{R}(s,a)$ and a state $s' \in \mathcal{S}$ which is randomly drawn according to the transition function $s' \sim \text{succ}(s,a)$; and

- can access the *declarative model* of $\mathcal{M}$ iff it is provided with an explicit description of $\mathcal{M}$, which includes, most importantly, access to the transition probabilities $\mathbb{P}_{\mathcal{T}}[s' \mid s,a]$ and the reward formula $\mathcal{R}(s,a)$ for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$.

Our notion of planning with a generative model follows the definition of Kearns et al. (2002). It is important that all three models determine the *a-priori* view of the agent on the environment, i. e., they describe the available knowledge *before* the interaction with the environment starts. The terms *model-based* and *model-free* algorithm that are often used in RL are not related to the a-priori model. Instead, they specify whether an agent learns a model of the MDP by interacting with the environment (model-based), or if it doesn't build a model but derives a policy directly (model-free). We have already touched the topic of different a-priori models of agents at the beginning of Chapter 2, where we distinguished *learning agents* (which are not provided with an a-priori model) from *planning agents* (with access to a generative or declarative model), and we continue to use the terms in this sense.

5.1.1 Multi-Armed Bandit

We start our discussion with the smallest possible MDP, the (discrete) Multi-Armed Bandit (MAB) problem (Robbins, 1952). Figuratively, the agent faces a slot machine (a bandit) with multiple arms, and it has to decide which arm to pull based only on the payouts it received so far. Each arm provides a random reward according to a probability distribution that is specific to the arm and constant over time, but the agent has no initial information on the distribution. This picturesque description does, of course, not restrict the potential applications of MAB to a single gambling scenario. All applications where actions that were taken in the past have no effect on the (transition and reward function of the) action that is taken now can be modeled as an MAB, including clinical trials where the best treatment for a disease is to be found (Robbins, 1952), scheduling of jobs with different payouts (Gittins, 1979), online advertising with the aim to figure out what ads a typical user is most likely to click (e.g., Chakrabarti et al., 2008; Chen et al., 2013), or ranking of results of a web-based search engine (Radlinski et al., 2008) to give just a few examples of an ever-growing list of applications.

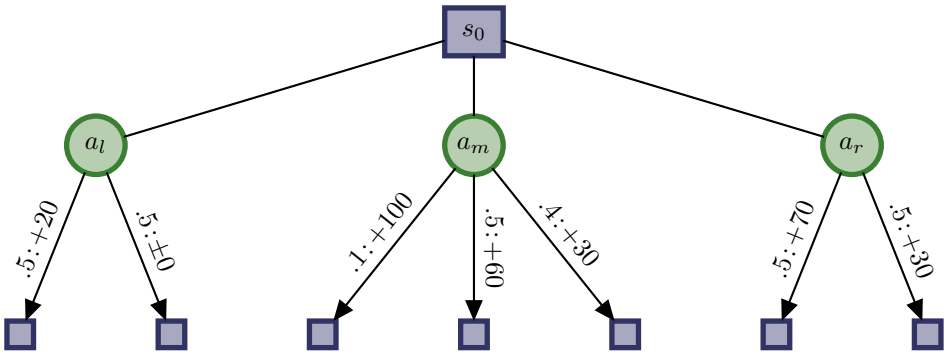

Figure 5.1: An example MAB problem with three arms. Edges are labeled with transition probabilities p and rewards r as $p{:}r$.

A discrete MAB can be modeled as an MDP where the only non-terminal state is the initial state s_0. Without loss of generality, each action $a \in \mathcal{A}$ is applicable in the initial state (i. e., $\mathcal{A}(s_0) = \mathcal{A}$) and corresponds to pulling one of the arms of an $|\mathcal{A}|$-armed gambling machine. In the context of MABs, we omit the state in all notations as s_0 is the only relevant state anyway. In contrast to the rest of this thesis, we consider a *stochastic* reward function $\mathcal{R}(a)$ that defines a discrete probability distribution for MABs. However, such an MAB can easily be compiled to an MDP with deterministic reward function that is in line with our definition of MDPs. We denote the arm that is pulled in the k-th run ϕ^k with a^k, the number of times action a was selected among the first k runs with $\mathcal{L}^k(a)$, and the expected reward of the optimal arm $a^\star$ with $Q_\star$. The action-value estimate of arm a after k pulls is $\hat{Q}^k(a)$, and it corresponds to the average result that has been obtained in runs where a has been selected.

The main challenge of the MAB lies in the fact that it is an optimization problem where the agent has no access to an a-priori model of the task. The central challenge of MABs is the famous *exploration-exploitation dilemma*: on the one hand, since the agent has no a-priori knowledge of the environment, it has to *explore* its possibilities to learn from the feedback the environment provides. And on the other, since it aims to maximize the accumulated reward over all runs, it has an incentive to *exploit* the knowledge it has gathered by executing the action it believes to be best. In other words, the agent learns from its trial-and-error interaction with the environment, and it has to make sure that the balance between exploration and exploitation is such that it learns the best action without sacrificing too much reward in early decisions.

Example 18. *An example MAB problem with three arms a_l, a_m, and a_r (for left, middle and right arm) is depicted in Figure 5.1. Pulling the left (the right) arm yields a reward of 20 or 0 (70 or 30) with equal probabilities, and playing the middle arm gives a payout of 100 with probability $.1$, of 60 with probability*

.5 and of 30 *otherwise. It is possible to compute the expected rewards of the three arms as* $Q_\star(a_l) = 10$*,* $Q_\star(a_m) = 52$ *and* $Q_\star(a_r) = 50$ *with access to the declarative model of the MAB. The choice with the highest expected reward, arm* a_m*, is hence slightly better than playing* a_r*, while the left arm is a clearly inferior choice.*

However, even though the computation of the expected values is trivial if the model can be accessed*, the MAB is a challenging problem since the agent has no a-priori knowledge on the distributions that define the payout. It therefore has to explore the arms of the machine to obtain a picture what the distributions (or the expected rewards) look like, and exploit its knowledge if it is sufficiently certain about its beliefs. If it gives up exploration too early, it might never find the best arm – consider, for instance, the not unlikely case where the middle option does not yield the reward of* 100 *in the first few runs. Its action-value estimate* $\hat{Q}^k(a_m)$ *can be expected to be below* 50 *in that case, and it is likely that* a_r *looks more promising. Therefore, if the agent commits to its beliefs too early, it will continue with selecting the right arm even though* a_m *is superior. On the other hand, if it continues to explore for too long, it pulls the suboptimal arms* a_l *and* a_r *too often.*

An important implication of the exploration-exploitation dilemma is that there is no algorithm that can guarantee that it selects the arm with the highest expected payout in all runs starting from the very first, and it is hence also impossible for a learning agent to achieve an expected reward that equals the action-value of the best arm (that is, unless the choice is irrelevant as all arms are optimal). However, given a sufficient number of runs, it is simple to come up with an algorithm that determines the best arm with high probability. It has become popular as *A/B testing* in the context of marketing with two choices. Here, it is called the *Explore-then-Exploit* (ETE) strategy. The main idea is to split up the agent-environment interaction in two consecutive phases, starting with a pure exploration phase of length l with the aim to determine the best arm with high probability, which is followed by a phase that purely exploits the insights of the first phase. Even though there are different possibilities how to design the exploration phase, we go with the simplest one where Explore-then-Exploit applies the *uniform action selection* (UNI) strategy where an arm is selected *uniformly at random*:

$$a^k_{\text{UNI}} \sim \mathcal{U}(\mathcal{A})$$

Note that ETE is often described with *round-robin* (RR) action selection

$$a^k_{\text{RR}} \sim \mathcal{U}\left(\{a \in \mathcal{A} \mid \mathcal{L}^{k-1}(a) \leq \mathcal{L}^{k-1}(a') \text{ for all } a' \in \mathcal{A}\}\right)$$

in place of UNI, which does not make a difference for this thesis since the behavior in the limit is identical. After the exploration phase has finished,

Explore-then-Exploit algorithms commit to executing a *greedy action*

$$a^k_{\text{GRD}} \sim \mathcal{U}\left(\arg\max_{a\in\mathcal{A}} \hat{Q}^{k-1}(a)\right)$$

in run ϕ^k for $k > l$. However, a good value for the length of the exploration phase can only be found when the number of runs is known in advance. Otherwise, it is impossible to balance the conflicting aims of finding the best arm with high probability and exploiting the insights sufficiently long.

Let us therefore turn our attention to the question *how close* an algorithm can get to an average reward of $Q_\star$ in expectation. Rather than answering this with a bound on the average reward that can be achieved, the analysis is usually performed in terms of an algorithm's *cumulative regret* over a sequence of runs. Berry and Fristedt (1985) define it for a policy π as the difference between the reward π yields in a sequence of runs $(\phi_1, \ldots, \phi_n)$ and the expected accumulated reward of a run under $\pi^\star$:

$$L^n_\pi = \sum_{k=1}^{n} Q_\star - \mathcal{R}(\phi^k_\pi)$$

Lai and Robbins (1985) were the first to show that the expected cumulative regret has to grow at least logarithmically in the number of runs, i. e.,

$$\mathbb{E}\left[L^n_\pi\right] \geq \alpha \cdot \ln n$$

for all $n > 0$, all policies $\pi \in \Pi$, and with a constant $\alpha \geq 1$. (To know that such a constant exists is sufficient for our needs, even though Lai and Robbins provide a more accurate bound where α is given in terms of the *Kullback-Leibler divergence* between the suboptimal arms and the optimal one.) discounted, infinite They also present policies for discrete MABs that are such that the incentive to explore an action grows with the uncertainty of the action's quality. This drives the agent to try out arms that have not been observed often until the model of the system is suitably accurate eventually. The main idea is to update an *upper confidence bound* $\hat{U}^k(a)$ for each action a such that $Q_\star(a) \leq \hat{Q}^k(a) + \hat{U}^k(a)$ with high probability. The action

$$a^k \sim \mathcal{U}\left(\arg\max_{a\in\mathcal{A}} \left(\hat{Q}^{k-1}(a) + \hat{U}^{k-1}(a)\right)\right)$$

is then selected for execution in ϕ^k. Lai and Robbins show that their policies achieve the lower bound as the optimal action is executed exponentially more often than all other actions.

While the policies of Lai and Robbins require the underlying probability distribution of the MAB to meet certain requirements and guarantee achievement of the logarithmic cumulative regret only in the limit, improvements towards general methods were made (e.g., Kaelbling, 1993; Agrawal, 1995)

until Auer et al. (2002) presented an algorithm, *UCB1* (for Upper Confidence Bounds), that achieves the cumulative regret bound uniformly over the runs for any bounded probability distribution. The confidence bounds are computed by applying *Hoeffding's inequality*, which states (in the context of MABs) that the probability that the sampled reward of an action differs from its expected reward by more than a constant is reduced exponentially in the number of runs. By application of Hoeffding's inequality to the probability distribution that is defined by the rewards of an MAB, the famous UCB1 formula is derived. It computes the upper confidence bound with $\hat{U}^k_{\text{UCB1}}(a) = \sqrt{\frac{2 \cdot \ln(k-1)}{\mathcal{L}^{k-1}(a)}}$, and hence selects an action according to

$$a^k_{\text{UCB1}} \sim \mathcal{U}\left(\arg\max_{a \in \mathcal{A}} \left(\hat{Q}^{k-1}(a) + \sqrt{\frac{2 \cdot \ln(k-1)}{\mathcal{L}^{k-1}(a)}}\right)\right)$$

in the k-th run. The UCB1 policy is an important part of the work presented in this thesis, and we revisit the results of Auer et al. later.

Two classical algorithms for action selection in MABs, *ϵ-greedy* (ϵ-G) and *Boltzmann Exploration* (BE), only achieve asymptotically cumulative regret that is linear in the number of runs. However, a surprising study of Kuleshov and Precup (2010) reveals that both outperform theoretically superior algorithms like UCB1 experimentally. In BE, the probability of selecting action $a \in \mathcal{A}$ in the k-th run is proportional to $\mathcal{Q}^k(a) := e^{\frac{\hat{Q}^{k-1}(a)}{\tau}}$, where $\tau > 0$ is the *temperature*, a parameter that specifies the degree of exploration. Each action is selected with probability

$$\mathbb{P}[a^k_{\text{BE}} = a] = \frac{\mathcal{Q}^k(a)}{\sum_{a' \in \mathcal{A}} \mathcal{Q}^k(a')}$$

in an action selection that is based on Boltzmann Exploration, while ϵ-G is such that, given a parameter ϵ with $0 < \epsilon < 1$, the greedy action is selected with probability $(1-\epsilon)$, and the uniform action selection is applied otherwise:

$$a^k_{\epsilon\text{-G}} = \begin{cases} a^k_{\text{UNI}} & \text{with probability } \epsilon \\ a^k_{\text{GRD}} & \text{otherwise.} \end{cases}$$

Cesa-Bianchi and Fischer (1998) present versions of both policies where a poly-logarithmic cumulative regret is achieved with a schedule for decreasing τ- and ϵ-values, and Auer et al. (2002) enhance on this work and present a method to decrease ϵ that achieves logarithmic cumulative regret – though only under the assumption that the difference between the expected reward of the best and second best action is known. We pick up the idea in Chapter 6 and discuss several different schemata to decrease the respective parameters.

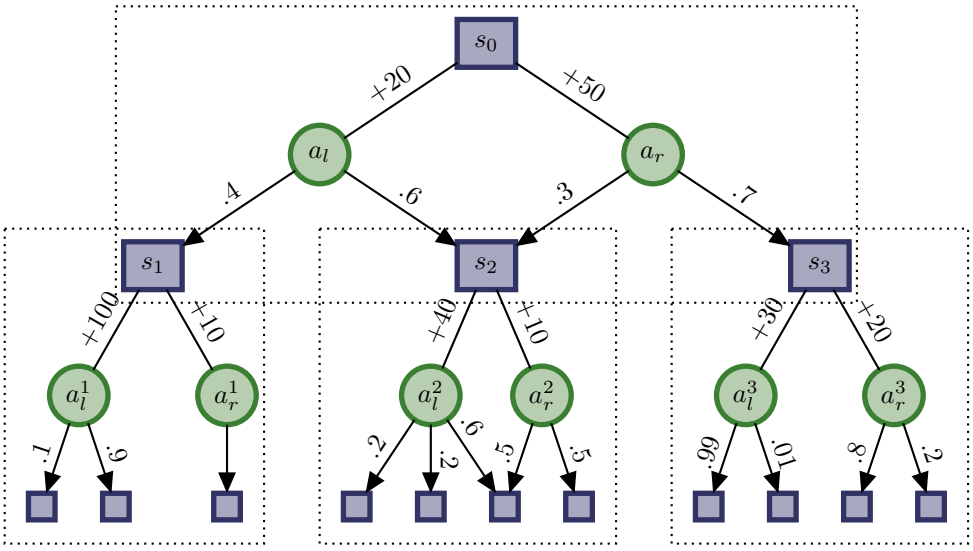

Figure 5.2: An MDP with horizon $\mathcal{H} = 2$ depicted as four nested MABs (one per area with dotted border).

5.1.2 Online Reinforcement Learning

Online Reinforcement Learning is the term that is commonly used for the generalization of MABs to larger state spaces (e.g., Sutton and Barto, 1998), i. e., the optimization problem of solving MDPs without access to an a-priori model. However, Online RL comprises a large variety of similar scenarios that are very different when examined theoretically. For instance, if there is no finite horizon and no option to restart from the initial state, reachability assumptions complicate the performance analysis as it becomes disproportionately important if a poor initial decision (or bad luck) obstructs a good result (e.g., Burnetas and Katehakis, 1997; Tewari and Bartlett, 2008; Jaksch et al., 2010). We restrict our discussion to *episodic* settings here where the agent is provided with the possibility to set its current state to the initial state at any point during the execution of a run. The option to stop a run $\phi = (s_0, a_0, \ldots, a_{l-1}, s_l)$ after l steps at will and start another run that starts in s_0 will become the foundation of the *trial length component* of the THTS framework in the next section.

Many algorithms for Online RL approach the problem by regarding the provided MDP as a structure of nested MABs. A finite-horizon example of this view is depicted in Figure 5.2. Each of the subgraphs in one of the dashed boxes can be regarded as an MAB problem that is rooted in s_0, s_1, s_2, or s_3 and with a pair of applicable actions with stochastic successor states.

Example 19. *Figure 5.2 depicts a small example MDP with emphasis on the way its structure can be regarded as four nested MABs (each one in an area with*

dotted border). The optimal policy $\pi^\star$ is such that $\pi^\star(s_i) = a_l^i$ for $i = 1, \ldots, 4$: in the states s_1, s_2, and s_3, the immediate reward of all left actions is apparently higher than the immediate reward of the corresponding actions to the right. And in the initial state s_0, we can compute the action-values (e. g., with the acyclic value computation algorithm that was presented in Chapter 2) as $Q_\star(s_0, a_l) = 84$ and $Q_\star(s_0, a_r) = 83$ and therefore $Q_\star(s_0, a_l) > Q_\star(s_0, a_r)$.

In the previous part on MABs, all action-value estimates $\hat{Q}^k(a)$ were computed as the *arithmetic mean* over all results that were obtained in runs where a was selected. This *backup function* is known as the *Monte-Carlo* (MC) backup function. Formally, it is such that the action-value estimate of a state-action pair (s, a) that has been encountered in the k-th run $\phi^k = (s_0, a_0, \ldots, a_{l-1}, s_l)$ is updated by applying the recursive formula

$$\hat{Q}^k_{\text{MC}}(s_t, a_t) = \hat{Q}^{k-1}_{\text{MC}}(a) + \frac{1}{\mathcal{L}^k(s_t, a_t)} \cdot \left(\mathcal{R}_{t..l}(\phi^k) - \hat{Q}^{k-1}_{\text{MC}}(a) \right),$$

where $\mathcal{R}_{t..l}(\phi^k) := \sum_{i=t}^{l-1} \mathcal{R}(s_i, a_i)$ denotes the accumulated reward that was achieved in the run starting from state s_t. Applying the MC backup function in an MAB is reasonable as each action-value estimate converges towards its true action-value (as long as each arm is selected sufficiently often) *independently from all other decisions*.

However, it is not necessarily the case that the computation of the arithmetic mean is reasonable in Online RL: consider the MDP from Figure 5.2 and assume that runs are performed by applying the uniform action selection. If $\hat{Q}^k_{\text{MC}}(s_0, a_l)$ and $\hat{Q}^k_{\text{MC}}(s_0, a_r)$ are updated by applying the MC backup function, the estimate of the left action will not converge to 84 but to $\hat{Q}^\infty_{\text{MC}}(s_0, a_l) = 57$ with $k \to \infty$ as a_l^1 and a_r^1 on the one hand and a_l^2 and a_r^2 on the other are executed with equal probability in the runs where s_1 and s_2 are encountered due to the UNI action selection. Even worse, the action value estimate of applying a_r in s_0 converges to $\hat{Q}^\infty_{\text{MC}}(s_0, a_r) = 75$, which combines to the wrong picture that a_r is the better action in the initial state. Note that this is not restricted to the UNI action selection, but holds for several popular action selection strategies including ϵ-greedy and Boltzmann Exploration.

One possibility to overcome this problem is to *flatten* the MDP into an equivalent MAB. The flattened version of the MDP from Figure 5.2 is depicted in Figure 5.3. The indices of the actions are such that the first letter of the index corresponds to the index of the action in s_0, then the index of the action that is taken in the outcome that is depicted to the left in Figure 5.2 and finally the index of the action that is applied in the outcome to the right. In a flattened MDP, all action selection strategies for MABs can be applied such that all action-value estimates converge towards the true action-values – obviously, as the MDP in Figure 5.3 is an MAB. However, the creation of a flattened MDP is practically intractable in all but prohibitively small MDPs, as it requires one arm for each policy. Nevertheless, it illustrates nicely that it is

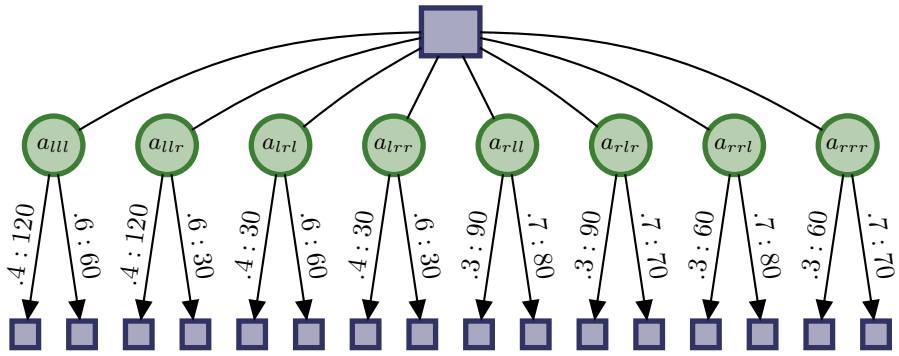

Figure 5.3: An MAB that is equivalent to the MDP from Figure 5.2.

possible to use the MC backup function if the action selection is such that a stationary policy is executed in each step, which is the case for many action selection strategies in the limit (we use this fact in our theoretical analysis of THTS algorithms in Chapter 7). Moreover (and following the same logic), if that policy is the optimal policy $\pi^\star$, $\hat{Q}^k_{\text{MC}}(s,a)$ even converges to $Q_\star(s,a)$ with an increasing number of runs.

There are also other possibilities to combine action selection strategies and backup functions in MDPs such that the action-values estimates converge towards the true action values. Singh et al. (2000) discuss the convergence of combinations of backup functions and action selections, a work we build upon with our analysis of anytime optimal THTS recipes in Chapter 7. A popular alternative to MC backups is the backup function that is used in *Temporal Difference Learning* (TD) (Sutton and Barto, 1987; Sutton, 1988). Given a run $\phi^k = (s_0, a_0, \ldots, a_{l-1}, s_l)$,[1] it updates all state-value estimates of states s_t that were visited in ϕ^k such that

$$\hat{V}^k_{\text{TD}}(s_t) = \hat{V}^{k-1}_{\text{TD}}(s_t) + \frac{1}{\mathcal{L}^k(s_t)} \cdot \Big(\mathcal{R}(s_{t+1}, a_{t+1}) + \hat{V}^{k-1}_{\text{TD}}(s_{t+1}) - \hat{V}^{k-1}_{\text{TD}}(s_t) \Big)$$

An equivalent notion is used in the famous SARSA algorithm (Rummery and Niranjan, 1994; Sutton, 1996) except that action-value estimates are updated instead by applying

$$\hat{Q}^k_{\text{TD}}(s_t, a_t) = \hat{Q}^{k-1}_{\text{TD}}(s_t, a_t) + \frac{1}{\mathcal{L}^k(s_t, a_t)} \cdot \Big(\mathcal{R}(s_t, a_t) + \hat{Q}^{k-1}_{\text{TD}}(s_{t+1}, a_{t+1}) - \hat{Q}^{k-1}_{\text{TD}}(s_t, a_t) \Big)$$

to all state-action pairs (s_t, a_t) that were visited in ϕ^k. In this form, it can be seen that TD backups differ from MC backups only in the fact that the *action-value estimate* $\hat{Q}^{k-1}_{\text{TD}}(s_{t+1}, a_{t+1})$ of the state-action pair that follows the current state-action pair (s_t, a_t) (and the latest immediate reward) is used rather than

[1] Actually, TD and (the latter introduced) QL backups are such that it is not necessary that a run is completed before a backup function is applied, which is of no relevance for our work.

the accumulated reward of the whole trajectory. The application that made TD Learning popular beyond the RL community is the Backgammon player TD-Gammon (Tesauro, 1995), which was one of the first computer programs to achieve a level of play in a game with chance that is competitive with the best human players.

The backup function that is considered in *Q-Learning* (Watkins, 1989) is closely related to the one that is used in TD. There, the action-value estimates of all visited state-action pairs are updated by

$$\hat{Q}^k_{\mathrm{QL}}(s_t, a_t) = \hat{Q}^{k-1}_{\mathrm{QL}}(s_t, a_t) + \frac{1}{\mathcal{L}^k(s_t, a_t)} \left(\mathcal{R}(s_t, a_t) + \hat{V}^{k-1}_{\mathrm{QL}}(s_{t+1}) - \hat{Q}^{k-1}_{\mathrm{QL}}(s_t, a_t) \right),$$

where $\hat{V}^k_{\mathrm{QL}}(s) = \max_{a \in \mathcal{A}(s)} \hat{Q}^k_{\mathrm{QL}}(s, a)$ is the action-value estimate of the best action in state s after k trials. MC and TD backups are typically described as *on-policy* backup functions as the updates are based on rewards that are achieved with the simulated policy. Q-Learning (QL), on the other hand, is an *off-policy* backup function where one policy is simulated and another one (namely the greedy one) is used to update encountered action-value estimates.

Before we proceed to the next optimization problem, we would like to discuss how the altered scenario influences the expected cumulative regret of the aforementioned action selection strategies. As some combinations of backup function and action selection do not even converge to optimal behavior, it can be expected that the regret bounds are different, which is indeed the case. The most important reason is that the selection to explore or exploit in the MAB implies that the whole run is explorative or exploitative. In MDPs, however, it is possible that an exploitative selection in an early decision is necessary to prepare an explorative step in the desired region of the MDP. Moreover, a couple of poor results with the same action in the initial state do not hint at the fact that the selection is indeed bad, as it is possible that the decisions following the first selection have been suboptimal. If an algorithm is too greedy, it is therefore possible that it takes prohibitively long for some states until they are encountered for the first time, and the regret bounds can therefore be entirely different. The most famous example where this is the case is probably the UCT algorithm (Kocsis and Szepesvári, 2006), where the UCB1 formula of Auer et al. (2002) is used for action selection in an MDP that is regarded as a tree (UCT is short for Upper Confidence Bounds applied to Trees). Even though UCT is an algorithm that has been applied successfully in many applications, Coquelin and Munos (2007) show that its worst case cumulative regret is not logarithmic as in the MAB, but

$$\mathbb{E}\left[L^n_{\mathrm{UCT}}\right] \in \Omega\Big(\underbrace{\exp(\exp(\ldots(\exp(n))))}_{\mathcal{H}-1} \Big)$$

in an Online RL optimization problem of an MDP with horizon $\mathcal{H}$. The reason for the poor behavior lies in the fact that UCB1 is too greedy, and it takes

hence too long until the right actions in states further from the initial state are even considered for the first time.

5.1.3 Planning with a Generative Model

A major problem of algorithms for Online RL is that it is not possible to design them in a way that reasonable behavior can be achieved without trying every single state-action pair several times simply because the agent has to *learn* from its interaction with an uncertain environment. If we want to find algorithms that achieve near-optimal behavior without executing each action in each state several times, we have to provide it with a model of the MDP. An optimization problem where the agent has a vague idea of its environment, which is given in form of a generative model of the MDP, is often called *Offline RL* in the RL community (e.g., Sutton and Barto, 1998). Since learning from the interaction with the environment can be replaced by planning on the provided model, we prefer to call this kind of optimization problem *planning with a generative model* (GMPlan) instead.

We start our brief survey on GMPlan algorithms by having a look at how the techniques we have seen so far can be adapted to the altered scenario. A first observation is that we can simply *ignore* the provided model and apply any Online RL algorithm in a GMPlan scenario. This is not very useful if we aim for improved performance, though. However, the generative model allows another, simple alternative how Online RL algorithms can be adapted to the GMPlan scenario: instead of learning by interaction with the environment (all the while increasing the cumulative regret that is inherent to learning), it is possible to *simulate trials* of the algorithm with the help of the model in an initial *planning phase* and execute the action that is determined most promising by a *recommendation function*. We will meet both trials and recommendation functions again when we introduce the THTS framework in the following section.

It is evident that theoretical performance bounds of Online RL algorithms can only improve when applied in GMPlan in this way. Even though the exploration-exploitation dilemma is still a central challenge, it is a problem of efficient simulation rather than a problem that incurs a theoretical upper bound on optimal behavior that is worse than applying the optimal policy. This is because the provided model allows the agent to perform exploration in the planning phase *at no cost* and *without accumulating regret*, while it executes only actions that are considered best (and therefore exploitative) by a *recommendation function*. The regret that is incurred in a planning scenario and that measures only the regret that is incurred by the executed actions is typically referred to as the *simple regret*. To emphasize the difference, let us compare the cumulative regret of the Explore-then-Exploit algorithm for MABs that was introduced earlier with the simple regret of the same algorithm in GMPlan. For simplicity, let us assume that the number of runs n is

known in advance. Busa-Fekete and Hüllermeier (2014) suggest to set the number of explorative runs to a value $l^\star$ such that the best arm is known after the exploration phase has finished with probability of at least $(1-\frac{1}{n})$, and show that the cumulative regret (assuming that rewards are in $[0,1]$) of this version of Explore-then-Exploit in Online RL is

$$\mathbb{E}\left[L^n_{\text{ETE}}\right] = \underbrace{\mathbb{E}\left[L^{l^\star}_{\text{UNI}}\right]}_{\leq l^\star \cdot 1} + \underbrace{\mathbb{E}\left[L^{n-l^\star}_{\text{GRD}}\right]}_{\leq (n-l^\star)\cdot \frac{1}{n}\cdot 1} = O\left(l^\star+1\right).$$

The *cumulative regret* is hence dominated by the cumulative regret of the action selection strategy that is applied in the exploration phase. However, if the exploration phase is *simulated* instead of executed, as it is possible in the GMPlan scenario, it is only the expected constant *simple regret* of $\mathbb{E}\left[L^n_{\text{ETE}}\right] = O(1)$ that remains.

Unlike in the learning scenarios and given enough *deliberation time*, it is therefore possible to execute the optimal policy $\pi^\star$ in the GMPlan scenario right from the start with probability one, and it is hence also possible to almost surely achieve no simple regret at all. There is even a connection between the cumulative and the simple regret of action selection strategies for the MAB. Bubeck et al. (2009) show that the simple regret is larger the smaller the cumulative regret. Even though this seems surprising at first glance, it is actually intuitive: the more we focus on exploitation during simulation, the more uncertainty is left about the actions we believe to be suboptimal, and the higher is the probability that our belief about the optimal action is wrong. In particular, Bubeck et al. show that the simple regret of the UCB1 formula decreases at best polynomially in the number of runs, i. e.,

$$L^k_{\text{UCB1}} \in O(\frac{1}{k^\alpha})$$

for some constant α, while the simple regret of the UNI strategy achieves an exponential-rate decrease

$$L^k_{\text{UNI}} \in O(\frac{1}{e^k}).$$

Please observe that the simple regret bounds are smaller than in Online RL for both cases, and that both approach zero with an increasing number of trials.

Even though both our Explore-then-Exploit example and the comparison of cumulative and simple regret of Bubeck et al. (2009) are good news for our goal of designing efficient algorithms with near-optimal behavior, it does not bring us close enough to where we would like to be if deliberation time is limited and the MDP is large. This is because both techniques have only relocated the problem that all state-action pairs must be *executed* multiple times to the problem that all state-action pairs must be *simulated* equally often. We therefore turn our attention to anytime algorithms that are specifically designed

for MDPs with a declarative model. One of the most notable pieces of work in this direction is due to Kearns et al. (2002), who were the first to show that efficient algorithms exist that are independent from the size of the state space: their *Sparse Sampling* algorithm samples each action in the initial state a constant β number of times (where β is a parameter), which yields at most β outcomes for each state-action pair. It continues by sampling each action in all generated outcomes β times until this procedure has built a tree of depth equal to the horizon that reflects a part of the search space. The main idea of Sparse Sampling is therefore that a good policy can be derived even if only a fraction of all possible outcomes is considered. Even though Sparse Sampling still considers a number of states that is exponential in the problem horizon – the size of the tree that is build by the Sparse Sampling algorithm is $(\beta \cdot |\mathcal{A}|)^{\mathcal{H}}$ – it is independent of the size of the state set. Sparse Sampling has been an influence for a wide variety of publications, and our Trial-based Heuristic Tree Search framework builds on the ideas of Kearns et al. as well.

Another framework that incorporates the ideas of Sparse Sampling is the popular *Monte-Carlo Tree Search* framework. The term Monte-Carlo Tree Search is, to our knowledge, due to Coulom (2006), who extend an algorithm described by Chang et al. (2005). Browne et al. (2012) define Monte-Carlo Tree Search in their seminal monograph as a framework that allows the specification of algorithms in terms of two policies, a *tree policy* and a *default policy*. Like the Optimistic Rollout approach that was presented in Chapter 4, Monte-Carlo Tree Search algorithms perform trials. The tree policy specifies how the *search tree* that is built iteratively is traversed until a state is encountered that is not yet represented in the tree. That state is *explicated* (i. e., it is added to the tree), and the default policy is applied until a terminal state is reached (without adding states to the tree). The result of the trial is *back-propagated* through the tree by applying Monte-Carlo backups to all explicated states that were visited in the trial. Monte-Carlo Tree Search algorithms perform trials until a timeout is reached, and a recommendation function is then used to decide which action looks the most promising and should be executed.

The most prominent Monte-Carlo Tree Search algorithm is the aforementioned UCT algorithm (Kocsis and Szepesvári, 2006). UCT uses the UCB1 action selection as tree policy and UNI action selection as default policy. It has been applied in several scenarios with tremendous success, and has become famous due to its application to the game of Go (e.g., Gelly and Silver, 2007; Lee et al., 2009; Gelly and Silver, 2011). Numerous further applications of UCT and other Monte-Carlo Tree Search variants to games are discussed in the work of Browne et al. (2012). We have also contributed to the list with our application of UCT to the Canadian Traveler's Problem (Eyerich et al., 2010), and to probabilistic planning with the version of the PROST planner that participated at IPPC 2011 (Keller and Eyerich, 2012).

Even though the insights of Bubeck et al. (2009) regarding the simple regret of UCB1 only hold in the MAB, it is not surprising that plain UCT performs

poorly in the GMPlan scenario, a discussion that is led in detail by Domshlak and Feldman (2013). They also show that the asymptotic reduction rate of the simple regret of UCT in MDPs is, just like the reduction rate of UCB1 in MABs, sub-exponential (Feldman and Domshlak, 2012, 2014b). Their solution to the problem is a whole family of algorithms dubbed BRUE. All BRUE algorithms are based on the idea of *separation of concerns* where exploration and exploitation are decoupled and pursued in subsequent algorithm periods. The core idea is to start each trial with an explorative part where only uniform action selection is applied until a switching point is encountered, from which on only the greedy action is simulated. The MC backup function is altered such that only those states are updated that were encountered after the switching point such that only rewards that are obtained under the greedy policy influence state- and action-value estimates. According to the authors, all BRUE variants that have been described (Feldman and Domshlak, 2013, 2014a,b) achieve a decrease rate of the simple regret that is exponential. We pick up the idea to separate concerns in this thesis, generalize it and propose two variants of the novel *selective backup* function in Section 6.3.

5.1.4 Planning with a Declarative Model

The final optimization problem that is considered here is *planning with a declarative model* (DMPlan). It differs from GMPlan only in the fact that the exact model of the MDP is given to the agent rather than a blackbox that produces state transitions and rewards on demand. While the difference appears small, the consequences are considerable. This is best illustrated with an example, so let us consider the MAB of Figure 5.1 again. In the learning scenario, it is necessary to use runs to collect information on rewards and transition probabilities, all the while acting suboptimally and generating regret. The generative model allows to learn without accumulating regret by simulating each decision a sufficient number of times until the belief on the sampled action-value is strong, but it takes several trials until an agent can say with sufficiently high certainty that it prefers to pull the second arm over the third, and it takes a large number of trials until the agent is almost certain. In DMPlan, solving an equation system that consists of three Bellman optimality equations as given in Definition 11 suffices to compute the expected reward of each arm, and a DMPlan agent will therefore incur no simple regret even with little deliberation time.

We have already discussed the computation of optimal policies in MDPs if the declarative model of the MDP is provided. The acyclic value computation algorithm is a *Dynamic Programming* (DP) approach that is a variant of value iteration (Bellman, 1957) for acyclic MDPs. Value iteration computes the fixed point $V_\star(s)$ for all states $s \in \mathcal{S}$ by iteratively computing estimates $\hat{V}^k(s)$ that are based on the previous estimate $\hat{V}^{k-1}(s)$ with the help of the Bellman optimality equation. The iterations continue until the highest change

over all states in the latest estimate is smaller than a predefined, small threshold value. However, value iteration – as well as acyclic value computation – need the whole state space in memory to even get started and hence do not scale to the problems we are interested in. The same is true for the related policy iteration (Howard, 1960) algorithm.

A step towards solving MDPs with large state spaces are asynchronous versions of value iteration that do not search the state space exhaustively. (Offline, Trial-based) Real-Time Dynamic Programming (RTDP) (Barto et al., 1995), for example, performs trials like Monte-Carlo Tree Search algorithms, but it always follows the greedy policy and samples outcomes according to their probability. In contrast to Monte-Carlo Tree Search, previously unvisited states are not initialized by following a default policy, but the declarative model is used to compute an admissible heuristic, and all encountered states are added to the tree until a terminal state is reached. All states that are visited in a trial are updated with the *Full Bellman* backup function

$$\hat{V}^k_{\text{FB}}(s) = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ \max_{a \in \mathcal{A}(s)} \hat{Q}^k_{\text{FB}}(s,a) & \text{otherwise,} \end{cases}$$
$$\hat{Q}^k_{\text{FB}}(s,a) = \mathcal{R}(s,a) + \sum_{s' \in \text{succ}(s,a)} \mathbb{P}_{\mathcal{T}}\left[s' \mid s, a\right] \cdot \hat{V}^k_{\text{FB}}(s'),$$

which has the advantage that it is possible to determine (exactly in acyclic MDPs, otherwise only approximate) when state- and action-value estimates have converged to the respective $V_\star$ and $Q_\star$ values. A popular extension of RTDP that speeds up this process is Labeled RTDP (Bonet and Geffner, 2003). In acyclic MDPs as considered here, a simple *solve labeling* procedure as discussed in the following chapter suffices.

On the other hand, Full Bellman backups suffer from the problem that an action-value estimate $\hat{Q}^k_{\text{FB}}(s,a)$ can only be computed if $\hat{V}^k_{\text{FB}}(s')$ is available for each $s' \in \text{succ}(s,a)$. As the number of outcomes is not restricted in our formalism, Full Bellman backups are intractable in practice (and in some domains of the IPPC 2011 and 2014 benchmarks as well). A possible solution is implemented in GLUTTON (Kolobov et al., 2012a) and GOURMAND (Kolobov et al., 2012b), two planning systems that competed at IPPC 2011 (GLUTTON) and IPPC 2014 (GOURMAND, yet under the name G-PACK). Both are RTDP-based planning systems that cope with the challenge of a large number of outcomes by *subsampling* of the transition function a predefined number of times. All outcomes that were not sampled this way are removed from the MDP and the probabilities of the remaining outcomes are extrapolated. However, both GLUTTON and GOURMAND lose optimality in the limit by this adaption. An alternative that still allows an optimal algorithm in the limit, the *Partial Bellman* backup function, is presented in Section 6.3.

Another popular algorithm for acyclic MDPs is AO* (e.g., Martelli and Montanari, 1973; Nilsson, 1980; Pearl, 1984), an algorithm that has been ex-

tended to cyclic MDPs as LAO* (Hansen and Zilberstein, 2001). This *heuristic search* approach gradually builds an optimal solution graph, beginning from the root node representing the initial state. It expands a single tip node in the current best partial solution graph, which is the subgraph that can be reached by applying only greedy actions. It assigns admissible heuristic values to the new tip nodes and propagates the collected information through the acyclic graph to the root. These steps are repeated until all tip nodes in the best partial solution are terminal nodes. The main difference between AO* and RTDP is the termination criterion of a trial: in AO*, trials last until a previously unvisited node is encountered, while trials end only in terminal states in RTDP. The THTS framework allows to model both algorithms with identical ingredients apart from the used *trial length component*.

The combination of an admissible heuristic and the Full Bellman backup function in RTDP and AO* allows pruning of parts of the search space, as all state- and action-value estimates are guaranteed upper bounds on the expected rewards at all times. To improve the pruning procedure, many extensions of RTDP or AO* compute an additional lower bound on action-value estimates as, e. g., in Bounded RTDP (McMahan et al., 2005), Focused RTDP (Smith and Simmons, 2006), Bayesian RTDP (Sanner et al., 2009), or Iterative Bounding AO* (Warnquist et al., 2010). The pruning inherent to RTDP and AO* is an alternative way of circumventing that the whole state space of an MDP must be considered for optimal behavior. However, the success of Monte-Carlo Tree Search algorithms in planning under uncertainty or of our THTS-based PROST planner at IPPC 2014 indicate that the necessity that an expensive, admissible heuristic is computed outweighs the possible advantage of pruning.

Bonet and Geffner (2012) also identify the need for an admissible heuristic as a general problem in anytime optimal planning. Their approach, *Anytime AO**, overcomes the limitations of AO* and can cope with inadmissible heuristics by replacing the greedy selection of a tip node with an ϵ-greedy one. While potential pruning effects inherent to greedy action selection with admissible heuristics are lost, the idea that it suffices to look at a fraction of the search space that has been shown with the development of Sparse Sampling is strengthened. However, the most notable property of Anytime AO* is another one: it does not only make use of the declarative model by supporting informed heuristics, but is additionally the only algorithm we are aware of where outcomes in the simulation are selected with an *outcome selection* strategy that is more than just sampling outcomes according to their probabilities. Anytime AO* achieves this by computing the *impact* of a state on the current best solution, and by ensuring that the state with the highest impact is selected.

5.2 The Framework

The previous section has sketched a variety of techniques and algorithms that are used in four related optimization problems. In the part on MABs, we have seen that different methods for *action selection* influence the expected quality of an MAB algorithm significantly. In Online RL, we have seen how *backup functions* are a crucial tool to propagate information that is collected in a run, and how the right choice of a backup function influences the convergence to optimal behavior of an algorithm. GMPlan not only liberated us from the (complicating) learning scenario, allowing the computation of near-optimal policies from the start even though only a fraction of the search space is considered, but it also introduced a *recommendation function* that generates a final decision from the collected information. Furthermore, the default policy of the Monte-Carlo Tree Search framework can also be regarded as a first attempt to include heuristics into the search. And finally, our discussion of DMPlan revealed that informed heuristic functions and a labeling procedure are well-suited to achieve policies of high-quality quickly, that different *trial lengths* significantly alter the performance of an algorithm and that sampling is not the only way for *outcome selection* in a trial.

In this section, we develop the *Trial-based Heuristic Tree Search* framework for planning with a generative or declarative model such that it allows the description of a large variety of algorithms with only a few ingredients. We are especially interested in a framework that subsumes the most successful classes of algorithms for learning and for planning with generative or declarative model: Dynamic Programming, Monte-Carlo Tree Search, Temporal Difference Learning, and heuristic search algorithms. Our aim is a formalization that allows the specification of a large number of different algorithms, but we also want to keep things as simple as possible.We believe that this can be achieved in a framework that allows the specification of algorithms by providing the six ingredients action selection, outcome selection, initialization, trial length, backup function, and recommendation function. All of these were already briefly introduced in the previous section and are discussed in detail in Chapter 6. Our view on THTS differs from original work on the topic (Keller and Helmert, 2013) only in the fact that we have replaced the static *greedy recommendation* with the adaptable ingredient of the same name. We decided to add the ingredient to our framework due to the work of Chaslot et al. (2008), who yield better results with the Monte-Carlo Tree Search algorithm UCT if the action that was simulated most often is recommended, and due to Bubeck et al. (2009), who show that different recommendation strategies lead to different bounds on the simple regret in MABs (and that it is not always the best decision to recommend the greedy action).

The term “Trial-based Heuristic Tree Search” already summarizes what is common among all THTS algorithms fairly well: the acyclic MDP is unfolded to a *tree*, which is *searched* in a sequence of *trials* that are guided by a *heuristic*.

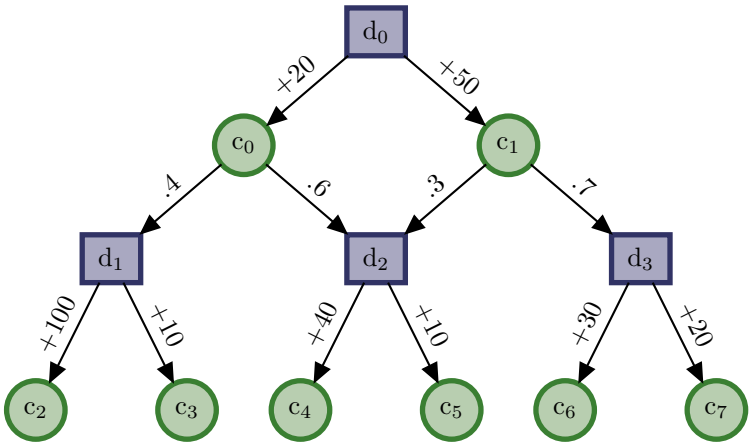

Figure 5.4: Complete AND/OR graph $\mathcal{G}$ of the example MDP of Figure 5.2.

Before we discuss possible ingredients in Chapter 6 and recipes that combine to anytime optimal behavior in Chapter 7, we present the underlying idea of the framework. Let us start with a look at the search space that is considered in THTS. We derive it in two steps: first, we show how an *AND/OR graph* is induced by an acyclic MDP, and in a second step we show how to convert the AND/OR graph to an equivalent *AND/OR tree*. The term AND/OR graph stems from the fact that it consists of two kinds of vertices (or *nodes*): AND nodes (which we call *chance nodes*) and OR nodes (which go under the name *decision nodes* here). The edges connect vertices such that decision and chance nodes alternate. The *root node* of an AND/OR graph is always a decision node, and the leaf nodes are always chance nodes.

Definition 18 (AND/OR-graph). *An* **AND/OR graph** *is specified by a tuple* $\mathcal{G} = \langle d_0, \mathrm{D}, \mathrm{C}, E \rangle$ *with a set of vertices* $N = \mathrm{D} \cup \mathrm{C}$*, which is given by the disjunct sets of* **decision nodes** D *and* **chance nodes** C*; with* **root node** $d_0 \in \mathrm{D}$*; and with edges* $E \subseteq N \times N$ *such that*

- $\langle N, E \rangle$ *is a directed acyclic graph;*
- *there is a* $c \in \mathrm{C}$ *for all* $d \in \mathrm{D}$ *such that* $(d, c) \in E$*; and*
- *all* $(n, n') \in E$ *are such that* $n \in \mathrm{D} \Rightarrow n' \in \mathrm{C}$ *and* $n \in \mathrm{C} \Rightarrow n' \in \mathrm{D}$.

If $(n, n') \in E$*, we call* n' *a* **successor** *of* n *and* n *a* **predecessor** *of* n'*. The set of all predecessors of* n' *is* $\mathrm{pred}(n) \subseteq N$*. Sequences* $(n_1, \ldots, n_k)$ *are* **trajectories** *in* $\mathcal{G}$ *if* $(n_i, n_{i+1}) \in E$ *for all* $i = 1, \ldots, k-1$*. We call an AND/OR graph* $\mathcal{G}$ *an* **AND/OR tree** *iff* $\langle N, E \rangle$ *is a tree.*

Each finite-horizon, factored MDP $\mathcal{M}$ induces an AND/OR graph $\mathcal{G}$. Since we have not excluded unreachable states from MDPs,[2] we exclude nodes from $\mathcal{G}$ if the corresponding state is not reachable from s_0. While it is not necessary to do this, it is convenient for a variety of reasons and simplifies the analysis that follows considerably. Then, for each state $s \in \mathcal{S}$ that is reachable from s_0, there is a decision node $d \in \mathsf{D}$ in $\mathcal{G}$ with assigned state $s(d) = s$; for each state-action pair (s, a) where s is reachable from s_0, there is a chance node $c \in \mathsf{C}$ with assigned state $s(c) = s$ and assigned action $a(c) = a$; and there are no other nodes in N. The edges of $\mathcal{G}$ connect nodes according to the transitions in $\mathcal{M}$: for each transition (s, a, s') where s is reachable from s_0, there is an edge (d, c) from the unique decision node d with $s(d) = s$ to the unique chance node c with $s(c) = s$ and $a(c) = a$; and there is an edge (c, d') from c to the unique decision node with $s(d') = s'$ unless $s' \in \mathcal{S}_\star$. Since acyclic MDPs and AND/OR graphs resemble each other so closely, we use the following abbreviations that were introduced for states and state-action pairs earlier in the obvious way for decision nodes and chance nodes as well:

- the successors of a decision node d are $\mathcal{A}(d) := \{c \in \mathsf{C} \mid (d, c) \in E\}$;
- the successors of a chance node c are $\mathrm{succ}(c) := \{d \in \mathsf{D} \mid (c, d) \in E\}$;
- the reward in a chance node c is $\mathcal{R}(c) := \mathcal{R}(s(c), a(c))$;
- the transition probability from c to d is $\mathbb{P}_{\mathcal{T}}[d \mid c] := \mathbb{P}_{\mathcal{T}}[s(d) \mid s(c), a(c)]$;
- for each trajectory $\theta_1 = (d_1, c_1, \ldots, c_n)$ in $\mathcal{G}$ and $\theta_2 = (c_1, d_2, \ldots, c_n)$:
  - the accumulated reward is $\mathcal{R}(\theta_1) = \mathcal{R}(\theta_2) := \sum_{t=1}^{n} \mathcal{R}(c_t)$;
  - the combined probability is $\mathbb{P}[\theta_1] = \mathbb{P}[\theta_2] := \prod_{t=1}^{n-1} \mathbb{P}_{\mathcal{T}}[d_{t+1} \mid c_t]$;
- and the set of leaf nodes is $N_\star := \{c \in \mathsf{C} \mid s(c)[h] = 1\}$.

An AND/OR graph $\mathcal{G}$ that is induced by a finite-horizon, factored MDP $\mathcal{M}$ differs from $\mathcal{M}$ only in a few details: even though it appears differently at first sight in the figures of MDPs we have seen so far in this thesis, an MDP does actually *not* consist of two different kinds of vertices that are connected by edges, but of a single kind of vertex (representing states) which are connected via *hyperarcs*. (A closer inspection of the figures reveals that we have actually taken this fact into account by omitting arrow heads in incoming edges of "action nodes", which are thus just a graphical way of representing hyperarcs.) The AND/OR graph of $\mathcal{M}$, on the other hand, contains a decision node for each state and a chance node for each state-action pair. The nodes that are depicted as circles in Figure 5.4, which shows the AND/OR graph that is induced by the MDP from Figure 5.2, are hence actual vertices in the AND/OR graph. To illustrate the connection between decision nodes and

[2] A state s is reachable from s_0 iff there is a trajectory $\theta = (s_0, a_0, \ldots, s) \in \Theta$ with $\mathbb{P}[\theta] > 0$.

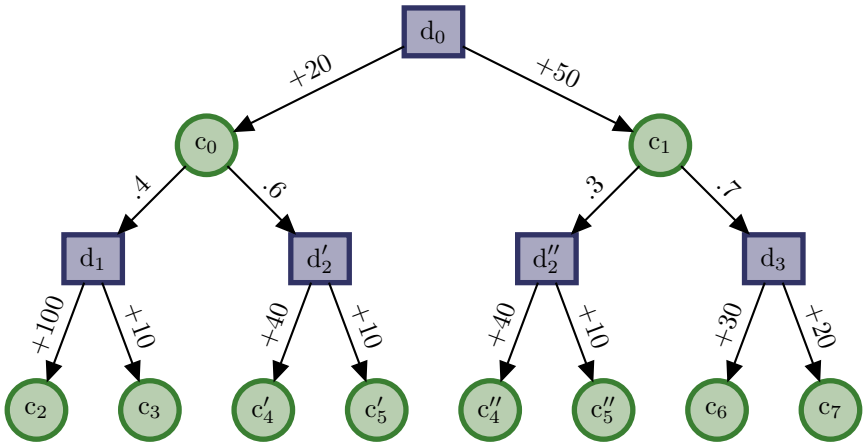

Figure 5.5: Complete AND/OR tree $\mathcal{G}$ of the example MDP of Figure 5.2.

states on the one hand and chance nodes and actions on the other, we use the same colors and forms for the respective elements in all figures. Other differences between the finite-horizon, factored MDP and the induced AND/OR graph are mostly of technical nature. Terminal states of the finite-horizon, factored MDP are not represented in the AND/OR graph, which is possible since they do not entail any information (recall that rewards are independent from the outcome) and because it allows for simpler and cleaner definitions in the remainder of this thesis. Therefore, the set of leaf nodes consists of chance nodes and not of decision nodes. Moreover, even though we have amended Figure 5.4 (and all other Figures that show AND/OR graphs) with edge labels that represent transition probabilities and rewards, they are technically not part of the AND/OR graph but induced by the corresponding MDP (but could, of course, be added easily as edge labels).

A Trial-based Heuristic *Tree* Search algorithm does, as the name implies, not build an AND/OR graph but an equivalent AND/OR *tree*. It can be derived from an AND/OR graph by iteratively replacing each subgraph that starts in a decision node with more than one predecessor with a copy of the subgraph for each predecessor until the AND/OR graph is an AND/OR tree. In our example AND/OR graph of Figure 5.4, there is only one decision node with multiple parents. Cutting the subgraph that is rooted in d_2 from the AND/OR graph removes it along with the nodes c_4 and c_5, and appending a copy to both parents c_0 and c_1 results in the tree that is depicted in Figure 5.5 where the respective nodes are replaced by a primed and a doubly primed version. However, this is only a theoretical procedure that allows the description of the *complete* AND/OR tree $\mathcal{G}$. We have already emphasized that it is a fundamental limitation that prevents the ability to scale to large MDPs if an algorithm requires that the whole MDP is represented in memory. Of course, this is also the case if the complete AND/OR tree has to be *explicated* in memory before

an algorithm starts its computations.

Each THTS algorithm solves this dilemma by maintaining a version of the AND/OR tree which is explicated iteratively step by step. The *explicit graph* is no *static* construct while a sequence of trials $(\theta^1, \ldots, \theta^n)$ is simulated, but it changes over time. We use a superscript to denote with $\mathcal{G}^k = \langle d_0, \mathrm{D}^k, \mathrm{C}^k, E^k \rangle$ the explicit AND/OR tree *at the end* of trial $\theta^k = (d_0, c_0, \ldots, c_l)$ (as opposed to the complete AND/OR tree $\mathcal{G}$ which goes without superscript). In THTS, all nodes in the explicit graph can have any number of additional annotations. However, we restrict ourselves to the following annotations that describe a node's property at the end of the k-th trial in this thesis:[3]

- *state-value estimates* $\hat{V}^k(d) \in \mathbb{R}$ for all $d \in \mathrm{D}$;
- *action-value estimates* $\hat{Q}^k(c) \in \mathbb{R}$ for all $c \in \mathrm{C}$;
- *solve labels* $\varphi^k(n) \in \{\texttt{true}, \texttt{false}\}$ for all $n \in N$;
- *backup counter* $\mathcal{B}^k(n) \in \mathbb{N}$ for all $n \in N$;
- *selection counter* $\mathcal{L}^k(n) \in \mathbb{N}$ for all $n \in N$; and
- sets of *explicated outcomes* $\mathrm{succ}^k(c) \subseteq \mathrm{succ}(c)$ for all $c \in \mathrm{C}$.

We denote the set of *greedy actions* of a decision node d at the end of the k-th trial with $\mathcal{A}^k_\star(d) \subseteq \mathcal{A}(d)$, which is technically an abbreviation for $\mathcal{A}^k_\star(d) = \{c \in \mathcal{A}(d) \mid \hat{Q}^k(c) \geq \hat{Q}^k(c') \text{ for all } c' \in \mathcal{A}(d)\}$ rather than a true annotation (i. e., it is not maintained in memory explicitly).

The pseudo code of the THTS framework is depicted in Algorithm 7. Instances of the framework start building the explicit graph with a graph $\mathcal{G}^0$, an AND/OR tree that consists only the root node d_0, a decision node with $s(d_0) = s_0$. The AND/OR tree is built iteratively from a finite-horizon, factored MDP description $\mathcal{M}$ by performing a sequence of trials $(\theta^1, \ldots, \theta^n)$. The number of trials n can be set directly via a parameter or indirectly via a timeout parameter in our implementation, and trials stop prematurely if the root node is labeled as solved. Most trials of an THTS algorithm add nodes and edges to the explicit graph. Each trial can be described in terms of two phases: an initial *expansion phase* which is followed by a *backup phase*. In the expansion phase, the explicit tree is traversed starting from the root node by alternatingly selecting a chance node that represents an action $a \in \mathcal{A}(s(d))$ in a decision node d and an outcome $s \in \mathrm{succ}(s(c), a(c))$ in a chance node c according to the action and outcome selection ingredients. Different strategies for action and (to a lesser extent) outcome selection are discussed in the

[3] The restriction comes from the fact that it is straightforward to extend the framework with additional (explicit) annotations, but supporting an arbitrary set of annotations complicates the notation significantly.

Algorithm 7: Trial-based Heuristic Tree Search

```
1  THTS(M):
2      G = initialize_explicit_graph(M)
3      while more_trials(d_0) do
4          visit_decision_node(M, G, (d_0))
5      return ~ recommend(G)
6  visit_decision_node(M, G, (d_0, c_0, ..., d_t)):
7      if d_t is not expanded then
8          for a ∈ A^k(s(d_t)) do add_chance_node(G, d_t, a);
9          initialize(G, d_t)
10     L(d_t) ← L(d_t) + 1
11     a ~ select_action(G, d_t)
12     visit_chance_node(M, G, (d_0, c_0, ..., d_t, cn(d_t, a)))
13     backup(G, d_t)
14 visit_chance_node(M, G, (d_0, c_0, ..., c_t)):
15     L(c_t) ← L(c_t) + 1
16     if c_t is not a leaf and continue_trial(G, (d_0, c_0, ..., c_t)) then
17         s ~ select_outcome(G, c_t)
18         if dn(c_t, s) = ⊥ then add_decision_node(G, c_t, s);
19         visit_decision_node(M, G, (d_0, c_0, ..., c_t, dn(c_t, s)))
20     backup(G, c_t)
```

Sections 6.5 and 6.2, respectively. And in the backup phase, one o several proposed backup functions is applied to all visited nodes in reverse order.

Please note that it is sometimes not sufficient to access the explicit graph or an annotation at the end of a trial, but at a point in the expansion phase where only a prefix of the k-th trial $\theta^{k,t} = (d_0, c_0, \ldots, c_t)$ with $t \leq l$ has been selected or at a point in the backup phase where only some of the nodes have been updated. We annotate the explicit AND/OR Graph or an annotation with an additional superscript, e. g., $\mathcal{G}^{k,t} = \langle d_0, \mathrm{D}^{k,t}, \mathrm{C}^{k,t}, E^{k,t} \rangle$, in both cases, and clarify if we mean the corresponding graph or value during the expansion or the backup phase if it is not clear from the context.

The THTS framework guarantees that each explicit graph $\mathcal{G}^k$ for $k > 0$ (the fact that this does not hold in $\mathcal{G}^0$ is irrelevant for our discussion) is such that all chance nodes $c \in \mathcal{A}(d)$ are explicated when decision node d is explicated. The result of the action selection is hence that the algorithm proceeds with the chance node that corresponds to the selected action. This is not the case with decision nodes that represent a state that is selected by the outcome selection ingredient. We have to keep track of the states that are explicated at the end of the k-th trial, and denote the (initially empty) set of *explicated outcomes* of a chance node c with $\mathrm{succ}^k(c) \subseteq \mathrm{succ}(c)$. Related to this, we abbreviate the

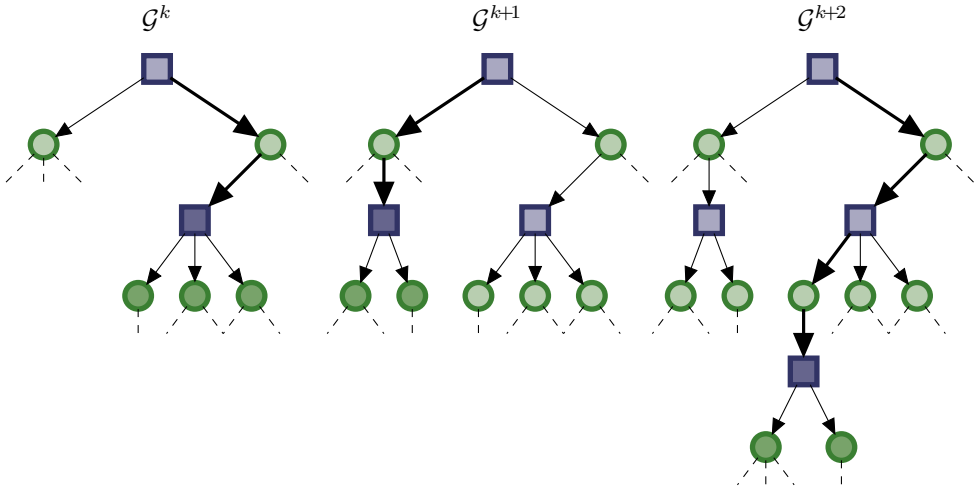

Figure 5.6: Example of three consecutive explicit trees $\mathcal{G}^k$, $\mathcal{G}^{k+1}$, and $\mathcal{G}^{k+2}$ that are extended in the expansion phase of an THTS algorithm. Thick edges specify decisions of action and outcome selection, and opaque nodes are explicated in the current trial. Dashed edges hint at outcomes that have not been explicated yet.

sum of the probabilities of all explicated outcomes of a chance node c with $\mathbb{P}\left[\text{succ}^k(c)\right] := \sum_{d \in \text{succ}^k(c)} \mathbb{P}_{\mathcal{T}}[d \mid c]$ and denote with $\mathbf{dn}^k(c, s)$ the function that maps to the (only possible) decision node $d \in \text{succ}^k(c)$ with $s(d) = s$ if that node exists, and to $\perp$ otherwise; and with $\mathbf{cn}^k(d, a)$ the unique chance node $c \in \mathcal{A}^k(c)$ (which must be explicated in the tree).

If the outcome selection ingredient chooses state s in node c and $s(d) \neq s$ for all $d \in \text{succ}^k(c)$, the THTS framework creates a new node d' with $s(d') = s$ and adds d' to D^k and (c, d') to E^k. Furthermore (and this ensures that the invariant that all successors of decision nodes are explicated holds), the created decision node d' is completely expanded, i. e., a chance node c_a with $s(c_a) = s(d')$ and $a(c_a) = a$ is created for each $a \in \mathcal{A}(d')$, added to C^k and connected to d' by adding (c_a, d') to E^k. Then the initialization ingredient initializes the newly created nodes by assigning initial values to all node annotations. The process of selecting actions and outcomes and of expanding the explicit tree is repeated until a leaf node $c_\star$ is encountered or until the trial length ingredient decides to terminate the trial.

Example 20. *An example of an expansion phase is given in Figure 5.6. Given that the explicit graph after k trials is the graph to the left ($\mathcal{G}^k$), the $(k+1)$-th trial starts with the decision of the action selection that the left action is simulated (as indicated by the thick edge). The outcome selection ingredient decides that the outcome to the middle is simulated, but the corresponding successor is not yet explicated (as can be seen by comparing $\mathcal{G}^k$ with $\mathcal{G}^{k+1}$). The respective*

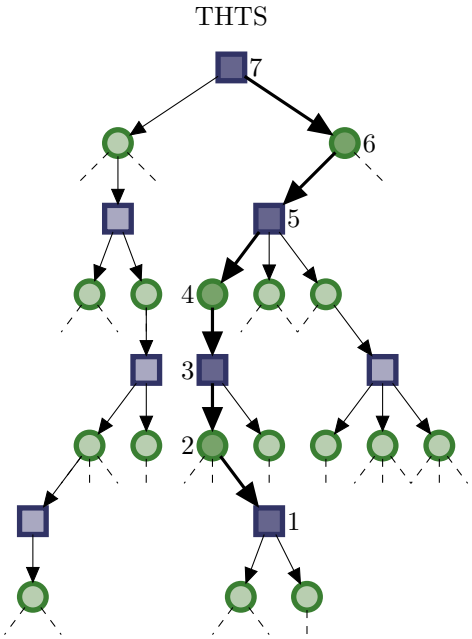

Figure 5.7: Example of the backup phase of a THTS algorithm. Thick edges mark the selected trajectory, and the number next to an (opaque) node gives the relative position of the node in the backup process (nodes with lower numbers are updated earlier).

decision node is appended to the tree, as are the two successor nodes that represent the two actions that are applicable in that node. In trial θ^{k+2}, the right action is selected initially, followed by the left outcome and the left action. As the only (deterministic) outcome has not been explicated yet, the tree is expanded in the same vein as in the previous trial by adding a decision node and all of its successors to the explicit graph.

In the subsequent backup phase, all nodes that were visited during a trial are updated in *reverse order*, which allows for an efficient propagation of collected information through the explicit tree. The trial finishes after the backup function has been applied to the root node, and a new trial starts if desired.

Example 21. *The tree in Figure 5.7 shows what has become of the explicit graph of Example 20 a few trials later. As before, the thick edges indicate the trajectory that was selected in the expansion phase, which ended in the appendage of the decision node with the number* 1 *to its right and of its two successor nodes. In the backup phase, the nodes that were visited in the trial during the expansion phase are updated in reverse order, which is given by the numbers next to a subset of all nodes. The advantage of updates in reverse order is that information that is collected in the trial can be propagated in the same trial all the way up to the*

root node. After the seventh call to the backup function, when the root node is updated, the trial ends and it is determined if another trial is performed.

Once the THTS algorithm decides to stop the simulation of trials, the recommendation function analyzes the information that is contained in the successor nodes of the root node (which represent the actions that are applicable in the current state of the environment), and executes the action it believes to be best. Upon receiving the new current state (with a number of step-to-go that is decreased by one) from the environment, the process is repeated and the next decision is derived with THTS until a terminal state is reached. In the following chapter, we continue our discussion of Trial-based Heuristic Tree Search algorithms by proposing several possible ingredients that allow the derivation of a large variety of algorithms within our framework.

CHAPTER 6

THTS Ingredients

In the previous chapter, we have presented the Trial-based Heuristic Tree Search framework, which allows the specification of a large number of algorithms in terms of only six ingredients: initialization, backup function, trial length, outcome selection, action selection, and recommendation function. In this chapter, we present a selection of well-known ingredients that are used in popular algorithms and can be used to derive novel variants within the THTS framework. The two ingredients that apply the recursive formulas to compute the annotations of search nodes are the initialization, which is described in Section 6.1 and the backup function, which is discussed in Section 6.3. The ingredients that define the expansion phase of a THTS algorithm are the trial length component (in Section 6.4), the outcome selection (in Section 6.2), and the action selection (in Section 6.5). Finally, Section 6.6 discusses the ingredient that is used to derive a final decision based on the results of the other components, the recommendation function.

6.1 Initialization

We start our discussion of THTS ingredients with the *initialization*, a component that is used to determine the starting values of to the recursive functions that describe the update rules of a node's annotations. In particular, an initialization receives an explicit graph $\mathcal{G}^k$ and a decision node $d \in \mathrm{D}^k$ as input and initializes its state-value estimate $\hat{V}^0(d)$, its selection counter $\mathcal{L}^0(d)$, its backup counter $\mathcal{B}^0(d)$, and its solve label $\varphi^0(d)$. The THTS framework asserts that the decision node d must have been expanded right before the initialization component is invoked – i. e., for all actions $a \in \mathcal{A}$ that are applicable in $s(d)$, there is a chance node $c \in \mathrm{C}^k$ with $a(c) = a$ and an edge $(d, c) \in E^k$. However, these successors are not yet initialized, so the initialization furthermore initializes the action-value estimate $\hat{Q}^0(c)$, the selection counter $\mathcal{L}^0(c)$, the backup counter $\mathcal{B}^0(c)$, and the solve label $\varphi^0(c)$ of all $c \in \mathcal{A}(d)$.

Definition 19 (Initialization). *An* **initialization** *is a function $\mathfrak{I}$ that takes an explicit graph $\mathcal{G}^k$ and a decision node d as input and determines the starting values $\hat{V}^0(d)$, $\hat{Q}^0(c)$, $\mathcal{B}^0(n)$, $\mathcal{L}^0(n)$, and $\varphi^0(n)$ of the corresponding recursive functions for all $c \in \mathcal{A}(d)$ and $n \in \{d\} \cup \mathcal{A}(d)$.*

We consider only a single initialization technique in this thesis, the *virtual trials initialization* $\mathfrak{I}_{\text{VT}}$, which is inspired from our work on the CTP (Eyerich et al., 2010) and has been adapted to domain-independent probabilistic planning in the PROST planner (Keller and Eyerich, 2012). In the virtual trials initialization, an *action-value heuristic* $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is used to compute a finite heuristic value for each state-action pair $(s(c), a(c))$. Based on the results of h, the virtual trials initialization initializes each $c \in \mathcal{A}(d)$ with

$$\begin{aligned} \hat{Q}^0(c) &= \omega \cdot h(s(c), a(c)), \\ \mathcal{B}^0(c) &= \chi, \\ \mathcal{L}^0(c) &= \chi, \text{ and} \\ \varphi^0(c) &= \texttt{false}, \end{aligned} \tag{6.1}$$

and, subsequently, the parent node d with

$$\begin{aligned} \hat{V}^0(d) &= \max_{c \in \mathcal{A}(d)} \hat{Q}^0(c), \\ \mathcal{B}^0(d) &= |\mathcal{A}(d)| \cdot \chi, \\ \mathcal{L}^0(d) &= |\mathcal{A}(d)| \cdot \chi, \text{ and} \\ \varphi^0(d) &= \texttt{false}. \end{aligned}$$

There are two parameters that can be used to adapt the virtual trials initialization. The first, $\omega > 0$, describes the *weight* of the heuristic. It plays a somewhat similar role to the weight parameter in the Weighted A* search algorithm (Pearl, 1984) in the sense that it balances to what extent the algorithm relies on heuristic information rather than the immediate rewards it has actually obtained during search. However, unlike the weight parameter in Weighted A*, our parameter ω does not have a clear cut influence on solution quality as the influence of the heuristic cancels out in combination with most backup functions (and in particular with all backup functions that are presented in Section 6.3) over the course of a large number of trials.

The main idea of the virtual trials initialization is reflected by the second parameter, $\chi \in \mathbb{N}^+$. All annotations of all nodes in the explicit graph are initialized by treating the heuristic values as if there have been χ *virtual trials* in earlier visits of the node, each with a result of $\omega \cdot h(s(c), a(c))$. To complete the impression that χ (virtual) trials have been performed prior to the initialization, all backup and selection counters of all chance nodes are set accordingly as well. The final annotation, the solve label, is set to `false` initially (if the chance nodes are leaves, we leave it to the backup function to label the nodes

as solved if desired). Please observe that all annotations are initialized to *finite* starting values if the virtual trials initialization is used, a property that is important in our theoretical evaluation of THTS recipes in Chapter 7.

An advantage of heuristic search algorithms over Monte-Carlo Tree Search is that it is not necessary to select each child node of a node at least once before an informed decision can be taken by an action selection ingredient. Especially when the deliberation time is sparse this has proven to be an significant benefit of heuristic search over Monte-Carlo Tree Search. Examples where virtual trials (or comparable techniques) improve the performance considerably include the winner of the AAAI 2007, 2008, and 2012 General Game Playing competition CADIA (Finnsson and Björnsson, 2008, 2010, 2011), the popular Go player of Gelly and Silver (2007, 2011) and PROST (Keller and Eyerich, 2012), the winner of IPPC 2011 and 2014 that is also the base of our implementation of the THTS framework.

The third – and likely most influential – way to customize the virtual trials initialization is by the choice of the heuristic function h. Algorithms for the computation of heuristic estimates have received much attention in recent years, especially in the context of classical planning (e.g., Haslum and Geffner, 2000; Bonet and Geffner, 2001; Richter et al., 2008; Helmert and Domshlak, 2009). Even though their work has shown the crucial influence of powerful heuristics, heuristic functions are *not* the focus of this thesis. However, we believe that heuristic functions for probabilistic planning are a promising research area for future work. In the empirical evaluation of THTS algorithms in Chapter 8, we use the base heuristic that comes with the implementation of the PROST planner. It performs an iterative deepening search on the most-likely determinization of the finite-horizon, factored MDP until the determinization has been searched exhaustively, or until a timeout is reached that is determined in a learning step before the interaction between planner and environment starts. An in-depth discussion of the determinization and the heuristic can be found in the description of the PROST planner (Keller and Eyerich, 2012).

6.2 Outcome Selection

While it is not possible to select the outcome of an *executed* action at will, it is entirely possible to decide during *simulation* which outcome of an action application deserves more attention. The THTS ingredient that is responsible for this decision is the outcome selection. Depending on the provided model, it is only possible to select an outcome that was encountered during previous trials (with a generative model) or to choose an arbitrary outcome (with a declarative model). However, there is only little work on the topic in the literature. Apart from the Anytime AO* algorithm of Bonet and Geffner (2012), who define a measure for the impact of a tip node that influences the outcome

selection, we are not aware of any efforts on the development of outcome selections. All other work uses an outcome selection component that samples among (unsolved) outcomes according to their probabilities (possibly extrapolated with the sum over the probabilities of solved outcomes). Even though the THTS framework is specifically designed to support arbitrary outcome selection strategies, our own work is (unfortunately) no exception. Let

$$\text{succ}^k_{\neg\varphi}(c) = \{s' \in \text{succ}(s(c), a(c)) \mid \mathbf{dn}^k(c, s') = \bot \text{ or } \neg\varphi(\mathbf{dn}^k(c, s'))\}$$

be the set of all unsolved successors of a chance node c (this includes outcomes that are not yet explicated and is hence a set of states rather than a set of decision nodes). Furthermore, let

$$\mathbb{P}^k_{\neg\varphi}[c] = \sum_{d \in \text{succ}^k_{\neg\varphi}(c)} \mathbb{P}_{\mathcal{T}}\left[s(d) \mid s(c), a(c)\right]$$

be the accumulated probability of all unsolved outcomes. These allow the definition of an *outcome selection* $\mathfrak{O}$ in general and of the *Monte-Carlo* (MC) outcome selection $\mathfrak{O}_{\text{MC}}$ in particular:

Definition 20 (Outcome Selection). *An* **outcome selection** *is a function $\mathfrak{O}$ that maps an explicit graph $\mathcal{G}^{k,t}$ and a trial $\theta^{k,t} = (d_0, c_0, \ldots, d_t, c_t)$ that is such that* $\text{succ}^k_{\neg\varphi}(c_t) \neq \varnothing$ *to a probability distribution over* $\text{succ}^k_{\neg\varphi}(c_t)$.

Given a trial $\theta^k = (d_0, c_0, \ldots, d_t, c_t)$, the MC outcome selection selects each state $s \in \text{succ}^k_{\neg\varphi}(c_t)$ in chance node c_t with probability

$$\mathbb{P}_{\text{MC}}(s(d_{t+1}) = s) = \frac{\mathbb{P}_{\mathcal{T}}[s \mid s(c_t), a(c_t)]}{\mathbb{P}^k_{\neg\varphi}[c_t]}$$

and proceeds the trial in the node that corresponds to the selected successor state d_{t+1}.

6.3 Backup Functions

In the previous chapter, we have seen that the role of the backup function is the propagation of information that is gathered in a trial or run. The most important difference between any two backup functions is the way how the new information is incorporated into the state- and action-value estimates of all decision and chance nodes that are affected by the current trial. Other annotations that are updated by the backup function are the backup counter $\mathcal{B}^k(n)$ and the solve label $\varphi^k(n)$ of all visited nodes n.

Definition 21 (Backup Function). *A* **backup function** *is a function $\mathfrak{B}$ that takes an explicit graph $\mathcal{G}^{k,t}$ and a decision or chance node n as input, and computes the values of the annotations $\hat{V}^k(n)$ (iff $n \in \mathsf{D}$), $\hat{Q}^k(n)$ (iff $n \in \mathsf{C}$), $\mathcal{B}^k(n)$, and $\varphi^k(n)$.*

The backup function of all considered annotations is given as a recursive formula with starting value that is determined by the initialization, and it is invoked for all nodes of the current trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$ *in reverse order*. An annotation $X^k(n)$ describes the value of annotation $X \in \{\hat{V}, \hat{Q}, \mathcal{B}, \varphi\}$ of node n at the end of the k-th trial. The recursive equation that allows the computation of the value $X^k(n)$ in the t-th timestep of the k-th trial may depend (for practical reasons) only on the latest values of all annotations at the moment the update takes place. In particular, for a node $n_t \in \{d_t, c_t\}$ that was visited in the current trial, this means that an annotation $X^k(n_t)$ may be updated (apart from parameters, constants, and information on the current trial like the accumulated reward) only based on (1) the value of annotations that belong to the same node with the values of the end of the previous trial (i. e., $\hat{Q}^{k-1}(n_t)$ or $\hat{V}^{k-1}(n_t)$, $\mathcal{B}^{k-1}(n_t)$, and $\varphi^{k-1}(n_t)$) and (2) on the (already updated) values $\hat{Q}^k(n')$ or $\hat{V}^k(n')$, $\mathcal{B}^k(n')$, and $\varphi(n')$ of all nodes n' that are part of the subgraph that is rooted at n_t. (As we are interested in anytime algorithms which must ensure that trials do not take prohibitively long, we restrict ourselves even further and consider only backup functions that depend on nodes that were visited in the current trial and on their direct successor nodes.) The dependencies reflect the fact that annotations are updated in an order that is given by an increasing number of trials (1) and within a trial from the leaves to the roots (2), and that they are updated *in-place* in practice, which we believe to be important for efficient implementations.

We nevertheless make one exception (yet for convenience of notation only), and assume that the backup counter $\mathcal{B}^k(n)$ is updated *before* $\hat{Q}^k(n)$ and $\hat{V}^k(n)$ for all $n \in N$ and can hence additionally be used in the recursive formulas for the computation of state- and action-value estimates. All backup functions that are considered in this thesis have in common that state-value estimates, backup counter and solve labels are updated by applying the equations

$$\hat{V}^k(d) = \max_{c \in \mathcal{A}(d)} \hat{Q}^k(c) \tag{6.2}$$

$$\mathcal{B}^k(n) = \mathcal{B}^{k-1}(n) + 1 \tag{6.3}$$

$$\varphi^k(n) = \texttt{false} \tag{6.4}$$

to all decision nodes d_t and all nodes $n_t \in \{d_t, c_t\}$ for $0 \leq t \leq l$ that are updated in the backup phase of a THTS trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$. The update of the action-value estimate differs between all backup functions, and the two remaining annotations that were introduced in the previous chapter, the selection counter $\mathcal{L}^k(n)$ and the set of explicated outcomes $\text{succ}^k(c)$, are maintained by the framework directly.

We have already seen some examples of backup functions in previous parts of this thesis. Chapter 2.3 presented the Bellman optimality function and showed how to turn it into an assignment operator (the Full Bellman backup

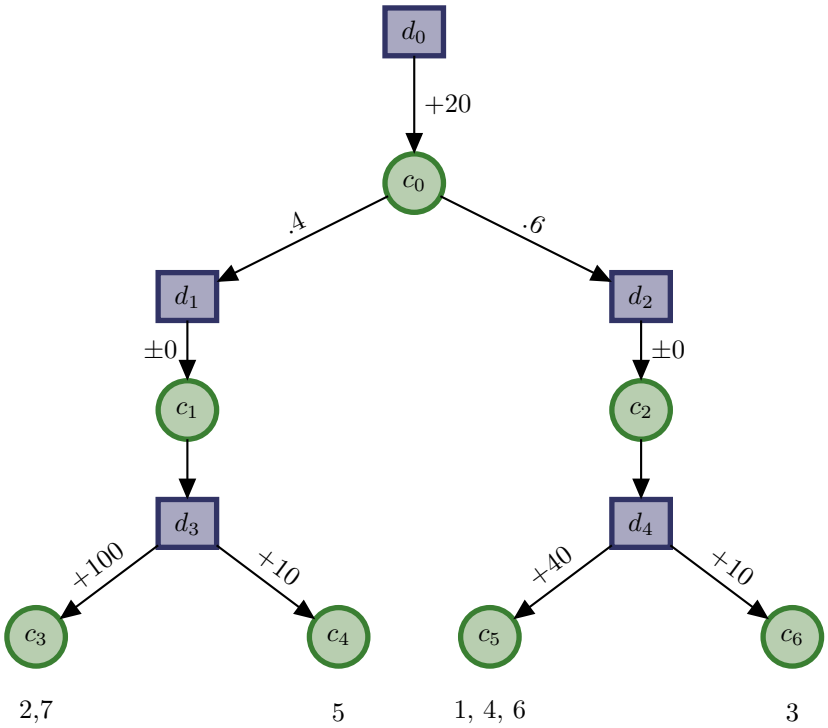

Figure 6.1: Explicit AND/OR tree that is used for the running example of this section: a number x below the chance nodes $c_3, \ldots, c_6$ specifies that we assume that the x-th trial ends in the corresponding node in the first example scenario (trial two and five are interchanged in the second example scenario).

function) that is used in the acyclic value computation algorithm, and we have met the same idea again in our discussion of algorithms for planning with declarative model in Section 5.1.4. However, we have also determined that Full Bellman backups are no tractable option in THTS as all successors of a chance node must be explicated for their computation, which can be a number that is exponential in the number of state variables in our framework. While there are possibilities to deal with this problem, e. g., by using a common estimate for all outcomes that are not yet explicated, we are not aware of an *anytime optimal* solver that uses a variant of Full Bellman backups. The GLUTTON and GOURMAND planners of Kolobov et al. (2012a,b) use a variant where a fixed number of outcomes is subsampled a-priori, but they lose anytime-optimality in the process. In this thesis, we do therefore *not* consider Full Bellman backups nor any other kind of full backup function.

Example 22. *We demonstrate the propagation of information with different backup functions with the help of an example MDP that is similar to the left arm of the MDP that was used to introduce Online RL, AND/OR graphs and AND/OR*

trees in Chapter 5. It differs only in the fact that we introduced additional decision nodes d_1 and d_2 (with a single applicable action) and (deterministic) chance nodes c_1 and c_2. The relevant part is depicted in Figure 6.1, along with markings (at the bottom) that describe the first seven trials that are considered in this example (they are such that $\theta^1 = (d_0, c_0, d_2, c_2, d_4, c_5)$, $\theta^2 = (d_0, c_0, d_1, c_1, d_3, c_3)$, etc.). To simplify the example, we assume that all chance nodes are initialized with an initialization function that sets $\hat{Q}^0(c_i) = 0$ for $i \in \{0, 1, 2, 3, 6\}$, and $\hat{Q}^0(c_4) = \hat{Q}^0(c_5) = \epsilon$ for some $\epsilon > 0$. The value for ϵ is sufficiently close to 0 that we can treat it as 0 when action-value estimates are updated, but it allows us to regard $a(c_4)$ and $a(c_5)$ as the initial greedy choice in d_3 and d_4, respectively. (This detail is only relevant in Examples 26 and 27).

An overview on the resulting action-value estimates $\hat{Q}^k(c_0)$ with all backup functions that are presented over the course of this section for the first seven trials is given in Table 6.1, along with the action-value estimates in an alternative set of trials where the second and fifth trial are exchanged. (To distinguish both sets of trials, we refer to them with first and second example scenario in the following.) Action-value estimates for node c_0 for the first and second example scenario are furthermore depicted graphically in Figures 6.2 and 6.3, respectively. For comparison, it might be interesting that the true action value estimate of the action that leads to c_0 is $Q_\star(c_0) = 20 + (.4{\cdot}100) + (.6{\cdot}40) = 84$. The performance of the individual backup functions in the set of example trials is discussed over the course of this section.

6.3.1 Monte-Carlo Backups

The most popular backup function in many applications for learning and planning under uncertainty is the Monte-Carlo backup function. We have already seen in Section 5.1 that MC backups maintain action-value estimates $\hat{Q}^k_{\text{MC}}(c)$ for all chance nodes c that are updated such that the current average is extended with the result of the latest trial. More formally, given a trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$ and with $\mathcal{R}_{t..l}(\theta^k) := \sum_{i=t}^{l} \mathcal{R}(c_i)$, the backup function

$$\hat{Q}^k_{\text{MC}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if } t = \mathcal{H}-1 \\ \hat{Q}^{k-1}_{\text{MC}}(c_t) + \frac{\mathcal{R}_{t..l}(\theta^k) - \hat{Q}^{k-1}_{\text{MC}}(c_t)}{\mathcal{B}^{k-1}(c_t)+1} & \text{otherwise} \end{cases} \tag{6.5}$$

is used to update all visited chance nodes c_t for $t = 0, \ldots, l$.

Example 23. *For the leaf nodes of Figure 6.1, the average result is equal to the immediate reward in that node starting from the trial where the node was visited first, i. e., $\hat{Q}^k_{\text{MC}}(c_5) = 40$ for all $k \geq 1$, $\hat{Q}^k_{\text{MC}}(c_3) = 100$ for all $k \geq 2$, $\hat{Q}^k_{\text{MC}}(c_6) = 10$ for all $k \geq 3$, and $\hat{Q}^k_{\text{MC}}(c_4) = 10$ for all $k \geq 5$. The seven subsequent estimates (without $\hat{Q}^0_{\text{MC}}$) for c_1 are 0 - 100 - 100 - 100 - 55 - 55 - 70 (in that order) and 40 - 40 - 25 - 30 - 30 - 32.5 - 32.5 for c_2 in the first example scenario. The evolution of the action-values estimates $\hat{Q}^k_{\text{MC}}(c_0)$ for chance node c*

is given in Table 6.1. After seven trials, the estimate is $\hat{Q}^k_{\text{MC}}(c_0) = 68\frac{4}{7}$ *and hence still quite far from the true action value* $Q_\star(c_0) = 84$*. This is mostly due to the fact that the average result of the three trials over* c_1 *is* 70 *instead of* 110*, which is owed to the explorative fifth trial in that part of the search tree.*

If MC backups are used, the action-value estimate of a chance node c_t is influenced by each performed trial that visits c_t with the same weight. While this does not matter as long as trials follow the optimal policy, it becomes a problem when trials explore other options as well – our first glimpse at MC backups in Section 5.1 hinted at the fact that this can even lead to suboptimal behavior in the limit. Our theoretical evaluation in Chapter 7 shows that there are nevertheless combinations with other ingredients that are such that the resulting THTS algorithm converges towards the true action values. As we will see, the main idea is that this is the case for action selection strategies that are *greedy in the limit*, i. e., such that (almost) only the optimal policy is simulated after a while.

However, it often takes a prohibitive number of trials until that point is reached in practice. On the other hand, one can observe that most action selection strategies do not switch their behavior all the sudden but resemble the optimal policy in more and more decision nodes with an increasing number of trials. It is therefore a popular idea to give trials that were performed later a *higher weight* and decrease the influence of early trials over time. We allow this by considering a *learning rate* $\eta^k(c)$ for MC backups, which is used to weight the result of the latest trial in c. The resulting backup function

$$\hat{Q}^k_{\text{MC}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if } t = \mathcal{H}-1 \\ \hat{Q}^{k-1}_{\text{MC}}(c_t) + \eta^k(c_t) \cdot \Big(\mathcal{R}_{t..l}(\theta^k) - \hat{Q}^{k-1}_{\text{MC}}(c_t) \Big) & \text{otherwise} \end{cases} \tag{6.6}$$

is equivalent to Equation 6.5 iff the learning rate is set to $\eta^k(c_t) = \frac{1}{\mathcal{B}^k(c_t)+1}$. We call this learning rate the *sample-average* learning rate. It is a well-known fact that MC backups converge in many applications if the learning rate is such that

$$\sum_{k=0}^{\infty} \eta^k(c_t) = \infty, \text{ and}$$

$$\sum_{k=0}^{\infty} \Big(\eta^k(c_t) \cdot \eta^k(c_t) \Big) = \alpha$$

for some finite constant α (see, for instance, Sutton and Barto, 1998). Put simply, the first condition is necessary to guarantee that the learning rate is high enough to overcome a poor initial value or poor decisions in early trials, and the second asserts that the learning rate becomes small enough eventually to allow convergence.

k	1	2	3	4	5	6	7
$\mathcal{R}(\theta_k)$	60	120	30	60	30	60	120
	60	30	30	60	120	60	120
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{MC}}(\mathbf{c_0})$	60	90	70	$67\frac{1}{2}$	60	60	$68\frac{4}{7}$
	60	45	40	45	60	60	$68\frac{4}{7}$
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{MC(0.5)}}(\mathbf{c_0})$	60	100	65	63	52	$54\frac{2}{7}$	$70\frac{5}{7}$
	60	40	35	45	70	$67\frac{1}{7}$	$80\frac{5}{14}$
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{TD}}(\mathbf{c_0})$	60	90	75	$68\frac{3}{4}$	70	$67\frac{1}{12}$	$70\frac{3}{14}$
	60	45	45	$46\frac{1}{4}$	52	$52\frac{1}{12}$	$57\frac{5}{12}$
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{LS\text{-}MC}}(\mathbf{c_0})$	60	60	60	60	60	60	75
	60	45	45	50	50	$52\frac{1}{2}$	66
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{LS\text{-}TD}}(\mathbf{c_0})$	60	60	60	60	60	60	75
	60	45	45	50	50	$52\frac{1}{2}$	60
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{ES\text{-}MC}}(\mathbf{c_0})$	60	90	90	80	80	75	84
	60	45	45	50	$67\frac{1}{2}$	66	75
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{QL}}(\mathbf{c_0})$	60	90	80	75	84	80	$85\frac{5}{7}$
	60	45	50	$52\frac{1}{2}$	66	65	$72\frac{6}{7}$
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{MaxMC}}(\mathbf{c_0})$	60	90	80	75	84	80	$85\frac{5}{7}$
	60	45	50	$52\frac{1}{2}$	84	80	$85\frac{5}{7}$
$\hat{\mathbf{Q}}^{\mathbf{k}}_{\mathbf{PB}}(\mathbf{c_0})$	60	84	84		84		
	60	48	48		84		

Table 6.1: Action-value estimates $\hat{Q}^k(c_0)$ under different backup functions at the end of the k-th trial. The gaps of $\hat{Q}^k_{\text{PB}}(c_0)$ are because the corresponding trials are not performed with PB backups due to the solve labeling procedure.

Learning rates that are determined by *progressive decay* comply with these restrictions: they are such that the learning rate is determined based on a parameter η_{d} (with $0 < \eta_{\text{d}} \leq 1$), the *learning rate decay*, as

$$\eta^k(c_t) = \frac{1}{(\eta_{\text{d}} \cdot \mathcal{B}^k(c_t)) + 1} \tag{6.7}$$

for chance node c, which can be used to model the sample-average learning rate by considering $\eta_{\text{d}} = 1$.[1]

[1] A second parameter η_0, the *initial learning rate*, is often used in place of the constant numerator of $\eta^k(c_t)$. Considering the learning rate decay suffices for our purposes, though.

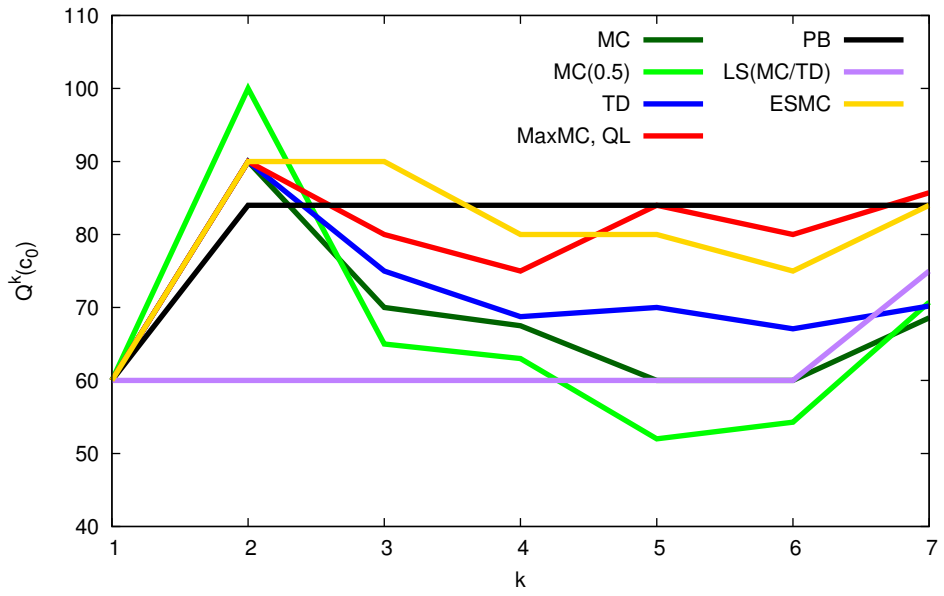

Figure 6.2: Action-value estimates $\hat{Q}^k(c_0)$ of different backup functions at the end of the k-th trial in the first example scenario.

Example 24. *The action-value estimates of a THTS algorithm that uses MC backups with a learning rate that is determined by progressive decay (with $\eta_d = 0.5$) are denoted with $\hat{Q}^k_{\text{MC}(0.5)}$ in our example. It can be observed that the action-value estimates tend to oscillate stronger in the direction of the latest sample than $\hat{Q}^k_{\text{MC}}$. Unlike in MC backups with $\eta_d = 1$, where the relative order of trials is irrelevant, the different relative order of both trial sequences leads to different final action value estimates: the final result is a better estimate of the true value if good decisions are taken later during the trials, as can be seen in the fact that the second example scenario, where the explorative selection in d_3 occurs already in the second instead of the fifth trial, yields a final result that is closer to the true action value of 84.*

6.3.2 Temporal Difference Backups

The introduction of a learning rate that is determined by progressive decay in MC backups is a first step towards a backup function that allows fast convergence towards true action values. However, MC backups that use progressive decay suffer from the problem that it takes several exploitative actions to compensate the error that is caused by an explorative one. In our example scenario, this is most obvious in the fifth trial (the second in the second example scenario), where the explorative selection of $a(c_4)$ leads to a large decrease of the involved action-value estimates. In the first example scenario, this is

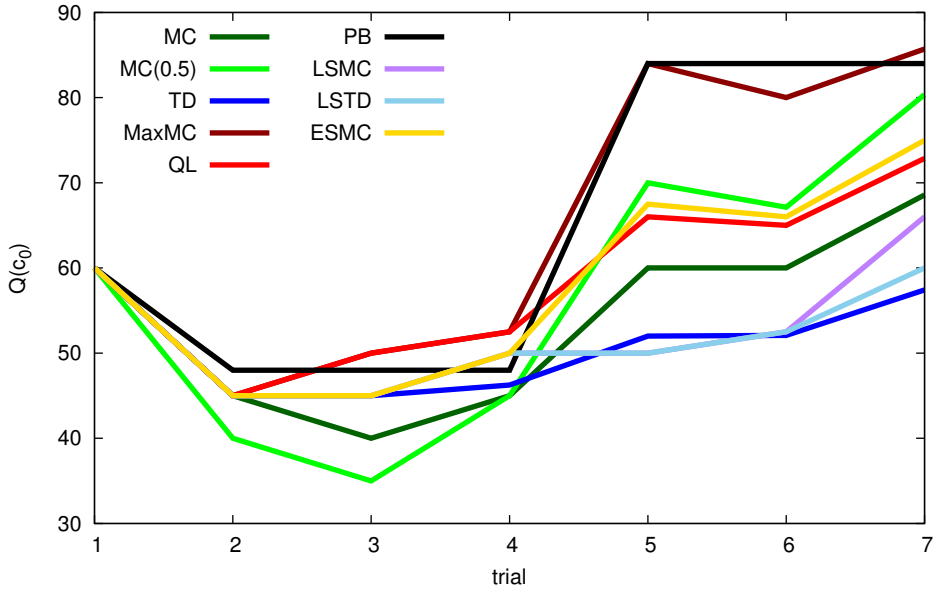

Figure 6.3: Action-value estimates $\hat{Q}^k(c_0)$ of different backup functions at the end of the k-th trial in the second example scenario.

barley compensated by the selection of $a(c_3)$ in the seventh trial, leading to a final estimate that is only little better than the estimate of MC backups with $\eta_d = 1$.

Unfortunately, this is not an artificial case that never occurs in practice: in domains where the initialization is such that the relative order of actions is reasonable[2], it is the case with many action selection strategies that *early and late trials* are such that the optimal action is selected (the early ones due to a good initialization, and the late ones due to informative decisions), while suboptimal actions are selected in the trials in between (due to the necessity to explore at some point). The ability of a backup function to adapt quickly to the latest samples is therefore a two-edged sword: on the one hand, it is desirable that action-value estimates are highly flexible and change quickly towards the results of the latest trials. But on the other hand, if they are too unstable, each explorative action or outcome with low probability can distort the final result significantly. It turns out that a technique that is very popular in the Reinforcement Learning community – Temporal Difference Learning – might provide us with a solution to this problem: like MC backups, their flexibility can be controlled with a learning rate function, but they produce more stable estimates as updates are based on the action-value estimate of the chance node that follows in the current trial instead of the accumulated reward of the current trial.

[2]The heuristic estimates can still be arbitrary bad, only the relative order is of relevance.

Temporal Difference Learning algorithms TD(λ) are typically described in terms of a parameter λ (e.g., Sutton and Barto, 1998). Even though there is no reason not to consider other values for λ, our backup function restricts to the basic version TD(0). *In Online RL*, where TD learning is very popular, it is an important property of TD backups that the update of action value estimates can be performed *right after* an action has been selected for execution and not only when a trial has been finished. As the new information is directly used to update an estimate this away, a better informed decision can be made if the same state is encountered again in the same trial. Furthermore, it allows TD backups to also work in scenarios that are not episodic (i.e., one where the learning process can be restarted from the initial state at any time). However, both of these properties are irrelevant for our needs, as it is impossible to reach the same state more than once in the same trial due to the acyclicity of the MDP. We therefore update action value estimates in the backup phase of k-th trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$ with the function

$$\hat{Q}^k_{\text{TD}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if } t = \mathcal{H}-1 \\ \hat{Q}^{k-1}_{\text{TD}}(c_t) + \eta^k(c_t) \cdot \left(\mathcal{R}(c_t) + \hat{Q}^{k-1}_{\text{TD}}(c_t) \right) & \text{if } \mathcal{H}-1 < t = l \\ \hat{Q}^{k-1}_{\text{TD}}(c_t) + \eta^k(c_t) \cdot \left(\mathcal{R}(c_t) + \hat{Q}^{k}_{\text{TD}}(c_{t+1}) - \hat{Q}^{k-1}_{\text{TD}}(c_t) \right) & \text{otherwise.} \end{cases} \tag{6.8}$$

This has the advantage that collected information is propagated faster while stable estimates are maintained. The TD backup function looks similar to the MC backup function, and it is essentially only the term $\mathcal{R}_{t..l}(\theta^k)$ that has been replaced by $\mathcal{R}(c_t) + \hat{Q}^k_{\text{TD}}(c_{t+1})$. In other words, an action-value estimate $\hat{Q}^k_{\text{TD}}(c_t)$ is no longer updated under consideration of the reward that was achieved in the current trial during and following the visit of c_t, but with the sum of the immediate reward $\mathcal{R}(c_t)$ (which is also contained in $\mathcal{R}_{t..l}(\theta^k)$) and the action-value estimate of the next chance node that was visited in the current trial (which has already been updated in the current trial and hence includes the information of $\mathcal{R}_{t+1..l}(\theta^k)$). As the action-value estimate of the selected successor is more stable than $\mathcal{R}_{t..l}(\theta^k)$, TD backups are more resilient to results that differ vastly from the current estimate, but they also adapt slower to a better policy. Basing updates on the current estimate of successor states instead of using the result of the latest trial is typically known as *bootstrapping*. From a theoretical point of view, there are good reasons both for the usage of MC and TD backups, and we will see in Chapter 7 that there are anytime optimal THTS recipes with both backup functions. However, it is unclear to date if one backup function has a theoretical advantage over the other, and a comparison of the application of TD backups to our example runs to the application of MC backups above also shows a better result in one run and a worse one in the other. We attempt to find an empirical answer on the advantages of both backup functions in Chapter 8.

Example 25. *Let us first observe that* $\hat{Q}^k_{\text{TD}}(c_1) = \hat{Q}^k_{\text{MC}}(c_1)$ *and* $\hat{Q}^k_{\text{TD}}(c_2) = \hat{Q}^k_{\text{MC}}(c_2)$ *for all* k*. The first action-value estimate for* c_0 *where MC and TD backups differ in the first example scenario is after the third trial, where the (comparably bad) trial over* c_6 *influences TD backups less since its update is based on* $\hat{Q}^3_{\text{TD}}(c_2) = 25$ *instead of* $\mathcal{R}(c_2) + \mathcal{R}(c_6) = 10$ *that is used by MC backups. Similarly, in the fifth trial (which includes the bad decision to simulate* $a(c_4)$*) the action-value estimate* $\hat{Q}^5(c_0)$ *even* increases *because the fact that the better outcome to the left was sampled weighs more than the poor decision in* d_3*. The estimates of TD in the first example scenario are therefore always at least as close to* $Q_\star(c_0)$ *as the corresponding estimates that are computed with MC backups.*

However, TD backups suffer from the problem that early backups influence the estimates stronger than later ones (and the order of the trials hence plays a role for TD backups even with the sample-average learning rate) – the poor decision in d_3*, which is taken already in the second trial in the second example scenario, influences TD backups far more than MC, leading to an estimate that is worse after seven trials in that case. It is, however, possible to counteract this effect by using a learning rate, but the influence of the learning rate with the same learning rate decay is lower in TD – the action-value estimates with* $\eta_{\text{d}} = 0.5$*, for instance, lead to* $\hat{Q}^7_{\text{TD(0.5)}}(c_0) \approx 71.79$ *in the first example scenario (which is only* 2.2% *higher than* $\hat{Q}^7_{\text{TD}}(c_0)$ *as opposed to an increase of* 3.1% *between the respective MC backup functions), and* $\hat{Q}^7_{\text{TD(0.5)}}(c_0) \approx 61.88$ *in the second (an increase of* 7.7% *for TD*(0.5) *in comparison to sample-average TD, while the estimate with MC*(0.5) *is* 17.2% *higher than the estimate of sample-average MC).*

6.3.3 Selective Backups

We have seen that both MC and TD backups suffer from the problem that the action-value estimate of a chance node c can be biased by explorative action selections in the subtree rooted at c. We have presented a possible solution with the introduction of a learning rate for MC and TD backups, but the value for the learning rate decay must be selected with care to avoid unstable estimates that oscillate strongly. In this section, we present a simple alternative solution to the problem that exploits the fact that we *know* during the backup phase if we are about to update a node based on decisions that contain explorative actions. *Selective backups* are based on the principle of *separation of concerns* that has been proposed by Feldman and Domshlak (2012), where chance nodes are only updated if the part of the trial that impacts the update has been derived with greedy action selections only. Feldman and Domshlak argue that the fact that the action selection strategy of an algorithm follows two different concerns – exploration and exploitation – should also be reflected in the backup function.

Each trial of the BRUE algorithm and its variants (Feldman and Domshlak, 2013, 2014a,b) starts with a purely explorative phase until a *switching point* is reached that is set following a predefined pattern. All action selections be-

fore the switching point are UNI and all other are greedy, and action-value estimates are only updated for all nodes starting with the chance node at the switching point. In BRUE, action selection and backup function are interdependent and only work in the specific combination. Here, we present a generalization of the BRUE backup function that realizes the separation of concerns concept *and* allows the combination with the other THTS ingredients presented in this thesis. Given a trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$, we define the Boolean flag $\sigma^k(c_t)$ for a chance node c_t in θ^k as

$$\sigma^k(c_t) := \begin{cases} \texttt{true} & \text{if } t = l \\ c_{t+1} \in \mathcal{A}_\star^{k-1}(d_{t+1}) \wedge \bigwedge_{i=t+2}^{l} \sigma^k(c_i) & \text{otherwise.} \end{cases} \tag{6.9}$$

In other words, $\sigma^k(c_t)$ holds for a chance node if it is the last visited node in the trial, or if all decisions following c_t in that trial were due to greedy choices. The role of σ^k is the same as in BRUE: the action-value estimate of a node $\hat{Q}^k(c_t)$ is updated iff $\sigma^k(c_t)$ is `true`, and it remains unaltered otherwise. This way, the concept of separation of concerns can be achieved in combination with any other backup function. In combination with a backup function $\mathfrak{B}$, we can derive the *lazy selective* $\mathfrak{B}$ backup function, where the action-value estimates and backup counters of all visited chance nodes c_t are update according to the rules

$$\hat{Q}^k_{\text{LS-}\mathfrak{B}}(c_t) = \begin{cases} \mathfrak{B}^k(c_t) & \text{if } \sigma^k(c_t) \\ \hat{Q}^{k-1}_{\text{LS-}\mathfrak{B}}(c_t) & \text{otherwise, and} \end{cases} \tag{6.10}$$

$$\mathcal{B}^k(c_t) = \begin{cases} \mathcal{B}^{k-1}(c_t) + 1 & \text{if } \sigma^k(c_t) \\ \mathcal{B}^{k-1}(c_t) & \text{otherwise,} \end{cases} \tag{6.11}$$

where $\mathfrak{B}^k(c_t)$ refers to the *function* of $\mathfrak{B}$ that computes the backup in the k-th trial based on the action-value estimates $\hat{Q}^k_{\text{LS-}\mathfrak{B}}$. Even though any backup function can be used in combination with the separation of concerns that is achieved by lazy selective backup functions, we restrict our analysis in the following to lazy selective Monte-Carlo (LS-MC) and lazy selective Temporal Difference (LS-TD) backups where the update rules that are specified in the Equations 6.6 and 6.8 are used in place of $\mathfrak{B}^k(c_t)$ in Equation 6.10, respectively. In combination with MC backups, for instance, this results in the backup function where all chance nodes c_t that are visited in a trial θ^k are updated according to Equation 6.11 and the rule

$$\hat{Q}^k_{\text{LS-MC}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if } t = \mathcal{H}-1 \\ \hat{Q}^{k-1}_{\text{LS-MC}}(c_t) + \eta^k(c_t) \cdot \left(\mathcal{R}_{t..l} - \hat{Q}^{k-1}_{\text{LS-MC}}(c_t) \right) & \text{if } t \neq \mathcal{H}-1 \text{ and } \sigma^k(c_t) \\ \hat{Q}^{k-1}_{\text{LS-MC}}(c_t) & \text{otherwise.} \end{cases}$$

Note that it is not necessary to check if $t \geq \sigma^k$ in the first case where $t = \mathcal{H}-1$ since Equation 6.9 ensures that the last action in each trial must be

updated. LS-MC is the backup function that resembles the backup function of the BRUE algorithms the most. However, it is slightly more general as LS-MC allows the usage of a learning rate, a concept that is incorporated comparably in BRUE(α) (Feldman and Domshlak, 2014b) where learning is achieved by removing obsolete results from a list of rewards that is maintained for each chance node in the explicit graph.

Example 26. *Recall from Example 22 that the actions $a(d_4)$ and $a(d_5)$ are considered the greedy choices initially in d_3 and d_4, respectively. In the trials over d_2 the result is as desired: the explorative trial where c_6 is selected is ignored in c_2 and $\hat{Q}^k_{\text{LS-MC}}(c_2) = \hat{Q}^k_{\text{LS-TD}}(c_2) = Q_\star(c_2) = 40$ for all $k \geq 1$. However, the trials over d_1 give a completely different picture: since c_4 is considered the better choice initially, it is not until the last considered trial of the first example scenario (which is the third over c_1) that a trial impacts the action-value estimate of c_1 because the selection of c_3 in the second trial is non-greedy but* switches *the relative assessment of c_3 and c_4 such that the fifth trial over c_4 is non-greedy as well (admittedly, it requires poor initialization and action selection ingredients for this case). The result is that only trials from the right branch impact $\hat{Q}^k_{\text{LS-MC}}(c_0)$ for $k \leq 6$, but starting with the seventh trial both LS-MC and LS-TD will only consider optimal action selections.*

Apparently, there is no difference between LS-MC and LS-TD in the first example scenario (which is due to the fact that the action-value estimates of c_1 and c_2 rely only on trials over c_3 and c_5, respectively). The second scenario is worth discussing only because it shows that LS-MC and LS-TD are in fact different backup functions: in the seventh trial, LS-MC makes an update based on the result of the trial $\mathcal{R}(c_0)+100 = 120$, while LS-TD bases its update on $\mathcal{R}(c_0)+\hat{Q}^7_{\text{LS-TD}}(c_1) = 55$.

A potential weakness of backup methods that consider the principles of separation of concerns is that only a comparably small number of trials impacts the action-value estimates of actions that are applicable in the environment's current state (i. e., of successors of the root node d_0). In the first scenario of our running example, already three out of seven trials are ignored in c_0. Even worse, in problems with a larger horizon this will occur a lot more often, as most action selection methods that are not designed to be used in conjunction with Selective backups (such as BRUE) are such that the probability that all decisions in a trial suffix are greedy is negatively correlated to the length of the trial. However, it is a valid argument that the information that is considered is so much more accurate that the sparseness of information is compensated for. Nevertheless, one has to keep in mind that lazy selective backups do by no means only consider trials where all actions are selected according to the optimal policy, but it considers trials where all actions are selected according to the a policy that is *believed* to be the optimal policy *in the current trial*. This is apparent in the second example scenario, where the second trial with the suboptimal choice of $a(c_4)$ leaves its mark on the action-value estimate of c_0.

Since it is anyway not the case that only trials where the optimal policy is simulated are considered it is a promising approach to think about ways how to increase the number of trials that impact action-value estimates. If we take the lazy selective approach to the extreme where all trials are considered in all visited nodes we end up with MC and TD backups as defined earlier in this section. However, there is a large space between that extreme and lazy selective backups.[3] We propose only a single such method that is closely related to lazy selective backups. It is inspired from the behavior of LS-MC and LS-TD in the first scenario of our running example, where both lazy selective backups ignore the result of second trial in the backup of c_1 because the greedy action *at the moment of action selection* has been c_4. However, after having updated c_3 and *before* updating c_1 it is already apparent that $a(c_4)$ is no longer the greedy choice in d_3 in future trials. This means that in the next trial where the same decision is made in c_1, the trial will be considered, and if it is reasonable to do so in the next trial over c_1 there is no reason not to include the information of the current trial right away.

For a given backup function $\mathfrak{B}$ we therefore define *eager selective* backups as the backup function that is identical to lazy selective backups except for the fact that the switching point σ^k is calculated in the k-th trial as

$$\sigma^k(c_t) := \begin{cases} \texttt{true} & \text{if } t = l \\ c_{t+1} \in \mathcal{A}_\star^{k-1}(d_{t+1}) \cup \mathcal{A}_\star^{k,t+1}(d_{t+1}) \wedge \bigwedge_{i=t+2}^{l} \sigma^k(c_i) & \text{otherwise,} \end{cases}$$

Please observe that the action-value estimate of chance node c_{t+1} in the k-th trial has already been performed when $\sigma^k(c_t)$ is computed, and $\mathcal{A}_\star^{k,t+1}(d_{t+1})$ therefore refers to the set of greedy actions after the update.

Example 27. *We only provide results for ES-MC backups in Table 6.1 and Figure 6.2 and 6.3 since ES-TD backups behave equally in the first example scenario and differ only slightly in the second. The most important difference is that ES-MC backups consider the second trial over c_3 in c_1 as the computation of $\hat{Q}^2_{\text{ES-MC}}(c_3)$ makes c_3 the greedy choice in d_3 in future trials. It is a promising sign that the final action-value estimate in c_0 is equal to $Q_\star(c_0) = 84$, but the result of the second example scenario shows that this is (of course) not necessarily the case.*

6.3.4 Q-Learning Backups

With the family of selective backup functions, we have presented a first solution to the problem that explorative actions have the potential to bias action-value estimates disproportionately for many trials. Selective backups achieve this by ignoring parts of trials that are not accomplished under the greedy policy. However, like MC and TD backups, they are on-policy methods that

[3] It is also possible to consider even less trials in node updates. However, we do not believe that is a promising idea and do hence not pursue it any further.

base their backup only on the result of the policy that was simulated in the latest trial. A popular alternative in RL are off-policy methods that are able to simulate one policy but learn from another one. That way, it becomes possible to explore in the simulation but base backups directly only on the policy that is believed to be optimal. In THTS algorithms, we achieve this by taking the state-value estimates of decision nodes d – which correspond to the value of the greedy policy in d according to Equation 6.2 – into account. All backup functions that are discussed in the remainder of this section are off-policy methods. The equivalent for TD backups that incorporates this idea is known under the name *Q-Learning* in the literature (Watkins, 1989), where the dependence on the action-value of the next chance node in TD backups is replaced by a dependence on the state-value of the direct decision node successor. The resulting QL-backup function computes its chance node estimates with

$$\hat{Q}^k_{\mathrm{QL}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if } t = \mathcal{H}-1 \\ \hat{Q}^{k-1}_{\mathrm{QL}}(c_t) + \eta^k(c_t) \cdot \Big(\mathcal{R}(c_t) - \hat{Q}^{k-1}_{\mathrm{QL}}(c_t) \Big) & \text{if } \mathcal{H}-1 < t = l \\ \hat{Q}^{k-1}_{\mathrm{QL}}(c_t) + \eta^k(c_t) \cdot \Big(\mathcal{R}(c_t) + \hat{V}^k_{\mathrm{QL}}(d_{t+1}) - \hat{Q}^{k-1}_{\mathrm{QL}}(c_t) \Big) & \text{otherwise} \end{cases} \tag{6.12}$$

for all chance nodes c_t that are encountered in a trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$.

Example 28. *The interesting trials in the first example scenario are the third and fifth where suboptimal actions are selected in d_4 and d_3, respectively. In contrast to TD, which bases its update on the reward of 10 that is obtained when c_6 is visited, QL exploits its knowledge that there is a better choice and bases its update on c_5 instead. QL is hence equivalent to a version of TD that* pretends *in the third trial that c_5 has been selected instead of the suboptimal c_6. The same behavior can be observed in the fifth trial, where QL effectively uses a trial over c_3 (which is known to be the max child of d_3) rather than c_4. Interestingly, after the fifth trial (when all trajectories in the AND/OR Tree have been simulated at least once), the decision of the action selection strategy becomes irrelevant for the backup function, since QL will always use the information of the trials over c_3 and c_5 in its updates.*

6.3.5 MaxMonte-Carlo Backups

When we compare the result of QL-backups in the first and second example scenario, it is apparent that the fact that the poor decision in d_3 of simulating c_4 comes *prior* to the right decision of selecting c_3 is reflected in the action-value estimates of c_0 forever (hence the lower estimates in the second scenario). The reason is that QL backups – as well as MC and TD backups – compute new action-value estimates by *altering* the old value with an *error* estimate that is given as the difference of the result in the current trial and

the old value. We have already seen that this is different in the Full Bellman backup function, where the old estimate is not part of the computation a new estimate, but only the estimates of the successor nodes. Such an approach is typical for Dynamic Programming algorithms. We have adopted the idea of DP in our original work on THTS (Keller and Helmert, 2013) and combined it with the principles of MC backups. In the *MaxMonte-Carlo* (MaxMC) backup function, decision nodes are updated by maximizing over all successor nodes, while action-value estimates are computed as the sum over the state-value estimates over all successors, each weighted proportionally to the number of times it has been visited. Formally, all chance nodes c_t that are encountered in a trial $\theta^k = (d_0, c_0, \ldots, d_l, c_l)$ are updated with MaxMC backups by applying

$$\hat{Q}^k_{\text{MaxMC}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if succ}^k(c_t) = \varnothing \\ \mathcal{R}(c_t) + \frac{\sum_{d \in \text{succ}^k(c_t)} \mathcal{B}^k(d) \cdot \hat{V}^k_{\text{MaxMC}}(d)}{\sum_{d \in \text{succ}^k(c_t)} \mathcal{B}^k(d)} & \text{otherwise.} \end{cases} \tag{6.13}$$

MaxMC backups are not only an off-policy backup function (since state-value estimates are considered in the updates of $\hat{Q}^k_{\text{MaxMC}}(c_t)$), but both kinds of nodes are also updated with DP methods where estimates are based on the estimates of successor nodes.

Example 29. *We have seen that QL backups overcome the problem of exploratives actions by pretending in their backups that each trial has been performed under application of the current greedy policy. However, if the greedy policy is inaccurate in a trial, the error of that – in hindsight explorative – action application impacts the estimates forever (albeit with a decreasing weight). An example for this effect is the second example scenario, where the action-value estimate is biased by the seemingly greedy decision in the second trial that turns out to be explorative in the fifth. This is different with MaxMC backups: as action-value estimates are not altered by closing the gap to the latest result but re-computed based on the estimates of all successor nodes, MaxMC backups have the ability to "alter the past" and pretend that not only the current trial is under the current greedy policy, but also all former trials. In the fifth trial of the second example scenario, it is therefore the case that* $\hat{Q}^5_{\text{MaxMC}}(c_0)$ *jumps from* 52.5 *to* 84*, as MaxMC realizes that the second trial over* c_4 *can be treated as if it were a trial over* c_3*. That way, MaxMC backups yield the (quite accurate) result of QL backups in both example scenarios.*

Another analogy to MC backups is the fact that, among all present backup functions, only MC backups (with sample-average learning rate) and MaxMC backups are such that the relative order of the trials is irrelevant – when the same set of trials has been sampled, all estimates will be the same regardless of the order.

6.3.6 Partial Bellman Backups

With Full Bellman backups, we have already seen a way how the Bellman optimality equation can be turned into a backup function. However, we have also seen that it is impossible to compute action-value estimates by accumulating over all outcomes as explicating all successors of a state-action pair is not always possible. To turn the Bellman optimality equation into a backup function that also works in the MDPs that are considered in this thesis, we are interested in a backup function that is equivalent to Full Bellman backups in the limit but allows the computation of a backup even in the presence of a large number of outcomes. We have developed the Partial Bellman (PB) backup function (Keller and Helmert, 2013) which meets these requirements:

$$\hat{Q}^k_{\text{PB}}(c_t) = \begin{cases} \mathcal{R}(c_t) & \text{if succ}^k(c_t) = \varnothing \\ \mathcal{R}(c_t) + \frac{\sum_{d \in \text{succ}^k(c_t)} \mathbb{P}_{\mathcal{T}}[d|c_t] \cdot \hat{V}^k_{\text{PB}}(d)}{\sum_{d \in \text{succ}^k(c_t)} \mathbb{P}_{\mathcal{T}}[d|c_t]} & \text{otherwise.} \end{cases} \tag{6.14}$$

PB backups extrapolate missing estimates of outcomes that are not explicated by assuming that they are consistent with the estimates of the outcomes we have seen so far. Unlike all other backup functions discussed so far, PB backups achieve this behavior by using the declarative model of the MDP which provides the transition probabilities. While this limits its use to DMPlan scenarios, it has the advantage that it is possible to *label states as solved*. PB backups replace the update function for the solve label that is given in Equation 6.4 by an update procedure that sets

$$\varphi(d) = \bigwedge_{c \in \mathcal{A}(d)} \varphi(c)$$

$$\varphi^k(c) = \begin{cases} \texttt{true} & \text{if } c \in N_\star \\ \texttt{false} & \text{if } c \notin N_\star \text{ and succ}^k(c) \neq \text{succ}(c) \\ \bigwedge_{d \in \text{succ}^k(c)} \varphi(d) & \text{otherwise} \end{cases}$$

for all decision nodes $d \in \text{D}$ and chance nodes $c \in \text{C}$ that are visited in the k-th trial.

Example 30. *Due to the solve labeling procedure of PB backups, trials where a solved node is selected would never happen in the THTS framework and the corresponding entries are hence omitted. In the third trial of the first example scenario, PB backups label d_4 as solved and, in turn, also c_2 and d_2 since all of their successors are both explicated and solved (c_5 is labeled as solved in the first trial, and c_6 in the third). The fourth and the sixth trial are therefore such that they* cannot *provide new information and they will not happen this way in a THTS algorithm. After the fifth trial (the fourth for a THTS algorithm with solve labeling), the left subtree is labeled as solved as well, and the action-value estimate in chance node c_0 is equal to the true action value $Q_\star(c_0)$.*

6.4 Trial Length

The trial length is the ingredient that determines if the expansion phase of a trial ends and the backup phase begins before a leaf node has been encountered. Formally, it is a function that maps the current system state (which is given by the partial trial and the explicit graph) to a decision if the trial is stopped (`true`) or not (`false`).

Definition 22 (Trial Length Component). *A* **trial length component** *is a Boolean function $\mathfrak{T}$ over a trial $\theta^{k,t} = (d_0, c_0, \ldots, d_t, c_t)$ and an explicit graph $\mathcal{G}^{k,t}$.*

We only use a single (yet configurable) trial length component in this thesis, the *expansion count* trial length component, which is defined by

$$\mathfrak{T}^{\nu}_{\text{EC}}(\mathcal{G}^{k,t}, (d_0, c_0, \ldots, d_t, c_t)) = \begin{cases} \texttt{true} & \text{if } |\mathrm{D}^{[k/k-1]}| = \nu \\ \texttt{false} & \text{otherwise,} \end{cases}$$

where $\mathrm{D}^{[k/k-1]}$ is the set of decision nodes that were expanded in the k-th trial and $\nu \in \mathbb{N}^+$ is a parameter that gives the number of decision nodes that must have been expanded for the expansion count trial length component to end the trial.

The expansion count trial length component suits our needs even though it is based on a simple concept. However, by adjusting its parameter ν, the expansion count trial length component allows the description of the two most popular trial lengths that are used in well-known algorithms: on the one hand, it models the trial length of Trial-based Real Time Dynamic Programming (Barto et al., 1995) and its variants (e.g., Bonet and Geffner, 2003; McMahan et al., 2005; Smith and Simmons, 2006) which *never* stop a trial before a leaf node is encountered by setting $\nu = \mathcal{H}$. And on the other, the expansion count trial length component can be used with $\nu = 1$ in a THTS recipe that uses the trial length of the popular *global search* algorithm AO* or the Monte-Carlo Tree Search algorithm UCT, where only a single node is added to the explicit graph before the collected information is propagated.

6.5 Action Selection

We have already introduced most action selections that are described in the following in Section 5.1 in the context of the Multi-Armed Bandit problem. Here, we generalize the action selection methods to their use in THTS algorithms, where they have the role of deciding during the expansion phase which action (or chance node) is selected in a state (or decision node) to continue the traversal of the tree. An alternative way to look at the action selection strategy is to describe it as a non-stationary, incomplete policy of the

MDP. In combination with the outcome selection that has been discussed in Section 6.2, action selection is the ingredient that decides ultimately at which node the explicit graph is extended. The only condition that must be met by an action selection strategy of a THTS algorithm is that only successor nodes that are not labeled as solved are selected (which is only relevant if the backup function supports solve labeling and is hence restricted to combinations with PB backups in this thesis). We denote the set of all successor nodes of an expanded decision node d that are not labeled as solved in the k-th trial with

$$\mathcal{A}^k_{\neg\varphi}(d) := \{c \in \mathcal{A}(d) \mid \neg\varphi^{k-1}(c)\}$$

and assume in the following without loss of generality that $|\mathcal{A}^0_{\neg\varphi}(d)| > 1$ (the set cannot be empty initially as dead-ends are prohibited in our setting, and if $\mathcal{A}^0_{\neg\varphi}(d)$ contains only a single action that action can be selected without further deliberation). Furthermore, we denote the set of (unsolved) *greedy decision* in the k-th trial with

$$\mathcal{A}^k_\star(d) := \operatorname*{arg\,max}_{c \in \mathcal{A}^k_{\neg\varphi}(d)} \hat{Q}^{k-1}(c),$$

and the *normalized action-value estimates* at the end of the k-th trial of a successor node $c \in \mathcal{A}^k_{\neg\varphi}(d)$ of d as

$$\tilde{Q}^k(d,c) := \frac{\hat{Q}^k(c) - \min_{c' \in \mathcal{A}(d)} \hat{Q}^k(c')}{\max_{c' \in \mathcal{A}(d)} \hat{Q}^k(c') - \min_{c' \in \mathcal{A}(d)} \hat{Q}^k(c')}.$$

Definition 23 (Action Selection). *An* **action selection** *is a function $\mathfrak{A}$ that maps a trial $\theta^{k,t} = (d_0, c_0, \ldots, c_{t-1}, d_t)$ with $\mathcal{A}^k_{\neg\varphi}(d_t) \neq \varnothing$ and an explicit graph $\mathcal{G}^{k,t}$ to a probability distribution over $\mathcal{A}^k_{\neg\varphi}(d_t)$.*

In the following, we denote the probability that an action selection strategy $\mathfrak{A}$ selects chance node $c \in \mathcal{A}_{\neg\varphi}(d)$ in trial $\theta^{k,t} = (d_0, c_0, \ldots, c_{t-1}, d_t)$ with

$$\mathbb{P}_{\mathfrak{A}}[c \mid d_t] := \mathbb{P}[c_t = c].$$

After a chance node c (or, equivalently, the action represented by c) has been drawn from the probability distribution according to the probabilities that are given by $\mathfrak{A}$, the selection counter $\mathcal{L}^k(c)$ is increased by one and the trial proceeds with a visit of c.

6.5.1 Greedy Action Selection

We include the greedy action selection strategy (GRD) here only as a discussion of action selection strategies feels incomplete without it. We have met GRD in the previous chapter in two places: as part of the Explore-then-Exploit

strategy and as part of the ϵ-greedy strategy (which is discussed below). Following the greedy policy in a decision node d corresponds to acting exploitative in d in most settings, and it is usually combined with an alternative action selection strategy that takes care of exploration. We will see in Chapter 7 that there is no combination of ingredients that is anytime optimal with GRD in this thesis (but there are exceptions to this, e. g., by using an optimistic initialization). For the sake of completeness, GRD is the action selection strategy that samples one of the unsolved greedy successor nodes $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$ of decision node d_t uniformly at random in the t-th decision of the k-th trial, i. e., the greedy action selection is such that for all $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$:

$$\mathbb{P}_{\text{GRD}}\left[c \mid d_t\right] = \frac{1}{|\mathcal{A}^k_\star(d_t)|}.$$

6.5.2 Uniform

The opposite extreme of the purely exploitative GRD action selection is an action selection strategy that explores in every step. The uniform action selection strategy (UNI) matches this description the closest, as the successor $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$ of d_t is selected in the t-th decision of the k-th trial with probability uniformly at random, i. e., $\mathfrak{A}_{\text{UNI}}$ is such that for all $c \in \mathcal{A}(d_t)$:

$$\mathbb{P}_{\text{UNI}}\left[c \mid d_t\right] = \frac{1}{|\mathcal{A}(d_t)|}.$$

6.5.3 ϵ-Greedy

A classical action selection strategy that balances exploration and exploitation with the help of a single parameter ϵ with $0 < \epsilon < 1$ is the famous ϵ-greedy action selection (ϵ-G). In the k-th trial in decision node d, it behaves like the uniform action selection with probability ϵ and like the greedy one otherwise, i. e., such that

$$\mathbb{P}_{\epsilon\text{-G}}\left[c \mid d_t\right] = \begin{cases} \frac{\epsilon}{|\mathcal{A}^k_{\neg\varphi}(d_t)|} + \frac{1-\epsilon}{|\mathcal{A}^k_\star(d_t)|} & \text{if } c \in \mathcal{A}^k_\star(d_t) \\ \frac{\epsilon}{|\mathcal{A}^k_{\neg\varphi}(d_t)|} & \text{otherwise} \end{cases}$$

for all $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$. The parameter ϵ allows to adjust the similarity of ϵ-G to GRD on the one hand and to UNI on the other: the closer ϵ is to one, the more does ϵ-G resemble uniform action selection, and both are identical for $\epsilon = 1$; and the closer ϵ is to zero, the greedier it gets, and ϵ-G with $\epsilon = 0$ is equivalent to the greedy action selection strategy.

We have seen in Section 5.1, that the theoretical properties of using ϵ-G are weak in all presented optimization problems. While a study of Kuleshov and Precup (2010) has shown that it is able to keep up with the state-of-the-art in MABs, we have also presented an example that shows that ϵ-G leads to

suboptimal decision if combined with MC backups. For anytime optimality, it is often required that action value estimates converge to the optimal action values. We will see in the next chapter that this imposes requirements on the action selection strategy as well, so we consider three additional ϵ-greedy action selections where the probability for exploration is computed dynamically rather than provided by the constant parameter ϵ. In *decreasing ϵ-greedy* action selection, each chance node $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$ is selected with probability

$$\mathbb{P}_{\mathfrak{A}}\left[c \mid d_t\right] = \begin{cases} \frac{\epsilon^k(d_t)}{|\mathcal{A}^k_{\neg\varphi}(d_t)|} + \frac{1-\epsilon^k(d_t)}{|\mathcal{A}^k_\star(d_t)|} & \text{if } c \in \mathcal{A}^k_\star(d_t) \\ \frac{\epsilon^k(d_t)}{|\mathcal{A}^k_{\neg\varphi}(d_t)|} & \text{otherwise,} \end{cases}$$

where $\mathfrak{A}$ is described by the *ϵ-decay function* that is used to compute $\epsilon^k(d)$:

$$\epsilon^k(d) = \frac{1}{\mathcal{L}^k(d)} \quad \text{in linear decreasing } \epsilon\text{-greedy } (\epsilon_{\text{LIN}}\text{-G}),$$
$$\epsilon^k(d) = \frac{1}{(\mathcal{L}^k(d))^\gamma} \quad \text{in root-valued decreasing } \epsilon\text{-greedy } (\epsilon_{\text{RT}}\text{-G}), \text{ and}$$
$$\epsilon^k(d) = \frac{1}{\ln(\mathcal{L}^k(d))} \quad \text{in logarithmic decreasing } \epsilon\text{-greedy } (\epsilon_{\text{LOG}}\text{-G}),^4$$

The first option, *linear decreasing ϵ-greedy*, is similar to the suggestions for a decay schema for ϵ-G of Singh et al. (2000) and Auer et al. (2002). However, their schemata aim for the optimization of cumulative regret in the Multi-Armed Bandit problem where fast decay is desirable. However, cumulative regret minimization is an optimization criterion that is both irrelevant in the results of MAB applications in practice (Kuleshov and Precup, 2010) and theoretically harmful in the MAB when a generative model is available (Bubeck et al., 2009). Even though the results of Bubeck et al. (2009) tempt to conjecture that an action selection that leads to high cumulative regret in MABs is a good choice for an action selection in MDPs with generative or declarative model, this is not the case because exploitation in a single decision node does not equal exploitation in the whole trial (which is the same thing in the MAB) – in MDPs, early exploitation *enables* exploration in "desired" parts of the AND/OR tree. We therefore consider *root-valued decreasing ϵ-greedy* (with a parameter γ, $0 < \gamma < 1$ that allows to use smaller and larger roots) and *logarithmic decreasing ϵ-greedy* in addition to ϵ_{LIN}-G, and analyze the influence of the different ϵ-decay functions on the convergence of THTS algorithm theoretically in Chapter 7 and on the simple regret empirically in Chapter 8.

[4] It holds that $\ln\left(\mathcal{L}^k(d)\right) > 0$ for all $d \in \mathrm{D}$ because $\mathcal{L}^k(d)$ is monotonically increasing with increasing k and $\mathcal{L}^0(d) = |\mathcal{A}(d)| \cdot \chi$ with $\chi \geq 1$ and $|\mathcal{A}(d)| \geq 2$.

6.5.4 Boltzmann Exploration

A weakness of all proposed ϵ-greedy action selections is that they use the action-value estimates only to separate the set of unsolved successor nodes in a set of greedy and a set of non-greedy actions, but then they treat all non-greedy actions equivalently regardless of the difference to the best action(s). However, if there are promising choices with estimates that are close to the estimate of the best action and other that are significantly worse, it appears reasonable to select promising actions with higher probability. This describes the main idea of *Soft-Max* action selections (Sutton and Barto, 1998), whose most popular member is Boltzmann Exploration (BE). In BE, the probability of selecting each chance node $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$ in the k-th trial is

$$\mathbb{P}_{\text{BE}}[c \mid d_t] = \frac{\mathcal{Q}^k(d_t, c)}{\sum_{c' \in \mathcal{A}^k_{\neg\varphi}(d_t)} \mathcal{Q}^k(d_t, c')},$$

where $\mathcal{Q}^k(d, c) := e^{\frac{\tilde{Q}^{k-1}(d,c)}{\tau}}$. The degree of exploration can be adapted by a adjusting the *temperature* parameter $\tau > 0$. Again, the extreme cases for τ turn BE into uniform and greedy action selection: with $\tau \to \infty$, BE turns into UNI action selection, and with $\tau \to 0$ it resembles GRD more and more. Another commonality with ϵ-G is that it is possible to adapt the temperature τ dynamically over time. In the *Boltzmann Exploration with Decreasing Temperature* (BE-DT) action selection of Singh et al. (2000) that is also considered in this work, the parameter $\tau^k(d)$ is computed as

$$\tau^k(d) = \frac{1}{\ln\left(\mathcal{L}^k(d)\right)}.$$

6.5.5 Upper Confidence Bounds

Even though BE fixes the mismatch between the probability that an explorative action is selected and its action-value estimate that is inherent to all variants of ϵ-greedy action selection, it still has a fundamental flaw: it does not take into account *how uncertain* the action-value estimate is. An example for this is given in Figure 6.4. It depicts a situation before (on the left) and after (on the right) the k-th trial as Gaussians that describe the available information for the three chance nodes c_{blue}, c_{red}, and c_{green}. (The action-value estimate that is maintained by all our backup functions is an estimate of the mean, and keeping track of the variance would be trivial and would allow the derivation of the depicted normal distributions.) If BE is faced with the depicted situation, it selects c_{blue} with the highest probability as it has the highest action-value estimate (since its mean is the rightmost one). However, the certainty about c_{blue} is also the highest (its mean has the highest probability) and the probability that the action-value estimate changes significantly with additional information is the lowest, so it might very well be better to

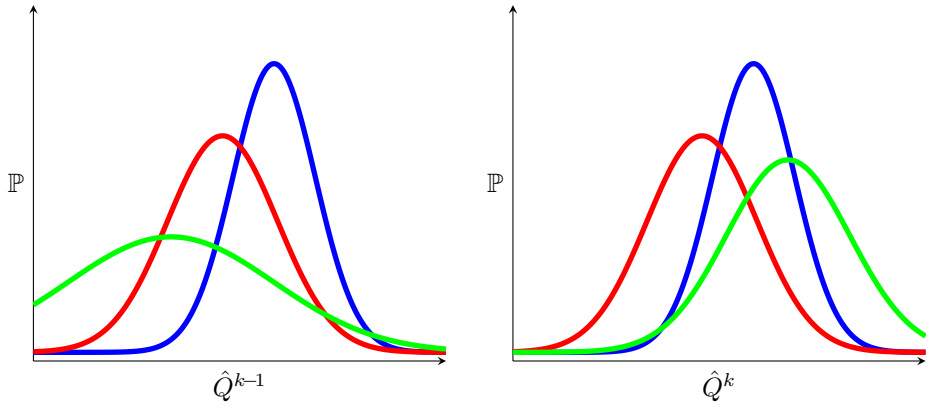

Figure 6.4: Example information of an action selection before (left) and after (right) the decision to select c_{green} is taken.

select the highly uncertain green chance node. If the result of the trial is such that distribution on the right of Figure 6.4 is the new believe, the selection of the uncertain chance node c_{green} has payed off since it has the highest action-value estimate of the three options now.

UCB1 (Auer et al., 2002) is an action selection strategy that incorporates an uncertainty estimate in its decisions that is based on the number of times each chance node has been selected. It maintains an *upper confidence bound* on the action-value estimate by computing a value $\hat{U}^k_{\text{UCB1}}(c)$ for each successor node c of d_t that is such that $Q_\star(c) \leq \hat{Q}^k(c) + \hat{U}^k_{\text{UCB1}}(c)$ with high probability. We denote with

$$\mathcal{A}^k_{\text{UCB1}}(d) := \operatorname*{arg\,max}_{c \in \mathcal{A}^k_{\neg\varphi}(d)} \left(\tilde{Q}^{k-1}(d,c) + \sqrt{\frac{2 \cdot \ln(\mathcal{L}^{k-1}(d))}{\mathcal{L}^{k-1}(c)}} \right)$$

the set of (unsolved) successor nodes of decision node d with highest upper confidence bound. The probability of selecting each chance node $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$ in the k-th trial with the UCB1 action selection strategy is

$$\mathbb{P}_{\text{UCB1}}\left[c \mid d_t\right] = \begin{cases} \frac{1}{|\mathcal{A}^k_{\text{UCB1}}(d_t)|} & \text{if } c \in \mathcal{A}^k_{\text{UCB1}}(d_t) \\ 0 & \text{otherwise.} \end{cases}$$

Even though the logarithmic cumulative regret in the Multi-Armed Bandit problem that led to the specific form of UCB1 is irrelevant in MDP planning with generative or declarative model (and even obstructive in MABs with declarative model), algorithms that use UCB1 have achieved convincing results in a wide variety of non-MAB applications. Examples for the successful application of UCT (Kocsis and Szepesvári, 2006) – the Monte-Carlo Tree Search algorithm that is based on UCB1 action selection – includes work in

MDPs (e.g., Balla and Fern, 2009; Eyerich et al., 2010; Keller and Eyerich, 2011) single-player games (e.g., Bjarnason et al., 2009), multi-player games (e.g., Finnsson and Björnsson, 2008; Gelly and Silver, 2011) or POMDPs (e.g., Silver and Veness, 2010; Geißer et al., 2014), to give just a small and by far not exhaustive selection.

Root-valued Upper Confidence Bounds

We believe that the success story of UCT is mostly due to the fact that the UCB1 formula is sufficiently exploitative close to the root node and sufficiently explorative far from it to allow exploration in relevant parts of the explicit graph. Of course, this has not been the *intention* of the developers of UCB1 (who were interested in MABs and cumulative regret and not in MDPs and simple regret), but it is merely a byproduct of the UCB1 formula. In our framework, this incurs the consequence that UCB1 performs poorly if the initial guidance is poor, as it turns into the greedy action selection too quickly. UCB1 can therefore commit to a suboptimal decision too early, which is a possible explanation for the fact that the optimization of parameters is an important part of many of the aforementioned successful applications of UCB1.

The most important consequence of this insight is that, while it is necessary that the best action is selected exponentially more often to achieve logarithmic cumulative regret in MABs, there is no theoretical justification to do so in MDPs with access to a model of the environment. On the contrary, the results of Bubeck et al. (2011) indicate that a higher degree of exploration might lead to a lower simple regret. We hence present a variant of UCB1 in the following. In line with the findings of Bubeck et al. (2011), we derive a method where the probability that $Q_\star(c) \leq \hat{Q}^k(c) + \hat{U}^k(c)$ is even smaller than with UCB1, which can only be achieved with a higher incentive for exploration. The *root-valued UCB* (RT-UCB) action selection achieves this by selecting a chance node in decision node d_t from the set

$$\mathcal{A}^k_{\text{RT-UCB}}(d_t) := \operatorname*{arg\,max}_{c \in \mathcal{A}^k_{\neg\varphi}(d_t)} \left(\hat{Q}^{k-1}(c) + \sqrt{\frac{2 \cdot \sqrt{\mathcal{L}^{k-1}(d_t)}}{\mathcal{L}^{k-1}(c)}} \right)$$

uniformly at random such that

$$\mathbb{P}_{\text{RT-UCB}}\left[c \mid d_t\right] = \begin{cases} \frac{1}{|\mathcal{A}^k_{\text{RT-UCB}}(d_t)|} & \text{if } c \in \mathcal{A}^k_{\text{RT-UCB}}(d_t) \\ 0 & \text{otherwise.} \end{cases}$$

for all chance nodes $c \in \mathcal{A}^k_{\neg\varphi}(d_t)$. Apparently, the only difference is that the logarithmic dependence on the selection counter of d in UCB1 has been replaced by a dependence in the square root of $\mathcal{L}^{k-1}(d)$. Before we have a look at the behavior of RT-UCB in MDPs, we analyze the ratio between exploration and exploitation of this novel action selection if it is applied to a Multi-Armed Bandit problem.

Theorem 5. *With RT-UCB action selection, each suboptimal arm a is selected at most*

$$\mathbb{E}\left[\mathcal{L}^k(a)\right] \leq \frac{8 \cdot \sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2} + 1 + \frac{\pi^2}{3}$$

times in expectation in an MAB problem with k trials and optimal arm $a^\star$.

Proof: We start the proof of Theorem 5 by using the Chernoff-Hoeffding inequality to bound the probability that the action-value estimate $\hat{Q}^k(a)$ in the k-th trial differs by more than $\hat{U}^k(a)$ from $Q_\star(a)$ by

$$\mathbb{P}\left[\hat{Q}^k(a) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a)}} \leq Q_\star(a)\right] \leq e^{-4 \cdot \sqrt{k}}. \tag{6.15}$$

Let us now assume that a suboptimal arm a is selected in the $(k+1)$-th trial. By definition of the RT-UCB action selection, this implies that

$$\hat{Q}^k(a) + \hat{U}^k(a) \geq \hat{Q}^k(a') + \hat{U}^k(a')$$

for all other arms $a' \neq a$ and in particular also for all optimal arms $a^\star$:

$$\hat{Q}^k(a) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a)}} \geq \hat{Q}^k(a^\star) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a^\star)}}. \tag{6.16}$$

There are several possible explanations for why Equation 6.16 holds in the $(k+1)$-th trial. The first is a consequence of the observation that all confidence bounds are large as long as the number of trials is low simply because there haven't been many opportunities to play an arm. In the initial phase of the interaction between agent and environment some exploration is unavoidable, and $\hat{Q}^k(a) + \hat{U}^k(a)$ can be larger than $\hat{Q}^k(a^\star) + \hat{U}^k(a^\star)$ even if the algorithm had access to the true action-values of all arms

$$Q_\star(a) + 2 \cdot \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a)}} \geq Q_\star(a^\star),$$

which can be transformed with simple arithmetic to

$$\mathcal{L}^k(a) \leq \frac{8 \cdot \sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2}.$$

This implies that, as soon as arm a has been selected at least

$$\mathcal{L}^{\min}(a) := \frac{8 \cdot \sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2} + 1 \tag{6.17}$$

times, all confidence bounds are tight enough that the upper bound of a and the lower bound of the estimate for the optimal action $a^\star$ overlap *only* because

the action-value estimates are far from the true action-values. Let us denote with $k_{\min}$ the minimal number of trials that is necessary in expectation that arm a has been played $\mathcal{L}^{\min}(a)$ times, i. e.,

$$k_{\min}(a) := \min_k \left(\mathbb{E}\left[\mathcal{L}^k(a) \right] = \mathcal{L}^{\min}(a) \right).$$

So far, we have seen that the sequence of trials can be divided into two phases: an initial phase with $k_{\min}(a)$ trials where a is selected $\mathcal{L}^{\min}(a)$ times, and a second phase (that is discussed below) where a is selected because Equation 6.16 holds. The expected number of pulls of a suboptimal arm a in $k > k_{\min}$ trials is hence

$$\mathbb{E}\left[\mathcal{L}^k(a)\right] = \mathcal{L}^{\min}(a) + \sum_{t=k_{\min}}^{k} \mathbb{I}\left[\hat{Q}^{t-1}(a) + \hat{U}^{t-1}(a) \geq \hat{Q}^{t-1}(a^\star) + \hat{U}^{t-1}(a^\star)\right] \tag{6.18}$$

$$\leq \mathcal{L}^{\min}(a) + \sum_{t=k_{\min}}^{k} \sum_{s=1}^{t} \sum_{u=k_{\min}}^{t} \mathbb{I}\left[\hat{Q}^{u}(a) + \hat{U}^{u}(a) \geq \hat{Q}^{s}(a^\star) + \hat{U}^{s}(a^\star)\right] \tag{6.19}$$

The inequality holds as all terms in Iverson brackets that are contained in Equation 6.18 are also contained in Equation 6.19, and there several additional comparisons that might hold in Equation 6.19.

Let us now have a look at the second phase in the sequence of trials where $\mathcal{L}^k(a) \geq \mathcal{L}^{\min}(a)$. As arm a is selected only if its upper confidence bound is larger than the lower confidence bound of $a^\star$, it is either the case that the current action-value estimate $\hat{Q}^k(a^\star)$ of the optimal action $a^\star$ is an underestimation of $Q_\star(a^\star)$

$$\hat{Q}^k(a^\star) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a^\star)}} < Q_\star(a^\star) \tag{6.20}$$

or if the current action-value estimate $\hat{Q}^k(a)$ of the selected action a is a poor estimate of the true action value of a:

$$\hat{Q}^k(a) > Q_\star(a^\star) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a)}}. \tag{6.21}$$

We can now use Equation 6.15 and the fact that $e^{-4 \cdot \sqrt{k}} \leq e^{-4 \cdot \ln(k)} = k^{-4}$ since $\sqrt{k} \geq \ln(k)$ for all $k \geq 1$ to derive an upper bound for the probability that Equation 6.20 is indeed the explanation that a has been selected in the $(k+1)$-th trial, which yields

$$\mathbb{P}\left[\hat{Q}^k(a^\star) + \sqrt{\frac{2 \cdot \sqrt{k}}{\mathcal{L}^k(a^\star)}} < Q_\star(a^\star) \right] \leq e^{-4 \cdot \sqrt{k}} \leq k^{-4} \tag{6.22}$$

and the same bound for Equation 6.21. Setting both bounds into Equation 6.19 allows us to derive the upper bound on the number of selections of a

suboptimal arm a that is given in Theorem 5:

$$\begin{aligned}
\mathbb{E}\left[\mathcal{L}^k(a)\right] &\leq \mathcal{L}^{\min}(a) + \sum_{t=k_{\min}}^{k}\sum_{s=1}^{t-1}\sum_{u=k_{\min}}^{t-1}\mathbb{I}\left[\hat{Q}^u(a) + \hat{U}^u(a) \geq \hat{Q}^s(a^\star) + \hat{U}^s(a^\star)\right] \\
&\leq \mathcal{L}^{\min}(a) + \sum_{t=1}^{k}\sum_{s=1}^{t}\sum_{u=1}^{t}\mathbb{P}\left[\hat{Q}^s(a^\star) + \hat{U}^s(a^\star) < Q_\star(a^\star)\right] + \\
&\qquad\qquad \mathbb{P}\left[\hat{Q}^u(a^\star) + \hat{U}^u(a) < Q_\star(a^\star)\right] \\
&\leq \frac{8\cdot\sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2} + 1 + \sum_{t=1}^{k}\sum_{s=1}^{t}\sum_{u=1}^{t} 2\cdot t^{-4} \\
&\leq \frac{8\cdot\sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2} + 1 + \sum_{t=1}^{k} 2\cdot t^{-2} \\
&\leq \frac{8\cdot\sqrt{k}}{\left(Q_\star(a^\star) - Q_\star(a)\right)^2} + 1 + \frac{\pi^2}{3}.
\end{aligned}$$

■

The most important consequence of Theorem 5 for our purposes is that the number of times the optimal arm is selected with RT-UCB is $k - O(\sqrt{k})$ times in k trials as the derived bound is dependent from the number of trials only in $\sqrt{k}$. This is asymptotically less often than the number of exploitative decisions of UCB1 just as we have aimed for. Finally, we would like to mention that the constant of 8 in the denominator of our bound can be brought arbitrarily close to 2 by using the same technique that was applied by Auer et al. (2002) in their UCB2 algorithm.

Generalization of RT-UCB

Unlike in the ϵ-decay function that is used in the ϵ_{RT}-greedy action selection, where a parameter γ allows to adapt the fractional exponent that specifies the root, we have used a constant value of $\gamma = \frac{1}{2}$ for RT-UCB so far. The reason is as follows: even though k^γ is *asymptotically* larger than $\ln(k)$ for all $\gamma > 0$, it takes a large number of trials until $k^\gamma > \ln(k)$ if γ is small: for instance, if $\gamma = \frac{1}{3}$, it is not until the 94-th visit that $k^{\frac{1}{3}} > \ln(k)$, and if $\gamma = \frac{1}{4}$ it already takes more than 5500 visits until $\ln(k)$ is larger than $k^{\frac{1}{4}}$. A function that displays the relation between k^γ and $\ln(k)$ is $f_\gamma(k) = k^\gamma - \ln(k)$, which is shown for values of $\gamma \in \{\frac{3}{4}, \frac{1}{2}, \frac{1}{e}, \frac{1}{3}, \frac{1}{4}\}$ in Figure 6.5. For all k where $f_\gamma(k)$ is positive, k^γ is larger than $\ln(k)$. This has both theoretical and practical consequences for a generalized RT-UCB action selection. First, the proof of Theorem 5 does not hold for the *generalized RT-UCB* action selection, where a chance node in

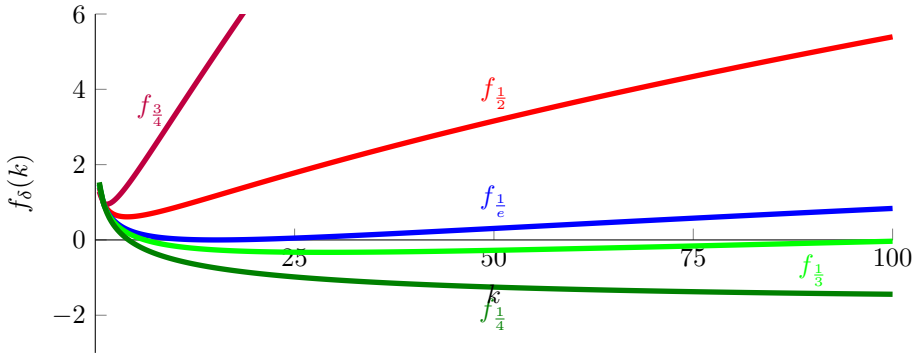

Figure 6.5: Comparison of root-valued functions and the natural logarithm.

decision node d is selected uniformly at random from the set

$$\mathcal{A}^k_{\text{RT-UCB}}(d) := \underset{c \in \mathcal{A}^k_{\neg\varphi}(d_t)}{\arg\max} \left(\hat{Q}^{k-1}(c) + \sqrt{\frac{2 \cdot (\mathcal{L}^{k-1}(d))^\gamma}{\mathcal{L}^{k-1}(c)}} \right).$$

The reason that the proof does not hold for RT-UCB is that we use the fact that $e^{-4 \cdot \sqrt{k}} \leq e^{-4 \cdot \ln(k)}$ for all $k \geq 1$ to derive Equation 6.22, which does, as we can see in Figure 6.5, not hold for all $0 < \gamma < 1$ if we extend from $\sqrt{k}$ to k^γ. We can nonetheless derive a bound for RT-UCB.

Corollary 1. *With generalized RT-UCB action selection with $0 < \gamma < 1$, each suboptimal arm a is selected at most*

$$\mathbb{E}\left[\mathcal{L}^k(a)\right] \leq k_{\text{min}} + 1 + \frac{\pi^2}{3}$$

times in expectation in an MAB problem with k trials and optimal arm $a^\star$, where

$$k_{\text{min}} := \max\{\frac{8 \cdot k^\gamma}{(Q_\star(a^\star) - Q_\star(a))^2}\} \cup \{k \geq 1 \mid k^\gamma - \ln(k) = 0\}.$$

Proof: The only difference between Theorem 5 and Corollary 1 is that we extend the first phase of the sequence of trials to

$$\max\{k \geq 1 \mid k^\gamma - \ln(k) = 0\} + 1$$

if that value exists and if it is larger than the value that was denoted with $\mathcal{L}^{\text{min}}(a)$ in the previous proof. The additional bound for the start of the second phase corresponds to the trial from where on it is guaranteed that $k^\gamma > \ln(k)$ for all k larger than the bound (which is the largest value k where $k^\gamma = \ln(k)$,

and since $\lim_{k\to\infty} k^\gamma - \ln(k) = \infty$), such that the rest of the proof of Theorem 5 holds for RT-UCB as well. ■

It is not hard to show that $\{k \geq 1 \mid k^\gamma - \ln(k) = 0\} = \varnothing$ iff $\gamma > \frac{1}{e}$ by studying some properties of $f_\gamma(k)$: it has a single critical point as its derivative

$$f'_\gamma(k) = \gamma \cdot k^{\gamma-1} - \frac{1}{k} = \gamma \cdot k^\gamma \cdot \frac{1}{k} - \frac{1}{k} = (\gamma \cdot k^\gamma - 1) \cdot \frac{1}{k}$$

can be used to compute the critical point(s) by setting $f'_\gamma(k)$ to 0:

$$\begin{aligned}
& f'_\gamma(k) = 0 \\
\Leftrightarrow \quad & (\gamma k^\gamma - 1) \cdot \tfrac{1}{k} = 0 \\
\Leftrightarrow \quad & \gamma k^\gamma - 1 = 0 \\
\Leftrightarrow \quad & \gamma k^\gamma = 1 \\
\Leftrightarrow \quad & k^\gamma = \tfrac{1}{\gamma} \\
\Leftrightarrow \quad & k = \left(\tfrac{1}{\gamma}\right)^{\frac{1}{\gamma}}.
\end{aligned}$$

Hence, $f_\gamma(k)$ has a single critical point at $\left(\frac{1}{\gamma}\right)^{\frac{1}{\gamma}}$ with value

$$\begin{aligned}
f_\gamma\left(\left(\frac{1}{\gamma}\right)^{\frac{1}{\gamma}}\right) &= \left(\frac{1}{\gamma}\right)^{\frac{1}{\gamma}\gamma} - \ln\left(\frac{1}{\gamma}\right)^{\frac{1}{\gamma}} \\
&= \frac{1}{\gamma} - \frac{1}{\gamma} \cdot \ln\left(\frac{1}{\gamma}\right) \\
&= \frac{1}{\gamma} + \frac{1}{\gamma} \cdot \ln(\gamma) \\
&= \frac{1}{\gamma} \cdot (1 + \ln(\gamma)),
\end{aligned}$$

which is larger than 0 for all $k \geq 1$ iff

$$\begin{aligned}
& \tfrac{1}{\gamma}(1 + \ln(\gamma)) > 0 \\
\Leftrightarrow \quad & 1 + \ln(\gamma) > 0 \\
\Leftrightarrow \quad & \ln(\gamma) > -1 \\
\Leftrightarrow \quad & \gamma > \tfrac{1}{e}.
\end{aligned}$$

With this, it is easy to see that the seemingly different bounds that are given in Theorem 5 and Corollary 1 are identical for $\gamma = \frac{1}{2} > \frac{1}{e}$. From a practical point of view, it is important to keep in mind that RT-UCB with $\gamma < \frac{1}{e}$ behaves more exploitative in the initial phase than even UCB1. This is significant especially when the deliberation time is small. For instance, with

$\gamma = \frac{1}{4}$, it takes until the 5504-th trial that $k^{\frac{1}{4}} > \ln(k)$, which is a number of trials that is not even reached in many of the experiments that are presented in Chapter 8.

6.6 Recommendation Functions

The recommendation function is the only ingredient that is discussed in this thesis that has not been included in our initial description of the THTS framework (Keller and Helmert, 2013) where a static recommendation of the action with the highest action-value estimate is considered. We have selected to extend the THTS framework with an adaptable recommendation function mostly due to the work of Bubeck et al. (2009), who show that there are combinations where it is *not* the best decision to recommend the action with the highest action-value estimate. Even though the work of Bubeck et al. is on the Multi-Armed Bandit problem with access to a generative model, it is to be expected that their results carry over to MDP planning. A first step in that direction is the MpaUCT algorithm of Feldman and Domshlak (2014a) which differs from UCT, among other things, in the fact that the *most played action* is recommended rather than the one with the highest action-value estimate.

Formally, a recommendation function ingredient is a mapping from the explicit graph $\mathcal{G}^k$ that is the result of a sequence of k trials to a probability distribution over all actions that are applicable in the initial state.

Definition 24 (Recommendation Function Ingredient). *A* **recommendation function ingredient** *is a function $\mathfrak{R}$ that maps an explicit graph $\mathcal{G}^k = \langle d_0, \mathrm{D}^k, \mathrm{C}^k, E^k \rangle$ to a probability distribution over $\mathcal{A}(d_0)$.*

A recommendation function may extract any information from the provided explicit graph, but it has to make a decision quickly if it is used in an anytime algorithm. We therefore only consider recommendation functions that base their decision on comparisons of annotations of the children of the root node only. There are some examples for recommendation functions in the literature. The solution that is often considered exclusively is called the *max child* recommendation by Chaslot et al. (2008) and the *expected best arm* by Bubeck et al.. Formally, it is the recommendation function ingredient where the probability distribution is such that one of the actions with highest action-value estimate is drawn uniformly at random:

$$\mathfrak{R}_{\text{EBA}}(\mathcal{G}^k) = \mathcal{U}\left(\arg\max_{c \in \mathcal{A}(d_0)} \hat{Q}^k(c)\right).$$

The second recommendation function that is considered here is also described both by Bubeck et al. and Chaslot et al., namely the *most played arm*

or *robust child* recommendation

$$\mathfrak{R}_{\text{MPA}}(\mathcal{G}^k) = \mathcal{U}\left(\arg\max_{c \in \mathcal{A}(d_0)} \mathcal{L}^k(c)\right),$$

where one of the actions that have been selected most often in the root node by the action selection ingredient is recommended uniformly at random. Even though we do not consider any more recommendation functions in the theoretical and empirical analysis of THTS algorithms in the following chapter, we would like to briefly mention some further possibilities that are described in related work. Chaslot et al. consider the *secure child* recommendation in addition to the aforementioned two functions

$$\mathfrak{R}_{\text{SecA}}(\mathcal{G}^k) = \mathcal{U}\left(\arg\max_{c \in \mathcal{A}(d_0)} \hat{Q}^k(c) + \frac{1}{\sqrt{\mathcal{L}^k(c)}}\right),$$

where one of the actions that maximize an upper confidence bound on the action-value estimate is recommended. And finally, Bubeck et al. propose the *empirical distribution of plays* recommendation function, where each arm is selected with a probability that is proportional to the number of times it has been selected by the action selection ingredient:

$$\mathfrak{R}_{\text{EDP}}(\mathcal{G}^k) = \bigcup_{c \in \mathcal{A}(d_0)} \{(p(c) : c)\}$$

with $p(c) := \frac{\mathcal{L}^k(c)}{\sum_{c' \in \mathcal{A}(d_0)} \mathcal{L}^k(c')}$. This concludes our discussion of specific ingredients for THTS algorithms, and we continue with a theoretical analysis of how they can be combined to anytime optimal algorithms in the next chapter.

CHAPTER 7

Theoretical Evaluation

In the last two chapters, we have presented the Trial-based Heuristic Tree Search framework and a selection of ingredients that allow the specification of both well-known and novel algorithms. Most theoretical properties of interest cannot be derived for ingredients in isolation. Instead, we analyze in Section 7.1 which *combinations* of ingredients yield a recipe that is such that all action-value estimates converge towards the respective true state- and action-values. In Section 7.2, we combine the insights with the available recommendation functions and derive which *anytime optimal* THTS algorithms can be created with the ingredients that have been presented in Chapter 6.

7.1 Convergence

We have deliberately decided to keep the THTS framework as generic as possible, which allows for a wide variety of recipes that result in anytime optimal algorithms. However, this means that there is no right or wrong when it comes to assessing individual ingredients, as it is only the whole recipe that can be analyzed. As we have introduced nine different action selections in the previous chapter (not counting the greedy action selection), nine different backup functions and two recommendation functions, this allows for an amazing 162 different THTS algorithms that have to be analyzed theoretically, 81 for each recommendation function. Luckily, we can reduce the analysis for all recipes that use the expected best arm recommendation function to an analysis of the *convergence* of the state- and action-value estimates to the true state- and action-values in the given recipe, and a few simple conclusions allow us to derive results for the anytime optimality of THTS recipes with the most played arm recommendation function as well.

Let us therefore put the recommendation functions aside for the moment (we consider it again in the following section) and focus on the convergence of state- and action-value estimates to the true state- and action-values. We start

with a brief survey of theoretical properties of the individual ingredients that were discussed in the previous chapter. For three of the six ingredients that describe a THTS algorithm we have presented only a single implementation, and all algorithms that are considered in this thesis hence share

- the Monte-Carlo outcome selection $\mathfrak{O}_{\text{MC}}$,
- the expansion count trial length component $\mathfrak{T}_{\text{EC}}$, and
- the virtual trials initialization $\mathfrak{I}_{\text{VT}}$.

We nonetheless extract the key properties of these implementations that are required for anytime optimality, which allows to obtain general results for all six ingredients.

7.1.1 Core Properties of Ingredients

Let us now have a look at some of the properties of the used ingredients that hold in isolation (i. e., independently of the other used ingredients). A property of the EC trial length component that is important for this analysis is that is it *explorative* independently of the choice of the parameter ν.

Definition 25 (Explorative Trial Length Component). *A trial length component $\mathfrak{T}$ is* **explorative** *iff it is such that for all $\theta^{k,t} = (d_0, c_0, \ldots, d_t, c_t)$ and all $\mathcal{G}^{k,t}$:*

$$\mathfrak{T}(\mathcal{G}^{k,t}, \theta^{k,t}) = \texttt{true} \Rightarrow |\mathrm{D}^{[k/k-1]}| > 0.$$

(Recall that $\mathrm{D}^{[k/k-1]}$ denotes the set of decision nodes that were added to the explicit graph in the k-th trial.) That $\mathfrak{T}^{\nu}_{\text{EC}}$ is explorative for all $\nu > 0$ follows directly from Definition 22.

It is also of relevance for this analysis that all of the backup functions that were presented in the previous chapter are *goal-aware* (a term that is used due to the similarity with goal-aware heuristics in classical planning), which holds if the action-value estimates of all leaf nodes are always backed up to the true action-values.

Definition 26 (Goal-aware Backup Functions). *A backup function $\mathfrak{B}$ is* **goal-aware** *iff it is such that for all $k > 0$ and all leaf nodes $c_\star \in N_\star$ that are expanded in $\mathcal{G}^k$:*

$$\hat{Q}^k_{\mathfrak{B}}(c_\star) = \mathcal{R}(c_\star).$$

It is easy to see that all considered backup functions are goal-aware, as the recursive functions that are given in Equations 6.6, 6.8, 6.10, 6.12, 6.13, and 6.14, which define how the value of $\hat{Q}^k_{\mathfrak{B}}(c_\star)$ is updated for all $c_\star \in N_\star$, list leaf nodes as special cases and assign the value of $\mathcal{R}(c_\star)$.

From hereon, we have to continue our analysis by considering only subsets of possible THTS recipes. The first distinction we make is to separate backup

functions that do not label nodes as solved from backup functions that do. We start with the former, which includes all backup functions that were presented in Section 6.3 apart from Partial Bellman backups. The convergence of THTS recipes that use PB backups is discussed at the end of this section.

Definition 27 (Solve Labeling). *An initialization*[1] *and a backup function do not support* **solve-labeling** *iff they are such that* $\varphi^k(n) = \texttt{false}$ *for all* $k \geq 0$ *and all* $n \in N^k$.

In the presence of an initialization and a backup function that do not support solve-labeling, the MC outcome selection that is used in all our algorithms is *unbiased*:

Definition 28 (Unbiased Outcome Selection). *An outcome selection* $\mathfrak{O}$ *is* **unbiased** *iff it is such that for all* $c \in \mathsf{C}$ *and all* $s' \in \text{succ}(s(c), a(c))$

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(c)\right] = \infty \Rightarrow \lim_{k\to\infty} \mathbb{E}\left[\frac{\mathcal{L}^k(\mathbf{dn}(c,s'))}{\sum_{s''\in\text{succ}(s(c))} \mathcal{L}^k(\mathbf{dn}(c,s''))}\right] = \mathbb{P}_{\mathcal{T}}[s' \mid s(c), a(c)].$$

In other words, outcomes are selected according to their true probabilities in all chance nodes that are visited infinitely often in expectation, which is given in the MC outcome selection since $\mathbb{P}^k_{\neg\varphi}[c] = 1$ for all $k \geq 0$ and all $c \in \mathsf{C}^k$.

7.1.2 Infinite Exploration

If nodes are not labeled as solved, all action selections that were discussed in the previous chapter (apart from GRD) *explore infinitely*.

Definition 29 (Infinite Exploration). *An action selection* **explores infinitely** *iff for all* $d \in \mathsf{D}$ *it holds that*

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(d)\right] = \infty \Rightarrow \lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(c)\right] = \infty \text{ for all } c \in \mathcal{A}(d).$$

As it is not as obvious as the previous properties of presented ingredients, we show that all our action selections explore infinitely in the following.

Theorem 6. *The UNI,* ϵ*-G,* ϵ_{LIN}*-G,* ϵ_{RT}*-G,* ϵ_{LOG}*-G, BE, BE-DT, UCB1 and RT-UCB action selections explore infinitely if combined with an initialization and a backup function that do not support solve-labeling.*

Proof: In the following, let d be a decision node with $\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(d)\right] = \infty$. As the initialization and the backup function do not support solve-labeling, it holds that $\varphi^k(n) = \texttt{false}$ for all $k \geq 0$ and all $n \in N^k$ according to Definition 27, and hence $\mathcal{A}^k_{\neg\varphi}(d) = \mathcal{A}(d)$ for all $k > 0$. Let $\mathbb{P}^k_{\mathfrak{A}}[c \mid d]$ be the probability

[1] The initialization is relevant because solve labels must be `false` for $k = 0$ as well.

that c is selected with $\mathfrak{A}$ in the i-th trial that actually visits d. With $\mathbb{E}\left[\mathcal{L}^k(c)\right] = \sum_{i=1}^{\mathcal{L}^k(d)} \mathbb{P}^k_{\mathfrak{A}}\left[c \mid d\right]$ it is then sufficient to show that

$$\sum_{i=1}^{\infty} \mathbb{P}^k_{\mathfrak{A}}\left[c \mid d\right] = \infty$$

holds for all $c \in \mathcal{A}(d)$ to show that action selection $\mathfrak{A}$ explores infinitely.

- UNI explores infinitely since for all $c \in \mathcal{A}(d)$

$$\sum_{k=1}^{\infty} \mathbb{P}^k_{\text{UNI}}\left[c \mid d\right] = \sum_{k=1}^{\infty} \frac{1}{|\mathcal{A}(d)|} = \frac{1}{|\mathcal{A}(d)|} \cdot \sum_{k=1}^{\infty} 1 = \infty.$$

- ϵ-G, ϵ_{LIN}-G, ϵ_{RT}-G, and ϵ_{LOG}-G explore infinitely since

$$\underbrace{\sum_{k=1}^{\infty} \alpha}_{(1)} = \underbrace{\sum_{k=1}^{\infty} \frac{1}{k}}_{(2)} = \underbrace{\sum_{k=1}^{\infty} \frac{1}{k^{\gamma}}}_{(3)} = \underbrace{\sum_{k=1}^{\infty} \frac{1}{\ln(k)}}_{(4)} = \infty$$

  for a constant $\alpha > 0$ are the infinite sums that describe the probability that a non-greedy successor of d is selected with (1) ϵ-G, (2) ϵ_{LIN}-G, (3) ϵ_{RT}-G, and (4) ϵ_{LOG}-G action selection, and the probability that a non-greedy successor is selected is a lower bound for all $c \in \mathcal{A}(d)$. Divergence of the infinite sums is obvious for (1) and well-known for the harmonic series (2), which induces divergence of (3) (with $0 < \gamma < 1$) and (4) as it is a lower bound for both.

- BE explores infinitely since for all $c \in \mathcal{A}(d)$

$$\begin{aligned}
\sum_{k=1}^{\infty} \mathbb{P}^k_{\text{BE}}\left[c \mid d\right] &= \sum_{k=1}^{\infty} \frac{e^{\tau^{-1} \cdot \tilde{Q}^{k-1}(d,c)}}{\sum_{c' \in \mathcal{A}(d)} e^{\tau^{-1} \cdot \tilde{Q}^{k-1}(d,c')}} \\
&\overset{(1)}{\geq} \sum_{k=1}^{\infty} \frac{1}{|\mathcal{A}(d)| \cdot e^{\tau^{-1}}} \\
&= \frac{1}{|\mathcal{A}(d)| \cdot e^{\tau^{-1}}} \cdot \sum_{k=1}^{\infty} 1 \\
&= \infty,
\end{aligned}$$

  where (1) holds because $\tilde{Q}^{k-1}(d,c) \in [0,1]$ due to the normalization.

- BE-DT explores infinitely since for all $c \in \mathcal{A}(d)$

$$\begin{aligned}\sum_{k=1}^{\infty} \mathbb{P}^k_{\text{BE-DT}}\left[c \mid d\right] &= \sum_{k=1}^{\infty} \frac{e^{\ln(k)\cdot\tilde{Q}^{k-1}(d,c)}}{\sum_{c'\in\mathcal{A}^k(d_t)} e^{\ln(k)\cdot\tilde{Q}^{k-1}(d,c')}} \\ &\overset{(1)}{\geq} \frac{1}{|\mathcal{A}(d)|}\sum_{k=1}^{\infty}\frac{1}{e^{\ln(k)}} \\ &= \frac{1}{|\mathcal{A}(d)|}\sum_{k=1}^{\infty}\frac{1}{k} \\ &\overset{(2)}{=} \infty,\end{aligned}$$

  where (1) holds because $\tilde{Q}^{k-1}(d,c) \in [0,1]$ due to the normalization and (2) due to the divergence of the harmonic series.

- The important argument for UCB1 comes from the analysis of Auer et al. (2002), who show that each non-greedy $c \in \mathcal{A}(d)$ is selected at least $O\left(\ln(k)\right)$ times (greedy actions are selected more often). Then UCB1 explores infinitely since for all $c \in \mathcal{A}(d)$

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(c)\right] \geq \lim_{k\to\infty} \ln\left(k\right) = \infty.$$

- Similarly, with Theorem 5 and Corollary 1, RT-UCB explores infinitely since for all $c \in \mathcal{A}(d)$ and $0 < \gamma < 1$

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(c)\right] \geq \lim_{k\to\infty} k^{\gamma} = \infty.$$

■

Before we proceed, we would like to mention that the reason why we consider a root-valued and linear ϵ-decay function for ϵ-greedy action selection but only a logarithmic τ-decay function is no arbitrary choice. We assume that functions where τ decreases faster than with the logarithmic pace that is used in BE-DT are impossible in a variant of Boltzmann Exploration that explores infinitely. We base this assumption on the following observation: it is not only the case that a BE variant with (1) a root-valued decay schema for τ explores likely and (2) a linear decay schema for τ explores certainly not infinitely, but (3) even a variant where $\tau^k(d)$ is computed as in BE-DT but with a constant factor $\gamma > 1$ as $\tau^k(d) = \frac{1}{\gamma\cdot\ln\left(\mathcal{L}^k(d)\right)}$ does not lead to an action selection that explores infinitely. We can only conjecture convergence of $\sum_{k=1}^{\infty} e^{-\sqrt{k}}$ for assumption (1), while it is well-known that (2) the geometric series $\sum_{k=1}^{\infty} e^{-k}$ and (3) a hyper-harmonic series $\sum_{k=1}^{\infty} k^{-\gamma}$ with $\gamma > 1$ converge.

7.1.3 Exhaustive Search

We can now combine Theorem 6 with some properties of other ingredients to derive that there are THTS recipes where it can be expected that all nodes in the AND/OR tree are eventually explicated and visited infinitely often.

Theorem 7. *In a THTS algorithm with an outcome selection $\mathfrak{O}$ that is unbiased, an action selection $\mathfrak{A}$ that explores infinitely, a trial length component $\mathfrak{T}$ that is explorative and an initialization and a backup function that do not support solve-labeling*

1. *there is a $k_0 > 0$ such that $\mathcal{G}^k = \mathcal{G}$ for all $k \geq k_0$, and*
2. *it holds that $\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(n)\right] = \infty$ for all $n \in N$.*

Proof: We show Theorem 7 by induction over the structure of the explicit AND/OR tree. The base idea is that all decision and chance nodes up to depth t are explicated in a finite number of trials $k_0(t)$, which implies that all nodes up to depth t are visited infinitely often as

- $\mathfrak{T}$ is explorative (no trial stops before a node in depth t is encountered)
- $\mathfrak{O}$ is unbiased (each decision node is visited with non-zero probability and hence infinitely often in the limit)
- $\mathfrak{A}$ explores infinitely (each chance node is selected infinitely often).

The root node d_0 and all its successors are explicated eventually (the latter because $\mathfrak{A}$ explores infinitely), so let us assume that the hypothesis holds up to depth t. As the trial length component is explorative, no trial stops in a node in depth t, but proceeds (at least) to depth $t+1$, and as the action selection strategy explores infinitely and the outcome selection is unbiased all nodes in depth $t+1$ are eventually explicated and are visited infinitely often as well. ■

We can conclude from Theorem 7 not only that all nodes are explicated eventually and visited infinitely often, but also that the implications inherent to an outcome selection that is unbiased (all outcomes are selected according to their probability) hold in the limit. It does not take much imagination to see that a THTS algorithm with a "clairvoyant" action selection that always simulates the optimal policy converges to the true state- and action-values with all our backup functions as long as outcomes are selected according to the true transition probabilities. Unfortunately, we do not have such a clairvoyant action selections – on the contrary, we have just used the fact that all our action selections explore infinitely, which implies that all actions are selected infinitely often. The backup functions that do not support solve-labeling have three different strategies to overcome the bias that is incurred by explorative action selections, and we discuss each of these strategies on their own in the following.

7.1.4 Convergence with On-policy Backups

The first strategy to overcome the bias of explorative actions is incorporated in the MC and TD backup functions. Both build upon the idea that, even though each chance node is selected infinitely often in each decision node, the chance nodes that represent optimal choices are selected "a significantly larger" number of times than all other options. Such an action selection is called *greedy in the limit*.

Definition 30 (Greedy in the Limit). *An action selection $\mathfrak{A}$ is* **greedy in the limit** *iff for all decision node $d \in \mathrm{D}$ it holds that*

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(d)\right] = \infty \text{ and } \lim_{k\to\infty} \hat{Q}^k(c) = \bar{Q}_c \text{ for all } c \in \mathcal{A}(d) \text{ and } \bar{Q}_c \in \mathbb{R}$$
$$\Rightarrow \lim_{k\to\infty} \frac{\mathbb{E}\left[\mathcal{L}^k(\bar{c}_\star)\right]}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \mathbb{E}\left[\mathcal{L}^k(c')\right]} = \infty \text{ for all } \bar{c}_\star \in \bar{\mathcal{A}}(d),$$

where $\bar{\mathcal{A}}_\star(d) := \arg\max_{c\in\mathcal{A}(d)} \bar{Q}_c$ and $\bar{\mathcal{A}}_\star(d) \neq \mathcal{A}^k(d)$ for $k \to \infty$.

Theorem 8. *The UNI, ϵ-G, and BE action selections are not greedy in the limit, and the ϵ_{LIN}-G, ϵ_{RT}-G, ϵ_{LOG}-G, BE-DT, UCB1, and RT-UCB action selections are greedy in the limit.*

Proof: In the following, let d be a decision node with $\lim_{k\to\infty} \mathcal{L}^k(d) = \infty$ and $\lim_{k\to\infty} \hat{Q}^k(c) = \bar{Q}_c$ for all $c \in \mathcal{A}(d)$. Furthermore, to ease notation, we use the abbreviations $\alpha := |\mathcal{A}(d)|$ and $\beta := |\bar{\mathcal{A}}_\star(d)|$ in this proof. Then:

- UNI, ϵ-G, and BE are not greedy in the limit since

$$\begin{aligned}\lim_{k\to\infty} \frac{\mathbb{E}\left[\mathcal{L}^k(\bar{c}_\star)\right]}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \mathbb{E}\left[\mathcal{L}^k(c')\right]} &= \lim_{k\to\infty} \frac{\mathbb{P}^k_{\mathfrak{A}}[\bar{c}_\star \mid d]\cdot k}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} (\mathbb{P}^k_{\mathfrak{A}}[c' \mid d]\cdot k)} \\ &= \frac{\mathbb{P}^k_{\mathfrak{A}}[\bar{c}_\star \mid d]}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \mathbb{P}^k_{\mathfrak{A}}[c' \mid d]} \\ &\neq \infty,\end{aligned}$$

  where the inequality holds because

  - $\mathbb{E}\left[\mathbb{P}^k_{\text{UNI}}[c \mid d]\right] = \frac{1}{\alpha}$ for all $c \in \mathcal{A}(d)$ is constant for all $k \geq 0$ in UNI
  - $\mathbb{E}\left[\mathbb{P}^k_{\epsilon\text{-G}}[\bar{c}_\star \mid d]\right] = \frac{\epsilon}{\alpha} + \frac{1-\epsilon}{\beta}$ for all $\bar{c}_\star \in \bar{\mathcal{A}}_\star(d)$ and $\mathbb{E}\left[\mathbb{P}^k_{\epsilon\text{-G}}[c \mid d]\right] = \frac{\epsilon}{\alpha}$ for all $\bar{c}_\star \in \bar{\mathcal{A}}_\star(d)$ are constant for all $k \geq 0$ in ϵ-G
  - $\mathbb{E}\left[\mathbb{P}^k_{\text{BE}}[c \mid d]\right] = \frac{e^{\tau^{-1}\cdot\bar{Q}_c}}{\sum_{c'\in\mathcal{A}(d)} e^{\tau^{-1}\cdot\bar{Q}_c}}$ for all $c \in \mathcal{A}(d)$ is constant in expectation for all $k \geq 0$ since $\hat{Q}^k(c)$ converges.

- ϵ_{LIN}-G, ϵ_{RT}-G, and ϵ_{LOG}-G are greedy in the limit since for all $\bar{c}_\star \in \bar{\mathcal{A}}_\star(d)$:

$$\begin{aligned}
\lim_{k\to\infty} \frac{\mathbb{E}\left[\mathcal{L}^k(\bar{c}_\star)\right]}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \mathbb{E}\left[\mathcal{L}^k(c')\right]} &= \lim_{k\to\infty} \frac{\mathbb{P}^k_{\mathfrak{A}}[\bar{c}_\star \mid d]\cdot k}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \mathbb{P}^k_{\mathfrak{A}}[c' \mid d]\cdot k} \\
&= \lim_{k\to\infty} \frac{(\frac{\epsilon^k(d)}{\alpha} + \frac{1-\epsilon^k(d)}{\beta})}{(\alpha-\beta)\cdot\frac{\epsilon^k(d)}{\alpha}} \\
&= \lim_{k\to\infty} \underbrace{\frac{1}{\alpha-\beta}}_{\text{const}} + \underbrace{\frac{\overbrace{1-\epsilon^k(d)}^{\to 1}}{\epsilon^k(d)\cdot\left(\beta-\frac{\beta^2}{\alpha}\right)}}_{\to 0} \\
&= \infty,
\end{aligned}$$

  where the annotated limits are correct since $\lim_{k\to\infty} \epsilon^k(d) = 0$.

- BE-DT is greedy in the limit as shown by Singh et al. (2000) in Appendix B.1.

- UCB1 is greedy in the limit since there is a constant $\delta > 0$ such that for all $\bar{c}_\star \in \bar{\mathcal{A}}_\star(d)$:

$$\begin{aligned}
\lim_{k\to\infty} \mathbb{E}\left[\frac{\mathcal{L}^k(c_\star)}{\sum_{c'\in\mathcal{A}(d)\setminus\mathcal{A}^\infty_\star(d)} \mathcal{L}^k(c')}\right] &\overset{(1)}{=} \lim_{k\to\infty} \frac{\frac{1}{\beta}\cdot(k-\delta\cdot\ln(k))}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \delta\cdot\ln(k)} \\
&= \lim_{k\to\infty} \frac{\frac{1}{\beta}\cdot(k-\delta\cdot\ln(k))}{(\alpha-\beta)\cdot\delta\cdot\ln(k)} \\
&= \frac{1}{\beta\cdot(\alpha-\beta)} \lim_{k\to\infty} \frac{k-\delta\cdot\ln(k)}{\delta\cdot\ln(k)} \\
&= \frac{1}{\beta\cdot(\alpha-\beta)} \lim_{k\to\infty} \frac{k}{\delta\cdot\ln(k)} - 1 \\
&= \infty,
\end{aligned}$$

  where (1) is due to the analysis of UCB1 of Auer et al. (2002) who show that each suboptimal action is selected $\delta\cdot\ln(k)$ times in expectation.

- RT-UCB is greedy in the limit since there is a constant $\delta > 0$ such that

for all $0 < \gamma < 1$ and all $\bar{c}_\star \in \bar{\mathcal{A}}_\star(d)$:

$$\begin{aligned}
\lim_{k\to\infty} \mathbb{E}\left[\frac{\mathcal{L}^k(c_\star)}{\sum_{c'\in\mathcal{A}(d)\setminus\mathcal{A}^\infty_\star(d)} \mathcal{L}^k(c')}\right] &\overset{(1)}{=} \lim_{k\to\infty} \frac{\frac{1}{\beta}\cdot(k-\delta\cdot k^\gamma)}{\sum_{c'\in\mathcal{A}(d)\setminus\bar{\mathcal{A}}_\star(d)} \delta\cdot k^\gamma} \\
&= \lim_{k\to\infty} \frac{\frac{1}{\beta}\cdot(k-\delta\cdot k^\gamma)}{(\alpha-\beta)\cdot\delta\cdot k^\gamma} \\
&= \frac{1}{\beta\cdot(\alpha-\beta)} \lim_{k\to\infty} \frac{k-\delta\cdot k^\gamma}{\delta\cdot k^\gamma} \\
&= \frac{1}{\beta\cdot(\alpha-\beta)} \lim_{k\to\infty} \frac{k}{\delta\cdot k^\gamma} - 1 \\
&= \infty,
\end{aligned}$$

where (1) is due to Theorem 5 and Corollary 1.

■

Please note that (similar to the reasons to omit a root-valued or linear variant of Boltzmann exploration) we do not consider a linear variant of UCB1 as the corresponding action selection is not greedy in the limit. Let us now state our first convergence result for a THTS recipe.

Theorem 9. *A THTS algorithm with an outcome selection $\mathfrak{O}$ that is unbiased, an action selection $\mathfrak{A}$ that explores infinitely and is greedy in the limit, a trial length component $\mathfrak{T}$ that is explorative, an initialization that does not support solve-labeling and the MC or TD backup function is such that for all $c \in \mathsf{C}$*

$$\lim_{k\to\infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c).$$

Proof: Let us start by having a look at the common form of the equation that is used to update action-value estimates in MC, TD, and QL backups. Consider the *target* $T^k_{\mathfrak{B}}(c)$ of a chance node c in the k-th trial in backup function $\mathfrak{B}$ as

$$\begin{aligned}
T^k_{\mathsf{MC}}(c) &:= \mathcal{R}_{t..l}(\theta^k) && \text{in MC backups,} \\
T^k_{\mathsf{TD}}(c) &:= \mathcal{R}(c) + \hat{Q}^k(c') && \text{in TD backups, and} \\
T^k_{\mathsf{QL}}(c) &:= \mathcal{R}(c) + \hat{V}^k(d') && \text{in QL backups,}
\end{aligned}$$

where c' and d' are the next chance and decision nodes that are visited in trial θ^k after c. The target allows the generalization of the three recursive backup functions for action-value estimates in MC, TD, and QL backups to

$$\begin{aligned}
\hat{Q}^k(c) &= \hat{Q}^{k-1}(c) + \eta^k(c)\cdot\left(T^k_{\mathfrak{B}}(c) - \hat{Q}^{k-1}(c)\right) \\
&= \hat{Q}^{k-1}(c) + \frac{1}{1+\eta^k_{\mathsf{d}}\cdot\mathcal{B}^k(c)}\cdot\left(T^k_{\mathfrak{B}}(c) - \hat{Q}^{k-1}(c)\right) \\
&= \frac{1}{1+\eta^k_{\mathsf{d}}\cdot\mathcal{B}^k(c)}\cdot T^k_{\mathfrak{B}}(c) + \frac{\eta^k_{\mathsf{d}}\cdot\mathcal{B}^k(c)}{1+\eta^k_{\mathsf{d}}\cdot\mathcal{B}^k(c)}\cdot\hat{Q}^{k-1}(c).
\end{aligned}$$

It is easy to see that the backup functions compute a weighted arithmetic mean of the target $T^k_{\mathfrak{B}}(c)$ and the previous estimate $\hat{Q}^{k-1}(c)$, as the weights sum up to one independently from k, $\hat{Q}^{k-1}(c)$, and $T^k_{\mathfrak{B}}(c)$:

$$\frac{1}{1+\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)} + \frac{\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)}{1+\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)} = \frac{1+\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)}{1+\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)} = 1.$$

The weight $\frac{1}{1+\eta^k_{\mathrm{d}}\cdot\mathcal{B}^k(c)}$ of the target $T^k_{\mathfrak{B}}(c)$, which was introduced as the learning rate $\eta^k(c)$ in Section 6.3, is often also referred to as the *step size* of the backup function (e.g., Sutton and Barto, 1998). It specifies how much of the gap between the current estimate and the target is closed in the update: the closer it is to 1 – or, the closer η^k_{d} is to 0 – the higher is the weight of the target in the computation of $\hat{Q}^k(c)$, and the lower is the weight of $\hat{Q}^{k-1}(c)$. As the step size is larger than 0 independently of the choice of the parameter η^k_{d} (that is, for $0 < \eta^k_{\mathrm{d}} \leq 1$), the backup is always such that the action-value estimate is closer (or equally close) to the target after the update than it was before, i. e.,

$$|\hat{Q}^k(c) - T^k_{\mathfrak{B}}(c)| \begin{cases} = |\hat{Q}^{k-1} - T^k_{\mathfrak{B}}(c)| & \text{if } \hat{Q}^{k-1} = T^k_{\mathfrak{B}}(c) \\ < |\hat{Q}^{k-1} - T^k_{\mathfrak{B}}(c)| & \text{otherwise.} \end{cases}$$

This implies than an action-value estimate converges towards a constant α if it holds that $\mathbb{E}\left[T^k(c)\right] = \alpha$. Moreover, it even suffices if there is a constant k_0 such that $\mathbb{E}\left[T^k(c)\right] = \alpha$ for all $k > k_0$ as $\hat{Q}^{k_0}$ is finite which does not change convergence.

From hereon, it is simple to show that Theorem 9 holds since $\mathbb{E}\left[T^k(c)\right] = Q_\star(c)$ for a sufficiently large k, which can be shown by induction over the structure of the tree from the leaf nodes to the root node, exploiting the same properties of ingredients that were used in the proof of Theorem 7, the goal-awareness of MC and TD backups, and the fact that the optimal policy is simulated in expectation as $\mathfrak{A}$ is greedy in the limit. ■

Lemma 1. *The requirement that the action selection $\mathfrak{A}$ is greedy in the limit is a necessary condition for Theorem 9 to hold.*

Proof: The lemma follows directly from a generalization of the example that was given at the beginning of Section 5.1.2. ■

As a result of this discussion we can conclude that all considered action selections except UNI, ϵ-G and BE can be combined with MC and TD backups to a THTS algorithm whose state- and action-value estimates converge towards their respective true values.

7.1.5 Convergence with Selective Backups

The convergence of selective backups incurs two contradictory anticipations: on the one hand, it appears that the results should match the convergence results of TD and MC backups, as only TD or MC backups are used in the backup phase. And on the other, as non-greedy actions are ignored in the backup phase, it appears that the constraint that the action selection has to be greedy in the limit is actually not required. We show in the following that the first anticipation is correct, before we give a counter-example that shows that it is not possible in general to relax the greedy in the limit requirement for selective backups.

Theorem 10. *A THTS algorithm with an outcome selection $\mathfrak{O}$ that is unbiased, an action selection $\mathfrak{A}$ that explores infinitely and greedy in the limit, a trial length component $\mathfrak{T}$ that is explorative, an initialization that does not support solve-labeling and the LSMC, LSTD, ESMC, or ESTD backup function is such that for all $c \in \mathsf{C}$*

$$\lim_{k \to \infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c).$$

Proof: The only difference between LSMC, ESMC, LSTD, and ESTD backups on the one hand and MC or TD backups is that the former are updated less often. However, as the induction base case holds for selective backups as well (they are also goal-aware), it is possible to show that in the limit all backups are considered as $\mathfrak{A}$ is greedy in the limit, and that selective backups converge to the same values. ■

Lemma 2. *The requirement that the action selection $\mathfrak{A}$ is greedy in the limit is a necessary condition for Theorem 10 to hold.*

Proof: Being greedy in the limit is a necessary condition for MC and TD backups because an even in the limit relevant proportion of runs under a non-optimal policy is used to update action-value estimates. This is different for selective backups, where only results that were obtained with greedy decisions are considered. However, we can show that the greedy decisions are not necessarily identical to optimal decisions as there is no guarantee that the probabilities of considered *outcomes* converges to the true probabilities even though $\mathfrak{O}$ is unbiased. We show this by providing an example with an action selection $\mathfrak{A}$ that is not greedy in the limit and show that Theorem 10 does not hold. Consider the explicit AND/OR tree from Figure 7.1, where an applicable action c_0 in d_0 has two outcomes that occur equally likely, ending up in either d_1 or d_2. In d_1, there are 100 applicable actions, all of which but one yield an immediate reward of ± 0, and the last yielding a reward of $+100$. In d_2, there is a single action with an immediate reward of -100. In combination, the expected reward of $a(c_0)$ is therefore $Q_\star(c_0) = \frac{100}{2} + \frac{-100}{2} = 0$. Now let

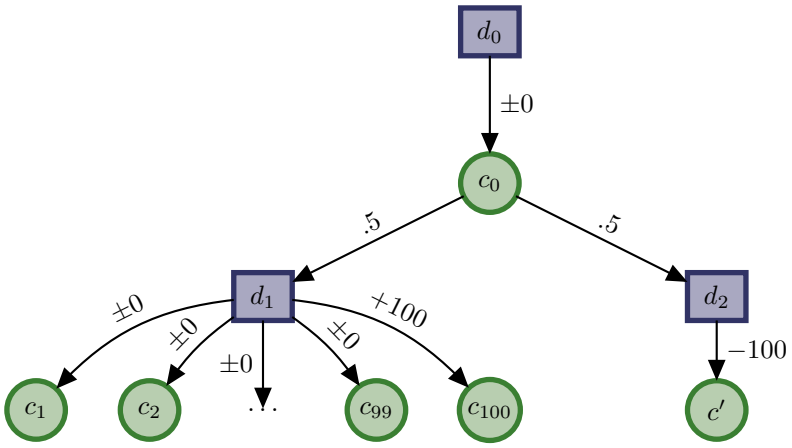

Figure 7.1: AND/OR tree where $\hat{Q}^k(c_0)$ does not converge to $Q_\star(c_0)$ in a THTS algorithm with a selective backup function and an action selection that is not greedy in the limit.

us assume that a selective backup function is combined with ingredients as required in Theorem 10 except that $\mathfrak{A}$ is not greedy in the limit.

As outcomes are sampled according to their true probabilities, d_1 and d_2 are visited equally often. Each trial that ends up in d_2 is considered when c_0 is updated with a selective backup functions as there is only a single (and hence greedy) option in d_2. As $\mathfrak{A}$ is not greedy in the limit, we know for d_1 that the relationship between greedy decision $a(c_{100})$ and non-greedy decisions (any other action) is finite, i. e.$\lim_{k\to\infty} \frac{\mathcal{L}^k(c_{100})}{\sum_{i=1}^{99} \mathcal{L}^k(c_i)} = \alpha$ for some finite constant α. Therefore, only α in $(\alpha{+}1)$ trials over d_1 are considered in the backup of c_0, and thus

$$\lim_{k\to\infty} \hat{Q}^k(c_0) = \underbrace{\left(\frac{\alpha}{\alpha{+}1} \cdot \frac{1}{2} \cdot 100\right)}_{d_1} + \underbrace{\left(\frac{1}{2} \cdot -100\right)}_{d_2} < 0 \neq Q_\star(c_0).$$

■

Please observe that it is possible to design an outcome function that compensates the imbalance by selecting outcomes such that the relation between the backup counters of outcomes are proportional to the true probabilities rather than the selection counters. Another possibility is used in the BRUE algorithm, where the switching point is not determined dynamically in the backup phase by checking from where on all selections were greedy, but is determined with a predefined computation schema that makes sure that state- and action-value estimates converge to the correct values.

7.1.6 Convergence with Off-policy Backups

The requirements for THTS algorithms with off-policy backups to converge to the right values are less constraint than in the previously discussed cases as it is always the result of the greedy action that is used for a backup. Therefore, neither of the reasons why an action selection must be greedy in the limit for convergence of on-policy or selective backups to true action values applies for the off-policy backup functions QL and MaxMC backups.

Theorem 11. *A THTS algorithm with an outcome selection $\mathfrak{O}$ that is unbiased, an action selection $\mathfrak{A}$ that explores infinitely, a trial length component $\mathfrak{T}$ that is explorative, an initialization that does not support solve-labeling and the QL or MaxMC backup function is such that for all $c \in \mathsf{C}$*

$$\lim_{k \to \infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c).$$

Proof: The proof for QL backups is similar to the proof for MC and TD backups with the addition that it is irrelevant which action is simulated as it is always the greedy action that is used to backup a node; and the proof for MaxMC backups is based on the observation that MaxMC backups are identical to the application of Full Bellman backup function in the limit (as $\mathfrak{O}$ is unbiased). ■

7.1.7 Convergence with Partial Bellman backups

The last backup function that was presented in the previous chapter is the only backup function that exploits the declarative model of the MDP in the backup function, which allows to label nodes as solved. This simplifies the convergence analysis significantly.

Theorem 12. *A THTS algorithm with a trial length component $\mathfrak{T}$ that is explorative and the PB backup function is such that for all $c \in \mathsf{C}$*

$$\hat{Q}^{k^\star}(c) = Q_\star(c)$$

and $\varphi^{k^\star}(d_0)$ for a finite $k^\star > 0$.

Proof: Eventually, all nodes are explicated as $\mathfrak{T}$ is explorative and as actions and outcomes are selected among unsolved successor nodes only. Then the Partial Bellman function is equivalent to the Full Bellman backup function. The finite $k^\star$ follows from the fact that it takes at most $|\mathsf{D}|$ trials until $\mathcal{G}^k = \mathcal{G}$ due to the exploorativeness of the trial length component, and then the root node must be labeled as solved. ■

7.2 Optimal Behavior

Let us now consider the recommendation function in addition to the other ingredients and use our convergence results to derive THTS recipes that behave

optimally in the limit. We start with the expected best arm recommendation, for which the result follows directly from the analysis up to this point.

Theorem 13. *A THTS algorithm with the expected best arm recommendation function* $\mathfrak{R}_{\text{EBA}}$ *and ingredients that are such that* $\lim_{k\to\infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c)$ *for all* $c \in \text{C}$ *executes the optimal policy in expectation with* $k \to \infty$.

Proof: Holds as $\lim_{k\to\infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c)$ for all $c \in \text{C}$ and in particular for all $c \in \mathcal{A}(d)$. ∎

For the ingredients that are considered in this thesis, Theorem 13 means that THTS algorithms are anytime optimal if the expected best arm recommendation function is combined with

- an on-policy (MC, TD) or selective (LSMC, LSTD, ESMC, ESTD) backup function and the ϵ_{LIN}-G, ϵ_{RT}-G, ϵ_{LOG}-G, BE-DT, RT-UCB or UCB1 action selection; or
- an off-policy (QL, MaxMC, PB) backup function and any of the considered action selections.

We have furthermore shown that all remaining combinations are not anytime optimal. An overview on the results is given in Table 7.1, where all non-red markings (the difference between yellow and green is irrelevant here) indicate that the corresponding THTS algorithm is anytime optimal in combination with the expected best arm recommendation function.

For the most played arm recommendation, we require a weaker version of greediness in the limit for an action selection.

Definition 31 (Favor the Greedy Action). *An action selection* $\mathfrak{A}$ **favors the greedy action in the limit** *iff for all decision node* $d \in \text{D}$ *it holds that*

$$\lim_{k\to\infty} \mathbb{E}\left[\mathcal{L}^k(d)\right] = \infty \text{ and } \lim_{k\to\infty} \hat{Q}^k(c) = \bar{Q}_c \text{ for all } c \in \mathcal{A}(d) \text{ and } \bar{Q}_c \in \mathbb{R}$$
$$\Rightarrow \mathbb{E}\left[\mathcal{L}^k(\bar{c_\star})\right] > \mathbb{E}\left[\mathcal{L}^k(c)\right] \text{ for } k \to \infty$$

for all $\bar{c_\star} \in \arg\max_{c\in\mathcal{A}(d)} \bar{Q}_c$ *and* $c \in \mathcal{A}(d) \setminus \arg\max_{c\in\mathcal{A}(d)} \bar{Q}_c$.

Theorem 14. *The UNI action selection does not favor the greedy action in the limit, and the* ϵ*-G,* ϵ_{LIN}*-G,* ϵ_{RT}*-G,* ϵ_{LOG}*-G, BE, BE-DT, UCB1, and RT-UCB action selections favor the greedy action in the limit.*

Proof:

- UNI does not favor the greedy action in the limit as it holds for all decision nodes $d \in \text{D}$ that

$$\mathbb{E}\left[\mathcal{L}^k(c)\right] = \mathbb{E}\left[\mathcal{L}^k(c')\right]$$

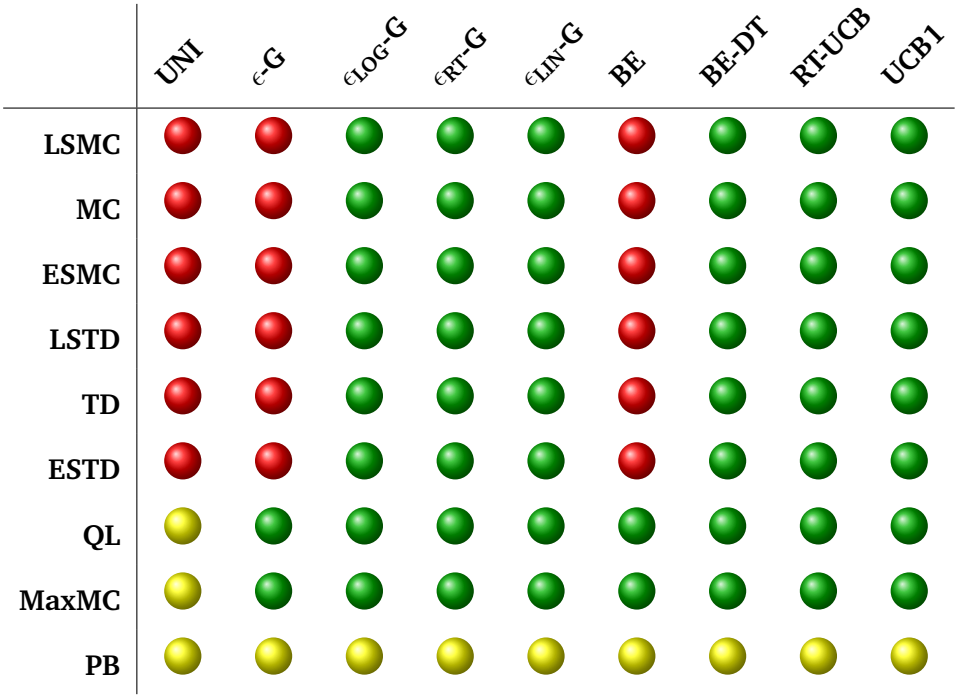

Table 7.1: Overview on anytime optimal THTS algorithms: green icons indicate that the corresponding recipe is anytime optimal in combination with both the expected best arm and the most played arm recommendation function, yellow icons that it is anytime optimal if combined with the expected best arm recommendation function and red icons that it is not anytime optimal.

for all $c, c' \in \mathcal{A}(d)$ and $k \to \infty$;

- ϵ-G favors the greedy action in the limit as it holds for all decision nodes $d \in \mathrm{D}$ that

$$\mathbb{E}\left[\mathcal{L}^k(\bar{c_\star})\right] = \frac{\epsilon}{\alpha} + \frac{1-\epsilon}{\beta} > \frac{\epsilon}{\alpha} = \mathbb{E}\left[\mathcal{L}^k(c)\right]$$

for all $\bar{c_\star} \in \arg\max_{c \in \mathcal{A}(d)} \bar{Q}_c$ and $c \in \mathcal{A}(d) \setminus \arg\max_{c \in \mathcal{A}(d)} \bar{Q}_c$ and $k \to \infty$ with $\alpha := |\mathcal{A}(d)|$ and $\beta := |\bar{\mathcal{A}_\star}(d)|$;

- BE favors the greedy action in the limit as it holds for all decision nodes $d \in \mathrm{D}$ that

$$\begin{aligned} & \frac{e^{\tau^{-1} \cdot \bar{Q}_{c_\star}}}{\sum_{c' \in \mathcal{A}(d)} e^{\tau^{-1} \cdot \tilde{Q}^{k-1}(d,c')}} > \frac{e^{\tau^{-1} \cdot \bar{Q}_c}}{\sum_{c' \in \mathcal{A}(d)} e^{\tau^{-1} \cdot \tilde{Q}^{k-1}(d,c')}} \\ \Leftrightarrow \quad & e^{\tau^{-1} \cdot \bar{Q}_{c_\star}} > e^{\tau^{-1} \cdot \bar{Q}_c} \\ \Leftrightarrow \quad & \bar{Q}_{c_\star} > \bar{Q}_c \end{aligned}$$

for all $\bar{c_\star} \in \arg\max_{c \in \mathcal{A}(d)} \bar{Q}_c$ and $c \in \mathcal{A}(d) \setminus \arg\max_{c \in \mathcal{A}(d)} \bar{Q}_c$ and $k \to \infty$; and

- ϵ_{LIN}-G, ϵ_{RT}-G, ϵ_{LOG}-G, BE-DT, UCB1, and RT-UCB favor the greedy action in the limit as this is implied by greediness in the limit.

■

Theorem 15. *A THTS algorithm with the most played arm recommendation function $\Re_{\text{MPA}}$, an initialization and a backup function that do not support solve-labeling, an action selection $\mathfrak{A}$ that favors the greedy action in the limit and where $\lim_{k \to \infty} \mathbb{E}\left[\hat{Q}^k(c)\right] = Q_\star(c)$ for all $c \in \mathsf{C}$ executes the optimal policy in expectation with $k \to \infty$.*

Proof: A THTS algorithm with the most played arm recommendation function behaves optimally in the limit if the action-value estimates converge to true action-values and additionally the action with the highest action-value estimate is selected most often. ■

Lastly, we would like to determine that the most played arm recommendation function does not combine well with a backup function that labels nodes as solved as it cannot be guaranteed that it takes more trials to solve the optimal part than a suboptimal one. Therefore, the PB backup function is only optimal in combination with the expected best arm recommendation function. This is depicted by the yellow markings in Table 7.1, which describe that a THTS algorithm is optimal with the expected best arm but not with the most played arm recommendation function.

Table 7.1 contains several well-known algorithms for planning under uncertainty. The combination of MC and UCB1 is the famous UCT algorithm, albeit with VT initialization and not with the typically used random walk heuristic (in the subsequent experimental evaluation both are distinguished as UCT and Blind UCT). Furthermore, the combination of MC backups and ϵ-G and BE is what is typically understood under ϵ-G and BE algorithms for planning under uncertainty. Silver et al. (2012) combine Temporal-Difference backups with Monte-Carlo Tree Search to Temporal-Difference search, an algorithm that can be derived in THTS by combining TD backups and ϵ-greedy action selection. Even though none of the BRUE algorithms of Feldman and Domshlak (2014b) is in Table 7.1 as we do not consider the action selection of BRUE, the row with LSMC backups behaves comparably. Furthermore, Partial Bellman backups and UNI action selection result in an algorithm that is very similar to breadth first search (they would be identical if we used RR action selection), and PB combined with UCB1 has been presented as UCT* in our initial work on THTS (Keller and Helmert, 2013). Other algorithms that are not considered are those that require Full Bellman backups (AO* and RTDP),

and even though variants with Partial Bellman backups are possible they are not considered as our initialization is not admissible.

This concludes our theoretical analysis of THTS algorithms, and we continue with an empirical comparison of all anytime optimal THTS algorithms that can be derived with the ingredients that are considered in this thesis.

CHAPTER 8

Empirical Evaluation

Over the course of the last three chapters, we have presented the THTS framework and a variety of ingredients and have analyzed which recipes combine to anytime optimal algorithms. In Section 8.1, we compare two representative THTS recipes to the suboptimal optimistic policy, hindsight optimization, and optimistic rollout that were presented in Chapter 4 of this thesis. In the subsequent main experiment that is discussed in Section 8.2, we compare all 115 anytime optimal THTS algorithms that can be derived with the ingredients that are considered in this thesis empirically.

8.1 THTS vs. Suboptimal Algorithms

Before we evaluate different combinations of ingredients that combine to an algorithm in the Trial-based Heuristic Tree Search framework, we examine if the theoretical superiority of anytime optimal THTS algorithms over the suboptimal methods that were presented in Chapter 4 can be confirmed empirically. The problem setup is the same as in the experiment that was presented in Chapter 4 on the CANADIAN TRAVELER'S PROBLEM. We use two variants of the famous UCT algorithm (Kocsis and Szepesvári, 2006), which are modeled as THTS algorithms by using MC backups, UCB1 action selection and the expected best arm recommendation function in combination with MC outcome selection and the expansion count trial length component that is configured such that it only stops when a terminal state is reached (i. e., ν is set to $\mathcal{H}$). Both UCT versions differ only in the used initialization procedure. The first version, UCTB (for *blind UCT*), uses the *blind heuristic* in the initialization that assigns an initial action-value estimate of 0 to all actions. It does hence not take into account any problem-specific information that would bias trials towards reaching the goal location. The second version, which is annotated with UCT, computes a heuristic based on the optimistic policy that was presented in Section 4.1. As the optimistic policy implementation is domain-specific and

	50 locations			100 locations		
	best subopt.	UCTB	UCT	best subopt.	UCTB	UCT
1	-214.3±7	-229.4±12	**-186.1**±7	-319.3±9	-464.5±21	**-286.8**±7
2	-375.4±7	-918.0±16	**-365.5**±7	-153.2±7	-185.9±12	**-151.5**±7
3	-268.5±7	-382.1±15	**-255.6**±7	-451.3±14	-811.1±39	**-412.2**±13
4	-241.6±7	-296.6±12	**-230.5**±7	-329.8±7	-552.3±20	**-314.3**±7
5	-229.5±7	-290.8±11	**-225.4**±7	**-348.1**±13	-654.6±43	-348.3±13
6	-238.3±9	-405.2±21	**-236.3**±8	-399.9±10	-741.7±29	**-396.2**±9
7	-209.3±7	-250.5±11	**-206.3**±7	-370.1±12	-716.2±39	**-358.2**±12
8	-300.4±8	-462.6±15	**-277.6**±8	-295.7±11	-405.7±25	**-293.3**±10
9	-238.1±9	-295.2±18	**-222.5**±9	-273.8±11	-382.1±27	**-262.0**±10
10	-249.0±6	-390.8±15	**-240.8**±6	-347.1±9	-735.1±32	**-342.3**±9
$\varnothing\,\mathcal{R}$	-256.4	-392.1±6	**-244.7**±2	-328.8	-564.9±10	**-316.5**±3
$\varnothing\,T_{\text{run}}$		40.48 s	13.99 s		162.00 s	43.74 s
$\varnothing\,T_{\text{dec}}$		2.65 s	1.23 s		7.38 s	2.87 s

Table 8.1: Average rewards with 95% confidence intervals for 1000 runs on ten roadmaps with 50 locations (left) and ten roadmaps with 100 locations (right). The columns labeled with "best subopt." repeat the highest result achieved by any of the optimistic policy, hindsight optimization, and optimistic rollout from Table 4.1.

based on the optimistic roadmap as described in Example 15 rather than an all-outcomes determinization, it can be computed efficiently. The number of virtual trials in the VT initialization has been optimized empirically and is set to $\chi = 20$ in the discussed results. We obtained comparable results for other values in the range from $\chi = 5$ to $\chi = 80$, but significantly worse performance for $\chi = 1$.

The results for both experiments that were discussed in Section 4.5 are given in Tables 8.1 and 8.2. We do not repeat all results for the optimistic policy, hindsight optimization, and optimistic rollout from Table 4.1, but combine the three suboptimal policies and present the highest result. (The entry can hence be regarded as the result of an artificial algorithm with an oracle that knows which of the optimistic policy, hindsight optimization, and optimistic rollout performs best in the instance.) UCT clearly dominates all other algorithms in both setups. It always provides the policy that yields the highest reward except for a single instance in each experiment where the difference to the best performance is below 1 and statistically not significant. Compared to the popular optimistic policy, UCT reduces the expected cost by 19.4% on the medium-sized instances and by 17.4% on the large instances. UCTB, on the other hand, does not fare well. Given that it is optimal in the limit as well, this must be due to a slow rate of convergence. The significant difference to

	best subopt.	UCTO	EXP	VOI
$p=0.1$ $\varnothing C$	-155.0±0.3	-155.5±0.3	**-154.9**±0.3	**-154.9**±0.3
$\varnothing T_{\text{run}}$		2.85 s	0.05 s	15.88 s
$\varnothing T_{\text{dec}}$		0.32 s	0.00 s	0.86 s
$p=0.3$ $\varnothing C$	-212.3±0.9	**-211.4**±0.9	-219.2±0.9	-218.9±0.9
$\varnothing T_{\text{run}}$		7.38 s	0.06 s	26.77 s
$\varnothing T_{\text{dec}}$		0.64 s	0.00 s	1.08 s
$p=0.5$ $\varnothing C$	-281.7±1.4	**-278.5**±1.4	-304.3±1.4	-309.3±1.5
$\varnothing T_{\text{run}}$		6.30 s	0.07 s	38.60 s
$\varnothing T_{\text{dec}}$		0.46 s	0.00 s	1.21 s
$p=0.6$ $\varnothing C$	-286.0±1.4	**-281.6**±1.5	-311.4±1.4	-316.4±1.5
$\varnothing T_{\text{run}}$		3.00 s	0.07 s	39.19 s
$\varnothing T_{\text{dec}}$		0.21 s	0.00 s	1.24 s

Table 8.2: Results for the CTP with sensing on the benchmarks of Bnaya et al. (2009) with fixed sensing cost of 5. Our algorithms (left half) do not make use of the sensing capabilities, while the others do. Each block summarizes the results for 30 roadmaps where all roads are blocked with the same probability.

UCT is not at all surprising since early trials of UCTB have to find the goal location with random walks.

Figure 8.1 shows the average reward in terms of the number of considered weathers (for hindsight optimization) and trials (for optimistic rollout, UCT and UCTB) for the benchmark instance 50-9. It takes roughly 10000 trials until UCT has converged, a slightly larger number than what is necessary for convergence of hindsight optimization and optimistic rollout. However, as UCT is optimal in the limit while hindsight optimization and optimistic rollout are not, UCT outperforms hindsight optimization after approximately 200 and optimistic rollout after roughly 1000 trials, which is a number of trials that is easily performed in less than a second. Figure 8.1 also indicates that the eventual convergence of the blind version of UCT to an optimal policy is of limited practical utility, as it would require a number of trials far beyond what is feasible in practice.

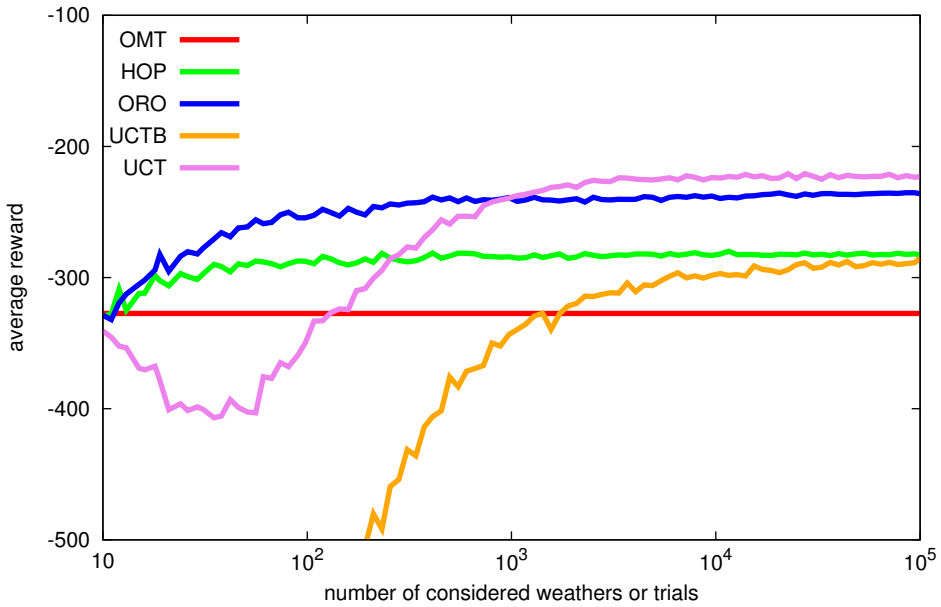

Figure 8.1: Average reward as a function of considered weathers or trials for benchmark instance 50-9.

8.2 THTS Recipes

8.2.1 Setting

Our first experiment shows that the use of an anytime optimal algorithm can lead to significantly improved behavior in comparison with a suboptimal alternative, but it also shows that it can take prohibitively long until near-optimal behavior is achieved. In our second experiment, we compare all anytime optimal THTS algorithms that are derived in Chapter 7 empirically. We use the benchmark sets of the latest two International Probabilistic Planning Competitions in 2011 and 2014, which sum up to a set of ten instances of each of the following twelve domains:

- ACADEMIC ADVISING (IPPC 2014)
- CROSSING TRAFFIC (IPPC 2011 & 2014)
- ELEVATORS (IPPC 2011 & 2014)
- GAME OF LIFE (IPPC 2011)
- NAVIGATION (IPPC 2011)
- RECON (IPPC 2011)

- SKILL TEACHING (IPPC 2011 & 2014)
- SYSADMIN (IPPC 2011)
- TAMARISK (IPPC 2014)
- TRAFFIC (IPPC 2011 & 2014)
- TRIANGLE TIREWORLD (IPPC 2014)
- WILDFIRE (IPPC 2014)

For each THTS recipe and parameter configuration that is considered in the following, we perform 100 runs and use the average over the 100 results to assess the quality of the evaluated policy. The experiments are performed on 2.60 GHz Eight-Core Intel Xeon computers with one task per core simultaneously and a memory limit of 2 GB per task. All algorithms are based on the same implementation in the state-of-the-art planning system PROST (Keller and Eyerich, 2012), so there should be no side effects of the implementation that bias the empirical evaluation towards one configuration or another. Recall that all tested configurations use

- the Monte-Carlo outcome selection $\mathfrak{O}_{\text{MC}}$,
- the expansion count trial length component $\mathfrak{T}_{\text{EC}}$, and
- the virtual trials initialization $\mathfrak{I}_{\text{VT}}$.

In the following, we denote a THTS algorithm with these three ingredients and backup function $\mathfrak{B}$, action selection $\mathfrak{A}$, and recommendation function $\mathfrak{R}$ with $\mathfrak{B}^{\mathfrak{A}}_{\mathfrak{R}}$ – for instance, the algorithm that uses UCB1 action selection, MC backups and the expected best arm recommendation function is denoted with $\text{MC}^{\text{UCB1}}_{\text{EBA}}$.

We compare the quality of algorithms with the evaluation schema that is also used at IPPC. The *instance score* of an MDP is computed for a given set of algorithms by normalizing all average rewards that are obtained in that MDP to a value in $[0, 100]$ such that best configuration is assigned a score of 100 and the worst a score of 0. For each algorithm, we furthermore compute the *domain score* by averaging over all ten instance scores that belong to the same domain, and the *total score* as the average of the twelve domain scores. The minimal average reward of all considered algorithms plays a role in the normalization process. To remove the influence of instances where all algorithms perform poorly, we use the reward of the artificial minimum policy of IPPC 2014 as a minimal limit, i. e., we also assign a score of 0 to all algorithms that perform worse than that policy. There are only a few THTS recipes and instances where this plays a role, but is is of strong relevance in the three ACADEMIC instances 5, 9, and 10 where none of our algorithms is able to achieve

a better average reward than the IPPC minimal policy, and where all configurations are hence assigned a score of 0. A hypothetical oracle that always selects the best THTS configuration would therefore achieve a domain score of 100 in all domains except for ACADEMIC, where it only achieves a domain score of 70, which combines to a total score of 97.5.

The goal of our main experiment is the comparison of all anytime optimal THTS algorithms that can be derived with the ingredients that are discussed in this thesis. All considered recipes use the MC outcome selection, the EC trial length component and the VT initialization, while the backup function, action selection and recommendation function are combined to the anytime optimal algorithms that are depicted in Table 7.1. Before we show our main experiment, there are some parameters where a default value is unclear. This does *not* include the trial length, which is set to $\nu = 1$ for all algorithms, a value that was shown to be superior to larger values in our earlier article on THTS (Keller and Helmert, 2013); the heuristic weight, which is set to $\omega = 0.5$ for all algorithms because the best results were obtained in an empirical evaluation of PROST prior to IPPC 2014 with that value; and of the number of virtual trials χ, which is set to $\chi = 1$ as this is a parameter that strongly interferes with the strengths and weaknesses of the proposed action selections and backup functions if it is set to a higher value. The remaining five parameters are:

- the exploration probability ϵ in the ϵ-G action selection;
- the fractional exponent γ in the ϵ_{RT}-G action selection;
- the temperature τ in the BE action selection;
- the fractional exponent γ in the RT-UCB action selection; and
- the learning rate decay η_{d} in the MC, TD, LSMC, LSTD, ESMC, ESTD, and QL backup functions.

All of these are core parameters of action selection and backup functions, which are two of the three ingredients where our anytime optimal algorithms differ. As we believe that it is important for a meaningful comparison to set these parameters to reasonable values, we start with an experiment that is designed to find good parameter values. To ensure that our main experiment does not get biased, we select two instances of each of the twelve domains, one each uniformly at random from the five instances with the lowest and the five instances with the highest indices (in most domains, the index roughly corresponds to the hardness of an instance). Moreover, we do not aim to find a global maximum for each parameter but aim for reasonable values for the four (pairwise independent) parameters of the action selections in a first step. We settle for only a few reasonable values such that we do not pick the best values for each configuration, but generalize over combinations of

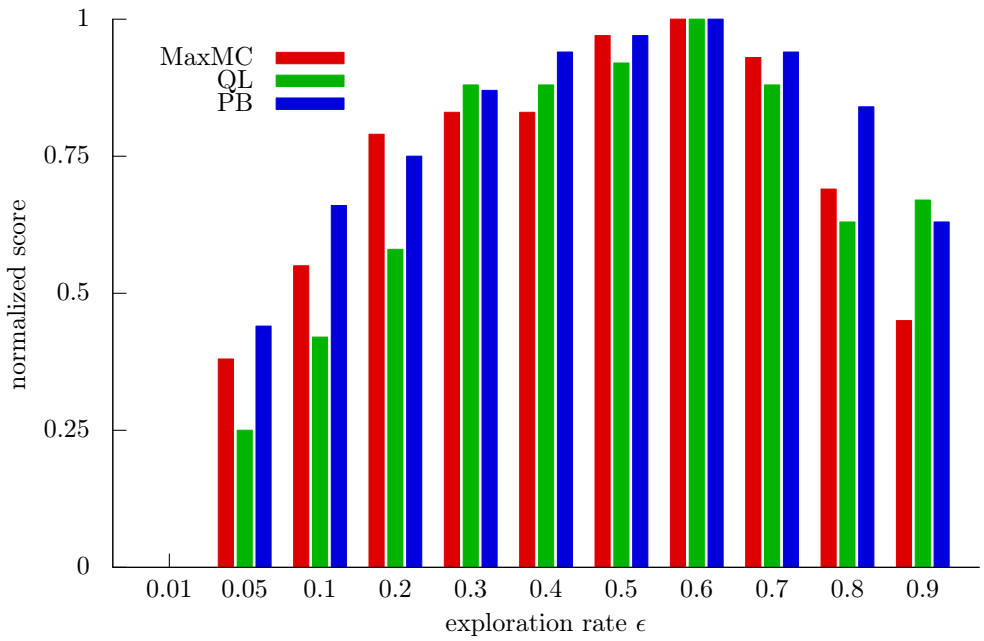

Figure 8.2: Normalized total score vs. exploration probability ϵ for the $\text{MaxMC}^{\epsilon\text{-G}}_{\text{EBA}}$, $\text{QL}^{\epsilon\text{-G}}_{\text{EBA}}$, and $\text{PB}^{\epsilon\text{-G}}_{\text{EBA}}$ THTS algorithms.

ingredients when possible. In a second step, we optimize the learning rate decay independently (i. e., with the fixed values of the first step), which might lead to local maxima as dependencies between the parameters are ignored. We furthermore only use the expected best arm recommendation function for the initial experiments and assign the same parameter values to recipes that differ only in the recommendation function.

8.2.2 Core Parameter

Exploration Probability of ϵ-greedy Action Selection. Let us start with the exploration probability of the ϵ-G action selection, which is used in only three anytime optimal THTS algorithms (if the recommendation function is ignored, otherwise there are five) where it is combined with the three off-policy backup functions QL, MaxMC, and PB. We computed policies for the three configurations $\text{PB}^{\epsilon\text{-G}}_{\text{EBA}}$, $\text{MaxMC}^{\epsilon\text{-G}}_{\text{EBA}}$, and $\text{QL}^{\epsilon\text{-G}}_{\text{EBA}}$ with a variety of exploration probabilities in $[0.01, 0.9]$. As we are interested in good values for each THTS recipe, we computed IPPC scores for each recipe independently. Figure 8.2 shows the total scores over all domains, but they are additionally normalized to values between 0 and 1, which does not alter the relative order of the configurations for each recipe (i. e., the relative length of bars of the same color is in relation to the respective total scores), but allows us to display the (incomparable)

results in a single figure without hinting at wrong conclusions.

All three THTS recipes have a common maximum for $\epsilon = 0.6$. This is surprisingly high at first glance, but it can be explained by the fact that all three recipes use an off-policy backup function that can benefit from exploration without suffering from biased action-value estimates. Even though all three configurations share a common maximum, it does not appear to be the case that there is a universal value for ϵ that works good across all domains: the best values for the exploration probability in the individual domains range from $\epsilon = 0.01$ in TRIANGLE to $\epsilon = 0.9$ in ELEVATORS, and there is a domain for almost all values that are considered in this experiment where the value performs best. Even though this is a problem of the ϵ-G action selection that has to be kept in mind, it is also the case that all results above a normalized score of roughly 0.75 (i. e., $\epsilon \in [0.3, 0.8]$ for PB and $\epsilon \in [0.3, 0.7]$ for QL and MaxMC) yield total scores that are not unreasonably far from the top result.

Temperature of Boltzmann Exploration. The BE action selection (with constant temperature) also combines only with the three off-policy backup functions to an anytime optimal algorithm. Figure 8.3 depicts normalized total scores in dependence on the temperature $\tau \in [0.01, 5]$. Like in the previous experiment, all three THTS algorithms share a common maximum, which can be found for $\tau = 0.15$. Unlike in the previous experiment, the best values for τ w.r.t. domain scores are fairly homogeneous, though: if BE is combined with a PB or MaxMC backup function, the best policy for eight of twelve domains is achieved with a temperature in $[0.1, 0.25]$, and the domain scores for the other four domains with $\tau \in [0.1, 0.25]$ are close to the best as well. While the picture is not quite as clear for $\text{QL}^{\text{BE}}_{\text{EBA}}$, it is still more stable than the corresponding algorithm with ϵ-G action selection. In general, it appears that the BE action selection leads to fairly stable results over all domains, which cannot be taken for granted as the twelve domains are far from similar.

Decrease schema of ϵ_{RT}-G. The γ parameter of the ϵ_{RT}-G action selection influences the trade-off between exploration and exploitation since ϵ decreases slower when γ is smaller, which leads to an higher expected exploration rate. We performed experiments with values for the fractional exponent γ in $[0.025, 0.5]$. The best result for the $\text{PB}^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$, $\text{MaxMC}^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$, and $\text{QL}^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$ algorithms is such that the exploration rate is comparably high with $\gamma \in [0.05, 0.1]$. Similar to the experiment with the ϵ-greedy action selection, the best results for the individual domains are scattered over all possible values. The ϵ_{RT}-G action selection can also be combined with the on-policy and selective backup functions in an optimal way, all of which yield the highest total score for $\gamma \in [0.1, 0.3]$. To pick as few values as possible without sacrificing too much performance, we decided to settle for values of $\gamma = 0.05$ for the off-policy backup functions, and of $\gamma = 0.2$ for the rest.

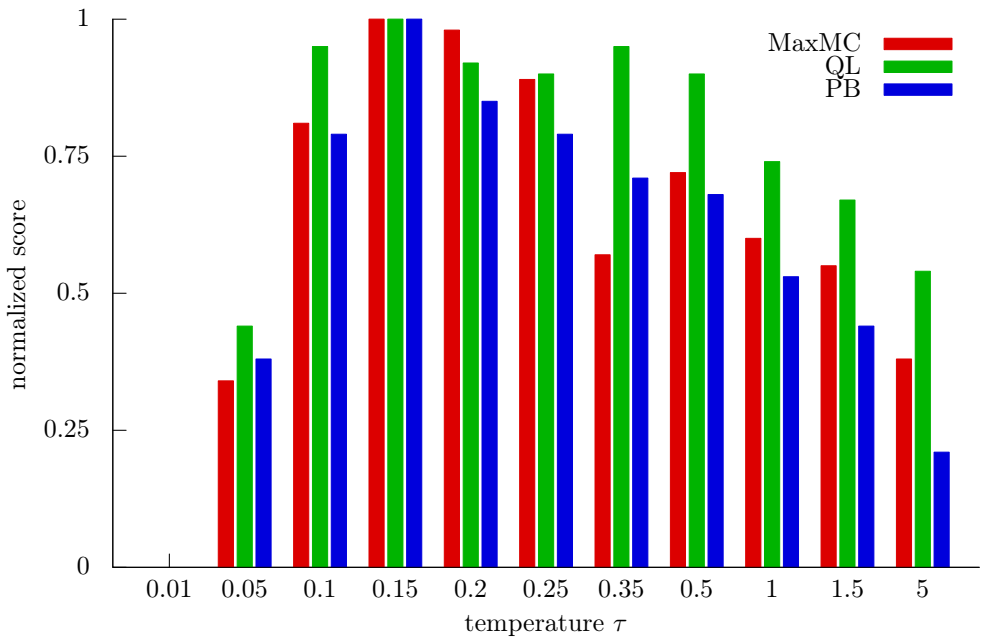

Figure 8.3: Normalized total score vs. temperature τ for the $\text{MaxMC}^{\text{BE}}_{\text{EBA}}$, $\text{QL}^{\text{BE}}_{\text{EBA}}$, and $\text{PB}^{\text{BE}}_{\text{EBA}}$ THTS algorithms.

Please observe that this means that the ϵ parameter decreases *slower* in ϵ_{RT}-G than in ϵ_{LOG}-G as the number of trials that is necessary until $\mathcal{L}^k(c)^\gamma > \ln(\mathcal{L}^k(c))$ holds is larger than the number of trials that is performed by our algorithms in each decision step (see our discussion of the RT-UCB action selection in Section 6.5 for details). This means that, in combination with the off-policy backup functions, the ϵ-greedy action selections are such that the exploration rate of ϵ-G (with $\epsilon = 0.6$) and ϵ_{RT}-G (with $\gamma = 0.05$) is the largest (with an average exploration rate of roughly 50% in the initial state), which is followed by ϵ_{LOG}-G with roughly 10% exploration rate and ϵ_{LIN}-G, where the ϵ parameter decreases so quickly that the average exploration rate is below 1%. For on-policy and selective backups, the highest amount of exploration is achieved with ϵ_{RT}-G (with $\gamma = 0.2$) with an average exploration rate of roughly 16% in the initial state, followed by ϵ_{LOG}-G and ϵ_{LIN}-G with identical rates.

Decrease schema of RT-UCB. The final action selection parameter is the fractional exponent γ of the RT-UCB action selection. To our surprise, it has a comparably low influence on the total score, with best results among all algorithms for $\gamma \in [0.3, 0.5]$. As the differences within the interval are negligible, we decided to settle for $\gamma = \frac{1}{e} \approx 0.37$ for all algorithms and in all further experiments, a value that is such that RT-UCB never explores less than UCB1.

	ACADEMIC	CROSSING	ELEVATORS	GAME	NAVIGATION	RECON	SKILL	SYSADMIN	TAMARISK	TRAFFIC	TRIANGLE	WILDFIRE	Total
$MC^{\epsilon_{LOG}\text{-G}}_{EBA}$	35	51	37	56	37	97	41	60	74	54	52	59	54
$MC^{\epsilon_{LOG}\text{-G}}_{MPA}$	26	53	78	60	35	93	49	64	78	73	47	89	62
$MC^{\text{RT-UCB}}_{EBA}$	28	67	47	87	43	93	78	91	84	77	48	87	69
$MC^{\text{RT-UCB}}_{MPA}$	28	64	71	87	47	91	80	86	88	73	47	81	70
$ESMC^{\epsilon_{LIN}\text{-G}}_{EBA}$	28	64	53	19	34	84	13	39	39	48	65	63	46
$ESMC^{\epsilon_{LIN}\text{-G}}_{MPA}$	22	68	83	26	31	88	19	43	52	70	64	89	55
$QL^{\epsilon_{LOG}\text{-G}}_{EBA}$	34	88	54	54	31	91	54	31	61	48	51	37	53
$QL^{\epsilon_{LOG}\text{-G}}_{MPA}$	21	87	87	47	33	85	50	30	62	49	45	55	54
$MaxMC^{\text{BE-DT}}_{EBA}$	27	93	61	82	35	64	66	69	71	59	49	54	61
$MaxMC^{\text{BE-DT}}_{MPA}$	32	90	81	77	36	68	59	66	68	50	53	40	60

Table 8.3: Influence of the recommendation function on the performance of a THTS recipe.

Learning Rate Decay. The learning rate decay of the MC, TD, LSMC, LSTD, ESMC, ESTD, and QL backup function is the last parameter we have been looking at prior to the main experiment. Given how different the action selections are, the results have been surprisingly homogeneous for all recipes that use an MC-based backup function (i. e., MC, LSMC, or ESMC), namely that it is always best to set $\eta_{\text{d}} = 1$ and use no decay at all. This underlines the behavior of the MC-based backup functions in the example from Chapter 6, where the backup functions adapt quickly to changed results. For the remaining backup functions, the result is slightly different. We have seen that both TD-based backup functions and QL backups are much more stable than the MC-based backup functions and take a long time until estimates adapt to significantly different results. The results of this experiment indicate that the rate of adaption is too low, as the total scores of most algorithms with TD, LSTD, ESTD, and QL backups improve with a decreasing learning rate decay until $\eta_{\text{d}} = 0.5$, which is the best value for most recipes. However, the picture is not as clear as it was with the MC-based backup functions, and the action selection plays a more important role.

8.2.3 Main Experiment

Our theoretical evaluation in the previous section has provided us with a large number of anytime optimal THTS algorithms: Table 7.1 shows 52 green

icons (which can be paired with two recommendation functions) and 11 yellow ones (which are anytime optimal only in combination with the expected best arm recommendation function), which combines to 115 different THTS recipes that are anytime optimal. In our main experiment, we ran each of the anytime optimal algorithms with values for the core parameters as described above and computed IPPC scores by considering all of these recipes (the scores are hence comparable to each other). As the result table is really large, we decided to move it to Appendix A. However, we repeat the most relevant entries in the following discussion.

Influence of the recommendation function

Before we have a look at the best performing algorithms in our main experiment, let us first analyze the influence of the recommendation function. Table 8.3 gives an excerpt of the results of our main experiment that shows the most important insights w.r.t the influence of the recommendation function. All combinations that use an on-policy or selective backup function have in common that there is a clear tendency towards better performance if the most played arm recommendation function is used, and the effect is strongest for the algorithms that use an MC-based backup function. The domain that is foremost responsible for this is the ELEVATORS domain, where every single THTS recipe $\mathfrak{B}^{\mathfrak{A}}_{\text{MPA}}$ outperforms the corresponding recipe $\mathfrak{B}^{\mathfrak{A}}_{\text{EBA}}$ with identical backup function $\mathfrak{B}$ and action selection $\mathfrak{A}$ by far (there are cases like the depicted $\text{MC}^{\epsilon_{\text{LOG}}\text{-G}}_{\mathfrak{R}}$ where the difference is more than 40 points). Similar results – albeit to a lesser extent – can be observed in the TAMARISK, TRAFFIC, and WILDFIRE domains. A possible explanation for the superiority of the most played arm recommendation function is that all of these domains have in common that things can only become worse with more information due to stochastic events and independently from the agent's policy. Therefore, an action-value estimate can only get worse the more often it is selected, and it is possible that a point is reached where the best action has been examined so much more thoroughly that it appears worse than an inferior yet less often visited action.

Since there is no domain where $\mathfrak{B}^{\mathfrak{A}}_{\text{EBA}}$ performs significantly better than the corresponding $\mathfrak{B}^{\mathfrak{A}}_{\text{MPA}}$ recipe over all action selections $\mathfrak{A}$, the empirical data clearly hints at the fact that the most played arm recommendation function is indeed a better choice in combination with on-policy and selective backup functions. At first glance, this seems to match the theoretical results of Bubeck et al. (2009) who show the same for the MAB with access to a generative model, but their results can not be transferred to MDPs without further considerations. A theoretical confirmation of the assumption is certainly an interesting avenue of research for future work.

Another, independent observation is that the set of algorithms that is described by $\text{ESMC}^{\mathfrak{A}}_{\text{MPA}}$ yields a higher total score than the respective $\text{LSMC}^{\mathfrak{A}}_{\mathfrak{R}}$,

	ACADEMIC	CROSSING	ELEVATORS	GAME	NAVIGATION	RECON	SKILL	SYSADMIN	TAMARISK	TRAFFIC	TRIANGLE	WILDFIRE	Total
PB^{BE}_{EBA}	35	**97**	72	82	**91**	84	**85**	71	79	51	**76**	55	**73**
$PB^{RT\text{-}UCB}_{EBA}$	30	92	72	83	88	67	**81**	67	74	63	**72**	63	**71**
MC^{UCB1}_{MPA}	27	65	78	**86**	45	92	77	**89**	**86**	71	46	84	**70**
$MC^{RT\text{-}UCB}_{MPA}$	28	64	71	**87**	47	91	80	**86**	**88**	73	47	81	**70**
PB^{UCB1}_{EBA}	24	91	69	77	**94**	71	77	69	66	64	**74**	62	**70**
$MC^{BE\text{-}DT}_{MPA}$	26	61	84	84	43	86	**83**	81	**82**	**74**	47	84	**70**
$PB^{BE\text{-}DT}_{EBA}$	30	89	73	84	89	67	73	68	70	56	63	54	68
$MaxMC^{BE}_{EBA}$	36	**95**	68	84	40	85	72	68	81	59	**67**	52	67
$PB^{\epsilon\text{-}G}_{EBA}$	**53**	93	76	85	**93**	36	79	64	73	55	52	43	67
$PB^{\epsilon_{RT}\text{-}G}_{EBA}$	**44**	89	70	82	**95**	56	75	62	71	45	52	43	65
QL^{BE}_{EBA}	31	**97**	69	85	37	90	73	67	72	49	56	51	65
$TD^{BE\text{-}DT}_{MPA}$	26	68	78	**87**	25	78	**85**	74	73	66	38	68	64

Table 8.4: Top 12 anytime optimal THTS algorithms. Best results (among all 115 algorithms) for each column are in red, and top five in bold.

$LSTD^{\mathfrak{A}}_{\mathfrak{R}}$, and $ESTD^{\mathfrak{A}}_{\mathfrak{R}}$ algorithms *for all* action selections $\mathfrak{A}$ and for both considered recommendation functions, and each $ESMC^{\mathfrak{A}}_{MPA}$ is a good representative of the other THTS algorithms that use a selective backup function as its performance in the individual domains is similar to the performance of the other THTS algorithms. It is not unexpected that the eager selective backup function improves over its lazy pendant because more trials of the same quality are considered. We believe that the fact that the poor behavior of selective TD backups is because they adapt slower to results that are different from the current estimate than MC backups, which combines poorly with the fact that only a fraction of all trials are considered in the backup process.

Finally, there is no clear advantage of one recommendation function over the other if combined with the off-policy backup functions QL or MaxMC (recall that PB cannot be paired with most played arm to an anytime optimal algorithm). In combination with one of the ϵ-greedy action selections, the algorithms that are paired with the most played arm recommendation function yield significantly higher scores in the ELEVATORS and WILDFIRE domains as well, while the remaining scores (including TAMARISK and TRAFFIC) tend slightly towards the usage of the expected best arm recommendation function. And in combination with the BE, BE-DT, RT-UCB, or UCB1 action selection it is only the ELEVATORS domain where the most played arm recommendation

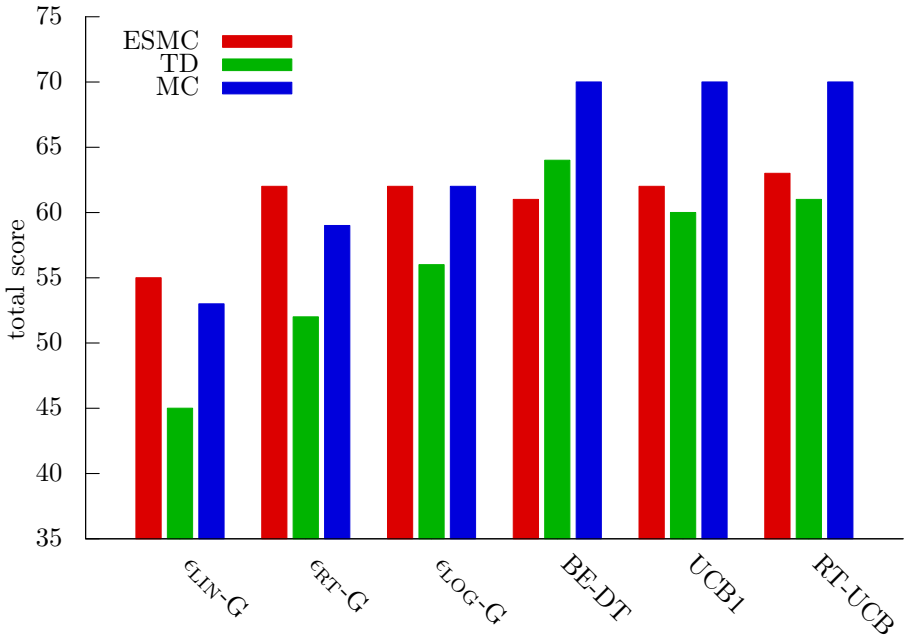

Figure 8.4: Total scores for the $\text{ESMC}^{\mathfrak{A}}_{\text{MPA}}$, $\text{TD}^{\mathfrak{A}}_{\text{MPA}}$, and $\text{MC}^{\mathfrak{A}}_{\text{MPA}}$ THTS algorithms in dependence on the action selection $\mathfrak{A}$

looks better. For the total scores, this means that the $\text{QL}^{\mathfrak{A}}_{\text{EBA}}$ and $\text{MaxMC}^{\mathfrak{A}}_{\text{EBA}}$ algorithms perform equally good to slightly better than their respective $\text{QL}^{\mathfrak{A}}_{\text{MPA}}$ and $\text{MaxMC}^{\mathfrak{A}}_{\text{MPA}}$ counterparts if $\mathfrak{A}$ is one of the ϵ-greedy action selections in most of the cases, and equally good to slightly worse for the majority of the recipes with BE, BE-DT, UCB1 and RT-UCB action selection.

Influence of the action selection

Table 8.4 shows the twelve THTS recipes that performed best in our final experiment.[1] At the top of the list is a recipe that uses the BE action selection and PB backups (and hence the expected best arm recommendation function) – one of ten recipes in the list that uses an action selection that has not been applied to the benchmark set of IPPC 2011 and 2014 prior to this thesis. Especially the two variants of Boltzmann Exploration and the RT-UCB action selection are represented in the list disproportionately often. And at the same time, it is not the case that only a few action selection strategies dominate, since all but the three action selections UNI, ϵ_{LOG}-G, andϵ_{LIN}-G are contained in the list of best performing recipes. Let us therefore have a closer look at the performance of the nine action selection strategies of this thesis.

[1]For each combination of action selection and backup function, only the recipe with the recommendation function that yields a higher total score is considered.

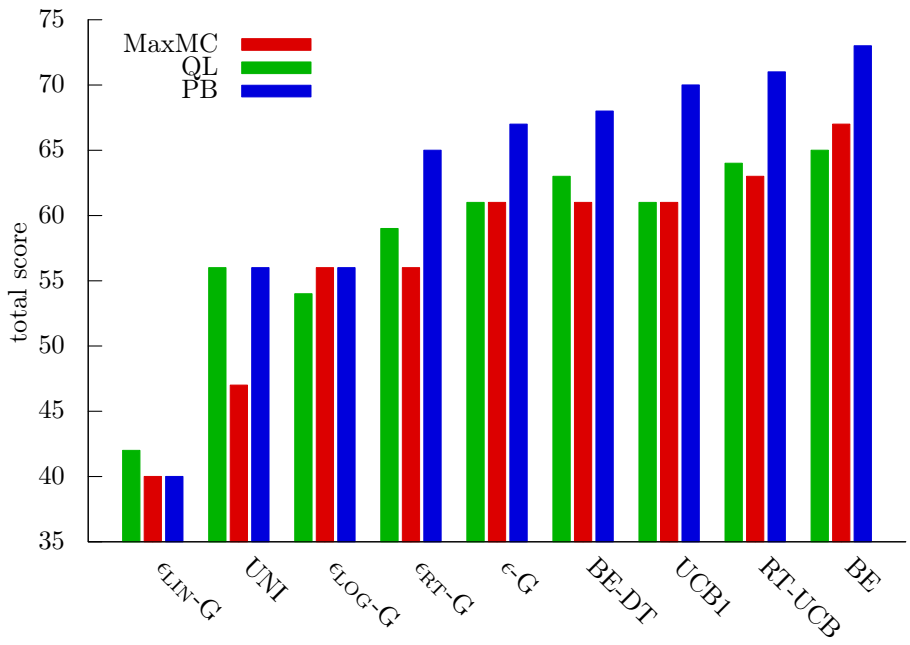

Figure 8.5: Total scores for the $\text{MaxMC}^{\mathfrak{A}}_{\mathfrak{R}}$, $\text{QL}^{\mathfrak{A}}_{\mathfrak{R}}$, and $\text{PB}^{\mathfrak{A}}_{\mathfrak{R}}$ THTS algorithms in dependence on the action selection $\mathfrak{A}$ and with the recommendation function $\mathfrak{R}$ that yields the highest score.

Figures 8.4 and 8.5 show the performance of different recipes in dependency on the used action selection – Figure 8.4 shows the total scores for the THTS recipes with on-policy and selective backup functions, and Figure 8.5 shows the total scores for the THTS algorithms that use an off-policy backup function. Except for ESMC backups, all recipes follow the same pattern: if the BE action selection combines to an anytime optimal algorithm, it performs best, followed by RT-UCB, UCB1, BE-DT, and ϵ-G (the latter for the Offline backup functions only) which yield similar total scores. All ϵ-greedy action selections with decaying ϵ and UNI perform significantly worse, with ϵ_{LIN}-G clearly at the end of the scale. Even though the results are clear, we have to keep in mind that an important parameter of both top performers, BE and RT-UCB, has been optimized prior to the main experiment, and that the instances that were used there are also considered here. While this should not alter the results by much, it certainly incurs a slight bias in favor of both action selections. Hence, and as the gap to UCB1 and BE-DT is comparably small, we believe that it is possible to introduce similar parameters for UCB1 and BE-DT (e. g., the base of the logarithm for UCB1 and a constant factor for BE-DT), optimize them and achieve comparable results.

The influence of the action selection on ESMC (and the other selective backup functions) is fairly different, as all action selections except for ϵ_{LIN}-G

yield similar results (and ϵ_{LIN}-G also achieves comparably high scores in combination with ESMC). It is likely that this is because all selective backup functions have in common that the decision to consider trials (or not) serves a comparable role w.r.t. the action-value estimates as the action selection ingredient itself. On the one hand, this has a positive effect on all ϵ-greedy action selections with decaying ϵ, all of which perform best in combination with ESMC backups. On the other hand, it does not seem to improve the results with BE-DT, UCB1 and RT-UCB action selections, but it results in significantly lower total scores than MC backups. Nevertheless, it shows that separation of concerns can have positive effects on the performance of THTS algorithms, and ways to transfer the advantages to stronger action selections seem like an interesting research direction for future work.

Let us also have a look at the domain and instance scores in dependence on the action selection. Figure 8.6 shows the number of domains and instances where a recipe with the given action selection is among the algorithms with the highest score. A first observation is that both pie charts are very multicolored, which is not entirely surprising for the instance scores where the confidence intervals are comparably large and the results hence quite uncertain. However, this is not true for the domain scores, where the 95% confidence intervals are smaller than ± 1 consistently and where better results do clearly hint at some fundamental advantage. Even though all action selections besides UNI are best in at least one domain, we can nevertheless not detect that there are domains where it is the action selection that is responsible for the fundamental advantage, as there is not a single domain where an action selection is particular good independently from the selected backup functions. Nevertheless, both the total results and the results on domains where one action selection dominates the others clearly indicate that some action selections – in particular the two variants of Boltzmann Exploration and RT-UCB – allow for stronger THTS recipes than others.

Influence of the backup function

In general, the picture of the best performers is pretty clear w.r.t the backup functions, with the seven best performers all being PB and MC backup functions. However, it is not the case that other backup functions are nowhere close. The best MaxMC recipe is the eighth best result, the best QL the eleventh, the best TD the twelfth and the best ESMC (14-th) and LSMC (17-th) just missed the list of the best recipes by a margin. In fact, it is only the two selective TD backup functions that do not perform particularly good in any combination.

As the action selections are not responsible for good or bad behavior in single domains, it must be the combination of action selection and backup function. Figure 8.7 shows the number of domains and instances where a THTS algorithm with the respective backup function is among the best per-

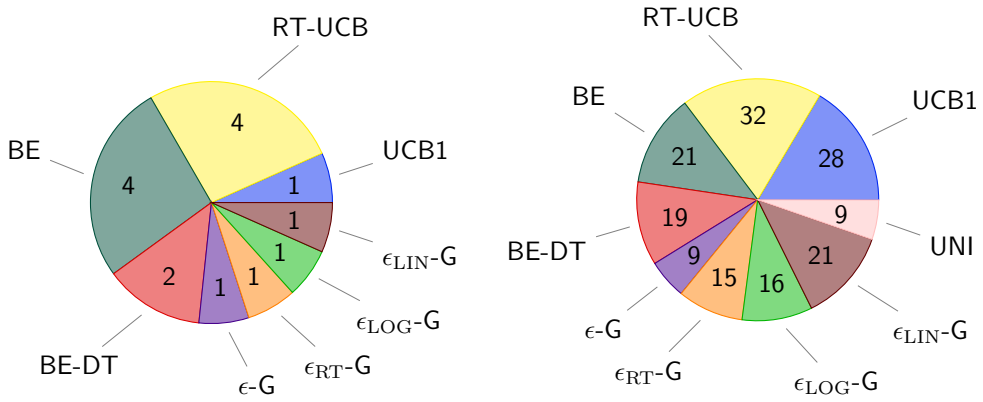

Figure 8.6: Number of domains (left) and instances (right) where a recipe with the given action selection is among the algorithms with the highest score.

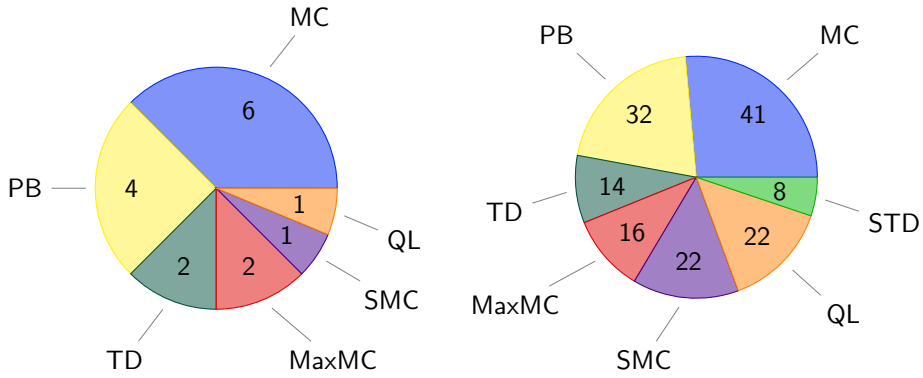

Figure 8.7: Number of domains (left) and instances (right) where a recipe with the given backup function is among the algorithms with the highest score. SMC (STD) contains both ESMC (ESTD) and LSMC (LSTD) backups.

formers. Again, both charts are fairly multi-colored, and again, all (except TD-based selective backups) are best in at least one domain. However, this time there is a clear dependency between the behavior in a domain and the backup function. There are a few domains – most notably CROSSING, ELEVATORS, and TRIANGLE – where the three off-policy backup functions perform best, followed by the selective backups and then the two on-policy backup functions. PB backups dominate another two domains, namely NAVIGATION and SKILL. And MC (and to a lesser extent TD) backups perform significantly better in GAME, RECON, SYSADMIN, TAMARISK, TRAFFIC, and WILDFIRE.

It is noticeable that all domains where off-policy backups dominate other backup functions are domains where best actions have a significantly higher expected reward than non-optimal actions. It is not hard to imagine that using the maximum of all actions in decision nodes is a reasonable choice. On-policy

	ACADEMIC	ELEVATORS	GAME	NAVIGATION	RECON	SKILL	TRIANGLE	WILDFIRE	Total
$PB_{EBA}^{\epsilon\text{-G}}$	7.2	12.2	1.4	25.1	9.4	14.8	15.8	5.4	8.0
$PB_{EBA}^{\epsilon_{LIN}\text{-G}}$	2.7	1.7	0.0	0.5	3.9	4.8	0.8	3.0	1.6
$PB_{EBA}^{\epsilon_{LOG}\text{-G}}$	6.5	8.6	0.8	13.2	5.9	17.6	3.6	5.0	5.4
$PB_{EBA}^{\epsilon_{RT}\text{-G}}$	**7.3**	12.2	1.5	25.1	9.4	14.8	15.8	5.4	8.0
PB_{EBA}^{UNI}	5.6	12.7	1.6	25.7	9.7	13.6	16.4	4.6	7.9
$PB_{EBA}^{RT\text{-UCB}}$	6.7	13.7	3.4	20.4	10.9	27.0	18.9	5.6	9.3
$PB_{EBA}^{BE\text{-DT}}$	6.6	12.8	2.3	23.7	11.2	21.0	18.9	5.0	8.8
PB_{EBA}^{BE}	7.0	12.9	2.4	26.9	**11.4**	27.3	**22.9**	**5.9**	10.1
PB_{EBA}^{UCB1}	7.0	**14.0**	**3.6**	**35.1**	11.2	**28.9**	19.2	4.7	**10.7**

Table 8.5: Average number of steps-to-go in the first root state that is labeled as solved. All instances have a horizon of $\mathcal{H} = 40$.

backup functions, on the other hand, dominate when all actions are highly uncertain and have many outcomes that are often caused by exogenous effects. We believe that it is advantageous in these domains to somehow use as much of the collected information as possible, and a higher amount of information is propagated in the tree by considering the results of all actions instead of just the best one. Finally, PB has an additional advantage over all other backup functions as it is able to label states as solved. This especially plays a role in small domains or in decisions with only few steps-to-go, and explains the good performance in the NAVIGATION and SKILL domains. Table 8.5 shows the average number of steps-to-go in the first root state that is labeled as solved in a run of all algorithms that use PB backups. Note that the horizon is set to 40 in all IPPC benchmarks, so PB_{EBA}^{UCB1} acts provably optimal almost from the beginning in all instances of NAVIGATION and after little more than 10 steps on average in the SKILL instances. (In practice, it acts optimally from the first step in the simpler instances and needs longer to solve the root state in harder instances.) Other domains where recipes with PB backups profit from the solve labeling procedure are ELEVATORS, RECON, and TRIANGLE.

In comparison to the algorithms that were presented in our first analysis of the THTS framework (Keller and Helmert, 2013) where UCT* (which is PB_{EBA}^{UCB1} here) dominated all other algorithms, it is apparently the case that the additional ingredients that are considered here are a noteworthy addition. The action selection strategies BE and RT-UCB improve slightly yet notable over the state-of-the-art algorithm UCT*, and the most played arm recommendation function improves the performance of algorithms that use MC backups

up to a point where some are competitive with the best recipes. This concludes our analysis of the Trial-based Heuristic Tree Search framework, and we turn our attention to a problem we became aware off during our participation at the last International Probabilistic Planning Competition in 2014.

CHAPTER 9

The MDP-Evaluation Stopping Problem

Since it is often impossible or intractable to evaluate MDP algorithms based on a theoretical analysis alone, the International Probabilistic Planning Competition (IPPC) was introduced to allow a comparison based on experimental evaluation. Just as our evaluation in the previous chapter, the idea is to approximate the quality of an MDP solver by performing a sequence of runs on a problem instance, and by using the average of the obtained results as an approximation of the expected reward. Following the optimal policy (i.e., the policy that maximizes the expected reward) leads to the best result in such a setting. In the final chapter of this thesis, we discuss the MDP-Evaluation Stopping Problem (MDP-ESP) (Keller and Geißer, 2015), an optimization problem that we got aware of in our preparation for IPPC 2014, where each solver had to perform *at least* 30 runs within a given time limit, while only *the last* 30 runs were used for evaluation.

The decision when to stop the sequence of runs could be taken at any point of the evaluation with knowledge of the rewards that were collected in all previous runs. We show in Section 9.1 how the MDP-ESP can be constructed as a meta-MDP with actions that correspond to the application of a policy on the base-MDP. Interestingly, the computation of the optimal policy is no longer the only objective of participating planners, and the fact that the execution of other policies on the base-MDP can be part of an optimal strategy for the MDP-ESP leads to a problem that is intractable in practice.

However, there are special cases where the MDP-ESP can be reduced to an instance of an optimal stopping problem. We analyze these cases theoretically in Section 9.2. Two functions that depend only on the number of remaining runs – one that specifies the target reward that is necessary to stop, and one that gives the policy that is applied otherwise – suffice to describe an optimal policy. Based on these observations, we present four approximate

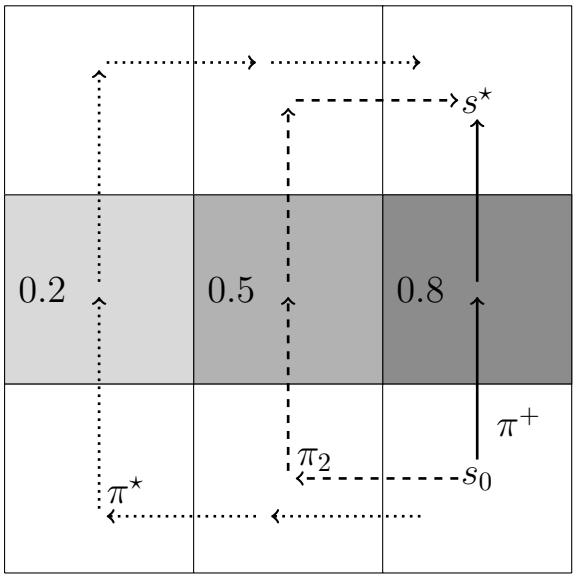

Figure 9.1: An example instance of the NAVIGATION domain with the policies $\pi^\star$ (dotted), π_2 (dashed) and π^+ (solid).

algorithms for the general problem in Section 9.3. The first strategy can be applied even when a policy for the base-MDP is computed online and not known in advance, while two other algorithms require the knowledge of the optimal policy and its expected reward. We show that the expected reward of the optimal policy is a lower bound for the expected performance of both strategies.

Our final algorithm switches between the application of the optimal policy and the policy with the highest possible outcome, which can be computed without notable overhead in the THTS framework. We show theoretically and empirically that all algorithms outperform the naïve base approach in Section 9.4 that ignores the potential of optimizing evaluation runs in hindsight, and that it pays off to take suboptimal base policies in addition to the optimal one into account. Finally, we discuss the influence of the MDP-ESP on the results of IPPC 2014, and propose applications of our algorithms by discussing them in the context of related work.

9.1 The MDP-ESP

Each policy π induces a set of outcomes $\mathcal{O}^\pi$, which consists of each accumulated reward r that can be achieved under application of π paired with the probability of r, i. e., $\mathcal{O}^\pi = \{(r, \mathbb{P}[R(\phi^\pi) = r]) \mid \mathbb{P}[R(\phi^\pi) = r] > 0\}$. We call the highest possible outcome $P^\pi := \max_{(r,p)\in\mathcal{O}^\pi} r$ of a policy π the potential of π. We abbreviate the policy with the highest potential among all policies

with π^+. Moreover, we abbreviate the expected reward of $\pi^\star$ (π^+) with $V^\star$ (V^+), its potential with $P^\star$ (P^+), its set of outcomes with $\mathcal{O}^\star$ ($\mathcal{O}^+$) and a run under the policy with $\phi^\star$ (ϕ^+).

The example MDP that is used in this Chapter is depicted in Figure 9.1. It shows an instance of the NAVIGATION domain of IPPC 2011, where an agent is initially located in grid cell s_0 and aims to reach cell $s^\star$ by moving within the grid. On its way, the agent has to cross the middle row at some point, where it gets stuck with increasing probability from left (20%) to right (80%). The agent has no possibility to break free once it is stuck, and it receives a reward of -1 in each step unless it is located in $s^\star$. If we consider the IPPC horizon of $\mathcal{H} = 40$, the agent receives an accumulated reward of $R(\phi^\star) = -6$, $R(\phi^{\pi_2}) = -4$, and $R(\phi^+) = -2$ if it successfully passes the middle row, and of -40 if it gets stuck regardless of the applied policy. The expected reward of $\pi^\star$ is $V^\star = -12.8$, and it induces the set of outcomes $\mathcal{O}^\star = \{(-6, 0.8), (-40, 0.2)\}$ with potential $P^\star = -6$. For π^+, we have $V^+ = -32.4$, $\mathcal{O}^+ = \{(-2, 0.2), (-40, 0.8)\}$, and $P^+ = -2$.

The problem we have faced at IPPC 2014 is the MDP-ESP^u_k, where a sequence of at least $k > 0$ and at most $u \geq k$ runs is performed on an MDP. A strategy σ assigns policies $\pi_1, \ldots, \pi_n$ to runs on the MDP and stops the evaluation after n ($k \leq n \leq u$) runs. The objective is to find a strategy σ that maximizes the average accumulated reward of the last k runs, i.e., where

$$\mathcal{R}^u_k(\sigma) := \frac{1}{k} \cdot \sum_{i=n-k+1}^{n} R(\phi_i^{\pi_i})$$

is maximal in expectation. The quality of a strategy σ is hence measured in terms of its expected average reward $\mathbb{E}[\mathcal{R}^u_k(\sigma)]$.

An instance of the MDP-ESP not only optimizes the evaluation of a sequence of policies on a base-MDP, it can be described in terms of a meta-MDP itself. A state in the meta-MDP is given by a sequence of rewards $(r_1, \ldots, r_n)$, where $r_i := R(\phi_i^{\pi_i})$ for $i = 1, \ldots, n$ is the accumulated reward of the runs that were performed before reaching a state. The meta-MDP provides an action a_π for each policy $\pi \in \Pi$, which encodes the execution of policy π on the base-MDP. Furthermore, there is a single action $a_\otimes$ that encodes the decision to stop the MDP-ESP and evaluate the meta-run under σ based on the result of the last k runs on the base-MDP. We describe the transition function of the meta-MDP in terms of its actions: $a_\otimes$ is not applicable in a state $(r_1, \ldots, r_n)$ if $n < k$, and it is the only applicable action in a state $(r_1, \ldots, r_u)$. Its application leads deterministically to an absorbing terminal state and yields a reward $R((r_1, \ldots, r_n), a_\otimes) = \frac{1}{k} \cdot \sum_{i=n-k+1}^{n} r_i$. The application of an action a_π in state $(r_1, \ldots, r_n)$ incurs no reward and leads to a state $(r_1, \ldots, r_n, r)$, where r is drawn according to the outcome function $r \sim \mathcal{O}^\pi$ of the executed policy π.

9.2 Theoretical Analysis

Upper and Lower Bounds. All optimization problems that were discussed in Chapter 5 have in common that the theoretical upper bound of the expected reward of a policy is less than or equal to the expected reward $V^\star$ of the optimal policy $\pi^\star$. This is different in the MDP-ESP. Since the agent is allowed to decide in hindsight if the last k runs were good enough to be used for evaluation, there are strategies that allow an expected performance that is at least as good as $V^\star$ for all instances of the MDP-ESP^u_k. However, it is impossible to achieve a result that is higher than P^+.

Theorem 16. *$V^\star \leq \max_\sigma \mathbb{E}[\mathcal{R}^u_k(\sigma)] \leq P^+$ for all $k > 0$ and $u \geq k$.*

Proof: We start with a discussion of the lower bound $V^\star$ by considering the subset of instances where $u = k$. The MDP-ESP^k_k, where each performed run is an evaluation run reduces to DMPlan, and the optimal strategy is hence the strategy that only executes $\pi^\star$. (We denote the strategy that executes $\pi^\star$ in each run and never stops prematurely with $\sigma_{\pi^\star}$ in this proof). Since the expected reward of each run under $\pi^\star$ is $V^\star$, the expected average reward of the whole sequence of k runs is $V^\star$ as well. If we apply $\sigma_{\pi^\star}$ to instances where $u > k$, the additional, prepended runs have no effect as they are not used for evaluation. Therefore, $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\pi^\star})] = V^\star$ for any instance of the MDP-ESP, and the lower bound is as stated in Theorem 16.

P^+ is an upper bound of the MDP-ESP^u_k since it is the highest possible outcome of all policies, and it is therefore impossible to achieve a higher expected reward in a run and a higher expected average reward in a sequence of k runs. Moreover, the bound is tight for the MDP-ESP$^\infty_k$, i. e., the subset of instances with an infinite number of runs and a finite number of evaluation runs. Since any sequence of outcomes will occur eventually in an infinite number of runs, the optimal strategy for the MDP-ESP$^\infty_k$ applies π^+ in every run until a sequence of k runs in a row yields P^+, and the expected average reward of this strategy is P^+. ■

Optimal Strategies. Even though we have provided tight upper and lower bounds for the MDP-ESP, the expected reward of optimal policies in the space between the discussed extreme cases is not yet clear. It is not hard to show that the expected reward of the MDP-ESP^u_k under an optimal strategy increases strictly from $V^\star$ to P^+ with increasing u for all k (unless $\pi^\star = \pi^+$ and $\pi^\star$ deterministic, in which case $\max_\sigma \mathbb{E}[\mathcal{R}^u_k(\sigma)] = V^\star = P^+$ for all $k > 0$ and $u \geq k$). We omit a general proof and discuss the figurative special case where $k = 1$ instead. The MDP-ESP$^u_1$ corresponds to a finite-horizon version of the house-selling problem (Karlin, 1962), where offers come in sequentially for a house an agent wishes to sell. The offers are drawn from a known probability distribution, and the agent has to accept or decline each offer right

n	$\pi^\star$	π_2	π^+	app(n)
1	$\mathbf{-12.8}$	-22	-32.4	$\pi^\star$
2	$\mathbf{-7.36}$	-8.4	-10.64	$\pi^\star$
3	-6.272	$\mathbf{-5.68}$	-6.288	π_2
4	-5.936	$\mathbf{-4.84}$	-4.944	π_2
5	-5.768	-4.42	$\mathbf{-4.272}$	π^+
6	-5.654	-4.136	$\mathbf{-3.8176}$	π^+

Table 9.1: The optimal strategy for the MDP-ESP$_1^u$ on the NAVIGATION instance of Figure 9.1 applies app(n) if the current result is less than $t(n)$ (in bold) and stops otherwise.

after receiving it. The agent aims to sell the house for the highest price among at most u offers. The subset of instances where only a single run is used for evaluation is interesting for our purposes because an optimal strategy can be described with two simple functions: the target reward function $t : \{1, \dots, u-k\} \to \mathbb{R}$ describes the average reward $t(n)$ of the last k runs that must have been achieved in a state $(r_1, \dots, r_{u-n})$ to apply $a_\otimes$, and the policy application function app : $\{1, \dots, u-k\} \to \Pi$ specifies the policy that is taken otherwise.

A solution for the MDP-ESP$_1^u$ is to compute these functions by applying backward induction, a popular method to solve full information optimal stopping problems where an increasing number of available runs u is considered (Gilbert and Mosteller, 1966). We know that it is optimal to apply $\pi^\star$ in the MDP-ESP$_1^1$, and the expected reward is $V^\star$, i. e., app$(1) = \pi^\star$. Now consider the MDP-ESP$_1^2$: if, after the first run, our current result is higher than $V^\star$, we stop the evaluation, since the remaining problem is exactly the MDP-ESP$_1^1$ with expected reward $V^\star$. Otherwise, we apply app$(1) = \pi^\star$. The target reward function is therefore such that $t(1) = V^\star$. The policy that is applied in the first run of the MDP-ESP$_1^2$, app(2), can be computed as the policy that maximizes the expected reward given $t(1)$, which in turn allows the computation of $t(2)$ and so on.

Take for example the NAVIGATION domain that was presented earlier. We have app$(1) = \pi^\star$ and $t(1) = V^\star = -12.8$. If we apply $\pi^\star$ in the first run of the MDP-ESP$_1^2$, we achieve a reward of -6 with probability 0.8 and of -40 with probability 0.2. Since we prefer not to stop in the latter case, we get $t(2) = (0.8 \cdot (-6)) + (0.2 \cdot t(1)) = -7.36$. Table 9.1 shows these computations for all three policies of the NAVIGATION instance that are depicted in Figure 9.1. It reveals that it is optimal to execute π^+ if five or more runs are left, and to stop only if a run successfully crosses the middle row and yields a reward

	$u = k$	$k < u < \infty$	$u = \infty$
$k = 1$	$O(1)$	$O(u \cdot \lvert\Pi\rvert \cdot \mathcal{O}_{\max})$	$O(1)$
$1 < k < \infty$	$O(1)$	$O\big((\lvert\Pi\rvert \cdot \mathcal{O}_{\max})^u\big)$	$O(1)$

Table 9.2: Complexity results for different instances of the MDP-ESP$_k^u$ given an oracle for the underlying base-MDP.

of -2. If three or four runs are left, the strategy proposes the execution of policy π_2, and $\pi^\star$ is executed only in the last two runs. The example shows that restricting to strategies that consider only $\pi^\star$ and π^+ is not sufficient for optimal behavior.

Complexity. It is not hard to see that finding an optimal strategy for the general MDP-ESP$_k^u$ is practically intractable for all but the most trivial cases. It corresponds to solving the meta-MDP with a search space of size $(|\Pi| \cdot \mathcal{O}_{\max})^u$ with $\mathcal{O}_{\max} = \max_{\pi \in \Pi} |\mathcal{O}^\pi|$, which is intractable even if $|\Pi|$ were manageable (which is usually not the case). We have discussed three special cases of the MDP-ESP, though, and we have shown that an optimal strategy for two of them – the MDP-ESP$_k^k$ and the MDP-ESP$_k^\infty$ – can be derived in constant time under the assumption that the cost of deriving policies in the base-MDP can be neglected. For the third, we have provided an algorithm that regards all outcomes of all policies $\pi \in \Pi$ in at most $(u - k)$ decisions, and it is hence linear in u, $|\Pi|$, and $\mathcal{O}_{\max}$. Even though the dependence on $|\Pi|$ is discouraging as the computation of all policies is intractable, it also shows that efficient approximations of good quality are possible if we consider only a subset of Π. The complexity results are summarized in Table 9.2. The manageable cases all have in common that two simple functions that map the number of remaining runs to a target reward and a policy suffice to describe the strategy. In the next section, we show how these ideas can be used to approximate the general case with strategies of high quality.

9.3 Strategies for the MDP-ESP

We consider three possible states of a-priori information: first, we look at the case where $\pi^\star$ and $V^\star$ are unknown, and assume that the computation of a policy and its execution are interleaved. We continue with MDPs where $\pi^\star$ and $V^\star$ can be computed, and present two strategies, one that aims at avoiding bad luck and one that pushes its luck under execution of $\pi^\star$. In the last part of this section, we present a strategy that mixes $\pi^\star$ and π^+ and prove that it is theoretically superior to the other considered strategies.

Secretary Problem. While most instances of the IPPC are such that they cannot be solved in the given time, it is always possible to perform more than k runs. Even if the available time is distributed equally among all planning steps beforehand, there are reasons for spare time: the PROST planner (Keller and Eyerich, 2012) that has been our entry for IPPC 2014 detects reward locks; it recognizes states with only one reasonable action; it is able to solve an encountered state even in larger MDPs if the remaining horizon is small; and it reuses decisions if it encounters a state more than once.

If the optimal policy is not available, the MDP-ESP$_k^u$ is similar to the secretary problem (Dynkin, 1963; Ferguson, 1989), which is a variant of the finite-horizon house-selling problem where the underlying probability distribution is not revealed to the agent. It involves a single secretarial position and u applicants which are interviewed sequentially in a uniformly random order. Applicants can be ranked unambiguously, and the decision to hire a candidate has to be made right after the interview and is irrevocable. The objective is to have the highest probability of selecting the best applicant of the whole group, and it can be shown that an optimal solution is to reject the first $\lfloor \frac{u}{e} \rfloor$ applicants ($\approx 36.8\%$) and select the first subsequent candidate that is ranked higher than all candidates before (e.g., Ferguson, 1989; Bruss, 2000).

To apply the secretary-problem strategy (SecP) to the MDP-ESP$_k^u$, we pretend that all sequences of k consecutive runs are independent, identically distributed data points. We perform $\lfloor \frac{u-k+1}{e} \rfloor$ runs and stop as soon as the last k runs yield a higher average reward than all data points before. It is important to note that the data points are of course not independent and identically distributed in our setting – each data point depends on the previous one(s) unless $k = 1$, since two consecutive samples differ only in a single reward. Our empirical evaluation (where only $\pi^\star$ is executed) shows that the secretary-problem strategy is a strategy that improves over $V^\star$ significantly nonetheless.

Meet-The-Expectations. The IPPC benchmarks offer MDPs of varying complexity, including some instances where $\pi^\star$ can be computed. However, it is always possible that the execution of a policy is unfortunate. Take, for example, the NAVIGATION instance from Figure 9.1. If we execute $\pi^\star$ for $k = 30$ runs, the expected reward $V^\star$ is achieved if the agent ends up stuck exactly six times. Figure 9.2, which depicts how likely it is that the agent gets stuck, reveals that the probability that it gets stuck more than six times is roughly 40%. A strategy that avoids bad luck if more than k runs are available is a first step in the right direction. We call the strategy with $t_{\sigma_{\text{MTE}}}(n) = V^\star$ and $\text{app}_{\sigma_{\text{MTE}}}(n) = \pi^\star$ for all n the meet-the-expectations strategy (MTE).

Theorem 17. *$V^\star \leq \mathbb{E}[\mathcal{R}_k^u(\sigma_{\text{MTE}})] \leq P^\star$ for all $k > 0$ and $u \geq k$.*

Proof: If $\pi^\star$ is deterministic, all inequalities are trivially equalities. Otherwise, both inequalities hold since only $\pi^\star$ is applied. The first is strict for $u > k$ since

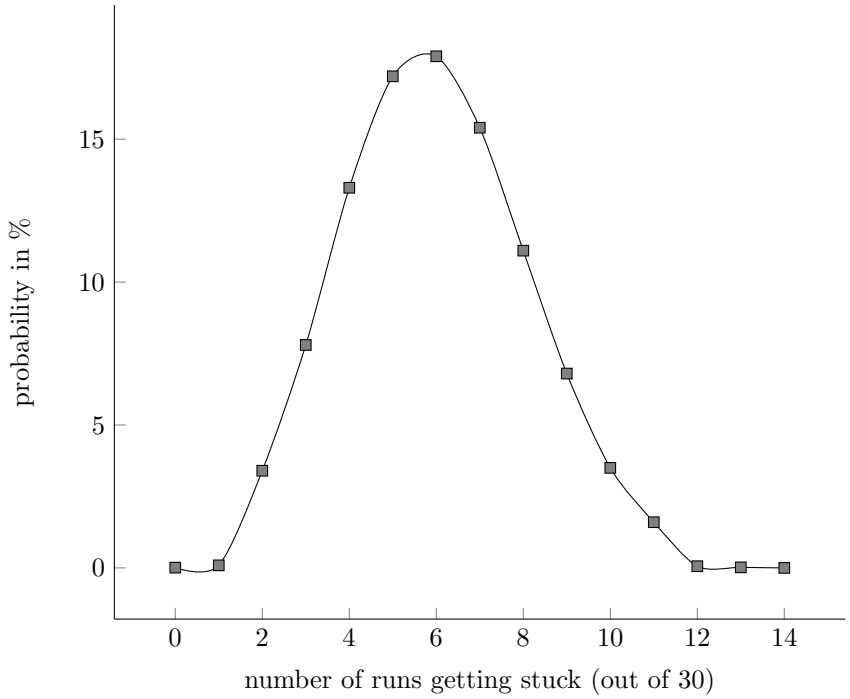

Figure 9.2: Probability of getting stuck x times in 30 runs of the NAVIGATION example instance.

we accept lucky results and improve unlucky ones, and the second is strict even for most instances of the MDP-ESP$_1^\infty$ since the meet-the-expectations strategy stops with a result between $V^\star$ and P^+. ■

Pure Strategy. We have presented a strategy that avoids bad luck while applying $\pi^\star$, so the question naturally arises how to push the envelope and aim for good luck. After all, Figure 9.2 shows that the probability of getting stuck less than six times is also approximately 40%. Since an optimal target reward function is intractable in practice even if app$_{\sigma_{\text{PS}}} = \pi^\star$ for all n, we use a simulation approach in the pure strategy (PS) to estimate $t_{\sigma_{\text{PS}}}$. The pure strategy performs a sequence of m simulations $(\oslash_1, \ldots, \oslash_m)$ (m is a parameter of the algorithm), where each $\oslash_i$ consists of u runs $(\phi^\star_{i1}, \ldots, \phi^\star_{iu})$. We use the simulations to compute the target reward function as

$$t_{\sigma_{\text{PS}}}(n) = \text{median}(\mathcal{R}^n_{\max}(\oslash_1), \ldots, \mathcal{R}^n_{\max}(\oslash_m)),$$

where $\mathcal{R}^n_{\max}(\oslash_i) = \max_{l \in \{1,\ldots,n\}}(\frac{1}{k}\sum_{s=l}^{l+k-1} R(\phi^\star_{is}))$.

Theorem 18. *$V^\star \leq \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})] \leq P^\star$ for all $k > 0$ and $u \geq k$. For all finite $k > 0$, $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{PS}})] = P^\star$ and $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$ is monotonically increasing in u and converges to $P^\star$.*

Algorithm 8: Mixed strategy for the MDP-ESP^u_k with $u > k$

compute_mixed_strategy(u, k, m)**:**
  let all $t(n) \leftarrow -\infty$, app$(n) \leftarrow \pi^\star$ and $n_0 \leftarrow 1$
  for $i = 0, \ldots, k$ **do**
    sample_run_sequences(u, k, m, i)
    update_strategy(u, k, i)

sample_run_sequences(u, k, m, i)**:**
  for $j = 1, \ldots, m$ **do**
    for $n = 1, \ldots, u$ **do**
      if $(n \bmod k) < i$ **then** $r_n \leftarrow$ sample(π^+)*;*
      else $r_n \leftarrow$ sample$(\pi^\star)$*;*
      if $n > k$ **then**
        $t_{ij}(n-k) \leftarrow \max_{l \in \{1,\ldots,n\}}(\frac{1}{k} \sum_{s=l}^{l+k-1} r_s)$
  for $n = 1, \ldots, u-k$ **do**
    $t_i(n) \leftarrow$ median$(t_{i1}(n), \ldots, t_{im}(n))$

update_strategy(u, k, i)**:**
  for $n = u-k, \ldots, n_0$ **do**
    if $t_i(n) > t(n)$ **then** $t(n) \leftarrow t_i(n)$*;*
    else
      for $l = n_0, \ldots, n$ **do**
        if $(l \bmod k) < i$ **then** app$(n) \leftarrow \pi^+$*;*
      $n_0 \leftarrow n$ and **return**

Proof: $V^\star$ and $P^\star$ are bounds since only $\pi^\star$ is applied and $\mathbb{E}[t_{\sigma_{\text{PS}}}(n)] \geq V^\star$ for a sufficiently large m. $\mathbb{E}[t_{\sigma_{\text{PS}}}(n)]$ increases monotonically in n from $V^\star$ to a value $\leq P^\star$ for $u > k$ since the number of considered data points in the simulations grows, and it reaches $P^\star$ for $u = \infty$ and therefore $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{PS}})] = P^\star$. $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$ is monotonically increasing since the expected reward is bounded from below by a probability weighted sum of all target rewards. ■

Mixed Strategy. π^+ can not only be derived when $\mathcal{O}^\pi$ is available for all π, but it can be computed as $\pi^+ := \max_{\pi \in \Pi} P^\pi(s_0)$, which is described by the set of equations

$$\begin{aligned} P^\pi(s) &= \begin{cases} 0 & \text{if } s \text{ is terminal} \\ W^\pi(s, \pi(s)) & \text{otherwise, and} \end{cases} \\ W^\pi(s,a) &= R(s,a) + \max_{s' \in \text{succ}(s,a)} P^\pi(s'). \end{aligned}$$

Note that the only difference to the Bellman optimality function in Definition 11 is that $W^\pi(s,a)$ only cares about the best outcome while $Q^\pi(s,a)$ uses the weighted average of all outcomes. In the PROST version that has been used

for IPPC 2014, we have turned these equations into assignment operators and have extended the used PB backup function to compute π^+ in addition. This way, it is in general possible to use the THTS framework to derive π^+ as a side-effect of the $\pi^\star$ computation and without notable overhead.

While it is possible to use π^+ as the base-policy of the pure strategy, it turns out that u has to be prohibitively large to outperform the pure strategy based on $\pi^\star$. Instead, we generate a policy that is inspired by our analysis of the MDP-ESP$^u_1$, where a function app$_{\sigma_{\text{MS}}}(n)$ is used to describe which policy is executed solely in terms of the number of remaining runs. We restrict ourselves to the policies $\pi^\star$ and π^+ in our version of a mixed strategy (MS), but adding more policies is an interesting (and certainly non-trivial) topic for future work. Initially, the mixed strategy computes a function t_0 (the index stands for the number of runs under π^+ in each data point) which is equivalent to $t_{\sigma_{\text{PS}}}$. The mixed strategy, which is depicted in Algorithm 8, continues by performing simulations where π^+ is executed in i out of k runs. The functions $t_{\sigma_{\text{MS}}}$ and app$_{\sigma_{\text{MS}}}$ are updated after the i-th iteration by finding the largest n where $t(n) \geq t_i(n)$, i. e., by finding the element in the sequence where the number of runs is small enough that an additional execution of π^+ does not pay off anymore. Note that our implementation stops the computation prematurely when a t_i does not alter $t_{\sigma_{\text{MS}}}$ in the update procedure (unlike the depicted Algorithm 8).

Theorem 19. *$\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})] \leq \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{MS}})]$ for all $k > 0$ and $u \geq k$, and for all finite $k > 0$ it holds that $\mathbb{E}[\mathcal{R}^\infty_k(\sigma_{\text{MS}})] = P^+$.*

Proof: If we assume that the number of simulations is sufficiently high, then it is either not beneficial to apply π^+ and the mixed strategy reduces to the pure strategy, or it is beneficial and $\mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{MS}})] > \mathbb{E}[\mathcal{R}^u_k(\sigma_{\text{PS}})]$. The mixed strategy converges towards P^+ with an increasing number of u since at some point it pays off to only apply π^+ with an expected reward of P^+ in the limit. ■

9.4 Experimental Evaluation

To evaluate our algorithms empirically, we perform experiments on the domains of IPPC 2011 and 2014. We use the UCT* algorithm of Keller and Helmert (2013), which corresponds to the PB$^{\text{UCB1}}_{\text{EBA}}$ recipe, to solve the base-MDP. We have altered the THTS framework to perform a sequence of searches with an increasing horizon, a change that is inspired by the Reverse Iterative Deepening approach that is used in GLUTTON (Kolobov et al., 2012a). A higher number of instances can be solved since state-values of solved states are reused, which occurs more often if the horizon is increased iteratively (the possibly weaker anytime performance is not important here). The resulting algorithm is able to solve 34 instances of the 120 existing IPPC benchmarks: four of CROSSING, five of ELEVATORS, three of GAME, all NAVIGATION

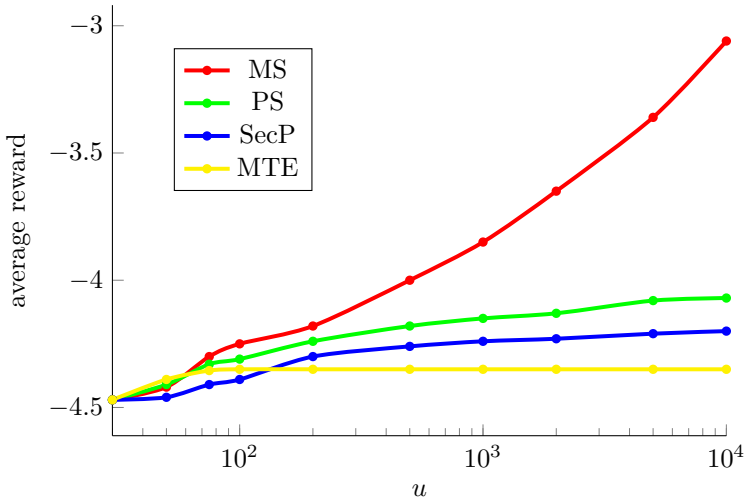

Figure 9.3: Results for an increasing number of available runs u on the first instance of the CROSSING domain with $V^\star \approx -4.43$, $P^\star = -4$, and $P^+ = -2$.

instances, and six both of SKILL and TRIANGLE. Apart from the ELEVATORS domain, where $\pi^\star$ can be derived for the instances 1, 2, 4, 7, and 10, the instances with the lowest indices are solved. The number of evaluation runs k is set to 30 in all experiments, which corresponds to the number of evaluation runs at both IPPC 2011 and 2014, and the values for u are increased from 30 to 10000. Each experiment is conducted 20 times and average results are reported.

Figure 9.3 shows the average reward of the experiments on the first instance of CROSSING with increasing u. We have selected the instance since comparably small values of u showcase our theoretical results. Nevertheless, if u is large enough, any instance could have been selected. Table 9.3 shows normalized IPPC scores which are computed over the average of the results of the experiment sets for $u = 200$ and $u = 1000$. As in the previous chapter, a score of 100 is assigned to the best performing strategy. Moreover, a score of 0 is assigned to an artificial baseline solver with an average reward of $V^\star$ in all instances. (Please observe that the *minimal* result is assigned to a solver that plays the *optimal* base policy in each run.) All other scores are normalized with the procedure that is used at IPPC with the described minimal and maximal values.

The expected reward in the depicted CROSSING instance is $V^\star \approx -4.43$. The simple meet-the-expectations strategy is already an improvement over the baseline solver. It reliably avoids bad luck already with only a few extra runs. However, it has converged to -4.35, a value that is only little above $V^\star$, in the CROSSING instance already with $u = 100$ and does not improve any further. The same can be observed in Table 9.3, where the result does

not improve when u is increased from 200 to 1000. In that experiment set, merely 35 to 40 runs suffice to avoid bad luck reliably over all instances, and in between 100 and 200 runs suffice for convergence. Except for special cases like an MDP-ESP$_1^u$ with $|\{(r,p) \in \mathcal{O}^\star \mid r \geq V^\star\}| = 1$, the meet-the-expectations strategy converges to a value that is greater than $V^\star$ yet significantly less than $P^\star$.

The secretary-problem strategy does not suffer from this problem – the larger u, the better the result in our experiments. It quickly outperforms the meet-the-expectations strategy even though the availability of the optimal policy is no condition for the applicability of the strategy. It should nevertheless be noted that the presented results of the secretary-problem strategy are based on an implementation that executes the optimal policy in all experiments to allow a better comparison of the strategies. In this setting, the secretary-problem strategy converges to $P^\star$ with growing u: since only $\pi^\star$ is applied, it cannot improve over $P^\star$, and since $\lfloor \frac{u-k+1}{e} \rfloor$ grows with u, the target reward and in turn the expected average reward approach $P^\star$.

The two sampling-based strategies yield comparable results in the experiment on all solvable IPPC instances that is given in Table 9.3, and both outperform the other considered algorithms significantly and in all domains. Obviously, simulation based approaches are well-suited to create strategies of high quality. It is not surprising that the pure strategy outperforms the meet-the-expectations strategy with increasing u since the pure strategy converges to $P^\star$ according to Theorem 18 while the meet-the-expectations strategy usually does not, and since the pure strategy reduces to the meet-the-expectations strategy if it ever were reasonable. The theoretical relation between the performance of the pure strategy and the secretary-problem strategy is an open question, but it appears that the latter converges to $P^\star$ with a slower pace. The pure strategy often has an edge over the mixed strategy when u and k are close, since applying π^+ is rarely reasonable in these cases and the mixed strategy can hence only be misled by its additional possibilities. Increasing the number of simulations m will neglect the slight advantage the pure strategy has in some instances. The larger u compared to k, the larger the advantage of the mixed strategy over the pure strategy. Figure 9.3, which depicts one of the smaller instances of the used benchmarks, shows this clearly: the mixed strategy quickly outperforms all other strategies (as soon as it starts to mix in runs under π^+) and converges to $P^+ = -2$. Table 9.3 also supports this claim since the mixed strategy outperforms the pure strategy in all domains with $u = 1000$. The only exception is the ELEVATORS domain, where $P^\pi(s_0) = 0$ for all policies π since there is a small chance that no passenger shows up. If ties were broken in favor of better action-value estimates in our implementation of π^+ (instead of uniformly at random), the mixed strategy and the pure strategy would perform equally in the ELEVATORS domain.

	MTE		SecP		Pure		Mixed	
	200	1000	200	1000	200	1000	200	1000
CROSSING	21	21	34	45	**59**	84	57	**100**
ELEVATORS	26	26	34	59	**59**	**98**	55	93
GAME	28	28	27	52	63	**98**	**68**	**98**
NAVIGATION	37	37	57	85	**74**	93	73	**100**
SKILL	23	23	37	63	57	**97**	**60**	96
TIREWORLD	27	27	38	68	**62**	96	**62**	**99**
Total	27	27	38	62	62	94	**63**	**98**

Table 9.3: IPPC scores of the proposed algorithms for the MDP-ESP$^{u}_{30}$ on instances of IPPC 2011 and 2014 that can be solved with PROST. The expected reward $V^\star$ of the optimal policy $\pi^\star$ is used as the minimum for normalization.

9.5 Discussion

We started with the work at hand due to the evaluation schema that was used at IPPC 2014. Only three IPPC solvers made use of the rule that more than 30 runs are allowed: both versions of PROST and G-PACK, a variant of the GOURMAND planner (Kolobov et al., 2012b). The latter does not reason over the MDP-ESP, though. It simplifies the original MDP by considering at most N outcomes for all actions, and computes and executes the policy with highest expected reward in the simplified MDP. If time allows, this process is repeated with a larger N, which is the only reason that more than k evaluation runs are performed.

Therefore, only our submissions actually considered the MDP-ESP as the relevant optimization problem. However, most of the work described in this paper was done *after* the competition – only the secretary-problem strategy and a variant of the pure strategy with a target reward that is independent from the number of remaining runs were applied at IPPC 2014. Note that both are strategies that aim at optimizing the evaluation of $\pi^\star$ or a near-optimal policy. PROST 2011 applied the secretary-problem strategy in 33 out of 80 instances, while PROST 2014 used it in 28 and the pure strategy in another six instances. Even though we were able to improve the average reward in 22 (19) instances with the secretary-problem strategy and in five with the variant of the pure strategy in the 2014 (2011) version, the total IPPC scores are mostly unaffected: had we stopped evaluation after 30 runs in all instances with both solvers, the final result would differ only slightly with total IPPC scores of 81.6 (-0.9) and 77.3 (+0.4) for the PROST versions

and of 73.9 (+0.5) for G-PACK (and hence still with a clear winner PROST).

There are many applications for optimal stopping problems, including sponsored search (e.g., Babaioff et al., 2007; Zhou and Naroditskiy, 2008), online auctions (e.g., Hajiaghayi et al., 2004), or optimal stock market behavior (e.g., Griffeath and Snell, 1974; Shiryaev et al., 2008). Most applications are based on variants of the secretary problem that differ in the number of applicants that must be selected as in the multiple-choice secretary problem (Freeman, 1983), that have full information as in the house-selling problem or where the selected values must be maximized under constraints on each element as in the online knapsack problem (Marchetti-Spaccamela and Vercellis, 1995). The MDP-ESP differs from the multiple-choice secretary problem in two details: first, the selected applicants must show up consecutively, and second, the probability distribution that gives the next sample must be selected from a known set of probability distributions.

The former difference does not alter the applicability of our algorithms, since scenarios where k consecutive data points must be selected exist, e. g., with resources that decay with time, goods on an assembly line, or in traffic control. Moreover, our algorithms can also be applied to a variant of the MDP-ESP where k results can be selected in arbitrary order. An example application is, as in the multiple-choice secretary problem, the plan to hire k employees. In our scenario, all applicants have provided application documents, which allow the estimation of a probability distribution over the candidate's aptitude (e. g., grades or experience influence the expectation and the rest of the CV the variance and hence the potential). We invite at most u of the applicants (more than u applicants are necessary since we do not restrict the number of times a "type of applicant" is selected), and we have to decide right after the interview if we hire the candidate with knowledge of the aptitude. This problem differs from the MDP-ESP only in the way the k applicants are selected. However, it is easy to see that Theorem 16 also holds, that the meet-the-expectations strategy can be applied with the same bounds on the expected reward (i. e., Theorem 17 holds) and that Theorems 18 and 19 hold for versions of the pure strategy and the mixed strategy where line 12 of Algorithm 8 is replaced with an equation that sums the k largest rewards independently from their position. Since this is a simple and useful generalization of the popular multiple-choice secretary problem, it is well-suited to describe existing optimal stopping problem applications more realistically.

Chapter 10

Conclusion

The focus of this thesis is the problem of planning and acting in a dynamic and uncertain environment. In Chapter 3, we discussed different determinization techniques that are used in the context of MDPs. We showed how two different single-outcome determinizations can be derived from a given MDP, and we have discussed a compilation schema that allows the conversion of an MDP to an equivalent MDP in forked normal form, which in turn allows the creation of an all-outcomes determinization of polynomial size. Among the algorithms that use determinizations in MDP planning are three suboptimal policies which are presented in Chapter 4. The optimistic policy, which bases its decisions solely on a determinization, is among the state-of-the-art in many planning problems under uncertainty despite its simplicity. We showed that it is overly optimistic of the optimistic policy to assume that it is possible to follow the optimal path in the best possible weather. Hindsight optimization is an alternative approach that also allows to reduce the policy computation to solving a set of deterministic MDPs. It samples a weather according to its probability in each step and computes action-value estimates as the average of the optimal solution in each weather. We showed both theoretically and empirically that hindsight optimization improves over the optimistic policy, but is still too optimistic due to the assumption of clairvoyance. The third approximate policy that has been presented is optimistic rollout, which overcomes this weakness by computing action-value estimates by simulating runs.

Even if provided with unlimited resources, optimistic rollout remains suboptimal. It is nevertheless possible to derive similar algorithms that also simulate trials but behave optimally in the limit. In Chapter 5, we introduced the main contribution of this thesis, the Trial-based Heuristic Tree Search framework. It is a general framework that allows the specification of algorithms in terms of only six ingredients: action selection, outcome selection, initialization, trial length, backup function, and recommendation function. Our discussion of related work both in MDP Planning and Reinforcement Learning provided us with a variety of basic ingredients for THTS algorithms: action

selection strategies can be derived from techniques that are used in the Multi-Armed Bandit Problem; backup functions can be derived from techniques that are used in Online RL; recommendation functions play a key role in MDP planning with generative model; and a declarative model allows the initialization of search nodes with well-informed heuristics that guide THTS algorithms in early trials.

A selection of different THTS ingredients is provided in Chapter 6, with a focus on backup functions and methods for action selection. We proposed backup functions that are inspired from Temporal Difference Learning and Q-Learning, and we showed that they differ significantly from the Monte-Carlo backups that are most often used in MDP planning. We derived four selective backup functions that are inspired from the concept of separation of concerns. We presented Partial Bellman backups, a variant of Full Bellman backups that are tractable even if the number of outcomes is large, and MaxMC backups, an off-policy backup function that can be used even if only a generative model of the MDP is accessible. We furthermore introduced several action selection methods, among them RT-UCB, a variant of UCB1 that behaves explorative more often but is still greedy in the limit.

The proposed ingredients can be combined to 162 different THTS algorithms, which we analyzed theoretically in Chapter 7 by determining the properties that are necessary for anytime optimal recipes. We showed in particular that on-policy and selective backup functions must be combined with an an action selection that is both greedy in the limit and explores infinitely, while infinite exploration alone is sufficient for off-policy backup functions. Partial Bellman backups can even be combined with any action selection strategy, but unlike most other recipes they can only be used with the expected best arm recommendation function and not in conjunction with the most played arm recommendation function. The main result of the theoretical evaluation is that we showed that 115 different, anytime optimal THTS algorithms can be derived with the considered ingredients.

All anytime optimal THTS algorithms are evaluated empirically by performing experiments on the benchmark sets of IPPC 2011 and 2014. Our implementation of the THTS framework is the PROST planning system, a state-of-the-art MDP planner that was developed during the author's doctoral process. We showed empirically in Chapter 8 that off-policy backup functions perform better if the rate of exploration is high while on-policy and selective backup functions prefer little exploration. We also showed that a decaying learning rate can be a useful tool and that the most played arm recommendation function outperforms the expected best arm recommendation significantly in some recipes. We showed empirically that the majority of well-performing THTS algorithms uses Monte-Carlo or Partial Bellman backups in combination with an action selection that is based on Boltzmann Exploration or the computation of an upper confidence bound. Most considered ingredients are nevertheless used in a recipe that performs best in at least one domain.

In the final Chapter 9, we identified the MDP-Evaluation Stopping Problem as the relevant problem of IPPC 2014. We showed how it can be constructed as a meta-MDP where actions encode the execution of a policy in the base-MDP, and where the decision when to stop is crucial. We showed that the expected reward of the optimal policy of the base-MDP is a lower bound for optimal strategies in the MDP-ESP. We derived four approximate strategies that achieve an expected result that exceeds the expected reward of the optimal policy significantly. The combination of the Trial-based Heuristic Tree Search framework, its efficient implementation in the MDP planning system PROST, and the insights of the MDP-ESP makes this thesis an integral view on the problem of planning and acting under uncertainty.

Even though we presented important contributions and encouraging results towards autonomous agents that are able to act reasonably over a large planning horizon in a complex and dynamic environment, we still see a lot of room for future research. The main contribution of this thesis, the Trial-based Heuristic Tree Search framework, offers a good starting point for further development. Especially the distinction of the six ingredients that allow the specification of algorithms for MDP planning induce clearly partitioned research topics. For instance, the only outcome selection that has been considered in this thesis is the Monte-Carlo outcome selection where effects of actions are sampled according to their probability. Methods that base their selection on the impact of each outcome on the current decision are the only missing piece in the guidance of the search procedure of THTS algorithms.

We also see possibilities to incorporate such an impact measurement into novel action selection methods. Even though we have extended the portfolio of action selection methods with new developments in this thesis, we have not been able to single out one action selection that outperforms all others in the majority of planning problems. All current methods balance exploration and exploitation by treating each decision node as a Multi-Armed Bandit problem. Integrating decisions that are deeper in the search tree in the current selection might give rise to THTS algorithms that outperform the current state-of-the-art significantly. However, the ingredient where the scope and need for progress is greatest is, in our opinion, the heuristic function that is used to compute initial action-value estimates. In the field of classical planning, the development of novel heuristics has shown that problems are now solvable that have a complexity that would have been unthinkable only a few years. Possibilities in MDP planning include the adaption of heuristics from classical planning (which also includes the development of new determinization techniques that are typically necessary to apply a classical heuristic), and the development of novel ideas that lead to heuristics that take the uncertainty of the MDP directly into account.

Finally, the ultimate aim of our research should be an autonomous planning systems that can be used in everyday applications. The current progress, e. g., in the context of autonomous cars, shows that there are applications

that need planning systems that are able to act reasonably and safely in an uncertain environment. However, there are still some hurdles to take. It is, for instance, impossible to describe the real world with discrete variables. An extension of the THTS framework such that continuous state variables are supported is a crucial first step. Incorporating ideas from optimal control theory, which deals with this kind of MDPs, opens up new challenges in the area of planning and acting under uncertainty, but also allows for a whole range of new possibilities.

Appendix A

Experimental Results

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	Total
$PB^{\epsilon\text{-G}}_{EBA}$	**53**	93	76	85	**93**	36	79	64	73	55	52	43	67
$PB^{\epsilon_{LIN}\text{-G}}_{EBA}$	19	73	46	20	39	78	33	10	14	36	62	48	40
$PB^{\epsilon_{LOG}\text{-G}}_{EBA}$	**47**	93	57	59	64	30	70	38	60	56	52	43	56
$PB^{\epsilon_{RT}\text{-G}}_{EBA}$	**44**	89	70	82	**95**	56	75	62	71	45	52	43	65
$PB^{\text{BE-DT}}_{EBA}$	30	89	73	84	89	67	73	68	70	56	63	54	68
PB^{BE}_{EBA}	35	**97**	72	82	**91**	84	**85**	71	79	51	**76**	55	**73**
$PB^{\text{RT-UCB}}_{EBA}$	30	92	72	83	88	67	**81**	67	74	63	**72**	63	**71**
PB^{UCB1}_{EBA}	24	91	69	77	**94**	71	77	69	66	64	**74**	62	**70**
PB^{Uni}_{EBA}	14	87	76	75	**94**	26	72	56	66	23	48	51	56

Table A.1: IPPC scores of THTS algorithms with PB backups and the given action selection and recommendation function. Best results among all recipes from Tables A.1 to A.5 in red and top five in bold.

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	Total
$MC^{\epsilon_{LIN}\text{-}G}_{EBA}$	28	56	50	26	37	88	16	35	50	53	62	61	47
$MC^{\epsilon_{LIN}\text{-}G}_{MPA}$	21	56	85	23	39	87	18	41	52	69	56	**92**	53
$MC^{\epsilon_{LOG}\text{-}G}_{EBA}$	35	51	37	56	37	**97**	41	60	74	54	52	59	54
$MC^{\epsilon_{LOG}\text{-}G}_{MPA}$	26	53	78	60	35	**93**	49	64	78	73	47	**89**	62
$MC^{\epsilon_{RT}\text{-}G}_{EBA}$	36	32	7	69	31	**93**	59	65	75	63	48	61	53
$MC^{\epsilon_{RT}\text{-}G}_{MPA}$	24	31	44	71	28	**95**	62	72	75	72	44	**91**	59
$MC^{\text{BE-DT}}_{EBA}$	28	62	54	85	38	82	78	**88**	**82**	72	61	77	67
$MC^{\text{BE-DT}}_{MPA}$	26	61	84	84	43	86	**83**	81	**82**	**74**	47	84	**70**
$MC^{\text{RT-UCB}}_{EBA}$	28	67	47	**87**	43	**93**	78	**91**	**84**	**77**	48	**87**	69
$MC^{\text{RT-UCB}}_{MPA}$	28	64	71	**87**	47	91	80	**86**	**88**	73	47	81	**70**
MC^{UCB1}_{EBA}	26	62	49	84	42	90	69	**88**	**83**	60	49	85	66
MC^{UCB1}_{MPA}	27	65	78	**86**	45	92	77	**89**	**86**	71	46	84	**70**
$TD^{\epsilon_{LIN}\text{-}G}_{EBA}$	26	63	44	22	25	83	26	20	22	45	49	49	40
$TD^{\epsilon_{LIN}\text{-}G}_{MPA}$	22	62	75	27	24	85	21	23	30	56	44	69	45
$TD^{\epsilon_{LOG}\text{-}G}_{EBA}$	32	61	37	62	30	86	62	46	61	55	40	51	52
$TD^{\epsilon_{LOG}\text{-}G}_{MPA}$	20	63	79	59	27	85	55	47	70	57	33	74	56
$TD^{\epsilon_{RT}\text{-}G}_{EBA}$	**43**	34	9	74	28	90	62	55	64	47	27	53	49
$TD^{\epsilon_{RT}\text{-}G}_{MPA}$	27	26	41	76	26	88	64	54	75	56	22	73	52
$TD^{\text{BE-DT}}_{EBA}$	40	70	54	**86**	26	76	**81**	71	75	59	38	69	62
$TD^{\text{BE-DT}}_{MPA}$	26	68	78	**87**	25	78	**85**	74	73	66	38	68	64
$TD^{\text{RT-UCB}}_{EBA}$	24	66	44	85	32	85	73	71	73	55	41	78	61
$TD^{\text{RT-UCB}}_{MPA}$	34	61	60	85	28	86	75	72	70	57	30	73	61
TD^{UCB1}_{EBA}	27	63	44	82	31	73	62	69	66	63	37	83	57
TD^{UCB1}_{MPA}	23	66	69	**86**	27	69	64	74	75	65	27	72	60

Table A.2: IPPC scores of THTS algorithms with MC and TD backups and the given action selection and recommendation function. Best results among all recipes from Tables A.1 to A.5 in red and top five in bold.

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	Total
LSMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	28	67	55	23	32	83	26	37	48	50	63	61	48
LSMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	32	67	82	20	33	76	21	40	52	65	62	**91**	53
LSMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	36	82	56	50	28	86	49	48	74	69	56	55	56
LSMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	23	79	85	57	27	88	48	55	72	71	49	86	62
LSMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	35	82	57	61	26	82	51	49	63	57	56	51	56
LSMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	24	78	**87**	56	22	84	42	57	66	69	55	77	60
LSMC$^{\text{BE-DT}}_{\text{EBA}}$	15	86	54	76	27	81	53	67	73	67	56	55	59
LSMC$^{\text{BE-DT}}_{\text{MPA}}$	21	85	71	73	27	88	60	65	68	62	56	56	61
LSMC$^{\text{RT-UCB}}_{\text{EBA}}$	18	82	50	73	28	92	53	56	72	67	56	69	60
LSMC$^{\text{RT-UCB}}_{\text{MPA}}$	14	78	61	79	28	91	53	60	75	71	56	56	60
LSMC$^{\text{UCB1}}_{\text{EBA}}$	13	74	54	75	33	90	49	56	68	66	59	66	59
LSMC$^{\text{UCB1}}_{\text{MPA}}$	16	78	59	75	28	90	52	56	65	68	56	63	59
ESMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	28	64	53	19	34	84	13	39	39	48	65	63	46
ESMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	22	68	83	26	31	88	19	43	52	70	64	**89**	55
ESMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	35	83	61	56	28	83	48	52	65	60	55	55	56
ESMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	24	80	**89**	52	27	82	47	56	71	**74**	51	**87**	62
ESMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	32	81	60	62	26	84	54	49	60	72	56	51	56
ESMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	21	79	82	59	23	86	50	61	74	70	62	80	62
ESMC$^{\text{BE-DT}}_{\text{EBA}}$	22	84	53	76	27	83	53	68	68	67	54	53	59
ESMC$^{\text{BE-DT}}_{\text{MPA}}$	20	83	70	74	26	83	55	59	69	**74**	56	59	61
ESMC$^{\text{RT-UCB}}_{\text{EBA}}$	19	79	57	74	28	91	53	65	67	73	56	64	61
ESMC$^{\text{RT-UCB}}_{\text{MPA}}$	21	85	67	72	28	90	56	68	70	**74**	65	56	63
ESMC$^{\text{UCB1}}_{\text{EBA}}$	18	80	61	76	31	**93**	60	67	71	73	56	66	63
ESMC$^{\text{UCB1}}_{\text{MPA}}$	17	83	71	71	28	**93**	50	67	64	77	54	62	62

Table A.3: IPPC scores of THTS algorithms with selective MC backups and the given action selection and recommendation function. Best results among all recipes from Tables A.1 to A.5 in red and top five in bold.

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	Total
$\text{LSTD}^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	23	65	45	17	21	90	28	26	23	40	48	49	40
$\text{LSTD}^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	21	69	76	18	20	77	28	20	33	48	46	69	44
$\text{LSTD}^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	32	77	56	48	21	64	51	40	61	63	46	49	51
$\text{LSTD}^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	25	74	84	56	19	63	48	37	59	66	37	62	53
$\text{LSTD}^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	32	81	56	53	19	70	47	38	56	64	47	44	51
$\text{LSTD}^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	27	73	83	54	17	71	49	31	59	57	40	62	52
$\text{LSTD}^{\text{BE-DT}}_{\text{EBA}}$	18	83	48	76	4	72	52	61	63	68	38	48	53
$\text{LSTD}^{\text{BE-DT}}_{\text{MPA}}$	21	84	65	73	8	75	50	56	60	63	40	36	53
$\text{LSTD}^{\text{RT-UCB}}_{\text{EBA}}$	23	78	52	74	4	81	33	48	63	59	39	50	50
$\text{LSTD}^{\text{RT-UCB}}_{\text{MPA}}$	21	82	64	76	4	84	38	49	60	54	34	38	50
$\text{LSTD}^{\text{UCB1}}_{\text{EBA}}$	16	81	50	70	0	81	36	45	57	56	35	47	48
$\text{LSTD}^{\text{UCB1}}_{\text{MPA}}$	17	76	61	72	0	81	35	45	59	56	33	35	48
$\text{ESTD}^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	26	72	48	24	22	87	25	19	26	48	47	45	41
$\text{ESTD}^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	21	70	77	19	21	78	25	19	30	46	44	69	43
$\text{ESTD}^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	33	81	53	60	21	66	50	39	62	64	44	46	52
$\text{ESTD}^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	21	76	85	59	19	65	47	40	71	65	37	63	54
$\text{ESTD}^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	32	79	56	57	19	70	49	32	55	60	46	42	50
$\text{ESTD}^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	24	77	84	56	17	65	52	37	59	64	39	62	53
$\text{ESTD}^{\text{BE-DT}}_{\text{EBA}}$	23	84	48	76	6	61	55	56	64	61	37	45	51
$\text{ESTD}^{\text{BE-DT}}_{\text{MPA}}$	22	77	65	69	7	64	54	57	59	66	36	37	51
$\text{ESTD}^{\text{RT-UCB}}_{\text{EBA}}$	20	83	53	71	4	73	39	48	59	60	38	52	50
$\text{ESTD}^{\text{RT-UCB}}_{\text{MPA}}$	21	79	60	71	5	80	40	50	65	64	33	37	50
$\text{ESTD}^{\text{UCB1}}_{\text{EBA}}$	15	83	57	66	2	75	35	48	59	61	36	52	49
$\text{ESTD}^{\text{UCB1}}_{\text{MPA}}$	17	81	62	70	3	77	36	42	60	56	35	39	48

Table A.4: IPPC scores of THTS algorithms with selective TD backups and the given action selection and recommendation function. Best results among all recipes from Tables A.1 to A.5 in red and top five in bold.

	Academic	Crossing	Elevators	Game	Navigation	Recon	Skill	Sysadmin	Tamarisk	Traffic	Triangle	Wildfire	Total
MaxMC$^{\epsilon\text{-G}}_{\text{EBA}}$	**51**	92	68	75	39	54	62	56	69	57	40	40	59
MaxMC$^{\epsilon\text{-G}}_{\text{MPA}}$	35	93	82	74	41	73	63	56	73	41	36	60	61
MaxMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	17	78	46	18	34	75	26	8	9	39	56	40	37
MaxMC$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	10	75	73	15	39	63	28	8	15	42	60	56	40
MaxMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	**43**	**95**	55	45	41	55	49	38	68	54	49	44	53
MaxMC$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	28	89	**91**	43	47	61	51	34	69	62	43	61	56
MaxMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	40	85	70	77	31	69	68	53	75	37	28	43	56
MaxMC$^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	30	83	84	77	31	85	68	54	64	30	24	57	56
MaxMC$^{\text{BE-DT}}_{\text{EBA}}$	27	93	61	82	35	64	66	69	71	59	49	54	61
MaxMC$^{\text{BE-DT}}_{\text{MPA}}$	32	90	81	77	36	68	59	66	68	50	53	40	60
MaxMC$^{\text{BE}}_{\text{EBA}}$	36	**95**	68	84	40	85	72	68	81	59	**67**	52	67
MaxMC$^{\text{BE}}_{\text{MPA}}$	30	92	85	76	39	75	75	63	72	51	**81**	42	65
MaxMC$^{\text{RT-UCB}}_{\text{EBA}}$	27	93	60	80	40	76	60	64	**82**	61	50	64	63
MaxMC$^{\text{RT-UCB}}_{\text{MPA}}$	25	94	78	81	40	75	62	62	73	69	45	52	63
MaxMC$^{\text{UCB1}}_{\text{EBA}}$	20	91	62	74	39	69	60	64	69	68	49	64	61
MaxMC$^{\text{UCB1}}_{\text{MPA}}$	22	**95**	80	73	38	70	54	66	75	61	47	51	61
MaxMC$^{\text{Uni}}_{\text{EBA}}$	11	79	72	71	30	30	53	53	69	23	12	56	47
QL$^{\epsilon\text{-G}}_{\text{EBA}}$	35	92	76	78	34	**94**	69	53	67	44	45	35	60
QL$^{\epsilon\text{-G}}_{\text{MPA}}$	24	91	**87**	72	36	92	66	45	66	49	40	60	61
QL$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{EBA}}$	25	80	49	23	27	85	23	18	18	38	54	34	39
QL$^{\epsilon_{\text{LIN}}\text{-G}}_{\text{MPA}}$	20	79	76	17	25	87	22	17	15	31	64	54	42
QL$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{EBA}}$	34	88	54	54	31	91	54	31	61	48	51	37	53
QL$^{\epsilon_{\text{LOG}}\text{-G}}_{\text{MPA}}$	21	87	**87**	47	33	85	50	30	62	49	45	55	54
QL$^{\epsilon_{\text{RT}}\text{-G}}_{\text{EBA}}$	33	90	71	77	28	**94**	75	49	67	40	36	46	59
QL$^{\epsilon_{\text{RT}}\text{-G}}_{\text{MPA}}$	24	87	82	75	32	92	71	46	64	35	36	54	57
QL$^{\text{BE-DT}}_{\text{EBA}}$	27	92	70	85	32	87	**82**	68	65	52	44	54	63
QL$^{\text{BE-DT}}_{\text{MPA}}$	28	91	86	80	32	91	79	56	64	45	42	44	61
QL$^{\text{BE}}_{\text{EBA}}$	31	**97**	69	85	37	90	73	67	72	49	56	51	65
QL$^{\text{BE}}_{\text{MPA}}$	27	**95**	86	78	36	83	80	60	70	57	56	50	65
QL$^{\text{RT-UCB}}_{\text{EBA}}$	25	92	68	82	36	90	70	61	68	49	50	63	63
QL$^{\text{RT-UCB}}_{\text{MPA}}$	26	**95**	84	**86**	35	92	68	61	71	56	41	48	64
QL$^{\text{UCB1}}_{\text{EBA}}$	21	91	75	78	35	91	65	57	68	50	38	62	61
QL$^{\text{UCB1}}_{\text{MPA}}$	19	93	86	80	34	90	62	53	66	56	40	50	61
QL$^{\text{Uni}}_{\text{EBA}}$	23	83	76	79	19	92	71	50	65	20	38	55	56

Table A.5: IPPC scores of THTS algorithms with MaxMC and QL backups and the given action selection and recommendation function. Best results among all recipes from Tables A.1 to A.5 in red and top five in bold.

List of Figures

List of Tables

List of Algorithms

Bibliography

Agrawal, R. (1995). *Sample Mean Based Index Policies with O(log n) Regret for the Multi-armed Bandit Problem*, volume 27, pages 1054–1078. Applied Probability Trust.

Antos, A., Szepesvári, C., and Munos, R. (2008). Fitted Q-iteration in Continuous Action-space MDPs. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems (NIPS 2008)*, pages 9–16. Curran Associates, Inc.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Journal of Machine Learning Research*, 47:235–256.

Babaioff, M., Immorlica, N., Kempe, D., and Kleinberg, R. (2007). A Knapsack Secretary Problem with Applications. In *Proceedings of the 10th International Workshop on Approximation (APPROX 2007)*, pages 16–28, Berlin, Heidelberg. Springer-Verlag.

Balla, R.-K. and Fern, A. (2009). UCT for Tactical Assault Planning in Real-Time Strategy Games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 40–45.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence (AIJ)*, 72(1–2):81–138.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Berry, D. and Fristedt, B. (1985). *Bandit Problems*. Chapmann and Hall.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

Bertsekas, D. P. and Tsitsiklis, J. N. (1991). An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research*, 16:580–595.

Bjarnason, R., Fern, A., and Tadepalli, P. (2009). Lower Bounding Klondike Solitaire with Monte-Carlo Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 26–33.

Bnaya, Z., Felner, A., Shimony, E., Kaminka, G. A., and Merdler, E. (2008). A Fresh Look at Sensor-Based Navigation: Navigation with Sensing Costs. In *AAAI 2008 Workshop on Search in Artificial Intelligence and Robotics*, pages 11–17.

Bnaya, Z., Felner, A., and Shimony, S. E. (2009). Canadian Traveler Problem with Remote Sensing. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 437–442.

Bonet, B. (2009). Deterministic POMDPs Revisited. In *Proceedings of the 25th Conference in Uncertainty in Artificial Intelligence (UAI 2009)*, pages 59–66.

Bonet, B. and Geffner, H. (2001). Planning as Heuristic Search. *Artificial Intelligence (AIJ)*, 129(1-2):5–33.

Bonet, B. and Geffner, H. (2003). Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 12–21.

Bonet, B. and Geffner, H. (2012). Action Selection for MDPs: Anytime AO* Versus UCT. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic Dynamic Programming with Factored Representations. *Artificial Intelligence (AIJ)*, 121(1–2):49–107.

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions Computational Intelligence and AI in Games*, 4(1):1–43.

Bruss, F. T. (2000). Sum the Odds to One and Stop. *The Annals of Probability*, 28(3):1384–1391.

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure Exploration in Multi-armed Bandits Problems. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory (ALT 2009)*, pages 23–37.

Bubeck, S., Munos, R., and Stoltz, G. (2011). Pure Exploration in Finitely-armed and Continuous-armed Bandits. *Theoretical Computer Science*, 412(19):1832–1852.

Buffet, O. and Aberdeen, D. (2007). FF + FPG: Guiding a Policy-Gradient Planner. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 42–48.

Burnetas, A. N. and Katehakis, M. N. (1997). Optimal Adaptive Policies for Markov Decision Processes. *Mathematics of Operations Research*, 22:222–255.

Buro, M., Long, J. R., Furtak, T., and Sturtevant, N. (2009). Improving State Evaluation, Inference, and Search in Trick-Based Card Games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1407–1413.

Busa-Fekete, R. and Hüllermeier, E. (2014). A Survey of Preference-Based Online Learning with Bandit Algorithms. In *Proceedings of the 25th International Conference on Algorithmic Learning Theory (ALT 2014)*, pages 18–39.

Bäckström, C. and Nebel, B. (1995). Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11:625–656.

Cesa-Bianchi, N. and Fischer, P. (1998). Finite-time Regret Bounds for the Multiarmed Bandit Problem. In *In Proceedings of the 5th International Conference on Machine Learning (ICML 1998)*, pages 100–108. Morgan Kaufmann.

Chakrabarti, D., Kumar, R., Radlinski, F., and Upfal, E. (2008). Mortal multi-armed bandits. In *Advances in Neural Information Processing Systems (NIPS 2008)*, pages 273–280.

Chang, H. S., Fu, M. C., Hu, J., and Marcus, S. I. (2005). An Adaptive Sampling Algorithm for Solving Markov Decision Processes. *Operations Research*, 53(1):126–139.

Chaslot, G., Winands, M., van den Herik, J., Uiterwijk, J., and Bouzy, B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4:343–357.

Chen, W., Wang, Y., and Yuan, Y. (2013). Combinatorial Multi-Armed Bandit: General Framework and Applications. In Dasgupta, S. and Mcallester, D., editors, *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, volume 28, pages 151–159. JMLR Workshop and Conference Proceedings.

Chong, E. K. P., Givan, R. L., and Chang, H. S. (2000). A Framework for Simulation-based Network Control via Hindsight Optimization. In *Proceedings of the 39th IEEE Conference on Decision and Control (CDC 2000)*, pages 1433–1438.

Condon, A. (1992). The Complexity of Stochastic Games. *Information and Computation*, 96(2):203–224.

Coquelin, P.-A. and Munos, R. (2007). Bandit Algorithms for Tree Search. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI 2007)*, pages 67–74. AUAI Press.

Coulom, R. (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of the 5th International Conference on Computers and Games (CG 2006)*, pages 72–83, Berlin, Heidelberg. Springer-Verlag.

Dai, P., Mausam, Weld, D. S., and Goldsmith, J. (2011). Topological Value Iteration Algorithms. *Journal of Artificial Intelligence Research (JAIR)*, 42(1):181–209.

Derman, C. (1970). *Finite State Markovian Decision Processes*. Academic Press, New York.

Dey, D., Kolobov, A., Caruana, R., Kamar, E., Horvitz, E., and Kapoor, A. (2014). Gauss Meets Canadian Traveler: Shortest-path Problems with Correlated Natural Dynamics. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 1101–1108, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Domshlak, C. and Feldman, Z. (2013). To UCT, or not to UCT? (Position Paper). In Helmert, M. and Röger, G., editors, *Proceedings of the 6th International Symposium on Combinatorial Search (SOCS 2013)*. AAAI Press.

Dynkin, E. B. (1963). The Optimum Choice of the Instant for Stopping a Markov Process. *Soviet Mathematics Doklady*, 4.

Eyerich, P., Keller, T., and Helmert, M. (2010). High-Quality Policies for the Canadian Traveler's Problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 51–58.

Feldman, Z. and Domshlak, C. (2012). Online Planning in MDPs: Rationality and Optimization. *CoRR*, abs/1206.3382.

Feldman, Z. and Domshlak, C. (2013). Monte-Carlo Planning: Theoretically Fast Convergence Meets Practical Efficiency. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI 2013)*, pages 212–221.

Feldman, Z. and Domshlak, C. (2014a). On MABs and Separation of Concerns in Monte-Carlo Planning for MDPs. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*.

Feldman, Z. and Domshlak, C. (2014b). Simple Regret Optimization in Online Planning for Markov Decision Processes. *Journal of Artificial Intelligence Research (JAIR)*, 51:165–205.

Ferguson, D., Stentz, A., and Thrun, S. (2004). PAO* for Planning with Hidden State. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA 2004)*, pages 2840–2847.

Ferguson, T. S. (1989). Who Solved the Secretary Problem? *Statistical Science*, 4(3):282–289.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI 1971)*, pages 608–620, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Finnsson, H. and Björnsson, Y. (2008). Simulation-based Approach to General Game Playing. In Fox, D. and Gomes, C. P., editors, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*. AAAI Press.

Finnsson, H. and Björnsson, Y. (2010). Learning Simulation Control in General Game-Playing Agents. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI 2010)*, pages 954–959.

Finnsson, H. and Björnsson, Y. (2011). CadiaPlayer: Search Control Techniques. *KI Journal*, 25(1):9–16.

Frank, I. and Basin, D. A. (2001). A Theoretical and Empirical Investigation of Search in Imperfect Information Games. *Theoretical Computer Science*, 252(1–2):217–256.

Freeman, P. R. (1983). The Secretary Problem and its Extensions: A Review. *International Statistical Review*, 51(2):189–206.

Geißer, F., Keller, T., and Mattmüller, R. (2014). Past, Present, and Future: An Optimal Online Algorithm for Single-Player GDL-II Games. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pages 357–362". IOS Press.

Gelly, S. and Silver, D. (2007). Combining Online and Offline Knowledge in UCT. In *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pages 273–280.

Gelly, S. and Silver, D. (2011). Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence (AIJ)*, 175:1856–1875.

Gilbert, J. P. and Mosteller, F. (1966). Recognizing the Maximum of a Sequence. *Journal of the American Statistical Association*, 61(313):35–73.

Ginsberg, M. L. (1999). GIB: Steps Toward an Expert-Level Bridge-Playing Program. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 584–593. Morgan Kaufmann.

Gittins, J. C. (1979). Bandit Processes and Dynamic Allocation Indices. *Journal of the Royal Statistical Society, Series B*, pages 148–177.

Godoy, J., Karamouzas, I., Guy, S. J., and Gini, M. (2014). Anytime Navigation with Progressive Hindsight Optimization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, pages 730–735.

Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

Griffeath, D. and Snell, J. L. (1974). Optimal Stopping in the Stock Market. *The Annals of Probability*, 2(1):1–13.

Hajiaghayi, M. T., Kleinberg, R., and Parkes, D. C. (2004). Adaptive Limited-supply Online Auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 71–80, New York, NY, USA. ACM.

Hansen, E. A. and Zilberstein, S. (2001). LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence (AIJ)*, 129(1–2):35–62.

Haslum, P. and Geffner, H. (2000). Admissible Heuristics for Optimal Planning. In Chien, S., Kambhampati, S., and Knoblock, C. A., editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, pages 140–149. AAAI.

Helmert, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246.

Helmert, M. (2008). *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*. Springer-Verlag, Berlin, Heidelberg.

Helmert, M. and 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)*.

Hertle, A., Dornhege, C., Keller, T., Mattmüller, R., Ortlieb, M., and Nebel, B. (2014). An Experimental Comparison of Classical, FOND and Probabilistic Planning. In *Proceedings of the 37th International Conference on Artificial Intelligence (KI 2014)*.

Hoffmann, J. and Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302.

Howard, R. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, USA.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research*, 11:1563–1600.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press, Cambridge, MA, USA.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285.

Kallenberg, L. (1994). Survey of Linear Programming for Standard and Nonstandard Markovian Control Problems. Part I: Theory. *ZOR – Mathematical Methods of Operations Research*, 40(1):1–42.

Karlin, S. (1962). Stochastic Models and Optimal Policy for Selling an Asset. *Studies in Applied Probability and Management Science*, pages 148–158.

Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *Journal of Machine Learning Research*, 49(2–3):193–208.

Keller, T. and Eyerich, P. (2011). A Polynomial All Outcome Determinization for Probabilistic Planning. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pages 331–334. AAAI Press.

Keller, T. and Eyerich, P. (2012). PROST: Probabilistic Planning Based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 119–127. AAAI Press.

Keller, T. and Geißer, F. (2015). Better Be Lucky Than Good: Exceeding Expectations in MDP Evaluation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press.

Keller, T. and Helmert, M. (2013). Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pages 135–143.

Kiesel, S. and Ruml, W. (2014). Planning Under Temporal Uncertainty Using Hindsight Optimization. In *24th International Conference on Automated Planning and Scheduling (ICAPS 2014): Planning and Robotics Workshop*.

Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, pages 282–293.

Koenig, S. and Likhachev, M. (2002). D* Lite. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002)*, pages 476–483.

Kolobov, A., Dai, P., Mausam, and Weld, D. S. (2012a). Reverse Iterative Deepening for Finite-Horizon MDPs with Large Branching Factors. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pages 146–154.

Kolobov, A., Mausam, and Weld, D. S. (2012b). LRTDP vs. UCT for Online Probabilistic Planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, pages 1786–1792.

Kuleshov, V. and Precup, D. (2010). Algorithms for the Multi-Armed Bandit Problem. *Journal of Machine Learning Research*.

Lai, T. L. and Robbins, H. (1985). Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22.

Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. (2009). The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):73–89.

Likhachev, M. and Stentz, A. (2006). PPCP: Efficient Probabilistic Planning with Clear Preferences in Partially-Known Environments. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI 2006)*, pages 7860–867.

Little, I. and Thiébaux, S. (2007). Probabilistic Planning vs Replanning. In *ICAPS Workshop International Planning Competition: Past, Present and Future*.

Littman, M. L. (1996). *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Providence, Rhode Island.

Marchetti-Spaccamela, A. and Vercellis, C. (1995). Stochastic On-line Knapsack Problems. *Mathematical Programming*, 68(1):73–104.

Martelli, A. and Montanari, U. (1973). Additive AND/OR Graphs. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI 1973)*, pages 1–11, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mattmüller, R. (2013). *Informed Progression Search for Fully Observable Nondeterministic Planning*. PhD thesis, Albert-Ludwigs-Universität Freiburg.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL – The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control.

McMahan, H. B., Likhachev, M., and Gordon, G. J. (2005). Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, pages 569–576.

Nebel, B. (2000). On the Compilability and Expressive Power of Propositional Planning Formalisms. *Journal of Artificial Intelligence Research (JAIR)*, 12(1):271–315.

Nikolova, E. and Karger, D. R. (2008). Route Planning under Uncertainty: The Canadian Traveller Problem. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 969–974.

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc.

Papadimitriou, C. H. and Yannakakis, M. (1991). Shortest Paths Without a Map. *Theoretical Computer Science*, 84(1):127–150.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Boston, MA, USA.

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Radlinski, F., Kleinberg, R., and Joachims, T. (2008). Learning Diverse Rankings with Multi-Armed Bandits. In *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 784–791.

Raghavan, A., Joshi, S., Fern, A., Tadepalli, P., and Khardon, R. (2012). Planning in Factored Action Spaces with Symbolic Dynamic Programming. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*.

Richter, S., Helmert, M., and Westphal, M. (2008). Landmarks Revisited. In Fox, D. and Gomes, C. P., editors, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 975–982. AAAI Press.

Rintanen, J. (2003). Expressive Equivalence of Formalisms for Planning with Sensing. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 185–194.

Robbins, H. (1952). Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-Learning Using Connectionist Systems. Technical report, Cambridge University.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Sanner, S. (2010). Relational Dynamic Influence Diagram Language (RDDL): Language Description.

Sanner, S., Goetschalckx, R., Driessens, K., and Shani, G. (2009). Bayesian Real-Time Dynamic Programming. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1784–1789.

Shiryaev, A., Xu, Z., and Zhou, X. Y. (2008). Thou Shalt Buy and Hold. *Quantitative Finance*, 8(8):765–776.

Silver, D., Sutton, R. S., and Müller, M. (2012). Temporal-Difference Search in Computer Go. *Machine Learning*, 87(2):183–219.

Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 2164–2172.

Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Journal of Machine Learning Research*, 38(3):287–308.

Smith, T. and Simmons, R. G. (2006). Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI 2006)*, pages 1227–1232.

Stentz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments. In *Proceedings of the 1994 International Conference on Robotics and Automation (ICRA 1994)*, pages 3310–3317.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8 (NIPS 1996)*, pages 1038–1044. MIT Press.

Sutton, R. S. and Barto, A. G. (1987). A Temporal-Difference Model of Classical Conditioning. pages 355–378. Lawrence Erlbaum.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Teichteil-Königsbuch, F., Kuter, U., and Infantes, G. (2010). Incremental Plan Aggregation for Generating Policies in MDPs. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1231–1238.

Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM (CACM)*, 38(3):58–68.

Tewari, A. and Bartlett, P. L. (2008). Optimistic Linear Programming Gives Logarithmic Regret for Irreducible MDPs. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1505–1512, Cambridge, MA. MIT Press.

Warnquist, H., Kvarnström, J., and Doherty, P. (2010). Iterative Bounding LAO*. In Coelho, H., Studer, R., and Wooldridge, M., editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 341–346. IOS Press.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University.

Yoon, S., Fern, A., Givan, R., and Kambhampati, S. (2008). Probabilistic Planning via Determinization in Hindsight. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1010–1016.

Yoon, S. W., Fern, A., and Givan, R. (2007). FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pages 352–360.

Yoon, S. W., Ruml, W., Benton, J., and Do, M. B. (2010). Improving Determinization in Hindsight for On-line Probabilistic Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, (ICAPS 2010)*, pages 209–217.

Younes, H. and Littman, M. (2004). PPDDL 1.0: The Language for the Probabilistic Part of IPC-4. In *Proceedings of the 4th International Planning Competition (IPC 2004), Probabilistic Part*.

Zhou, Y. and Naroditskiy, V. (2008). An Algorithm for Stochastic Multiple-Choice Knapsack Problem and Keywords Bidding. In *Proceedings of the 17th International World Wide Web Conference (WWW 2008)*, pages 1175–1176.