Optimistic planning for nonlinear optimal control and networked systems

Habilitation thesis

Lucian Buşoniu

Contents

List of Algorithms

Part I

Preliminaries

Chapter 1

Introduction

1.1 Research overview

My research work revolves around the overarching goal of developing control methods for complex, possibly unknown systems. This is motivated by the high complexity of modern controlled systems, manifested in properties such as nonlinearity, stochasticity, large scale, and distributed nature. Moreover, some of these systems cannot be accurately modeled, e.g. due to being insufficiently understood. For such a case where the model is unknown, I worked during my PhD research on the promising class of reinforcement learning (RL) methods (Sutton and Barto, 1998; Szepesvári, 2010; Lewis and Liu, 2012), which learn how to control a nonlinear stochastic system without requiring a model. Even when the model is known, the nonlinear stochastic control problem remains highly challenging: model-based counterparts to RL, like planning and dynamic programming (La Valle, 2006; Bertsekas, 2012; Powell, 2012), can address this setting. The focus of my recent, post-PhD work has been placed on this latter case, and specifically on optimistic planning algorithms and their applications.

Optimistic planning (OP) methods (Munos, 2014) solve optimal control problems modeled as Markov decision processes (MDPs) (Puterman, 1994). In this framework, a controller measures at each discrete time step the state of a process, and applies an action according to a control policy. As a result of this action, the process transits into a new state, and a scalar reward signal is sent to the controller to indicate the quality of this transition. The controller measures the new state, and the whole cycle repeats. The goal is to find an optimal policy, i.e., a policy that maximizes the cumulative reward over the course of interaction (the return). This framework can be applied to lower-level control problems such as regulation to a desired state, where the rewards are defined based on the distance to this state. It works also in higher-level problems such as a robot learning to solve a task in an unknown environment, where the rewards encode success or failure in the desired task.

OP techniques work in a local fashion by finding actions on demand for each encountered state. The local nature of planning methods reduces their dependence on state dimensionality in comparison to dynamic programming or RL, and allows most methods to naturally deal with continuous state variables, which is essential in the control of physical systems. At each step, an explorative search is made through the space of possible action sequences from the current state, represented as a tree. Then, the best first action found is applied, and entire process repeats at the next step. Planning techniques are thus a very general type of model-predictive control. Since computation is limited in the online setting, the search must be efficient, and a good way to achieve this is by applying the principle of *optimism* in the face of uncertainty: given the choice between action sequences having uncertain values, more promising sequences are explored first. Formally, the most promising sequence is one with the largest upper bound on its return. Optimistic methods combine in a novel way ideas from optimization, planning and graph search (La Valle, 2006), and RL (Sutton and Barto, 1998) with bandit theory in particular playing an important role (Auer et al., 2002).

Several OP algorithms had already been developed when I entered the field, mostly in the discrete-action case (Kocsis and Szepesvári, 2006; Hren and Munos, 2008; Bubeck and Munos, 2010; Walsh et al., 2010), with Upper Confidence Trees perhaps the most widely known technique (Kocsis and Szepesvári, 2006). Several algorithms also worked for continuous actions (Mansley et al., 2011; Weinstein and Littman, 2012). Some of these methods have very useful features: they are applicable to general, nonlinear dynamics with general, nonquadratic cost functions; and they provide near-optimality guarantees that are placed in a tight relationship with the computational budget invested by the algorithm (Hren and Munos, 2008; Bubeck and Munos, 2010). However, they achieved this under restrictive conditions: discrete actions; and either for deterministic systems without uncertainties and disturbances, or for stochastic systems but only searching in the suboptimal class of open-loop action sequences. The first major part of this thesis will present my research to overcome these limitations.

Specifically, two deterministic algorithms are introduced: one that works for continuous actions, and one for handling disturbances in a robust, minimax approach that treats them as the actions of an opponent. Then, the case of stochastic systems is considered, which can be used to model e.g. noises or disturbances with known probability distributions. An algorithm is first provided for distributions with discrete and finite support, which is then extended to a class of continuous density functions via sigma-point discretization. The near-optimality of all the algorithms with the exception of the continuous-action one is characterized, introducing several novel measures of problem complexity appropriate for each particular class of problem. Additionally, all the algorithms are extensively evaluated in simulation experiments.

Beyond their fundamental interest in optimal control, the generality of OP methods also makes them useful to address other challenges in nonlinear control. In par-

ticular, networked systems are becoming extremely important in today's world: communication networks, power and transport grids, decentralized computing networks, and social networks are just some examples of such systems influencing everyday life. The second major part of the thesis therefore investigates applications of optimistic methods to the control of networked systems, treating these systems from two complementary perspectives. The first perspective tackles the coordinated behavior of multiple, interconnected systems called agents, under the constraints imposed by the interconnection topology. In this context, an algorithm based on optimistic optimization of fixed-length action sequences is first proposed, in order to achieve consensus over the agents' state variables under a fixed communication topology. Then, OP over variable-length action sequences is applied to achieve flocking, where the topology is dictated by a proximity relationship between the agents; here the main analytical aim is to guarantee the preservation of the interconnection topology under this constraint.

The second perspective deals with communication constraints induced by a network interposed between a single system and its controller. Two optimal, networked control strategies using OP are proposed to reduce the number of transmissions over the network. In the first strategy, action sequences are transmitted to the plant at a fixed period. In the second strategy, the algorithm decides the next transmission instant according to the last state measurement (leading to a self-triggered policy), working within a fixed computation budget. The algorithms are thoroughly analyzed, showing that they effectively solve the problems they target.

In addition to their analysis, all the algorithms for networked systems are evaluated in numerical examples.

The above two directions, into OP algorithms and their applications to nonlinear control, comprise the main thrust of my post-PhD research. Additional directions in RL, applications, and secondary planning and control topics are reviewed in separate chapters. Direct offshoots of my PhD research are not discussed, even if they were done or published after the PhD. The same applies to work where I participated but did not take a leading role.

In the future, I will start from the planning and control research presented in this thesis and I will integrate novel control insights, together with machine learning and RL ideas, in order to approach my overall objective of *an algorithmic framework for the learning and planning control of complex systems*. An important component will be the integration of OP with RL ideas from my previous expertise, in order to achieve hybrid algorithms with the advantages of both techniques. Stability guarantees for the solution obtained will also be very important. Applicative directions will be continued and new applications will be sought, both in general classes of nonlinear control problems and also in the specific area of assistive robotics. All these results will serve as a solid platform from which to explore new directions in decision and control on the one hand, and machine learning and artificial intelligence on the other.

1.2 Advising and management activities

Throughout my research I have been managing a wide group of student projects, both in the Netherlands and in Romania. I have been leading as coadvisor 2 PhD students, one of whom graduated in March 2015; 9 Master and 9 Bachelor students; among these, two Master projects finalized *cum laude* in the Dutch system. With my students, I investigated an agenda of planning, learning, and control topics complementary to my main research lines, see Chapters 5, 8, and 9.

I have successfully acquired funding for my research in one national project which is funding a young team over the period 2013-2016; two bilateral, international cooperation projects (one PHC-Brâncuşi and a PICS, the latter funded by the French side); as well as an internal grant within a funding initiative of the Research Center for Automatic Control of Nancy, France. I have also been involved in local project management in French, Dutch, and Romanian research projects.

Recently, I have initiated a new research group on "Robotics and Nonlinear Control" at the Department of Automation of the Technical University of Cluj-Napoca. I have been coorganizing the 2014 (Orlando) and 2015 (Cape Town) editions of the main event at the intersection of reinforcement learning and engineering: the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. In addition, I have been leading several special sessions in previous years.

1.3 Teaching activities

In my work so far, I have been involved in three different academic systems: Romanian, Dutch, and French, in several complementary roles: professor, student, project advisor, and researcher. My teaching career has begun 12 years ago, during my postgraduate studies at the Technical University of Cluj-Napoca, when I taught practical classes for the disciplines *Reliability and diagnosis*, *Computer-aided design*, and *Programming*. At the same place, but in the complementary role of student, I was exposed directly to the scientific literature, in course projects that required the critical investigation and evaluation of methods proposed in published papers. This experience widened my horizons and was essential in my decision to follow an academic career. Later, at the Technical University of Delft, I experimented myself with this way of stimulating critical thinking in students, using a literature project that I helped develop and coordinate. This was for a Master-level course on *Knowlege-based control systems*, within which I also taught invited lectures on reinforcement learning. I was also invited to hold such a lecture for the course *Control methods for robotics*. In general, my teaching work at TUDelft taught me to coordinate projects and exams for a large number of students, and introduced me to the development and presentation of new lectures.

My *independent* teaching career started in 2011 at the Technical University of

Cluj-Napoca, where I was first an invited lecturer (March 2011), then lecturer (October 2011), and finally associate professor (October 2014). Here, I was responsible with creating from the grounds up a new Master discipline on *Learning control*. I also fully restructured the English-line Bachelor discpline of *System identification*. I am currently leading both disciplines, including lecture, laboratory, project, and examination work.

In addition, I was invited to present at several workshops and tutorials at top conferences in the control and robotics fields, an experience which combines cutting-edge research with a didactic approach. Overall, my expertise familiarized me with the full spectrum of teaching skills needed in a University teaching career.

1.4 Outline of the thesis

Figure 1.1 shows a roadmap of the thesis in a graphical form. The main content is structured hierarchically along the two major threads of work presented above. Namely, Part II focuses on fundamental developments in planning algorithms for optimal control, and Part III on applications of these algorithms to the control of networked systems. Before this, Part I describes some necessary background, after the present introduction. At the end, in Part IV, other research directions are outlined, and an overall plan for the future is delineated. For easy reference, local lists of references are provided at the end of each part.

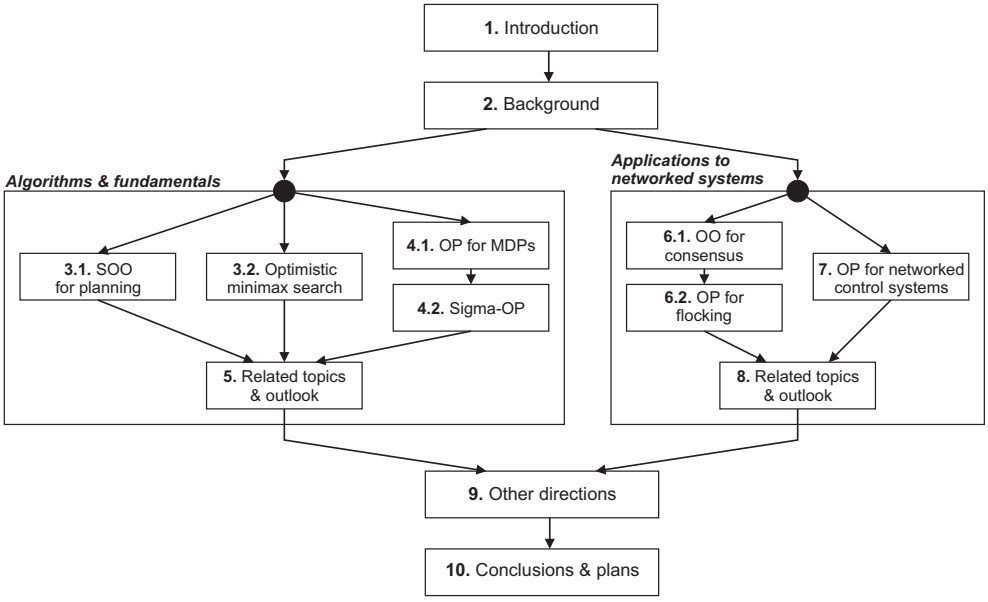

Figure 1.1: A roadmap for the thesis. Arrows indicate dependencies between topics. Chapter and section numbers are provided.

Both of the main Parts, II and III, are structured in a similar way: after a brief outline (not shown in the figure), the major research contributions in that direction are presented, while at the end secondary topics are briefly discussed, together with an outlook of open issues and ongoing work. In particular, the main algorithmic developments in Part II concern simultaneous optimistic optimization for planning, which handles continuous control actions; optimistic minimax search for adversarial problems; and OP for MDPs together with sigma-OP for stochastic problems. In Part III, optimistic optimization and planning are applied to the control of multiagent systems to achieve consensus or flocking goals; and optimistic planning is used in (single-controller) networked control systems.

The two parts can be read largely independently, with the first one being tailored more for planning and RL researchers; while the second part is more directly palatable to control engineers. Within each part, some of the topics are themselves independent so they can be read separately. All these interdependencies are shown in Figure 1.1, and each possibility of following the arrows from the top to the bottom of this figure indicates one short way of reading the thesis. In addition, one can interrupt such a thread at any time in order to jump back to the black disks and follow another arrow. One particular way of doing this leads to reading the thesis sequentially.

There are some minor cross-dependencies that are left out of the figure, but these do not affect the main flow. E.g. OP for MDPs (Section 4.1) is needed to understand some of the related topics for networked systems in Chapter 8; and some occasional backtracking may be necessary to find the description of example problems, because the same example is used multiple times but in order to avoid redundancy it is only presented in detail once, the first time it is used.

1.5 Acknowledgments

I am grateful to all my students and collaborators for their contributions to the work presented here. Among the students, I would like to mention especially my PhD students Ivo Grondman and Koppány Máthé; and my Master students Elöd Páll, Sander Adam, Lex Daniels, Maarten Vaandrager, and Thijs Wensveen, all of whom did significant work ending up in journal and conference publications. Long-term, extremely fruitful collaborations with Rémi Munos, Jamal Daafouz, Constantin Morărescu, Romain Postoyan, and Damien Ernst should also be highlighted; and of course, the contribution of my PhD advisors Robert Babuška and Bart De Schutter did not stop with my degree – instead, we are continuing to work together and coauthor papers. Countless other people have contributed, formally or informally, with scientific ideas, material, and opportunities, including for example Dragan Nešic, Frank Lewis, Dimitri Bertsekas, Michail Lagoudakis, Csaba Szépesvari, Warren Powell, Raphael Fonteneau, Boris Defourny, Liviu Miclea, Levente Tamás, and on and on – too many to mention here. All this is without even mentioning the help of family and friends,

which has been and is absolutely fundamental.

I would also like to acknowledge funding from the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (projects PNII-RU-TE-2012-3-0040 and PHC Brâncuşi 781/2014), CNRS (PICS project No 6614), CRAN, and Siemens Germany (reference no. 7472/3202246859); and my host institutions after my PhD, in chronological order: TUDelft, INRIA Lille, CRAN Nancy, and the Technical University of Cluj-Napoca.

Finally, much of this thesis is based on existing papers and books with several publishers, including among others IEEE, Elsevier, Springer, and Wiley. The copyright for the material remains with the respective publishers, and I am grateful to them for hosting my publications and reusing the material here.

To recognize that the work presented here is the result of the concerted effort and contributions of all these individuals and organizations, the remainder of the thesis will be written in the first person plural.

1.6 List of acronyms

This list below collects the most frequently used acronyms in this thesis. To avoid clutter, acronyms that are only used locally are not included.

MDP	Markov decision process
RL	reinforcement learning
OO	optimistic optimization
DOO, SOO	deterministic, simultaneous optimistic optimization
OP	optimistic planning
OPD	optimistic planning for deterministic systems
OPMDP	optimistic planning for Markov decision processes
SOOP	simultaneous optimistic optimization for planning
LP	Lipschitz planning
HOLOP, OLOP	(hierarchical) open-loop optimistic planning
OMS	optimistic minimax search
COP, STOP	clock-triggered, self-triggered optimistic planning
NCS	networked control systems

Chapter 2

Background

Several technical preliminaries from the preexisting literature are necessary to construct the rest of the thesis, and we present them here. We start in Section 2.1 with the optimal control problem that most algorithms we develop aim to solve, either in its general form or in some specific cases. Then, we present in Section 2.2 two optimistic algorithms for *optimization* problems, which form the basis of the *planning* algorithms for control that the thesis focuses on. One such algorithm, optimistic planning for the specific case of deterministic systems with discrete actions, is described and characterized in Section 2.3.

2.1 Optimal control problem and Markov decision process

Throughout this thesis, we consider problems in which a nonlinear, possibly stochastic dynamic system must be optimally controlled in discrete time. Optimality is measured by a cumulative reward signal which must be maximized: the return. Such problems arise in many fields, including artificial intelligence, automatic control, operations research, economics, medicine, etc. They are often modeled as Markov decision processes (MDPs).

In an MDP, the system is described by a state signal x varying in the state space X, and can be influenced by actions u in the action space U. State space X may be countable or uncountable (e.g. continuous). The probability of reaching next state x' after action u is taken in state x is given by the probability density $f(x,u,\cdot)$, where $f : X \times U \times X \to [0,\infty)$ is called a transition function. It collects the density functions for all pairs (x,u) and describes the dynamics of the system. After the transition to x', a reward $r' = \rho(x,u,x')$ is received, where $\rho : X \times U \times X \to \mathbb{R}$ is the reward function. A control policy $\pi : X \to U$ indicates how the controller chooses actions to interact with the system. Denoting by k the discrete time index, the expected infinite-horizon

discounted return of state x under a policy π is:

$$V^\pi(x) = \mathrm{E}_{x_{k+1}\sim f(x_k,\pi(x_k),\cdot)}\left\{\sum_{k=0}^{\infty}\gamma^k r_{k+1}\right\} \tag{2.1}$$

where $x_0 = x$, $r_{k+1} = \rho(x_k, \pi(x_k), x_{k+1})$, $\gamma \in (0,1)$ is the discount factor, and the notation $x_{k+1} \sim f(x_k, \pi(x_k), \cdot)$ means that after each step k, x_{k+1} is drawn from the density $f(x_k, \pi(x_k), \cdot)$ over next states. Other types of return can also be used, such as finite-horizon or averaged over time, but we will only consider discounted returns in this thesis. We call $V^\pi : X \to \mathbb{R}$ a value function.

The goal is to control the system using an optimal policy π^*, so that the value function is maximized for every $x \in X$. This maximal, optimal value function is denoted by V^* and is unique, so it does not depend on the particular optimal policy. It is also useful to consider an action-dependent optimal value function, the optimal Q-function:[1] $Q^*(x,u) = \mathrm{E}_{x'\sim f(x,u,\cdot)}\{\rho(x,u,x') + \gamma V^*(x')\}$. Note that optimal control problems are often stated so that a cost is minimized, rather than a return being maximized – but the two formulations are equivalent. For the technical conditions under which the expectation in (2.1) is well-defined and an optimal solution is guaranteed to exist, see Bertsekas and Shreve (1978).

We make a standing assumption of reward boundedness, that will be required to hold for all the optimal control algorithms in this thesis.

Assumption 2.1 *Rewards are bounded in* $[0,1]$.

Reward boundedness is often assumed in the MDP literature, see e.g. Ch. 4 of (Bertsekas and Shreve, 1978) and (Szepesvári, 2010), since it ensures boundedness of the value in (2.1). The main way to achieve boundedness is by saturating a possibly unbounded original reward function. This changes the optimal solution, but is often sufficient in practice. Then, the resulting bounded rewards can be normalized to $[0,1]$. On the other hand, the physical limitations of the system may be meaningfully modeled by saturating the states and actions. In this case, a reward bound follows from the saturation limits.

2.2 Optimistic optimization

Next, two optimistic optimization (OO) algorithms are introduced (Munos, 2011). The problem is to maximize some function $g(z)$, $g : Z \to \mathbb{R}$. The optimization proceeds by hierarchically partitioning the domain Z. This partitioning is represented by a tree structure $\mathscr{T}$ in which each node (d,i) is labeled by a point $z_{d,i}$ and represents a subset of Z denoted $Z_{d,i}$, and containing $z_{d,i}$. Here, $d \geq 0$ is the depth in the three

[1] Note the "prime" notation, as in e.g. x', is used to generically indicate variables at the next step, whereas if the actual time step is important, it is included as a subscript, e.g. x_k.

and i is the node index at a given depth. The root of the tree represents the entire domain Z, and the tree is defined so that the children of a node form a partition of the set represented by their parent. From a computational perspective the partitioning should be easy to generate. Figure 2.1 exemplifies such a partitioning. Finally, the set of leaves of the currently explored tree is denoted by $\mathscr{L}$.

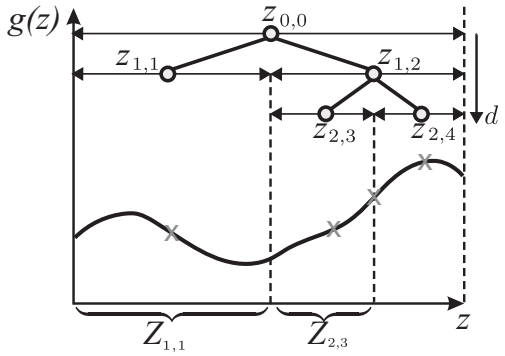

Figure 2.1: Illustration of the tree structure that is used by optimistic optimization. In this example, Z is an interval and binary partitions are used.

The core requirements of the algorithms are stated in terms of a semimetric $\ell : Z \times Z \to [0,\infty)$, where a semimetric is a function with all the properties of a metric except possibly the triangle inequality.

Assumption 2.2 *The objective function and the partitioning satisfy the following conditions:*

2.2.i There exists an optimum z^* so that:

$$g(z^*) - g(z) \leq \ell(z, z^*) \quad \forall z \in Z \tag{2.2}$$

2.2.ii There exist $c > 0$ and $\lambda \in (0,1)$ such that for any d, $\delta_{d,j} \leq c\lambda^d$ for all nodes j at depth d, where $\delta_{d,j} := \sup_{z \in Z_{d,j}} \ell(z_{d,j}, z)$ is a pseudo-diameter of $Z_{d,j}$.

2.2.iii There exists a constant μ such that any subset $Z_{d,j}$ contains a ball with center $z_{d,j}$ and radius $\mu c\lambda^d$ in the semimetric ℓ.

Intuitively, Assumption 2.2.i is a local, one-sided Lipschitz property. The other two conditions define the properties of a good partitioning procedure, which should provide well-shaped sets (Assumption 2.2.iii) that shrink with the depth in the tree (Assumption 2.2.ii). E.g. Assumption 2.2.iii forbids sets that, as the number of splits grows, become infinitely thin along some directions but remain of a constant size along others. Note that the guarantees can be generalized to the case of diameters that decrease in a different way than exponentially, but for simplicity we only handle the exponential case here.

Denote $\delta(d) = c\lambda^d$, the upper bound on diameters at d. OO for Deterministic functions (DOO) works by partitioning a set that may contain the optimum of g. It does this by assigning upper bounds to all leaf sets $Z_{d,i}, (d,i) \in \mathscr{L}$:

$$b(Z_{d,i}) = g(z_{d,i}) + \delta(d) \tag{2.3}$$

so that $b(Z_{d,i}) \geq g(z),\ \forall z \in Z_{d,i}$. Then at each iteration, an *optimistic* leaf set, which maximizes the upper bound, is further partitioned. At the end, the point with the largest value in the tree is returned. DOO is summarized in Algorithm 2.1.

Algorithm 2.1 Deterministic optimistic optimization (DOO)

Input: function g, computation budget n, partitioning of Z

1: initialize $\mathscr{T}$ with root $Z_{0,0}$
2: **for** $t = 1$ to n **do**
3: $\quad (d^\dagger, i^\dagger) \leftarrow \arg\max_{(d,i)\in\mathscr{L}} b(Z_{d,i})$
4: $\quad$ expand $(d^\dagger, i^\dagger)$ by partitioning $Z_{d^\dagger,i^\dagger}$, and add children to $\mathscr{T}$
5: **end for**

Output: $\widehat{z} = z_{d^*,i^*}$ where $(d^*, i^*) = \arg\max_{(d,i)\in\mathscr{T}} g(z_{d,i})$

DOO assumes knowledge of ℓ by using $\delta(d)$ in the upper bounds. The alternative, Simultaneous OO (SOO), does not require this knowledge. Instead, at each round, SOO simultaneously expands all potentially optimal leaf sets: those for which the upper bound could be largest under *any* semimetric ℓ satisfying the conditions. With a little thought, a set can only contain a largest upper bound if its sample value is at least as good as the values of all sets with diameters larger than its own; we say that the set is not dominated by larger sets. Since further, $\delta(d)$ decreases with d, we only have to compare with leaves higher up the tree. At each iteration t, the algorithm expands at most one leaf set at each depth. If we define $\mathscr{L}_{\leq d}$ as the set of leaf nodes having depths $d' \leq d$, then a leaf (d,i) is only expanded if $g(z_{d,i}) = \max_{(d',i')\in\mathscr{L}_{\leq d}} g(z_{d',i'})$; if there are several such leaves one is chosen arbitrarily. This selection procedure is illustrated in Figure 2.2. SOO additionally limits the tree depth at each iteration t with a function $d_{\max}(t)$, a parameter of the algorithm that controls the tradeoff between deeper or more uniform exploration. Algorithm 2.2 summarizes SOO. Note that in the form given here, SOO may take more than the budget n to finish the last iteration.

The convergence of DOO and SOO depends on the smoothness of the function g in the semimetric ℓ, formalized next. Define first the near-optimal sets $Z_\varepsilon = \{z \in Z \,|\, g(z^*) - g(z) \leq \varepsilon\}$. For any ε, the packing number of Z_ε is defined as the maximal number of disjoint ℓ-balls with centers in Z_ε and equal radii $\mu\varepsilon$ (recall μ from Assumption 2.2.iii). Finally, the *near-optimality dimension* is the smallest $\beta > 0$ so that there exists a positive constant C, such that the packing number is smaller than $C\varepsilon^{-\beta}$. The optimization problem is easier to solve when the semimetric ℓ captures

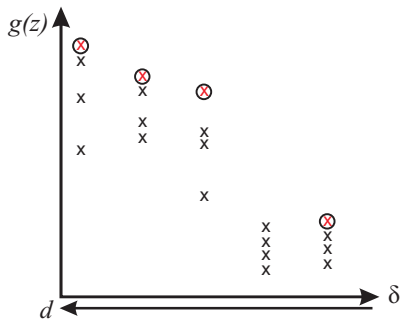

Figure 2.2: Illustration of set selection in SOO. Depth d increases to the left, while set diameter δ increases to the right. Samples are shown as 'z', and the samples of sets selected for expansion are circled and colored red.

more closely the smoothness of the objective function g around z^*, in which case β is smaller, with the ideal β being 0. In some contrived cases β may be infinite, e.g. when g is constant and a 'malicious' semimetric is defined that shrinks super-exponentially around z^*. In order to eliminate these edge cases, another technical assumption is needed.

Assumption 2.3 *A finite β exists.*

Then, the following results hold (Munos, 2011).

Proposition 2.1 (Near-optimality) *Given a budget n, DOO returns a solution $\widehat{z}$ satisfying:* [2]

$$g(z^*) - g(\widehat{z}) = \begin{cases} O(n^{-1/\beta}) & \text{if } \beta > 0 \\ O(\lambda^{n/C}) & \text{if } \beta = 0 \end{cases}$$

where C is the constant from the near-optimality dimension. SOO returns a solution $\widehat{z}$ satisfying:

$$g(z^*) - g(\widehat{z}) = \begin{cases} O(n^{-(1-a)/\beta}) & \text{if } \beta > 0 \\ O(\lambda^{\sqrt{n}/C'}) & \text{if } \beta = 0 \end{cases}$$

where $C' \geq 1$ is a constant, and we choose $d_{\max}(t) = t^a$ (with $a > 0$) when $\beta > 0$ and $d_{\max}(t) = \sqrt{t}$ when $\beta = 0$.

Thus, sub-optimality decreases as a power of n which depends on the problem complexity as expressed by β; and when $\beta = 0$, the decrease is faster – exponential,

[2] Let $g, h : (0,\infty) \to \mathbb{R}$. Statement $g(t) = O(h(t))$ (or $g(t) = \Omega(h(t))$) for large t means that $\exists t_0, c > 0$ so that $g(t) \leq ch(t)$ (or $g(t) \geq ch(t)$) $\forall t \geq t_0$. When the statement is made for small t, it means that $\exists t_0, c > 0$ so that the same inequalities hold for $\forall t \leq t_0$. $f(t) = \tilde{O}(g(t))$ for large (or small) t when $\exists a, b, t_0 > 0$ so that $f(t) \leq a(\log g(t))^b g(t)$ $\forall t \geq t_0$ (or $\forall t \leq t_0$). The logarithmic term asymptotically becomes negligible compared to $g(t)$.

Algorithm 2.2 Simultaneous optimistic optimization (SOO)

Input: function g, depth function $d_{\max}(\cdot)$, budget n,
partitioning of Z

initialize $\mathscr{T}$ with root $Z_{0,0}$; $t \leftarrow 1$
while $t \leq n$ **do**
  $g_{\max} \leftarrow -\infty$
  for $d = 0$ to $\min(\text{depth}(\mathscr{T}), d_{\max}(t))$ **do**
    $i^\dagger \leftarrow \arg\max_i g(z_{d,i})$
    if $g(z_{d,i^\dagger}) \geq g_{\max}$ **then**
      expand $(d, i^\dagger)$, add children to $\mathscr{T}$
      $g_{\max} \leftarrow g(z_{d,i^\dagger})$
      $t \leftarrow t+1$
    end if
  end for
end while

Output: $\widehat{z} = z_{d^*,i^*}$ where $(d^*, i^*) = \arg\max_{(d,i)\in\mathscr{T}} g(z_{d,i})$

confirming that the problem is 'easy' and solved efficiently by the algorithm. Note that DOO and SOO have similar guarantees. Since it does not use the semimetric, SOO must expand more sets, and it converges at a somewhat slower rate. However, for the same reason, SOO has a surprising advantage: it converges at the fastest rate allowed by any valid semimetric.

2.3 Optimistic planning for deterministic systems

In this section we consider a deterministic variant of the problem in Section 2.1, where the transition function reduces to $x' = f(x,\ u)$, since a single state x' is reached with probability 1; and the reward function to $r' = \rho(x,u)$, since the next state x' – and hence the reward – are fully determined by x and x.[3] Because the system is deterministic, a solution from a given initial state x can be represented by an infinitely-long sequence of actions $\boldsymbol{u}_\infty = (u_0, u_1, \dots) \in U^\infty$, and the discounted return (value) of this sequence is:

$$V^{\boldsymbol{u}_\infty}(x) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{2.4}$$

where $x_0 = x, x_{k+1} = f(x_k, u_k)$ for $k \geq 0$, and $\gamma \in [0,1)$ is the discount factor. Then, the optimal value function satisfies $V^*(x) = \sup_{\boldsymbol{u}_\infty} V^{\boldsymbol{u}_\infty}(x)$. Note that action sequences

[3] Here as well as in the sequel, we slightly abuse the notation by using the same symbols to denote analogous but mathematically different objects in the stochastic and deterministic case. For example, the deterministic $\rho(x,u)$ is obtained by plugging x' in the stochastic reward function, $\rho(x,u,x')$. It will usually be clear from the context to which variant the text refers; when it is not, we make it explicit.

are more general than a state-feedback policy π, which would also be sufficient to represent the solution, as in Section 2.1. We will nevertheless use the sequence-based formulation since it better fits our approach.

In addition to reward boundedness from Assumption 2.1, we restrict U to a finite set.

Assumption 2.4 *The action space is discrete (or discretized),* $U = \{u^1, \ldots, u^M\}$.

Many systems have inherently discrete and finitely-many actions, because they are controlled by switches. This is the case e.g. in traffic signal control (De Schutter and De Moor, 1998) or water level control by barriers and sluices (van Ekeren et al., 2013). When the actions are originally continuous, discretization reduces performance, but the loss is often manageable. Discretized actions may sometimes even be preferable, e.g. actuator saturation can be dealt with by simply discretizing within the operating ranges.

To introduce the algorithm, we focus on a particular state x where it must be applied, and by convention set the current time to 0, so that $x_0 = x$. Of course, the procedure works at any time step.

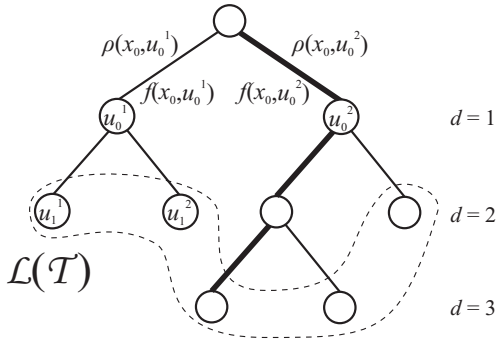

Figure 2.3: Illustration of an OPD tree $\mathcal{T}$. Nodes are labeled by actions, arcs represent transitions and are labeled by the rewards and next states resulting by applying the corresponding action. Subscripts are depths, superscripts index the M possible actions/transitions from a node (here, $M = 2$). The leaves $\mathcal{L}(\mathcal{T})$ are enclosed in a dashed line, while the thick path highlights an action sequence. Note that the root corresponds to the empty sequence.

Optimistic planning for deterministic systems (OPD) (Hren and Munos, 2008) explores a tree representation of the possible action sequences from the current state, as illustrated in Figure 2.3. OPD starts with an unlabeled root node, and iteratively expands nodes, where each expansion adds new children nodes corresponding to all the M actions $u^1, \ldots, u^M$. Each node at some depth d is reached via a unique path through the tree, and can thus be uniquely associated to the sequence of actions

Algorithm 2.3 Optimistic planning for deterministic systems (OPD)

Input: state x, budget n or depth d (set the other to ∞)

initialize tree: $\mathscr{T} \leftarrow \{\text{root}\}$, $i = 0$
repeat
  find optimistic sequence: $\boldsymbol{u}^\dagger \in \arg\max_{\boldsymbol{u} \in \mathscr{L}(\mathscr{T})} b_x(\boldsymbol{u})$
  add children $u^j, j = 1, \ldots, M$ to the node of $\boldsymbol{u}^\dagger$
  $i \leftarrow i + 1$
until $i = n$ or $\Delta(\mathscr{T}) = d + 1$
$n \leftarrow i$; $d \leftarrow \Delta(\mathscr{T}) - 1$

Output: $\boldsymbol{u}^* \in \arg\max_{\boldsymbol{u} \in \mathscr{L}(\mathscr{T})} l_x(\boldsymbol{u})$, d, n

$\boldsymbol{u}_d = (u_0, u_1, \ldots, u_{d-1})$ on this path. In what follows, we will work interchangeably with sequences and paths, keeping this equivalence in mind.

For a sequence $\boldsymbol{u}_d$, we define three quantities:

$$l_x(\boldsymbol{u}_d) = \sum_{d'=0}^{d-1} \gamma^{d'} \rho(x_{d'}, u_{d'}), \quad b_x(\boldsymbol{u}_d) = l_x(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma}$$
$$v_x(\boldsymbol{u}_d) = l_x(\boldsymbol{u}_d) + \gamma^d V^*(x_d)$$

where the states are generated with the action sequence $\boldsymbol{u}_d$, like in (2.1). Subscript x indicates that the three quantities depend on the state $x = x_0$ where OPD is applied. Due to Assumption 2.1, the rewards below depth d are in $[0, 1]$, so $l_x(\boldsymbol{u}_d)$ provides a lower bound on the value of any infinite sequence that starts with $\boldsymbol{u}_d$, while $b_x(\boldsymbol{u}_d)$ is an upper bound. Value $v_x(\boldsymbol{u}_d)$ is obtained by continuing optimally after $\boldsymbol{u}_d$. We call functions b and l respectively "b-values" and "l-values" in the text.

Recall notation $\mathscr{L}(\mathscr{T})$ for the set sequences corresponding to leaves of $\mathscr{T}$. OPD *optimistically* explores the space of action sequences, by always expanding further a most promising leaf sequence: one with the largest upper bound $b_x(\boldsymbol{u})$. At the end, a sequence that maximizes the lower bound $l_x(\boldsymbol{u})$ among the leaves is returned. Since leaves sit at varying depths d in the tree so that $\gamma^d/(1-\gamma)$ varies, maximizing $l_x(\boldsymbol{u})$ is different from maximizing $b_x(\boldsymbol{u})$, and can intuitively be seen as making a safe choice. Algorithm 2.3 summarizes the entire procedure, where function $\Delta(\cdot)$ gives the depth of a tree, and any ties among several sequences maximizing upper or lower bounds are broken arbitrarily. We allow the algorithm to terminate either after a given number of expansions, or after a node at given depth d has been expanded, leading to $\Delta(\mathscr{T}) = d + 1$. Sometimes a sequence of length $\Delta(\mathscr{T})$ may be returned, in which case the last action is removed for uniformity of analysis.

The computational budget is measured by the number of expansions, since an expansion takes M calls to the model f and to the reward function ρ, and for nonlinear systems computing f dominates the execution time. Other tree operations (such as computing b-values or traversing the tree to find the optimistic sequence) are signifi-

cantly cheaper, but can be bounded between $O(n \log n)$ and $O(n^2)$, depending on the branching factor $\kappa(x)$ defined in the next section.

To characterize the complexity of finding the optimal sequence from a given state x, we use the branching factor $\kappa(x)$ (average number of children per node) of the near-optimal subtree. This subtree contains only nodes that, given the rewards obtained down to them in the tree, cannot be ruled out as belonging to optimal sequences. In general, exploring these nodes is unavoidable, and $\kappa(x)$ is in this sense necessary to characterize the problem. OPD *only* explores the near-optimal subtree, leading to a priori guarantees on the relation between computation, sequence length, and near-optimality. Since $\kappa(x)$ is generally unknown, actual values for e.g. near-optimality cannot be determined in advance. Nevertheless, the analysis provides confidence that OPD automatically adapts to the complexity of the problem at state x, described by $\kappa(x)$. We return to detail these properties after the formal development is in place.

The near-optimal subtree is defined as:

$$\mathscr{T}^*(x) = \{\boldsymbol{u}_d \,|\, d \geq 0, V^*(x) - v_x(\boldsymbol{u}_d) \leq \gamma^d/(1-\gamma)\} \tag{2.5}$$

Let $\mathscr{T}_d^*(x)$ be the set of nodes at depth d on $\mathscr{T}^*(x)$ and $|\cdot|$ denote set cardinality, then the asymptotic branching factor is defined as $\kappa(x) = \limsup_{d\to\infty} \left|\mathscr{T}_d^*(x)\right|^{1/d}$.

A sequence $\boldsymbol{u}_d$ is said to be ε-optimal when $V^*(x) - v_x(\boldsymbol{u}_d) \leq \varepsilon$. The upcoming theorem is a consequence of the analysis in (Hren and Munos, 2008; Munos, 2014). Part (i) of the theorem shows that OPD returns a long and near-optimal sequence, and parts (ii), (iii) show that sequence length and near-optimality are closely related to the computation budget, via branching factor $\kappa(x)$.

Theorem 2.2 *Let $x \in X$. When OPD is called at x:*

(i) The length of the sequence $\boldsymbol{u}^*$ returned is $d = \Delta(\mathscr{T}) - 1$. Denoting $\varepsilon(x) = V^*(x) - l_x(\boldsymbol{u}^*)$, we have $\varepsilon(x) \leq \frac{\gamma^d}{1-\gamma}$.

(ii) When OPD is called with large target depth d: • If $\kappa(x) > 1$ it will require a number of expansions $n(x) = O(\kappa(x)^d)$. • If $\kappa(x) = 1$, $n(x) = O(d)$.

(iii) When OPD is called with large budget n: • If $\kappa(x) > 1$ it will reach a depth of $d(x) = \Omega(\frac{\log n}{\log \kappa(x)})$, and $\varepsilon(x) = O(n^{-\frac{\log 1/\gamma}{\log \kappa(x)}})$. • If $\kappa(x) = 1$, $d(x) = \Omega(n)$ and $\varepsilon(x) = O(\gamma^{c(x)n})$, where $c(x)$ is a constant. □

Proof:(sketch) Item (i) follows from the proof of Theorem 2 in (Hren and Munos, 2008), (ii) from the proof of Theorem 3 in (Hren and Munos, 2008), and (iii) from the proofs of Theorems 2 and 3 in (Hren and Munos, 2008). An outline for part (ii) is given here, since it will be useful later in our analysis. OPD has the important property that it only expands nodes in $\mathscr{T}^*(x)$, although it uses solely reward information from the current tree (Hren and Munos, 2008; Munos, 2014). According to item (i),

performance is dominated by the depth reached. Thus the worst case is when nodes in $\mathscr{T}^*(x)$ are expanded in the order of their depth. Now, $\mathscr{T}^*(x)$ contains $n = O(\kappa(x)^d)$ nodes up to depth d when $\kappa > 1$, and $n = O(d)$ otherwise. Inverting these relationships obtains the formulas for d in the Theorem statement, and replacing these expressions for d into $\frac{\gamma^d}{1-\gamma}$ provides the corresponding near-optimality bounds. ■

The sequence returned is immediately $\varepsilon(x)$-optimal, since $V^*(x) - v_x(\boldsymbol{u}^*) \leq V^*(x) - l_x(\boldsymbol{u}^*) \leq \varepsilon(x)$ in view of part (i); the second inequality here is stronger than $\varepsilon(x)$-optimality.

To build more intuition on $\mathscr{T}^*(x)$ and $\kappa(x)$, note that $\mathscr{T}^*(x)$ contains sequences for which it is impossible to tell, from their rewards down to d, whether or not they are part of an optimal solution, because their near-optimality is smaller than the amount of reward $\gamma^d/(1-\gamma)$ they might accumulate below depth d. Usually only some sequences have this property, therefore $\mathscr{T}^*(x)$ is smaller than the complete tree and $\kappa(x)$ is smaller than the number of actions M. The smaller $\kappa(x)$, the more easily near-optimal sequences can be distinguished, and the better OPD does. The best case is $\kappa(x) = 1$, obtained e.g. when a single sequence always obtains rewards of 1, and all the other rewards are 0. In this case the algorithm must only develop this sequence, and suboptimality decreases exponentially. In the worst case, $\kappa(x) = M$, obtained e.g. when all the sequences have the same value, the algorithm must explore the complete tree in a uniform fashion, expanding nodes in order of their depth.

OPD is closely related to the classical A* heuristic search algorithm, see Ch. 2 of (La Valle, 2006), and can in fact be seen as an extension of A* to infinite-horizon problems using the heuristic $\gamma^d/(1-\gamma)$ on the leaf values.

It is also instructive to examine the relation between OPD and DOO. OPD is, in fact, DOO applied to optimize the return $g(z) = V^{\boldsymbol{u}_\infty}(x)$, for fixed x, over the space of infinitely-long action sequences $z = \boldsymbol{u}^\infty$. The semimetric is:

$$\ell(\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty) = \frac{\gamma^{\text{diff}(\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty)}}{1-\gamma}$$

where $\text{diff}(\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty)$ is the smallest depth where the two sequences are different. This is equal to the largest difference between the returns provided by the two sequences in any possible MDP, so clearly the objective function $V^{\boldsymbol{u}_\infty}$ satisfies the required Lipschitz-like property of Assumption 2.2 with respect to semimetric ℓ. The set of sequences Z_d corresponding to each node $\boldsymbol{u}_d$ consists of all infinitely-long sequences starting with $\boldsymbol{u}_d$. Under the chosen metric, the diameter of each such set is $\delta(Z_d) = \frac{\gamma^d}{1-\gamma}$ since any two sequences in Z_d can differ at the earliest at index d. Each such set is split by making the choices for u_d definite, so that we obtain M different children sets Z_{d+1} which are specified up to $d+1$ actions, one set for every value of $u_d \in \{u^1, ..., u^M\}$. There is one technical difference from DOO: an exact point cannot be sampled inside a set Z_d, since any such point is an infinitely long sequence of actions. This is fortunately not a problem, since upper and lower bounds

bound on the values of sequences in the can still be obtained with (2.5), and they can be used in place of the DOO b-values and function values, respectively, as show in Algorithm 2.3.

Bibliography

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47(2-3):235–256.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition.

Bertsekas, D. P. and Shreve, S. E. (1978). *Stochastic Optimal Control: The Discrete Time Case*. Academic Press.

Bubeck, S. and Munos, R. (2010). Open loop optimistic planning. In *Proceedings 23rd Annual Conference on Learning Theory (COLT-10)*, pages 477–489, Haifa, Israel.

De Schutter, B. and De Moor, B. (1998). Optimal traffic light control for a single intersection. *European Journal of Control*, 4(3):260–276.

Hren, J.-F. and Munos, R. (2008). Optimistic planning of deterministic systems. In *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, pages 151–164, Villeneuve d'Ascq, France.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings 17th European Conference on Machine Learning (ECML-06)*, pages 282–293, Berlin, Germany.

La Valle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Lewis, F. and Liu, D., editors (2012). *Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control*. Wiley.

Mansley, C., Weinstein, A., and Littman, M. L. (2011). Sample-based planning for continuous action Markov decision processes. In *Proceedings 21st International Conference on Automated Planning and Scheduling*, pages 335–338, Freiburg, Germany.

Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 783–791.

Munos, R. (2014). The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search. *Foundations and Trends in Machine Learning*, 7(1):1–130.

Powell, W. B. (2012). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2 edition.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Szepesvári, Cs. (2010). *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers.

van Ekeren, H., Negenborn, R., van Overloop, P., and De Schutter, B. (2013). Time-instant optimization for hybrid model predictive control of the Rhine-Meuse delta. *Journal of Hydroinformatics*, 15(2):271–292.

Walsh, T. J., Goschin, S., and Littman, M. L. (2010). Integrating sample-based planning and model-based reinforcement learning. In *Proceedings 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, US.

Weinstein, A. and Littman, M. L. (2012). Bandit-based planning and learning in continuous-action Markov decision processes. In *Proceedings 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, São Paulo, Brazil.

Part II

Optimistic planning for optimal control: Algorithms and fundamentals

Introduction and outline

The optimistic planning algorithm from Section 2.3 has very useful features: it is applicable to systems with general, nonlinear dynamics and general, nonquadratic cost functions; it provides near-optimality guarantees that are placed in a tight relationship with the computational budget invested by the algorithm via Theorem 2.2; and, as a side-benefit, it also gives sequences of actions that are guaranteed to be long, which will prove useful later, in Chapter 7. However, it achieves all this under restrictive conditions: the actions must be discrete and finitely many, and the system must be fully known, without any disturbances. Many real control problems do not satisfy these conditions, and therefore we dedicate Part II of this thesis to developing algorithms that do not require them.

Specifically, we start by discussing two deterministic algorithms in Chapter 3: one that works for continuous actions (Section 3.1), and one for handling disturbances in a robust, minimax approach that treats them as the actions of an opponent (Section 3.2). Then, in Chapter 4, we consider the case of stochastic systems, where the dynamics are affected by noises or disturbances with known probability distributions. We provide first an algorithm for distributions with discrete and finite support in Section 4.1, and then we extend it to a class of continuous density functions via sigma-point discretization, in Section 4.2. We analyze the near-optimality of all the algorithms with the exception of the continuous-action one, for which work is ongoing. This ongoing direction is outlined in Chapter 5, together with future work and other related directions of research, which are not discussed in detail in the thesis.

The material in this part is based on the following publications:

(P1) L. Buşoniu, A. Daniels, R. Munos, R. Babuška, "Optimistic Planning for Continuous-Action Deterministic Systems", *Proceedings 2013 Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-13)*, Singapore, 15–19 April 2013 (Section 3.1).

(P2) L. Buşoniu, R. Munos, E. Páll, "An analysis of optimistic, best-first search for minimax sequential decision making", *Proceedings 2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-14)*, Orlando, USA, 10–13 December 2014 (Section 3.2).

(P3) L. Buşoniu, R. Munos, "Optimistic Planning for Markov Decision Processes", accepted at the *15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, Canary Islands, Spain, 21–23 April 2012 (Section 4.1).

(P4) L. Buşoniu, R. Munos, B. De Schutter, R. Babuška, "Optimistic Planning for Sparsely Stochastic Systems". *Proceedings 2011 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-11)*, pages 48–55, Paris, France, 11–15 April 2011 (Section 4.1).

(P5) L. Buşoniu, R. Munos, R. Babuska, "A Survey of Optimistic Planning in Markov decision processes". In *Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control*, F. Lewis and D. Liu, Editors, series *Computational Intelligence*, Wiley, 2012 (parts of Section 4.1).

(P6) L. Buşoniu, L. Támas, "Optimistic Planning for the Near-Optimal Control of General Nonlinear Systems with Continuous Transition Distributions". *Proceedings 19th IFAC World Congress (IFAC-14)*, Cape Town, South Africa, 24–29 August 2014 (Section 4.2).

Chapter 3

Advances in deterministic systems

3.1 Optimistic planning with continuous actions

In this section we consider deterministic problems with continuous, scalar actions. In these problems, an infinite-dimensional space of action sequences must be explored. We devise an optimistic planning algorithm to perform the search starting from Lipschitzian planning (LP), see Chapter 5 of (Hren, 2012). LP splits the infinite-dimensional space into hyperboxes of increasing dimensionality, guided by upper bounds on the return of all sequences within a hyperbox. To compute the bounds, LP requires globally Lipschitz dynamics and rewards, with a known Lipschitz constant. This is an important limitation, since the system may not be Lipschitz, or even if it is, its smoothness will usually vary across the state-action space. In the latter case, the Lipschitz constant will be conservative, leading to poor performance in smoother regions.

We therefore propose a method that does not rely on the restrictive assumption of a known, global Lipschitz constant. To achieve that, we exploit the principles behind simultaneous optimistic optimization (SOO) (Munos, 2011), see Section 2.2, which only requires local smoothness around an optimum, without knowing the Lipschitz constant or indeed even the metric. We develop a nontrivial extension of SOO to the optimization of the return over infinitely long action sequences, and call the resulting algorithm *SOO for planning (SOOP)*. The main idea is to select at every iteration *all* hyperboxes that are potentially optimal for any metric – rather than the box with the largest upper bound in the given metric, like LP. Then for each selected box, a choice is made on the dimension to split further, guided by a tuning parameter.

Compared to SOO, the main novelty introduced by SOOP is a relaxed selection procedure for potentially optimal boxes. This is necessary because (roughly speaking) SOO would require sorting boxes by their diameter in the unknown metric, which is not possible. The relaxation works under a weaker, reasonable assumption on the ordering of diameters. Due to this difference and other particularities of plan-

ning, the analysis of SOOP is currently open. However, we expect good performance due to the features of SOO, which guarantee convergence to an optimum at the most favorable rate given by any valid metric.

Concerning related work, several OP methods for continuous actions existed prior to SOOP. HOOT (Mansley et al., 2011) and SP (Hren, 2012) rely on the principle of Upper Confidence Trees: they explore the space of sequences of a given length (planning horizon) K, optimizing for the kth action the return obtained over subsequent steps. HOLOP[1] (Weinstein and Littman, 2012) optimizes directly the K-horizon return relative to the initial state (at $k = 0$). All three methods are limited by searching for a sequence that is only optimal over horizon K, whereas the control problem is infinite-horizon. In principle, K can be taken sufficiently large, but this will waste computation, and in practice K is a problem-dependent parameter. The actual space that should be explored is that of infinitely long continuous-action sequences. To our knowledge, the only existing OP algorithm that does this is LP, discussed above. Note that HOOT and HOLOP also work in stochastic problems, whereas SOOP works in deterministic ones.

3.1.1 Problem statement and Lipschitz planning background

Like for OPD in Section 2.3, we consider a current state x_0 and by convention set the current time to 0. A sequence of K actions is denoted $\boldsymbol{u}_K = (u_0, u_1, \ldots, u_{K-1}) \in U^K$. It is important to note that here we do not use notation d for the sequence length, because this length no longer corresponds to depth in a tree. The discounted value of an infinitely-long sequence $\boldsymbol{u}_\infty \in U^\infty$ is given by $V^{\boldsymbol{u}_\infty}(x_0)$ from (2.4), while the truncated return of a sequence with finite length K is taken from (2.5):

$$l(\boldsymbol{u}_K) = \sum_{k=0}^{K-1} \gamma^k \rho(x_k, u_k) \tag{3.1}$$

In addition to the standing Assumption 2.1 on bounded rewards, the following structure of the action space is required.

Assumption 3.1 *The action space is the scalar unit interval: $U = [0, 1]$.*

Scalar actions are only assumed for convenience, as they allow us to introduce the derivations and the algorithm in a simple fashion. The unit interval can accommodate any other closed interval by translation and scaling, but noncompact action spaces cannot be considered.

In the scalar case, U^∞ can be visualized as an infinite dimensional hypercube on which each dimension represents the action space at that step. The goal of continuous-action planning is to explore U^∞ in such a way that after a computational budget is exhausted, a near-optimal action sequence $\boldsymbol{u}$ is returned. The method we

[1] The acronyms stand for: hierarchical optimistic optimization applied to trees (HOOT), hierarchical open-loop optimistic planning (HOLOP), and sequential planning (SP).

develop explores U^∞ by iteratively splitting it into hyperboxes (boxes, for short). Such a box $\mathscr{U}_i \subseteq U^\infty$ is the cross-product of a sequence of subspaces (intervals) $(\mu_0^i, \ldots, \mu_{K_i-1}^i, U, U, \ldots)$ where $\mu_k^i \subseteq U$ and $K_i - 1$ is the deepest discretized dimension; for all further dimensions $\mu_k^i = U$. Thus K_i is the number of discretized dimensions, and might be seen as a "length" of $\mathscr{U}_i$. A box is further explored by trisecting either the subspace of an already discretized dimension, or the space U for the first undiscretized dimension, K_i. Thus trisecting dimension k corresponds to refining further the action at step k. In Figure 3.1 an example of exploring U^∞ is shown.

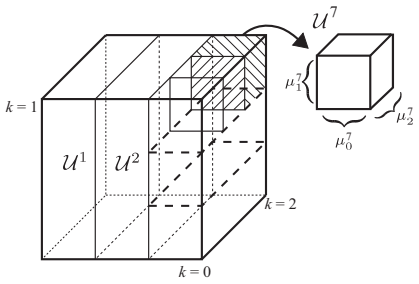

Figure 3.1: Example partition of U^∞ after 3 trisections. Dimensions 4 and higher are left out of the figure.

Define δ_k^i to be the size of subspace μ_k^i in box $\mathscr{U}_i$:

$$\delta_k^i = \begin{cases} \max_{u \in \mu_k^i} |u_k^i - u| & \text{for } 0 \le k < K_i, \\ 1 & \text{for } k \ge K_i \end{cases} \tag{3.2}$$

where u_k^i is the action at the center of μ_k^i.

Lipschitz planning (LP) (Hren, 2012) applies DOO to optimize the return (2.4) over the space U^∞ of infinite action sequences. The form of LP we introduce makes some mild changes to the version in (Hren, 2012), which we point out later. The dynamics f and rewards ρ need to be Lipschitz with a known constant L (we do not make this a formal assumption since our method does not require it):

$$\begin{aligned} \|f(x,u) - f(x',u')\| &\le L(\|x - x'\| + |u - u'|) \\ |\rho(x,u) - \rho(x',u')| &\le L(\|x - x'\| + |u - u'|) \end{aligned} \tag{3.3}$$

To apply DOO, first a semimetric ℓ is needed. After some simple calculations that exploit the Lipschitz property, the difference between the rewards obtained at step k by two sequences $\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty$ is bounded as follows:

$$|\rho(x_k, u_k) - \rho(x'_k, u'_k)| \le \sum_{j=0}^{k} L^{k-j+1} |u_j - u'_j|$$

Using this, we construct the semimetric as the following upper bound on the difference between the *returns* of the sequences:

$$\ell(\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty) = \sum_{k=0}^{\infty} \gamma^k \min\{1, \sum_{j=0}^{k} L^{k-j+1} |u_j - u'_j|\} \tag{3.4}$$

where the reward difference bounds are additionally capped by 1, since rewards are bounded in $[0,1]$.

The partitioning scheme is trisection-based. Since the samples are infinite action sequences, the algorithm never has access to a complete sample or its value (the infinite-horizon return), which would be needed for a "vanilla" implementation of DOO. Nevertheless, like in OPD the metric ℓ can still be used to provide an upper bound on the returns of sequences in a box discretized only up to some finite dimension $K_i - 1$:

$$b(\mathscr{U}_i) = \sum_{k=0}^{K_i-1} \gamma^k \min\{1, \rho_k^i + \sum_{j=0}^{k} L^{k-j+1} \delta_j^i\} + \frac{\gamma^{K_i}}{1-\gamma} \tag{3.5}$$

where ρ_k^i is the reward obtained at step k by applying the (finite) sequence $\boldsymbol{u}_{K_i}^i$ at the center of the box, and δ_k^i are the subspace sizes. Each term of the outer sum bounds the reward attainable at step k by any sequence in the box, while the fraction covers the reward attainable from step K_i onwards. The difference $b(\mathscr{U}_i) - l(\boldsymbol{u}_{K_i}^i)$, see (3.1), can be informally thought of as the diameter of $\mathscr{U}_i$.

LP works by following the principle of DOO: at each iteration, it selects an *optimistic* box $\mathscr{U}_{i^\dagger}$ that has the largest upper bound $b(\mathscr{U}_i)$, and further refines this box by trisecting one of its dimensions. To complete the algorithm, we only have to specify the dimension selection procedure. Each dimension $k < K_{i^\dagger}$ in turn is assumed trisected, and the upper bound for the resulting middle box is computed, which will be smaller due to the reduced subspace size $\delta_k^{i^\dagger}/3$. To rank the first undiscretized dimension $K_{i^\dagger}$, the center reward is assumed to be 0, and the subspace size will be $1/3$. Finally, the selected dimension is one that reduces the bound the most. Once an imposed budget n of calls to the model (f, ρ) has been depleted, the algorithm returns a center sequence with the largest return among all the boxes.

The original LP in (Hren, 2012) is different in the following ways. (i) The semimetric (3.4) and upper bound (3.5) are changed to cap individual reward bounds to 1 only after reaching the last k for which the reward bound is smaller than 1 (denote it by K'); thus (3.4) and (3.5) are tighter. (ii) If $K' < K_{i^\dagger} - 1$ for the optimistic box, only dimensions up to K' are considered for trisection, whereas we still consider all dimensions including $K_{i^\dagger}$. This avoids some pathological behavior such as when the first-step rewards ρ_0 are always 1, in which case the original LP would keep refining the first action dimension without ever going deeper. Finally, (iii) when a dimension $k < K_{i^\dagger} - 1$ is trisected, we compute all the rewards up to $K_{i^\dagger} - 1$ for the left and right center action sequences, whereas original LP only computes the kth rewards. This

allows a fair comparison with SOOP, which trisects in the same way. It may either increase or decrease performance with respect to the original LP (increase because the initial upper bounds of the left and right boxes are tighter, decrease because more model calls are spent).

3.1.2 Sequential optimistic optimization for planning

Determining the Lipschitz L constant is hard, and, in fact, it must usually be treated as a tuning parameter of LP. Even so, f or ρ may simply not be Lipschitz. If they are, a global Lipschitz constant may underestimate their smoothness in large parts of the domain, leading to inefficient partitioning. Conversely, overestimating the smoothness (taking L too small) is dangerous because the upper bounds become invalid and the DOO guarantees are lost. Therefore, we now propose an optimistic planning method that does not require a Lipschitz constant or knowing the semimetric, by exploiting the principles of SOO. Since the trisection scheme of LP is also used, many of the building blocks for the new method are already available. We still have to introduce the main insight that connects the pieces together into the overall, novel algorithm. We call this algorithm SOOP (Simultaneous Optimistic Optimization for Planning).

The main step in SOO is selecting potentially optimal sets. This is ideally done by sorting the sets by their diameters, and then only selecting sets with values undominated by the values of larger-diameter sets. Note that the diameters themselves need not be known, only their ordering; in Algorithm 2.2, because diameter decreases with increasing depth, the depth d acts as a proxy for the ordering. Unfortunately, such a global ordering is very difficult to define for the planning problem. To address this difficulty, we relax the SOO set selection procedure.

First, because depth no longer translates into a diameter ranking, we stop looking at the sets as being organized into a tree. Instead, the algorithm just works with a collection of sets (boxes in the planning context), which does not affect its validity. We define a notion of partial ordering on these boxes, and impose an assumption. For any box $\mathcal{U}_i$, denote by $s_i^k \geq 0$ the number of times the box has been trisected along dimension k.

Definition 3.1 *A box $\mathcal{U}_j$ is said to be partially greater than $\mathcal{U}_i$, denoted $\mathcal{U}_j \succeq \mathcal{U}_i$, iff* $\forall k \geq 0$, $s_k^j \leq s_k^i$.

Assumption 3.2 *If $\mathcal{U}_j \succeq \mathcal{U}_i$, then diameters $\delta(\mathcal{U}_j) \geq \delta(\mathcal{U}_i)$, where: $\delta(\mathcal{U}) = \sup_{\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty \in \mathcal{U}} \ell(\boldsymbol{u}_\infty, \boldsymbol{u}'_\infty)$ is the box diameter in the unknown semimetric ℓ.*

We expect that many useful semimetrics will satisfy Assumption 3.2. For instance, it can be shown that the Lipschitz semimetric (3.4) satisfies it. Under Assumption 3.2, we modify the box selection procedure as follows: a box $\mathcal{U}_i$ is potentially optimal and will be expanded if it is undominated by any $\mathcal{U}_j \succeq \mathcal{U}_i$; that is, if for

all $\mathscr{U}_j \succeq \mathscr{U}_i$, $l(\boldsymbol{u}^i_{K_i}) \geq l(\boldsymbol{u}^j_{K_j})$. So, $\mathscr{U}_i$ will be compared only with *some* of the boxes with larger diameters: those that are partially greater than it. It will still be expanded if it is dominated by some larger box that is not partially greater. Thus the new criterion is safe (all boxes that should be expanded are indeed expanded) but conservative (some boxes that ideally should not be expanded perhaps will be). Conservativeness implies the algorithm requires more samples than an ideal application of SOO.

The final step is specifying how to select the dimension (action step) for trisection. Ideally, the dimension that leads to the largest decrease of the diameter in ℓ should be trisected, but of course finding this decrease is not possible since ℓ is unknown. We leave this procedure open in the general method, summarized as Algorithm 3.1, and discuss alternatives below. Note that the algorithm may take more than n transitions to complete the last iteration (expand the last batch of potentially optimal boxes); alternatively, if running time is strictly limited, the latest iteration can simply be interrupted immediately upon reaching the budget n.

Algorithm 3.1 Simultaneous optimistic optimization for planning (SOOP)

Input: state x_0, model (f,ρ), budget of model calls n

initialize collection of boxes with $\mathscr{U}_1 = U^\infty$
repeat
  select potentially optimal boxes:
    $I^\dagger = \{i \,|\, \forall\ j \text{ s.t. } \mathscr{U}_j \succeq \mathscr{U}_i, l(\boldsymbol{u}^i_{K_i}) \geq l(\boldsymbol{u}^j_{K_j})\}$
  for $i \in I^\dagger$ **do**
    select dimensions $\kappa \subseteq \{0, K_i\}$ to trisect
    for $k \in \kappa$ **do**
      trisect dimension k,
        add resulting boxes to the collection
    end for
    remove parent $\mathscr{U}_i$ from the collection
  end for
until budget n has been depleted

Output: best sequence found $\boldsymbol{u}^{i^*}_{K_{i^*}}$, $i^* \in \arg\max_i l(\boldsymbol{u}^{K_i}_i)$

(i) The simplest alternative for dimension selection is to just trisect all dimensions $\{0, K_i\}$. This is safe, but very costly in terms of model calls and computation. Otherwise, one can conjecture that due to the discounting, which makes earlier actions more important, these actions should be discretized more finely. Thus a second alternative is to (ii) trisect those dimensions for which the resulting boxes are discretized more finely for smaller k, formally: $s^i_k \geq s^i_{k+1}\ \forall k \geq 0$. Then by induction, all boxes created by the algorithm satisfy the property. (iii) With the same conjecture, an even less costly heuristic may be derived that only selects one dimension. This is done by

ranking dimensions with a new discount factor $\alpha \in (0,1)$:

$$\kappa = \min\{\arg\max_{k \in \{0, K_i\}} \alpha^k (1/3)^{s_i^k}\} \tag{3.6}$$

The tuning parameter α trades off discretization accuracy and planning depth: small values will lead to finer discretizations close to the root, while with a larger value larger planning horizons are reached. In this sense, α is similar to the depth function $d_{\max}$ in SOO. With this criterion as well, all boxes produced are discretized more finely for smaller k.

Since in preliminary experiments trisecting many dimensions greatly increased computational costs without large performance benefits, we use (iii) in the sequel.

To extend the algorithm to multiple action variables, the partial ordering and the dimension selection must be changed. Denoting the action variable index by m, the partial ordering can be changed by requiring that *all* variables m at every step k are split at most as many times in $\mathcal{U}_j$ as in $\mathcal{U}_i$. Dimension selection can be performed by extending (3.6) to compare also between the variables at each k; thus a pair (k,m) that maximizes the discounted size would be selected, breaking ties in favor of small k and arbitrarily among m.

We close this section by discussing the amount of model calls required for trisections. Trisecting a box $\mathcal{U}$ of depth K along dimension k requires 3 model calls when $k = K$, and $2(K-k)$ if $k < K$. This is because in the former case all three new boxes inherit the entire center sequence $\boldsymbol{u}_K$ of $\mathcal{U}$, with the associated rewards, and must only simulate the next action (step K). When $k < K$, the center box retains again the complete information, whereas the left and right boxes only inherit the subsequence and rewards up to $k-1$, and the tails from k to $K-1$ must be simulated.

3.1.3 Experimental results

To determine the practical effectiveness of SOOP, it will be tested on three problems, in which it will be compared with three other OP algorithms. The first algorithm is OPD, which serves as a discrete-action baseline. The other two algorithms support continuous actions: they are LP, the closest relative of SOOP, and HOLOP (Weinstein and Littman, 2012). The latter is selected as a representative for the class of finite-horizon planning algorithms, which also includes HOOT (Mansley et al., 2011) and SP (Hren, 2012).

For each problem, the algorithms are executed for several values of the budget n of model calls. Like for SOOP above, the algorithms are not stopped mid-iteration, so they may take more than n calls to complete. For each value of n, the other algorithm parameters are optimized over a grid, and the best performance is reported. The parameters are: for SOOP, the discount factor α for dimension selection; for OPD, the number of discrete actions M (for every M, a uniform grid of actions is generated,

covering the whole action space); for LP, the Lipschitz constant L; and for HOLOP, the horizon K. The following parameter values are tested in all problems: for SOOP, $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$; for OPD, $M \in \{3, 5, \ldots, 15\}$; for LP, $L \in \{0.1, 0.2, \ldots, 1.5\}$; and for HOLOP, $K \in \{5, 10, 15, 20, 25, 30, 40, 50, 75, 100\}$. Since HOLOP generates solutions randomly, it is run 10 times for each experiment and a 95% confidence interval on the mean performance is computed. The best HOLOP experiment is then the one with the largest upper confidence bound.

DC motor. Consider a DC motor with two state variables: shaft angle $x_1 \in [-\pi, \pi]$ rad, angular velocity $x_2 \in [-15\pi, 15\pi]$ rad/s, and one action variable: voltage $u \in [-10, 10]$ V. The dynamics are linear:

$$f(x,u) = Ax + Bu, \quad A \approx \begin{bmatrix} 1 & 0.0095 \\ 0 & 0.9100 \end{bmatrix}, \; B \approx \begin{bmatrix} 0.0084 \\ 1.6618 \end{bmatrix}$$

The goal is to stabilize both states at zero, and is described by the unnormalized reward function:

$$\tilde{\rho}(x,u,x') = -x^T Q_{\text{rew}} x - u^T R_{\text{rew}} u, \; Q_{\text{rew}} = \text{diag}(1, 0.001), \; R_{\text{rew}} = 0.05 \qquad (3.7)$$

with discount factor $\gamma = 0.95$. Using the known variable bounds, the reward is normalized (scaled and translated) into $[0, 1]$, and for the sake of applying the continuous-action algorithms, the same is done for the action.

This first problem is chosen because it is simple and can be solved with short planning horizons. Nevertheless, continuous (or finely discretized) actions are necessary for good performance, due to the quadratic action penalty. The four planning algorithms are applied in receding horizon, from the initial state $[-\pi, 0]^T$ and for a duration of 1 s (100 steps). Table 3.1 shows the best parameters of the algorithms. Each row corresponds to an algorithm, and each column to a budget value n, and the content cells show the best value of each algorithm's parameter for the corresponding n. Figure 3.2 shows the best returns obtained.

Table 3.1: DC motor: the best values of the algorithm parameters for each n.

$n =$	100	500	1000	2500	5000
SOOP, $\alpha =$	0.8	0.7	0.8	0.7	0.7
OPD, $M =$	3	3	3	3	5
LP, $L =$	0.9	0.6	0.6	0.7	0.5
HOLOP, $K =$	5	5	5	5	5

SOOP is clearly better than OPD, as expected from the fact that coarse actions are not sufficient. An interesting observation is that despite this, discretizing finely is not worth the additional price paid in terms of model calls in OPD (since a larger tree

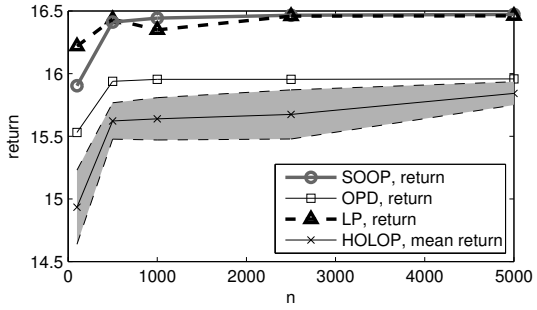

Figure 3.2: Performance for the DC motor. For HOLOP, the mean performance with its 95% confidence interval is shown.

must be explored), not even for larger budgets. Only for $n = 5000$ do we get better performance by taking $M = 5$ discrete actions.

SOOP and LP are performing similarly: LP is better for small budgets, while SOOP overtakes it for larger ones. Apparently, a global Lipschitz assumption works in this problem, which is not surprising due to its simplicity.

HOLOP is doing worse than all others, and looking at controlled trajectories (not shown here due to space limitations) this is due to very coarse actions which are not able to stabilize the system. Thus, for the budgets considered here, HOLOP cannot sufficiently refine the solution.

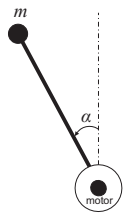

Figure 3.3: Inverted pendulum schematic.

Inverted pendulum swingup. The second problem is swinging up and stabilizing an underactuated inverted pendulum rotating in a vertical plane, see Figure 3.3. Due to limited power, from certain states (e.g., pointing down) the pendulum needs to be swung back and forth to gather energy, prior to being pushed up and stabilized. The first state $x_1 = \alpha$ is the angle and wraps around in the interval $[-\pi, \pi)$ rad; the second state is the angular velocity $x_2 = \dot{\alpha} \in [-15\pi, 15\pi]$ rad/s. The action $u \in [-3, 3]$ V is the motor voltage, see (Buşoniu et al., 2010a), Section 4.5.3 for the dynamics. The goal of stabilizing the pendulum pointing up is expressed by quadratic rewards of the form (3.7) with $Q_{\text{rew}} = \text{diag}(1, 0)$, $R_{\text{rew}} = 0.3$, and the discount factor is $\gamma = 0.95$.

Like before, rewards and actions are normalized into $[0,1]$.

While it is a standard benchmark in control and dynamic programming, this problem nevertheless supplies an interesting challenge to planning algorithms: the solution must be planned over a longer horizon, and solutions that seem good over a short horizon will not work, instead just pushing the pendulum in one direction. Furthermore, continuous actions are necessary, firstly due to the action penalty, and secondly to properly stabilize the pendulum in the unstable, pointing-up position. The planning algorithms are applied from an initially pointing down position, $x = [-\pi, 0]^T$, for a duration of 5 s (100 steps). Table 3.2 shows the algorithm parameters and Figure 3.4 the best returns.

Table 3.2: Parameters for the inverted pendulum.

$n =$	500	1000	5000	10000	15000
SOOP, $\alpha =$	0.9	0.8	0.7	0.7	0.7
OPD, $M =$	3	3	3	3	5
LP, $L =$	0.1	0.2	0.1	0.1	0.7
HOLOP, $K =$	10	10	10	10	10

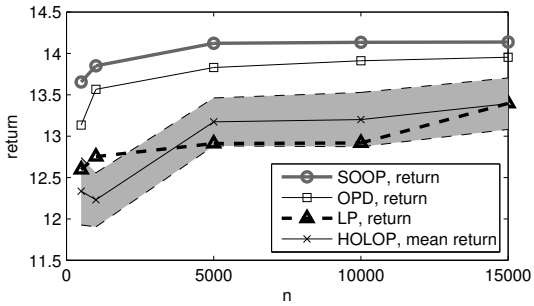

Figure 3.4: Performance for the inverted pendulum.

The relationships between SOOP, OPD, and HOLOP mirror those in the DC motor problem. However, LP now ranks as poorly as HOLOP. Figure 3.5 (on the next page) shows representative controlled trajectories with SOOP and LP. LP applies very coarse actions, while SOOP uses fine discretization to behave near-optimally.[2] The reason is found in the small values of L: LP prefers to search longer-horizon solutions rather than discretize finely. Unfortunately, even for this coarse discretization it does not manage a good swing-up. While the reasons are not entirely clear, one hypothesis is that unlike for the DC motor, in the swing-up problem the Lipschitz

[2]This is determined by comparing with near-optimal solutions found with dynamic programming, which is possible in this low-dimensional problem.

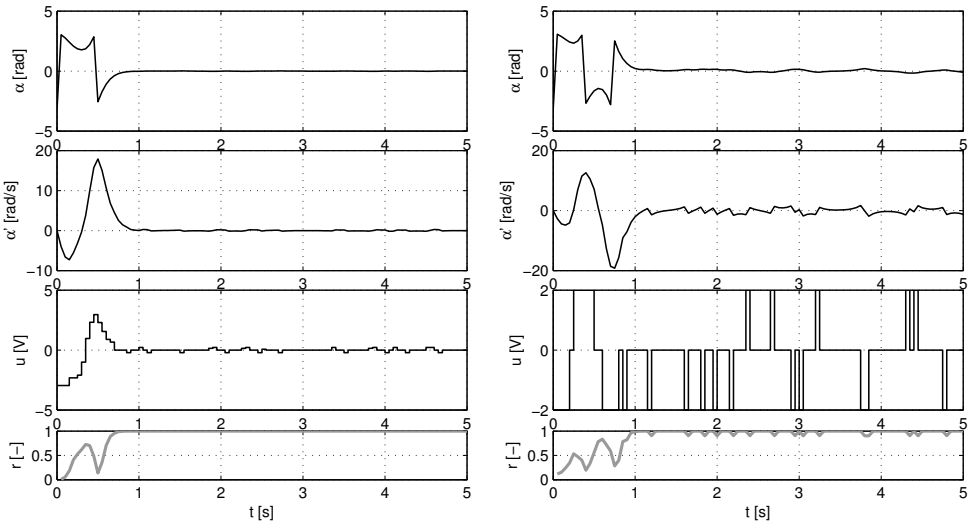

Figure 3.5: Swing-ups of the inverted pendulum with SOOP and LP, for $n = 5000$ and optimized parameters.

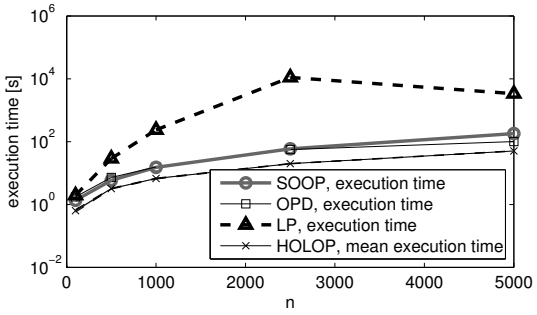

Figure 3.6: Execution time for the inverted pendulum, for optimized parameters.

constant varies, with the system behaving differently around equilibria than around the critical swing-up points; and that LP cannot deal with that.

Regarding α in SOOP, for tight budgets larger values are preferred, which means a longer horizon is sought at the expense of discretization; as more samples become available and a sufficient horizon is ensured, the balance shifts back towards discretization. This behavior is intuitive, since for too short horizons a good swing-up cannot be achieved, and fine actions become irrelevant.

Finally, we look at the computational cost of the algorithms, see Figure 3.6. Besides the fact that in our Matlab implementation the algorithms are not yet ready for real-time control, we notice that SOOP and OPD have similar costs, and HOLOP is somewhat faster. LP is slower, but this is at least partly due to our implementation,

which is optimized for many sequences with similar lengths; whereas around the goal state, LP typically expands a few very long sequences.

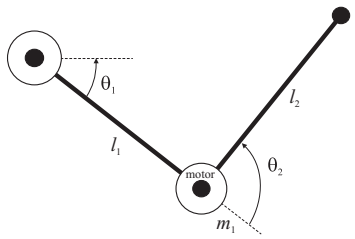

Figure 3.7: Robot arm.

Two-link robot arm. Finally, we consider a two-link robot arm actuated only in the middle joint, which has 4 states (angles θ_1, θ_2 of the joints plus their angular velocities) and 1 action u (motor torque). It can also be seen as a horizontally-oriented acrobot. The model equations are found in (Buşoniu et al., 2010a), Section 4.5.2. The link lengths are 0.15 and 0.25 m, both masses are 1 kg and concentrated at the ends of the links, and there is neither inertia nor friction. The task is stabilization to zero starting with both links at rest at angle $-\pi$, and the reward is quadratic with $Q_{\text{rew}} = \text{diag}(1,0,1,0)$ and no action penalty. Table 3.3 and Figure 3.8 show the results. OPD and discrete actions do well also in this problem, with SOOP trailing closely behind and doing better than LP and HOLOP.

Looking at Table 3.3 and Figure 3.8, OPD and discrete actions do well in this problem, with SOOP trailing close behind. Note that in problems where discrete actions work well, SOOP cannot be expected to outperform OPD, mainly because OPD searches the smaller space of discrete-action sequences, which still contains a good solution. Nevertheless, here SOOP still manages to find a good solution in the larger, continuous-action space, obtaining similar performance to OPD and still outperforming LP and HOLOP, which apparently search the larger space less efficiently.

Table 3.3: Parameters for the two-link robot arm.

$n =$	500	1000	5000
SOOP, $\alpha =$	0.5	0.6	0.5
OPD, $M =$	5	7	7
LP, $L =$	0.1	0.9	1.1
HOLOP, $K =$	5	5	5

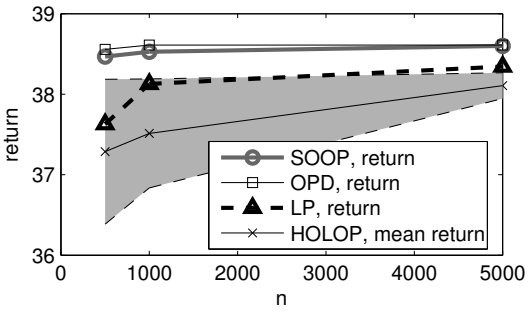

Figure 3.8: Performance for the two-link robot arm.

3.2 Optimistic best-first search for minimax control

We consider next the extension of optimistic ideas to sequential, adversarial decision-making problems, see e.g. Chapter 10 of (La Valle, 2006). Two adversarial agents take discrete actions in turn, one of them aiming to maximize the infinite-horizon cumulative value of the actions, and the other to minimize it. This framework can model important classes of problems, including e.g. turn-based games such as go or chess, as well as our main interest: optimal control under uncertainty, where the uncertainty is conservatively treated as the action of the opponent agent. It turns out that applying optimism in the adversarial setting naturally leads to the *best-first search* variant (Palay, 1982) of B*, a classical minimax algorithm proposed by (Berliner, 1979) in 1979. The name "best-first search" has been used for many other methods (including A* and even two minimax techniques: fixed-depth (Plaat et al., 1996) and adaptive-depth best-first search (Korf and Chickering, 1996)), so to avoid confusion we call the algorithm *optimistic minimax search* (OMS), always keeping in mind its relation to B*.

OMS explores a tree representation of the possible sequences of max and min agent actions, as do other minimax search algorithms such as alpha-beta pruning (Knuth and Moore, 1975) or those in (Plaat et al., 1996; Korf and Chickering, 1996). At each leaf node, OMS requires lower and upper bounds on the values of action sequences passing through that node, and it propagates these bounds upwards in the tree by maximization or minimization according to the type of node. The next leaf to expand is selected optimistically, by starting from the root and recursively moving to a child that maximizes the upper bound at max nodes, or minimizes the lower bound at min nodes. OMS can stop after any number of iterations, after which it returns the deepest expanded node. Thus it is an adaptive-depth, anytime algorithm.

By exploiting the optimistic framework, we are able to develop theoretical performance guarantees for OMS – which to our best knowledge were missing from the literature on B* search. Specifically, we provide conditions under which OMS is guaranteed to approach the minimax-optimal solution as the budget of node expansions increases. These conditions impose structure on the value function so that earlier decisions are more important than later ones, and require this structure to be reflected in the bounds. *A posteriori*, OMS is then near-optimal to the extent of the gap between the upper and lower bounds at the deepest expanded node. To obtain an *a priori* bound, we characterize the size of the subset of nodes that OMS expands by its asymptotic branching factor, and use this factor to provide a tight relationship between computation invested and near-optimality. In particular, when the gaps decrease exponentially with the depth, the convergence rate is directly characterized.

Throughout this section, we illustrate the theoretical framework in several classes of problems, including function optimization, games, and optimal control under uncertainty. In these examples, we study the value of the branching factor, illustrating that it is a meaningful measure of problem complexity. An empirical study illustrates

the analytical properties of OMS, and also includes the control problem of optimal treatment of HIV infection under uncertainty on drug effectiveness.

The importance of the effective branching factor in the analysis of minimax algorithms was understood as early as (Knuth and Moore, 1975; Pearl, 1982), where it was applied to alpha-beta pruning, see also (Korf, 1998). However, OMS is adaptive-depth and behaves quite differently from fixed-depth methods like alpha-beta. It is closer to the adaptive-depth best-first method of (Korf and Chickering, 1996), which does not have an analysis and in fact may converge to suboptimal solutions, as we will show in an example. Here we provide a general analysis of OMS near-optimality and branching factor, placing them in direct connection with (smoothness) properties of the value function – something that is largely missing in works analyzing classical minimax methods. From this perspective our branching factor is closer to other complexity measures in optimistic methods, such as the branching factor in (Hren and Munos, 2008a), the near-optimality dimension in Section 2.2 and (Munos, 2011), the near-optimality exponent in Section 4.1, etc. Different from these however, it works in minimax problems and filters nodes using a nontrivial, nonlocal property, which must hold for the entire path to the node. Finally, it must be noted that the B* search algorithm, of which OMS is a special case, aims only to find the optimal action at the root, whereas OMS as applied here further refines the value at the root even after the first action is clear, which is useful in optimization.

Throughout this section we preserve the full generality of the approach, by staying in the high-level setting of an adversarial decision-making process. We provide several examples of problems to which this setting can be applied. In particular, in Example 3.3, we specialize it to an optimal control problem with disturbance, which is closely related to the framework of Section 2.1.

3.2.1 Problem statement and examples

Consider an adversarial, sequential decision-making problem where a maximizer (max) and a minimizer (min) agent take actions in turn. The max and min actions are respectively denoted u and w, and belong to action spaces U and W. We assume that U and W contain finitely many elements, M and M respectively. A generic action is denoted $z \in Z := U \cup W$, and can be either a max or min action. Denote an infinite sequence of actions by $\boldsymbol{z}_\infty = (z_0, z_1, z_2, z_3, \dots) = (u_0, w_0, u_1, w_1, \dots) \in (U \times W)^\infty$, and a finite sequence of d actions by $\boldsymbol{z}_d = (z_0, z_1, \dots, z_{d-1})$, with $\boldsymbol{z}_0$ the empty sequence by convention. The truncation of $\boldsymbol{z}_\infty$ to d initial elements is denoted $\boldsymbol{z}_{\infty|d}$. Finally, define a sequence of reward functions $r_d : (U \times W)^{\lfloor d \rfloor} \times U^{\lceil d \rceil} \to \mathbb{R}$, $d \geq 1$ were notation $\lfloor d \rfloor$ means the result of the integer division of d by 2 and $\lceil d \rceil$ the remainder. Here, the convention is that a set to power 0 is omitted. The meaning of $r_d(\boldsymbol{z}_d)$ is that of immediate reward following a sequence of d decisions. Then, the overall infinite-horizon

value of sequence z_∞ is:

$$v(z_\infty) := \sum_{d=1}^{\infty} r_d(z_{\infty|d}) \tag{3.8}$$

The goal is to find the minimax-optimal value, defined as:

$$v^* := \lim_{k\to\infty} \left[\max_{u_0} \min_{w_0} \cdots \max_{u_{k-1}} \min_{w_{k-1}} \sum_{d=1}^{2k} r_d(z_d) \right] \tag{3.9}$$

when this limit exists.[3] This problem is similar e.g. to the one in Chapter 10 of La Valle (2006).

Define $\mathscr{Z}(z_d) = \left\{ z_\infty \,\middle|\, z_{\infty|d} = z_d \right\}$, the set of sequences starting with z_d. The following requirement sits at the core of our approach.

Assumption 3.3 *There exist functions l and b and a decreasing sequence $\{\delta(d)\}_{d\geq 0}$ of positive real numbers so that for any action sequence z_d:*

$$l(z_d) \leq v(z_\infty) \leq b(z_d), \forall z_\infty \in \mathscr{Z}(z_d) \tag{3.10}$$

$$b(z_d) - l(z_d) \leq \delta(d) \tag{3.11}$$

Thus, (3.10) says that l and b are lower and upper bounds on values of sequences starting with z_d. Our algorithm will require access to such bounds. Equation (3.11) intuitively restricts to problems where later decisions matter less than earlier ones. We will also call $\delta(d)$ the *gap* (between the two bounds).

Example 3.1 *Adversarial optimization.* Our first example is academic, and will later provide important insight into the behavior of the algorithm. Consider a function $g(x,y)$, $g : [0,1]\times[0,1] \to \mathbb{R}$. Both agents take binary decisions, $U = W = \{0,1\}$, with the following meaning. The max agent takes the domain $[0,1]\times[0,1]$ and splits it in half along dimension x, selecting the first half if $u = 0$ and the second if $u = 1$. The min agent then takes the resulting set and similarly splits it in half along dimension y. The max agent takes over and splits along x, and so on, see Figure 3.9. An infinite sequence z_∞ corresponds to a point and its value is $v(z_\infty) = g(x,y)$, assuming that g can be decomposed in the form (3.8).

Take, for example, function $g(x,y) = x+y$, which satisfies this property. For this function, upper and lower bounds can be easily found as follows. Each finite sequence z_d corresponds to a box (X,Y,Δ_x,Δ_y) where X,Y are the lower-left coordinates and Δ_x, Δ_y the lengths of the sides. Then, $l(z_d) = X+Y$, $b(z_d) = X+Y+\Delta_x+\Delta_y$. Further, $\Delta_x = 2^{-\lfloor d+1\rfloor}, \Delta_y = 2^{-\lfloor d\rfloor}$, so that $b(z_d) - l(z_d) \leq 2\cdot 2^{-d/2+1} \leq 4\cdot(1/\sqrt{2})^d =: \delta(d)$. The minimax-optimal value is $v^* = \max_x \min_y g(x,y) = 1$ and the corresponding minimax solution is the lower-right corner of the domain. □

[3] Decisions u, w, and index k are used when the max and min actions are regarded separately; otherwise, we use generic decision z and index d. Note that d corresponds to depth in a planning tree.

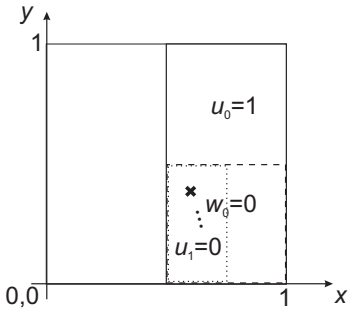

Figure 3.9: Adversarial optimization. The max agent takes action 1 choosing the continuous-outline box, the min agent 0 choosing the dashed box, and the max agent then applies 0 to choose the dotted box. Any infinite sequence of decisions is uniquely associated to a point.

Example 3.2 *Two-player games with discount.* Consider a turn-based game such as go, where the state of the board is represented by vector x. At turn $k \geq 0$, the player takes decision $u_k = z_{2k}$ and the opponent responds with $w_k = z_{2k+1}$. These decisions affect the board according to a transition function, $x_{d+1} = f(x_d, z_d)$, and the player attains rewards $\rho(x_d, z_d, x_{d+1})$, e.g., in go related to the territory and the number of pieces taken. The goal is to achieve discounted, minimax-optimal play:

$$\lim_{k\to\infty} \left[\max_{u_0} \min_{w_0} \cdots \max_{u_{k-1}} \min_{w_{k-1}} \sum_{d=0}^{2k} \gamma^d \rho(x_d, z_d, x_{d+1}) \right]$$

This is modeled in our framework by taking $\rho_d(\mathbf{z}_d) := \gamma^{d-1}\rho(x_{d-1}, z_{d-1}, x_d)$, while noting that the dependence of the rewards on the sequence of previous actions is collapsed into the state signal.

To ensure Assumption 3.3, we impose the following:

Assumption 3.4 *Rewards are bounded to the unit interval,* $\rho : X \times Z \times X \to [0,1]$.

This may require rescaling the original, nonunit rewards. Since all rewards after applying $\mathbf{z}_d$ are in $[0,1]$, we have $l(\mathbf{z}_d) = \sum_{j=0}^{d-1} \gamma^j \rho(x_j, z_j, x_{j+1})$ and $b(\mathbf{z}_d) = l(\mathbf{z}_d) + \frac{\gamma^d}{1-\gamma}$, with the convention that an empty sum is 0. Therefore, $\delta(d) = \frac{\gamma^d}{1-\gamma}$, and Assumption 3.3 is satisfied.

There may be terminal, game-over states, from which any transition ends up in the same state with reward 0. □

Example 3.3 *Discounted optimal control with disturbance.* Finally, take an optimal control problem for a system affected by disturbances. The dynamics at discrete-time step k are: $x_{k+1} = f(x_k, u_k, w_k)$, where u is now the applied action and w is the disturbance. A reward $r_{k+1} = \rho(x_k, u_k, w_k, x_{k+1})$ is obtained, and the goal is to achieve the best possible discounted return, conservatively taking into account the worst possible disturbances, as usually done in robust control:

$$\lim_{k\to\infty}\left[\max_{u_0}\min_{w_0}\cdots\max_{u_{k-1}}\min_{w_{k-1}}\sum_{j=0}^{k-1}\gamma^j\rho(x_j,u_j,w_j,x_{j+1})\right]$$

To place this in our framework, take:

$$\rho_d(\boldsymbol{z}_d) := \begin{cases} 0, & \text{if } d = 2k+1 \\ \gamma^k\rho(x_k,u_k,w_k,x_{k+1}) & \text{if } d = 2k+2 \end{cases}$$

We again impose reward boundedness to the unit interval:

Assumption 3.5 *The reward function satisfies* $\rho : X \times U \times W \times X \to [0,1]$.

Then, $l(\boldsymbol{z}_d) = \sum_{k=0}^{\lfloor d\rfloor-1}\gamma^k\rho(x_k,u_k,w_k,x_{k+1})$ and $b(\boldsymbol{z}_d) = l(\boldsymbol{z}_d) + \frac{\gamma^{\lfloor d\rfloor}}{1-\gamma}$, so that $\delta(d) = \frac{\gamma^{\lfloor d\rfloor}}{1-\gamma} \le \frac{\gamma}{1-\gamma}\sqrt{\gamma^d}$, and Assumption 3.3 is satisfied.

To make an explicit link with the optimal control problem of Section 2.1, consider the case where the disturbance w_k is a random variable with a probability distribution that may depend on the current state and action: $p(w|x,u)$. Then, we can eliminate the explicit dependence on the disturbance by defining the stochastic dynamics: $\tilde{f}(x,u,f(x,u,w)) := p(w|u,x)$ and the reward function $\tilde{\rho}(x,u,f(x,u,w)) := \rho(x,u,w,f(x,u,w))$. Functions $\tilde{f}$ and $\tilde{\rho}$ define a Markov decision process as in Section 2.1. Of course, the goal imposed there, of maximizing the *expected* discounted return (2.1), only makes sense because w is assumed to be a random variable. In the more general setting here, we do not make this assumption – instead, the disturbance w can have any behavior and we solve the problem for the worst-case behavior. □

It must be emphasized that in contrast to Example 3.1, Examples 3.2 and 3.3 comprise entire classes of practical problems.

More generally, lower and upper bounds can be derived if v is Lipschitz under a metric ℓ on the space of sequences:

$$\left|v(\boldsymbol{z}_\infty) - v(\boldsymbol{z}'_\infty)\right| \le \ell(\boldsymbol{z}_\infty, \boldsymbol{z}'_\infty)$$

and if for any set $\mathscr{Z}(\boldsymbol{z}_d)$, we have access to the value of a sample $\boldsymbol{z}_\infty \in \mathscr{Z}(\boldsymbol{z}_d)$. Define $\operatorname{diam}(\mathscr{Z}(\boldsymbol{z}_d)) := \sup_{\boldsymbol{z}'_\infty\in\mathscr{Z}(\boldsymbol{z}_d)}\ell(\boldsymbol{z}_\infty,\boldsymbol{z}'_\infty)$, then $\forall \boldsymbol{z}'_\infty \in \mathscr{Z}(\boldsymbol{z}_d)$:

$$\begin{aligned} v(\boldsymbol{z}'_\infty) &\ge v(\boldsymbol{z}_\infty) - \operatorname{diam}(\mathscr{Z}(\boldsymbol{z}_d)) =: l(\boldsymbol{z}_d) \\ v(\boldsymbol{z}'_\infty) &\le v(\boldsymbol{z}_\infty) + \operatorname{diam}(\mathscr{Z}(\boldsymbol{z}_d)) =: b(\boldsymbol{z}_d) \end{aligned}$$

Then, $\delta(d) = 2 \cdot \mathrm{diam}(\mathscr{Z}(\boldsymbol{z}_d))$ and the condition on $\delta(d)$ from Assumption 3.3 turns into a requirement on the diameters and thus on the smoothness of v.

In fact, in e.g. Example 3.1 the bounds follow from the Lipschitz property of $g(x,y) = x + y$ in the L_1 metric. In Example 3.2, a Lipschitz property of v holds for the metric $\ell(\boldsymbol{z}_\infty, \boldsymbol{z}'_\infty) = \frac{\gamma^{d(\boldsymbol{z}_\infty, \boldsymbol{z}'_\infty)}}{1-\gamma}$, where $d(\boldsymbol{z}_\infty, \boldsymbol{z}'_\infty)$ is the first index where the two sequences are different (a similar property holds for Example 3.3). However, the bounds in the examples were computed in a smarter way that did not require access to the exact value of a sample; indeed such a value will often be difficult to obtain since it is an infinite sum.

For more insight, the bounds l and b can be compared to those in DOO (Section 2.2), where they were also obtained from a Lipschitz condition; and to those in OPD (Section 2.3) where they had to be obtained directly due to the infinite sum, like in Examples 3.2 and 3.3, but they were again related to a Lipschitz condition.

In general, Assumption 3.3 allows any procedure for computing the bounds (3.10) as long as they satisfy together with v the smoothness property (3.11).

3.2.2 Optimistic minimax search

Optimistic minimax search (OMS) explores a tree representation of the possible action sequences, as illustrated in Figure 3.10. OMS starts with a root node corresponding to the empty sequence, and iteratively expands n nodes. Expanding a node adds new children nodes corresponding to all the M max actions (for max nodes) or N min actions (for min nodes). Each node at some depth d is reached via a unique path through the tree, and is thus uniquely associated to the sequence of actions $\boldsymbol{z}_d = (z_0, z_1, \ldots, z_{d-1})$ on this path. In what follows, we will work interchangeably with sequences and nodes, keeping this equivalence in mind.

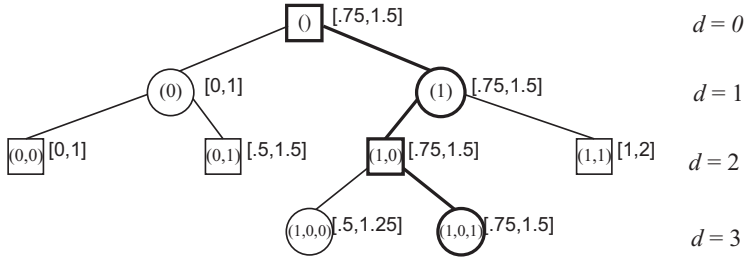

Figure 3.10: Illustration of a minimax tree developed by the algorithm when applied to Example 3.1. Squares are max nodes, and circles min nodes. Nodes are labeled by action sequences, shown inside the node, as well as by the interval $[L,B]$, shown outside. Four nodes have been expanded, and the thick path leads to the node that the algorithm would expand at iteration five.

Using the same notation as for the other trees so far, let $\mathscr{T}$ denote the current tree,

$\mathscr{L}(\mathscr{T})$ the leaf nodes of this tree, and $\mathscr{C}(\boldsymbol{z})$ the children of node $\boldsymbol{z}$. The algorithm computes lower and upper bounds $L(\boldsymbol{z})$ and $B(\boldsymbol{z})$ for each node. They are initialized at the leaves using l and b from Assumption 3.3, and propagated upwards in the tree:

$$
\begin{aligned}
L(\boldsymbol{z}) &= \begin{cases} l(\boldsymbol{z}), & \text{if } \boldsymbol{z} \in \mathscr{L}(\mathscr{T}) \\ \max_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} L(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a max node, } \boldsymbol{z} \notin \mathscr{L}(\mathscr{T}) \\ \min_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} L(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a min node, } \boldsymbol{z} \notin \mathscr{L}(\mathscr{T}) \end{cases} \\
B(\boldsymbol{z}) &= \begin{cases} b(\boldsymbol{z}), & \text{if } \boldsymbol{z} \in \mathscr{L}(\mathscr{T}) \\ \max_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} B(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a max node, } \boldsymbol{z} \notin \mathscr{L}(\mathscr{T}) \\ \min_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} B(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a min node, } \boldsymbol{z} \notin \mathscr{L}(\mathscr{T}) \end{cases}
\end{aligned}
\tag{3.12}
$$

To choose the next leaf to expand, the algorithm starts from the root and constructs a path by recursively selecting an optimistic child for the agent at the current node. That is, at max nodes a child with the largest upper bound is selected (optimistic for the max agent), while at min nodes the algorithm moves to a child with the smallest lower bound, which is optimistic for the min agent (it is pessimistic for the max agent).

OMS stops after n node expansions, and returns the deepest node expanded: its sequence $\hat{\boldsymbol{z}}$ and bounds. Algorithm 3.2 summarizes the entire procedure, where $(\cdot,\cdot)$ means the concatenation of the argument sequences and $d(\cdot)$ yields the depth (length) of the argument sequence. Ties in the maximizations and minimizations can be broken arbitrarily. Measuring computation by the number of node expansions is moti-

Algorithm 3.2 Optimistic minimax search (OMS)

Input: budget n

initialize: $\mathscr{T} \leftarrow \{\boldsymbol{z}_0\}$, the root
for iteration $t = 1$ to n **do**
  $\boldsymbol{z} \leftarrow \boldsymbol{z}_0$
  while $\boldsymbol{z} \notin \mathscr{L}(\mathscr{T})$ **do**
    $\boldsymbol{z} \leftarrow \begin{cases} \arg\max_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} B(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a max node} \\ \arg\min_{\boldsymbol{z}' \in \mathscr{C}(\boldsymbol{z})} L(\boldsymbol{z}'), & \text{if } \boldsymbol{z} \text{ is a min node} \end{cases}$
  end while
  $\boldsymbol{z}(t) \leftarrow \boldsymbol{z}$
  expand $\boldsymbol{z}(t)$, by adding to $\mathscr{T}$ its children:
    $(\boldsymbol{z}(t), u)$ $\forall u \in U$, if $\boldsymbol{z}(t)$ is a max node
    or $(\boldsymbol{z}(t), w)$ $\forall w \in W$, if $\boldsymbol{z}(t)$ is a min node
  compute bounds for all $\boldsymbol{z} \in \mathscr{T}$ with (3.12)
end for
$\hat{\boldsymbol{z}} \leftarrow \arg\max_{\boldsymbol{z}(t), t=1,\ldots,n} d(\boldsymbol{z}(t))$

Output: $\hat{\boldsymbol{z}}$, $l(\hat{\boldsymbol{z}})$, $b(\hat{\boldsymbol{z}})$

vated by the fact that these operations are often the most expensive, such as e.g. in the control problem of Example 3.3, where expansion requires the simulation of the dynamics. OMS is an anytime algorithm: n does not have to be specified in advance, and the algorithm can be stopped after any number of expansions.

Sometimes, OMS will be used with the intention of finding a decision to apply, rather than an approximation of the optimal value. In this case, the first action of the sequence $\hat{\mathbf{z}}$ is applied by the max agent, which then waits for the min agent's response and then reapplies OMS from the resulting situation (e.g., state). This can be seen as receding-horizon control (Maciejowski, 2002). Note that the min agent could itself apply OMS to find the actions, simply by starting with a min root node and then applying the algorithm as usual. Finally, the bounds L and B can be efficiently maintained by only updating at iteration t the path from the last expanded node $\mathbf{z}(t)$ to the root.

3.2.3 Analysis

Let us first establish a basic property of OMS.

Lemma 3.2 *At any iteration t, for any nodes $\mathbf{z}$, $\mathbf{z}' \in \mathscr{C}(\mathbf{z})$ on the optimistic path, we have $[L(\mathbf{z}), B(\mathbf{z})] \subseteq [L(\mathbf{z}'), B(\mathbf{z}')]$.*

Proof: If $\mathbf{z}$ is a max node, $B(\mathbf{z}) = B(\mathbf{z}')$ and $L(\mathbf{z}) \geq L(\mathbf{z}')$ since $L(\mathbf{z})$ is the maximum among the children's L-values. The situation is symmetrical at min nodes. ■

Define for any node $\mathbf{z}_d$ of finite depth d the minimax value $v(\mathbf{z}_d)$ among infinite sequences starting with $\mathbf{z}_d$. Formally:

$$v(\mathbf{z}_d) = \sum_{j=1}^{d} \rho_j(\mathbf{z}_j) + \begin{cases} \lim\limits_{k\to\infty} [\max\limits_{u_0} \min\limits_{w_0} \cdots \max\limits_{u_{k-1}} \min\limits_{w_{k-1}} \sum\limits_{j=1}^{2k} \rho_{d+j}((\mathbf{z}_d, \mathbf{z}_{\uparrow j}))] \\ \qquad \text{if } \mathbf{z}_d \text{ is a max node} \\ \lim\limits_{k\to\infty} [\min\limits_{w_0} \cdots \max\limits_{u_{k-1}} \min\limits_{w_{k-1}} \sum\limits_{j=1}^{2k-1} \rho_{d+j}((\mathbf{z}_d, \mathbf{z}_{\downarrow j}))] \\ \qquad \text{if } \mathbf{z}_d \text{ is a min node} \end{cases} \tag{3.13}$$

again assuming that the limits exist. Here, $\mathbf{z}_{\uparrow j} = (u_0, w_0, \ldots, u_{k-1}, w_{k-1})$, while $\mathbf{z}_{\downarrow j} = (w_0, \ldots, u_{k-1}, w_{k-1})$.

The second and final Lemma is essential to the analysis below, since it characterizes a restricted subset of nodes outside which the algorithm will never expand.

Lemma 3.3 *At depth d in the tree, OMS only expands nodes in the set:*

$$\mathscr{T}_d^* := \left\{ \mathbf{z}_d \middle| \, |v^* - v(\mathbf{z}_p)| \leq \delta(d), \forall \mathbf{z}_p \text{ on path from root to } \mathbf{z}_d \right\} \tag{3.14}$$

Proof: We will show by induction from leaves to the root that:

$$v(\mathbf{z}) \in [L(\mathbf{z}), B(\mathbf{z})], \quad \forall \mathbf{z} \in \mathscr{T}$$

At any leaf, the base case holds by definition: $v(\mathbf{z}) \in [l(\mathbf{z}), b(\mathbf{z}_d)] = [L(\mathbf{z}_d), B(\mathbf{z}_d)]$. For the general case, consider an inner node $\mathbf{z}$, and assume the property is true at all its children $\mathbf{z}'$. We have by definition (3.13):

$$v(\mathbf{z}_p) = \begin{cases} \max_{\mathbf{z}' \in \mathscr{C}(\mathbf{z}_p)} v(\mathbf{z}') & \text{if } \mathbf{z}_p \text{ is a max node} \\ \min_{\mathbf{z}' \in \mathscr{C}(\mathbf{z}_p)} v(\mathbf{z}') & \text{if } \mathbf{z}_p \text{ is a min node} \end{cases}$$

We first show that $L(\mathbf{z}) \leq v(\mathbf{z})$. If $\mathbf{z}$ is a max node, take child $\mathbf{z}'$ so that $L(\mathbf{z}) = L(\mathbf{z}')$, then $L(\mathbf{z}) = L(\mathbf{z}') \leq v(\mathbf{z}') \leq v(\mathbf{z})$. If $\mathbf{z}$ is a min node, take child $\mathbf{z}'$ so that $v(\mathbf{z}') = v(\mathbf{z})$, therefore: $L(\mathbf{z}) \leq L(\mathbf{z}') \leq v(\mathbf{z}') = v(\mathbf{z})$. Property $B(\mathbf{z}) \geq v(\mathbf{z})$ is shown in a symmetrical way: If $\mathbf{z}$ is a max node, take child $\mathbf{z}'$ so that $v(\mathbf{z}') = v(\mathbf{z})$, therefore $B(\mathbf{z}) \geq B(\mathbf{z}') \geq v(\mathbf{z}') = v(\mathbf{z})$. If $\mathbf{z}$ is a min node, take child $\mathbf{z}'$ so that $B(\mathbf{z}) = B(\mathbf{z}')$, then $B(\mathbf{z}) = B(\mathbf{z}') \geq v(\mathbf{z}') \geq v(\mathbf{z})$.

Consider now any leaf $\mathbf{z}_d$ selected for expansion on the current tree, and any ancestor node $\mathbf{z}_p$ at depth p on the path from the root to this leaf. Applying Lemma 3.2 iteratively from $\mathbf{z}_p$ down to the leaf $\mathbf{z}_d$, we have $[L(\mathbf{z}_p), B(\mathbf{z}_p)] \subseteq [L(\mathbf{z}_d), B(\mathbf{z}_d)] = [l(\mathbf{z}_d), b(\mathbf{z}_d)]$. Then:

$$v(\mathbf{z}_p) \in [l(\mathbf{z}_d), b(\mathbf{z}_d)] \tag{3.15}$$

At the root, $v(\mathbf{z}_0) = v^*$. For any $\mathbf{z}_p$ the values v^* and $v(\mathbf{z}_p)$ are in the interval $[l(\mathbf{z}_d), b(\mathbf{z}_d)]$, which has length at most $\delta(d)$, so the property in (3.14) holds. This concludes the proof. ■

At this point, we can already provide an *a posteriori* bound for the algorithm that can be directly evaluated after the algorithm has run.

Theorem 3.4 *Let d^* be the largest depth of any expanded node. Then, $|v^* - v(\hat{\mathbf{z}})| \leq \delta(d^*)$ and $v^* \in [L(\mathbf{z}_0), B(\mathbf{z}_0)]$.*

Proof: Follows immediately from (3.15) and Algorithm 3.2. ■

Note again that $B(\mathbf{z}_0) - L(\mathbf{z}_0) \leq b(\hat{\mathbf{z}}) - l(\hat{\mathbf{z}}) = \delta(d^*)$. To obtain a more refined bound, which works *a priori*, we characterize the size of the expanded subset $\mathscr{T}^* = \bigcup_{d \geq 0} \mathscr{T}_d^*$. Let $|\cdot|$ denote the cardinality of the argument set.

Definition 3.5 *Let ϑ be the smallest positive number so that $\exists C > 0, \left|\mathscr{T}_d^*\right| \leq C\vartheta^d$, $\forall d \geq 0$.*

The quantity ϑ is an asymptotic branching factor of $\mathscr{T}^*$, and it quantifies the complexity of the search problem. The smaller ϑ, the simpler the problem. The smallest possible value for ϑ is 1, when $\mathscr{T}_d^*$ contains a constant number of nodes at every d (e.g., just one minimax-optimal path), and the largest value is $\sqrt{MN}$, when

$\mathscr{T}_d^*$ contains all the nodes at d, namely $M^{\lfloor d+1 \rfloor} N^{\lfloor d \rfloor}$ nodes. Below we exemplify ϑ in several problems. Note that ϑ is similar with other complexity measures used in optimistic optimization and planning (Munos, 2014), such as the branching factor κ in OPD (see Section 2.3), or the near-optimality dimension β in OO (see Section 2.2). A crucial feature of ϑ and $\mathscr{T}^*$ is the nonlocal character of the inequality in (3.14), which must hold for any parent and not just the expanded node. This is important since it significantly reduces the size of the tree in some problems, as we will illustrate in Example 3.4.

Theorem 3.6 *(i) Let $d(n)$ be the smallest depth d so that $\sum_{j=0}^{d} C\vartheta^j \geq n$. Then, $|v^* - v(\hat{\mathbf{z}})| \leq \delta(d(n))$. (ii) Further, when $\exists c > 0, \lambda \in (0,1)$ so that $\delta(d) \leq c\lambda^d$, i.e. when the gap sequence decreases exponentially fast, then:*

$$\delta(d(n)) \leq \begin{cases} c(\frac{\vartheta-1}{Ck} \cdot n)^{-\frac{\log 1/\lambda}{\log \vartheta}} = O(n^{-\frac{\log 1/\lambda}{\log \vartheta}}) & \text{if } \vartheta > 1 \\ c\lambda^{n/C-1} = O(\lambda^{n/C}) & \text{if } \vartheta = 1 \end{cases} \tag{3.16}$$

Proof: For arbitrary d, OMS expands at most all the nodes up to d in $\mathscr{T}^*$ before expanding a node at $d+1$. Hence, since $\mathscr{T}^*$ contains at most $\sum_{j=0}^{d(n)-1} C\vartheta^j$ nodes until $d(n)-1$, and the algorithm expands more nodes than this (since by assumption $n > \sum_{j=0}^{d(n)-1} C\vartheta^j$). So, at least one node at $d(n)$ is expanded. From this $d^* \geq d(n)$ and since sequence $\delta(d)$ is decreasing, part (i) follows from Theorem 3.4.

To show part (ii), let $\vartheta > 1$. Then $n \leq \sum_{j=0}^{d(n)} C\vartheta^j = C\frac{\vartheta^{d(n)+1}-1}{\vartheta-1}$, and solving this for $d(n)$ we get $d(n) \geq \frac{\log n(\vartheta-1)/C\vartheta}{\log \vartheta}$, which when replaced in $\delta(d(n))$ gives the desired inequality. Similarly, if $\vartheta = 1$, we have $n \leq \sum_{j=0}^{d(n)} C = C(d(n)+1)$, from where $d(n) = \frac{n}{C} - 1$ which is substituted in $\delta(d(n))$. ■

Part (ii) of Theorem 3.6 is of practical importance, since in many problems the gap $\delta(d)$ will decrease exponentially with d, as e.g. in Examples 3.1–3.3, where λ is respectively $1/\sqrt{2}$, γ, and $\sqrt{\gamma}$. The big-O expressions in (3.16) highlight the qualitative, asymptotic behavior of the algorithm, whereas the detailed expressions preceding them make the constants explicit. Suboptimality decreases as a logarithmic power of the computation n when $\vartheta > 1$ (since the expanded tree grows exponentially), and exponentially fast with n when $\vartheta = 1$ (since only a constant number of paths must be explored). Since ϑ is generally unknown, the near-optimality of OMS cannot be determined in advance. However, Theorem 3.6 provides confidence that the algorithm automatically adapts to the complexity of the problem.

Example 3.4 *Adversarial optimization: branching factor.* We will find ϑ for Example 3.1 with $g(x,y) = x+y$. For any finite sequence $\mathbf{z}_d$ the minimax-optimal value $v(\mathbf{z}_d)$ is the value of g in the lower-right corner of the corresponding box. Consider some arbitrary odd depth d; boxes $\mathbf{z}_d$ at this depth are small tall rectangles like those shown in continuous outline at the bottom-right of Figure 3.11. Then the gap of these

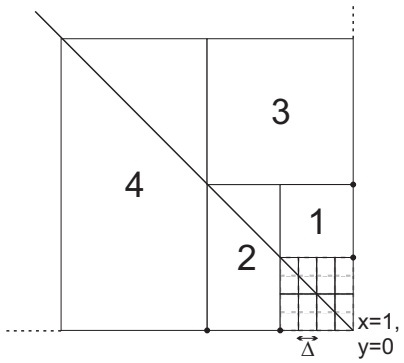

Figure 3.11: Counting the nodes in $\mathscr{T}_d^*$. Some minimax-optimal points are highlighted with black disks.

boxes is $\delta(d) = 3\Delta = 3 \cdot 2^{-(d+1)/2}$. Recall (3.14): if we can find a larger box containing $\boldsymbol{z}_d$ which is more than $\delta(d)$ away from the optimal value, then $\boldsymbol{z}_d$ will not be expanded. Now the suboptimality of any box is the distance between its lower-right corner and the main diagonal of the unit square. Since boxes 1 and 2 are 4Δ-away from optimum, no subbox $\boldsymbol{z}_d$ inside these larger boxes will be expanded. Boxes 3 and 4 are 8Δ-away so no subbox will be expanded there either, and continuing iteratively like this we can fill the entire domain *except* the lower-right corner, which contains 8 boxes. At depth $d+1$, boxes are square and have diameter 2Δ (shown in gray dotted line) so we can eliminate in a similar way all of them except the 16 in the corner. It follows that $\left|\mathscr{T}_d^*\right| \leq 16, \forall d$, and $\vartheta = 1$: the problem is easy. The regret bound, including constants, is $4(\frac{1}{\sqrt{2}})^{n/16-1}$.

It must be emphasized that checking the suboptimality of *parent* boxes is crucial: if we only checked boxes for their own suboptimality $|v^* - v(\boldsymbol{z}_d)|$, no box with the lower-right corner on the main diagonal could be eliminated, leading to a number of boxes growing with the depth and a large branching factor: the difficulty of the problem would be misrepresented.

Note that such applications of tree search methods to minimax optimization have been studied before, see e.g. (Ratschan, 2002), although that paper uses a different theoretical framework. □

Example 3.5 *Two-player games: branching factor.* We will illustrate the meaning of ϑ and the regret bounds for Example 3.2 with discount factor γ, in two representative special cases.

Consider first that all rewards are equal, say they are all 1. Then, $v^* = 1/(1-\gamma)$ and *any* sequence has this value. So no nodes can be eliminated with the condition in (3.14), $\mathscr{T}^*$ contains the whole tree, and OMS will in fact explore nodes uniformly, in the order of their depth. As shown above, in this case $\vartheta = \sqrt{MN}$. Therefore, this

uniform type of problem is an interesting worst case, where ϑ is the largest possible. From Theorem 3.16 with $\lambda = \gamma$, near-optimality is $O(n^{-\frac{\log 1/\gamma}{\log \sqrt{MN}}})$.

Next, an example with $\vartheta = 1$ is constructed, see Figure 3.12. At each max node along the path on the left of the tree, one child has reward 1 and all other children have reward 0, and the same is true of their complete subtrees. The situation is reversed at min nodes. Thus, the leftmost path is minimax-optimal, with value $v^* = \gamma^0 + \gamma^2 + \ldots = \frac{1}{1-\gamma^2}$.

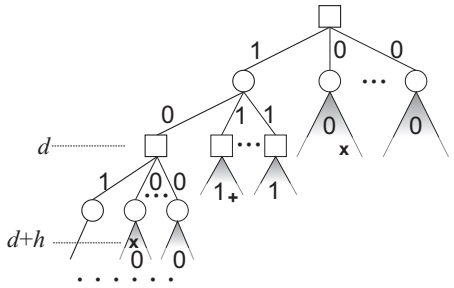

Figure 3.12: A game tree with $\vartheta = 1$. Rewards are shown along the transitions and inside subtrees where they remain constant. The thick path is minimax-optimal.

To study T^*, consider an arbitrary node $\boldsymbol{z}_{d+h}$ at depth $d+h$ that is *not* on the optimal path, but does belong to the subtree of some max node $\boldsymbol{z}_d$ which is on this path at an even depth d. Two examples of such nodes are shown by 'x' symbols in the figure. Then, the value of $\boldsymbol{z}_{d+h}$ is $v(\boldsymbol{z}_{d+h}) = \gamma^0 + \gamma^2 + \ldots + \gamma^{d-2} = \frac{1-\gamma^d}{1-\gamma^2}$. Since $\delta(d+h) = \frac{\gamma^{h+d}}{1-\gamma}$, see Example 3.2, node $\boldsymbol{z}_{d+h}$ can be excluded when:

$$|v^* - v(\boldsymbol{z}_{d+h})| > \frac{\gamma^{h+d}}{1-\gamma}, \text{i.e. } \frac{\gamma^d}{1-\gamma^2} > \frac{\gamma^{h+d}}{1-\gamma}$$

which boils down to $h > \frac{\log(1-\gamma^2)/(1-\gamma)}{\log 1/\gamma}$, a positive constant which we call H. Similarly, take now a node at depth $d+h$ on a non-optimal subtree of a *min* node at an odd d, as exemplified by a '+' in the figure. The value of such a node is $\gamma^0 + \gamma^2 + \ldots + \gamma^{d-1} + \gamma^d + \gamma^{d+1} + \gamma^{d+2} + \ldots = \gamma^0 + \gamma^2 + \ldots + \gamma^{d-1} + \gamma^{d+1} + \gamma^{d+3} + \ldots + \gamma^d + \gamma^{d+2} + \ldots = \frac{1+\gamma^d}{1-\gamma^2}$ where the intermediate step separated the odd and even powers of γ at depth d and larger. Solving the exclusion condition results in the same lower bound H on h as for max nodes.

Defining $N = \max(M,N)$, at some arbitrary depth d' the set $\mathscr{T}^*_{d'}$ contains at most the following amount of children coming from optimal nodes at various depths $d < d'$, which cannot be excluded via the conditions above:

$$1 + N + N^2 + \ldots + N^H =: C$$

so that $\mathscr{T}^*_{d'} \leq C$ and the branching factor $\vartheta = 1$. So this is an easy problem for OMS, and suboptimality is $O(\gamma^{n/C})$. □

3.2.4 Experimental results

First, in the optimization problem of Examples 3.1 and 3.4, we experimentally illustrate the practical effects of the theoretical properties studied above. Then, we show that OMS also works well in a challenging problem different from the games where it (and other minimax search algorithms) are usually applied: controlling infection with the human immunodeficiency virus (HIV), under uncertainty on the effectiveness of the drugs. This problem is in the class of Example 3.3.

Adversarial optimization. In addition to illustrating the properties of OMS, in this example we also compare it with two classical minimax search algorithms: alpha-beta pruning (Knuth and Moore, 1975) and adaptive-depth best-first minimax search (BFMS) (Korf and Chickering, 1996). Alpha-beta pruning is well-known so we do not review it here. BFMS is less widely used but it is an anytime algorithm similar to OMS. It develops the tree of Figure 3.10, but instead of maintaining an interval at each node it uses just one value, which is initialized using a heuristic function at the leaves and then propagated upwards as in (3.12). At each iteration, BFMS expands the leaf of the *principal variation*, a path along which the root inherited its value. After expanding a given amount of nodes it returns the principal variation. For both alpha-beta pruning and BFMS, we use l and b as a heuristic at respectively max and min leaves.

The computational requirements of alpha-beta pruning are not directly controlled, instead it searches until a given depth in the tree. It also expands a varying amount of children per node. Thus, to keep the comparison fair, we vary the depths d in the range $3, 4, \ldots, 15$ and measure for each d the required computation, in the form of the number of nodes n_t created on the tree. Figure 3.13, top shows the resulting values of n_t. Then, we allow OMS and BFMS to create as many nodes. This is different from the number n of *expanded* nodes that we used in the theory above, but only up to a constant factor so there are no changes in the asymptotic behavior. We only show OMS and BFMS results corresponding $d \leq 9$, since for the other budgets upper and lower bounds can become equal in double precision, and the results are not meaningful.

Figure 3.14, top shows the depths reached by OMS and BFMS. Clearly, expanding nodes in the order of their importance is better than up to a fixed depth like in alpha-beta: the depths reached by OMS and BFMS, as well as the corresponding confidence in the solution, are much better. Finally, Figures 3.13 and 3.14, bottom show the near-optimality of the returned solutions. This is measured here by the regret, defined for alpha-beta and BFMS as the distance between $v^* = 1$ and the value

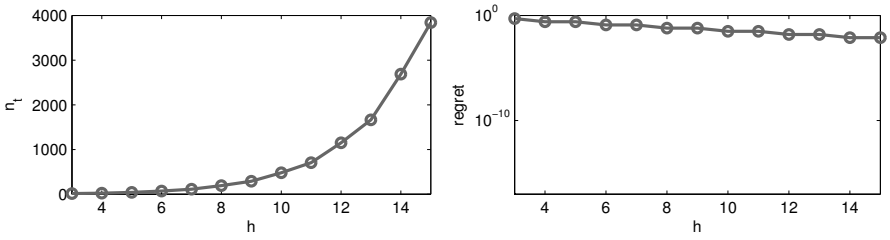

Figure 3.13: Results of alpha-beta pruning for adversarial optimization.

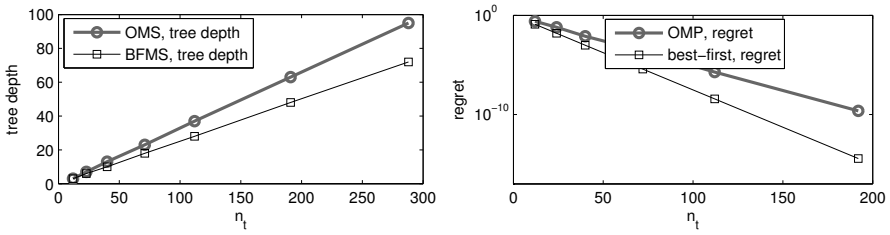

Figure 3.14: Results of OMS and BFMS. BFMS regrets because they are 0 for all $n_t > 12$ in this problem, so they are not shown.

returned by the algorithm, and for OMS as half the size of the interval $[L(\boldsymbol{z}_0), B(\boldsymbol{z}_0)]$ at the root (equal to the average distance of L and B to v^*).

As expected from the analysis and the branching factor $\vartheta = 1$ obtained in Example 3.4, OMS depths grow linearly with the computation budget and its regret shrinks exponentially with this depth. BFMS behaves surprisingly well: it often finds the optimal solution, and its depth also grows linearly with a larger slope than for OMS (in contrast, in alpha-beta n_t grows fast with d due to the exhaustive nature of the search, see again Figure 3.13, top). Unfortunately, unlike for OMS, a good behavior of BFMS cannot be guaranteed, and indeed BFMS is *inconsistent* over the class of problems satisfying Assumption 3.3, which means that for some problems it may entirely fail to converge to the optimal solution. The following counterexample illustrates this property.

Example 3.6 *BFMS is inconsistent.* Consider again adversarial optimization, Example 3.1, but with a different function $g(x,y)$, equal to 0.8 when $x \leq 0.5$, and $x+y$ otherwise. Take $l(\boldsymbol{z}) = b(\boldsymbol{z}) = 0.8$ for any box $\boldsymbol{z}$ in the left half of the domain, and use the bounds from Example 3.1 elsewhere. These l and b functions satisfy Assumption 3.3. BFMS develops the right (optimal) branch of the tree only until iteration 2, see Figure 3.15, and at any subsequent iteration it only expands nodes in the left branch of the root. Thus for any budget $n \geq 2$ it returns value 0.8, a constant away from the optimum $v^* = 1$. □

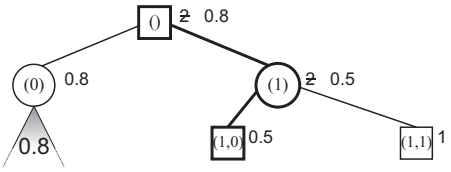

Figure 3.15: BFMS tree in the counterexample. Struck-through values are those changed after iteration 1.

HIV infection control. We consider the HIV infection dynamics described by Adams et al. (2004), with six state variables: T_1 and T_2 [cells/ml], the counts of healthy type 1 and type 2 target cells, T_1^{t} and T_2^{t} [cells/ml] the counts of infected type 1 and type 2 target cells, V the number of free virus copies [copies/ml], and E [cells/ml] the number of immune response cells. The system is controlled in discrete time with a sampling time of 5 days. In the strategy of structured treatment interruptions, two drugs are independently either fully administered (they are 'on'), or not at all (they are 'off'); thus there are two binary control variables u_1 and u_2, leading to $M = 4$. In other authors' work a one-to-one mapping was assumed between drug application and effectiveness. Here we use a variant where the effectiveness values ε_1 and ε_2 of the two drugs are, more realistically, uncertain by depending randomly on the inputs:

$$\varepsilon_1 = \begin{cases} 0 & \text{w.p. 1, if } u_1 = 0 \\ 0.77 & \text{w.p. 0.5, if } u_1 = 1 \\ 0.63 & \text{w.p. 0.5, if } u_1 = 1 \end{cases}$$

$$\varepsilon_2 = \begin{cases} 0 & \text{w.p. 1, if } u_2 = 0 \\ 0.33 & \text{w.p. 0.5, if } u_2 = 1 \\ 0.27 & \text{w.p. 0.5, if } u_2 = 1 \end{cases} \tag{3.17}$$

where "w.p." stands for "with probability". So depending on action u, there can be up to $N = 4$ possible outcomes. OMS is easy to modify for this varying-N case.

The system is initialized to the *unhealthy* equilibrium $x_{\mathrm{u}} = [163573, 5, 11945, 46, 63919, 24]^{\mathrm{T}}$, which represents a patient with dangerous infection levels and low immune response. STI is used to control the drugs such that the immune response of the patient is maximized and the number of virus copies is minimized, while penalizing the drugs administered due to their side-effects. We use the reward function of Adams et al. (2004) and normalize it to $[0, 1]$. An ideal solution would drive the state to the *healthy* equilibrium $x_{\mathrm{d}} = [967839, 621, 76, 6, 415, 353108]^{\top}$, which represents a patient whose immune system controls the infection without the need of drugs.

OMS is applied to plan a solution in receding horizon, while treating the uncertainties as an opponent that aims to minimize the return, like in Example 3.3. The budget is $n_t = 9000$, specified again as the number of created nodes, since the amount

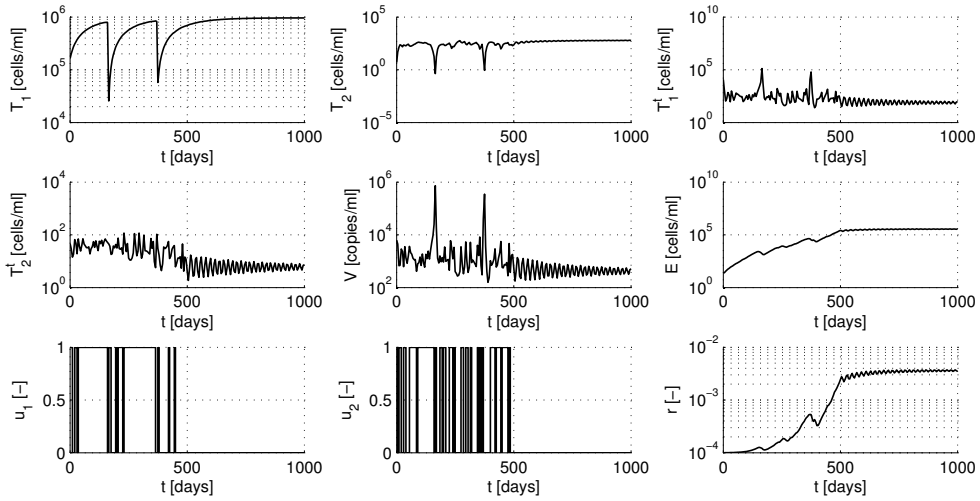

Figure 3.16: HIV system controlled online with OMS. The trajectories of the six system states are shown on the top and middle rows, while the two applied actions and the (normalized) rewards obtained are shown on the bottom row.

N of children of min nodes varies and using n would not result in a consistent computational load. The resulting trajectory is shown in Figure 3.16.[4] As hoped, the algorithm eventually stops administering drugs ($u_1 = u_2 = 0$), and the state reaches the healthy equilibrium x_{d} (although this particular solution has a 'lucky' disturbance realization for which the equilibrium is reached quickly).

[4]This experiment was run with a variant of OMS where the node to expand was directly selected to satisfy the interval inclusion property in Lemma 3.2, rather than by selecting bound extrema as in Algorithm 3.2.

3.3 Summary and conclusions

In the first part of this chapter, we described *SOO for Planning*, a planning algorithm for deterministic, continuous-action Markov decision processes. In extensive experiments, SOOP consistently ranked among the best algorithms, fully dominating competing methods when the problem requires both long horizons and fine discretization. In problems where discrete actions do well, discrete-action planning starts at an advantage; nevertheless, in our example that had this property, SOOP could still be applied with minimal loss of performance, unlike its continuous-actions competitors.

In the second part, we have showed analytically that, under appropriate conditions, *optimistic minimax search* (also known as the best-first search variant of B*) converges in a well-characterized way towards the optimal minimax value, and illustrated the analysis in an empirical evaluation. This is useful among others in control problems with disturbance, where OMS can be applied by treating the disturbance as an opponent, but also in other settings such as two-player games.

Chapter 4

Solving stochastic problems

4.1 Optimistic planning for Markov decision processes

In this section we move away from the deterministic problems considered so far. We describe and study an online, optimistic planning algorithm for problems that have a finite number M of actions, and in addition a finite number N of possible random next states for every transition. This includes complete, finite MDPs (Puterman, 1994) as well as infinite (e.g. continuous-state) MDPs that satisfy the condition on the next states. The algorithm is thus called optimistic planning for Markov decision processes (OPMDP).

At a given step of interaction with the system, OPMDP develops a tree starting from a node containing the current system state and then iteratively expanding well-chosen nodes. Rather than the open-loop action sequences of OPD, a solution here must be a closed-loop assignment of actions to stochastic state outcomes, represented as a subtree. To choose which node to expand, first an *optimistic* subtree is constructed. Then, among the leaves of this subtree, a node is selected that maximizes the contribution of the node to the uncertainty on the value. After n such iterations, the algorithm returns a tree policy by maximizing a lower bound.

Our analysis revolves around a *near-optimality guarantee* for OPMDP as a function of the number of expansions n. We show that OPMDP adapts to the complexity of the planning problem, by only expanding nodes with significant contributions to near-optimal policies. This notion is formalized so that the quantity of nodes with ε contribution to ε-optimal policies is of order $\varepsilon^{-\psi}$, with ψ a positive *near-optimality exponent*. Then, we show that the near-optimality is of order $n^{-1/\psi}$ for large n. When there are few near-optimal policies, as well as when the transition probabilities are nonuniform — both corresponding to having more structure in the MDP — ψ is small and the bound is better. To our knowledge, this is the first near-optimality bound available for closed-loop planning in stochastic MDPs.

While the bound does not directly depend on N and M, in practice they should not

be too large, e.g. the probability mass should be concentrated on a few discrete next states. Fortunately this is true in many problems of interest. For example, combining continuous-state deterministic dynamics with random variables that only take a few discrete values leads to a small N. Such variables could be failure modes, job arrivals into a resource management system (e.g., elevator scheduling, traffic signal control), user input, etc.

OPMDP extends OPD to stochastic problems, and in fact, in the deterministic case, OPMDP and the bounds we derive here reduce to OPD and its bounds, respectively. OPMDP can also be seen as an application of classical AO* heuristic search (Nilsson, 1980) to infinite-horizon discounted MDPs, similar to the AO* variant from (Hansen and Zilberstein, 1999). AO* builds a complete plan, which can only be done in finite MDPs with goal states and may require arbitrarily long computation. OPMDP finds general near-optimal solutions while the expansion budget is limited to n. It can actually be applied in an any-time fashion, without fixing n in advance, which gives it potential for online real-time control.

The exponent ψ is related to other measures of complexity used in the literature on bandits with many arms (Kleinberg et al., 2008; Bubeck et al., 2009; Wang et al., 2008). However, a direct application of those results would look at the closed-loop policies as the elementary entity (arm), and would thus not take into account essential problem structure: the fact that a node belongs to many policies, and expanding it improves knowledge about all these policies. A more refined notion of complexity is needed to capture this global structure, and ψ serves this purpose.

OPMDP requires the full MDP model, including probability distributions over next states, whereas the class of sample-based planning methods only need to generate next states according to this distribution. Kearns et al. (2002) first proposed a "sparse-sampling" method that builds a uniform planning tree by sampling a fixed number of states for each action, up to some horizon, without adapting to the structure of the MDP. An adaptive-horizon extension was given by Péret and Garcia (2004). An optimistic sample-based algorithm is "upper-confidence-trees" (Kocsis and Szepesvári, 2006), which travels an optimistic path along the planning tree by choosing actions with maximal upper confidence bounds on the returns, and sampling states independently. UCT often works well in practice (Wang and Gelly, 2007) but good performance cannot be guaranteed in general since it may exhibit pathological behavior (Coquelin and Munos, 2007). Walsh et al. (2010) avoid this problem with an optimistic extension of sparse sampling that comes with so-called probably-approximately-correct guarantees, of a different nature from the bounds we introduce here. Bubeck and Munos (2010) do provide bounds for optimistic planning in stochastic problems, but only for open-loop sequences of actions, which are generally suboptimal.

4.1.1 Problem statement and proposed optimistic planning algorithm

We are now in the setting of Section 2.1, with stochastic dynamics $x_{k+1} \sim f(x_k, u_k, \cdot)$ and rewards $r_{k+1} = \rho(x_k, u_k, x_{k+1})$. We still require bounded rewards and discrete actions, per Assumptions 2.1 and 2.4. Moreover, we focus on the case where transitions have a *finite* number of outcomes with known probabilities.

Assumption 4.1 *For any pair (x,u), the number of next states reachable with non-zero probability is at most integer $N > 0$.*

This class of problems disallows continuous next-state distributions, see Section 4.2 for an extension to a class of such distributions. Even discrete distributions are however very relevant, since they include many discrete-event systems (Cassandras and Lafortune, 1999) such as Markov jump systems (Costa et al., 2005), and have important applications in power systems (Billinton and Allan, 1996), fault detection (Mahmoud et al., 2001), and building automation (Meyer et al., 2013).

Some preparatory steps are necessary. Like OPD, OPMDP works at the current system state, conventionally denoted x_0. It explores iteratively an infinite tree that represents all possible stochastic evolutions of the system starting from x_0. Denote a state node by s, labeled by an actual state x. The planning tree $\mathscr{T}_\infty$, of which Figure 4.1 only shows a few top nodes, is defined recursively as follows. First, the root node s_0 is labeled by the current state x_0, and then each node s is expanded by adding, for any state x' for which $f(x,u,x') > 0$ for some u, a new child node s' labeled by x'. So each node has at most NM children, corresponding to all possible states reachable by applying all possible actions. Note that Figure 4.1 explicitly includes also the action nodes.

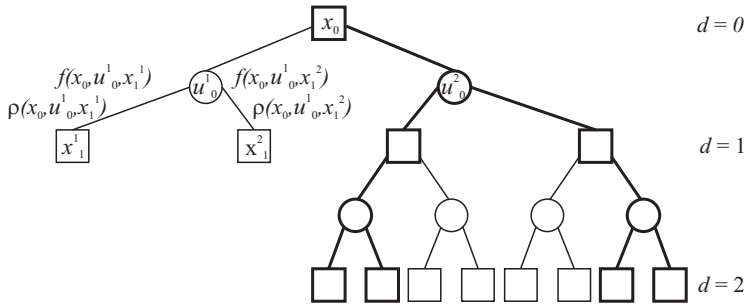

Figure 4.1: Illustration of OPMDP tree for $N = M = 2$. The squares are state nodes labeled by states x, and the actions u are explicitly included as circle nodes. Transition arcs to next states are labeled by probabilities f and rewards ρ. Superscripts index the possible actions and state outcomes, while subscripts are depths, which only increase with the state node levels. The thick subtree highlights a tree policy.

A closed-loop local solution concept called a *tree policy* is needed to react to the realization of the random transitions. At depth d, this tree policy is an assignment

of actions to all state outcomes under the previous action choices, thereby selecting only some nodes of $\mathscr{T}_\infty$:

$$\mathscr{T}_0 = \{s_0\}, \text{ and for any } d \geq 0\text{: } h_d : \mathscr{T}_d \to U,$$
$$\mathscr{T}_{d+1} = \{s' \in \mathscr{T}_\infty | s' \text{ is a child of } s \text{ along action } h_d(s)\}$$

where h_d assigns actions as desired. Then, the overall selected tree is $\mathscr{T}_{h_\infty} = \bigcup_{d\geq 0} \mathscr{T}_d$, and the policy itself is $h_\infty : \mathscr{T}_\infty \to U$, $h_\infty(s) = h_{d(s)}(s)$ with $d(s)$ the depth of s.

The objective is then to find, locally at x_0, a policy h_∞ maximizing the *expected* return:

$$V^{h_\infty}(x_0) = \mathrm{E}_{h_\infty}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1}\right\} \tag{4.1}$$

where the expectation is taken over all trajectories in $\mathscr{T}_{h_\infty}$. The optimal value is then $V^*(x_0) = \sup_{h_\infty} V^{h_\infty}(x_0)$. The optimal value is the same as in Section 2.1, but the class of solutions is larger than the state-feedback policies π considered there, since tree policies are allowed to apply different actions when they encounter the same state at a different node. While this extra generality is not necessary to represent an optimal solution, tree policies are convenient for planning so they will be used throughout this section.

OPMDP works of course with finite tree policies, exemplified in Figure 4.1. We denote such finite policies simply by h, and to keep the difference clear, we always write h_∞ explicitly when referring to an infinitely deep, complete policy tree. Notation $h_\infty \in h$ means that h_∞ "starts with" h (the subtree of h is fully contained in that of h_∞, so when h is regarded as a set of policies, h_∞ belongs to this set). Finite policies must correspond to well-defined subtrees $\mathscr{T}_h$ at the top of $\mathscr{T}_\infty$, so that any node is either fully expanded or not at all. The leaves of $\mathscr{T}_h$ are denoted by $\mathscr{L}_h$. We will treat policies h and their corresponding trees $\mathscr{T}_h$ interchangeably. Similarly to OPD, define three values:

$$\begin{aligned} l(h) &= \sum_{s\in\mathscr{L}_h} P(s)\, R(s) \\ b(h) &= \sum_{s\in\mathscr{L}_h} P(s)\, [R(s) + \frac{\gamma^{d(s)}}{1-\gamma}] \\ v(h) &= \sum_{s\in\mathscr{L}_h} P(s)\, [R(s) + \gamma^{d(s)} V^*(x(s))] \end{aligned} \tag{4.2}$$

where function $x(s)$ returns the state label associated to node s, $R(s)$ is the discounted return accumulated along the path from the root to s, and $P(s)$ is the probability of reaching leaf s:

$$R(s) = \sum_{d'=0}^{d-1} \gamma^{d'} \rho(x(s_{d'}), u_{d'}, x(s_{d'+1}))$$
$$P(s) = \prod_{d'=0}^{d-1} f(x(s_{d'}), u_{d'}, x(s_{d'+1}))$$

So, $l(h)$ is the expected partial return accumulated by h, and is a lower bound on the expected returns of all complete, infinite policies h_∞ starting with h; $b(h)$ is an upper bound on these expected returns; and $v(h)$ is the expected return when continuing optimally below h. It is important to note that $b(h) = l(h) + \sum_{s \in \mathscr{L}_h} P(s) \frac{\gamma^{d(s)}}{1-\gamma}$. We denote the sum in this expression by $\delta(h)$, called the diameter of h; and $c(s) = P(s) \frac{\gamma^{d(s)}}{1-\gamma}$, the contribution of node s to the diameter. Since the lower and upper bounds on the values of policies starting with h are separated by $\delta(h)$, this diameter is an uncertainty on $v(h)$, and $c(s)$ quantifies the contribution of s to this uncertainty.

OPMDP builds a subtree $\mathscr{T}$ of $\mathscr{T}_\infty$ by refining at each iteration an optimistic policy that maximizes b; and at the end, it returns a policy maximizing l. Thus the approach is similar to OPD, with the major difference that now a policy has multiple leaf nodes so a choice between them must be made. This is resolved by selecting for expansion a node with maximal contribution to the diameter. Algorithm 4.1 summarizes the approach. The algorithm stops after n node expansions. Note that expanding a node takes N times more simulations than in the deterministic case.

Algorithm 4.1 Optimistic planning for Markov decision processes (OPMDP)

Input: state x, budget n

1: initialize tree: $\mathscr{T}_0 \leftarrow \{\text{root}\}$, $i = 0$
2: **for** $t = 1, \ldots, n$ **do**
3: find optimistic policy $h_t^\dagger \in \arg\max_{h \in \mathscr{T}_t} b(h)$
4: expand node $s_t \in \arg\max_{s \in \mathscr{L}_{h_t^\dagger}} c(s)$, obtaining $\mathscr{T}_{t+1}$
5: **end for**

Output: $h^* \in \arg\max_{h \in \mathscr{T}_n} l(h)$

To develop more intuition, note that $\delta(h)$ is indeed a diameter, $\delta(h) = \sup_{h_\infty, h'_\infty \in h} \ell(h_\infty, h'_\infty)$, under the following metric on the space of policies:

$$\ell(h_\infty, h'_\infty) = \sum_{s \in \mathscr{L}(\mathscr{T}_{h_\infty} \cap \mathscr{T}_{h'_\infty})} c(s)$$

The sum is taken over the shallowest nodes where h_∞ and h'_∞ are different. The formula follows by noticing that for any s and u, $\sum_{s' \in \mathscr{C}(s,u)} c(s') = \gamma c(s)$ and so the contribution decreases monotonically with the depth, which implies that to maximize the distance between h_∞ and h'_∞, we should make these policies different at the shallowest possible places in the tree. Since the policies must be the same at inner nodes of $\mathscr{T}_h$, the leaves of this tree are the shallowest nodes where we can make h_∞ and h'_∞ different. So, OPMDP is similar to an application of DOO to search the space of tree policies, with the selection of the maximally-contributing node s_t corresponding to splitting the set h along the largest dimension. However, a highly nontrivial new effect is that splitting this node also refines many other policies, which the DOO analysis does not take into account.

The algorithm form above clearly brings out how the solution space is explored. However, it does not immediately make obvious how an implementation should proceed, e.g. how to search through all the tree policies compatible with the current tree. Fortunately efficient implementations exist, and in Algorithm 4.2 we provide one based on lower and upper bounds on the optimal values of the *states at the nodes*, rather than entire policies. These are computed for an existing tree as follows:

$$\begin{aligned} B(s) &= \begin{cases} \frac{1}{1-\gamma}, \text{ if } s \text{ is leaf} \\ \max_u \sum_{s' \in \mathscr{C}(s,u)} f(x(s),u,x(s'))[\rho(x(s),u,x(s')) + \gamma B(s')], \text{ else} \end{cases} \\ L(s) &= \begin{cases} 0, \text{ if } s \text{ is leaf} \\ \max_u \sum_{s' \in \mathscr{C}(s,u)} f(x(s),u,x(s'))[\rho(x(s),u,x(s')) + \gamma L(s')], \text{ else} \end{cases} \end{aligned} \tag{4.3}$$

where $\mathscr{C}(s,u)$ are the next-state children for action u at node s. The expansion rules of this algorithm are equivalent to those in original version, but they can be directly implemented, since they involve working with individual nodes rather than tree policies. For simplicity this version only returns the first action of the tree policy h^*, but the full h^* can be constructed as the optimistic subtree in lines 3-8, but by navigating towards maximal L-values instead of B-values.

Algorithm 4.2 OPMDP: Implementable version

1: initialize tree: $\mathscr{T}_0 \leftarrow \{s_0\}$
2: **for** $t = 0, \ldots, n-1$ **do**
3: starting from s_0, build optimistic subtree $\mathscr{T}_t^\dagger$:
4: **while** $\mathscr{L}(\mathscr{T}_t^\dagger) \not\subseteq \mathscr{L}(\mathscr{T}_t)$ **do**
5: retrieve a node $s \in \mathscr{L}(\mathscr{T}_t^\dagger) \setminus \mathscr{L}(\mathscr{T}_t)$
6: find optimistic action at s:
$u^\dagger = \arg\max_u \sum_{s' \in \mathscr{C}(s,u)} f(x(s),u,x(s'))[\rho(x(s),u,x(s')) + \gamma B(s')]$
7: add children $\mathscr{C}(s,u^\dagger)$ to $\mathscr{T}_t^\dagger$
8: **end while**
9: select leaf to expand: $s_t \leftarrow \arg\max_{s \in \mathscr{L}(\mathscr{T}_t^\dagger)} c(s)$
10: create $\mathscr{C}(s_t)$ and add them to $\mathscr{T}_t$, obtaining $\mathscr{T}_{t+1}$
11: update B and L-values upwards along the path from s_t to s_0
12: **end for**
13: **output** $u_0 = \arg\max_u \sum_{s' \in \mathscr{C}(s_0,u)} f(x(s_0),u,x(s'))[\rho(x(s_0),u,x(s')) + \gamma L(s')]$

4.1.2 Analysis

The complexity of a given MDP will be characterized in terms of a constant called *near-optimality exponent*. Then, the dependence of the sub-optimality of OPMDP on this exponent and the number of expansions n will be studied.

Consider any node s on the complete, infinite planning tree $\mathscr{T}_\infty$. Define $n(s)$ to be the largest number, for any policy h_∞ whose subtree contains s, of *leaves* of the subtree $\mathscr{T}_{h_\infty s} \in \mathscr{T}_{h_\infty}$ containing only nodes with larger contributions than s:

$$n(s) = \sup_{\mathscr{T}_{h_\infty} \ni s} |\mathscr{L}(\mathscr{T}_{h_\infty s})|, \ \mathscr{T}_{h_\infty s} = \left\{ s' \in \mathscr{T}_{h_\infty} \,\middle|\, c(s') \geq c(s) \right\}$$

$\mathscr{T}_{h_\infty s}$ is indeed a proper subtree since $c(s)$ decreases monotonically along paths in $\mathscr{T}_{h_\infty}$. A policy subtree is schematically represented in Figure 4.2, together with an example of subtree $\mathscr{T}_{h_\infty s}$.

Define using $n(s)$ also the quantity $\alpha(s) = n(s)c(s)$. An essential property of α is that it relates s to the diameter of some policies that have it as a leaf. Specifically, the diameter of any policy among whose leaves $c(s)$ is largest, is upper-bounded by $\frac{N}{\gamma}\alpha(s)$.

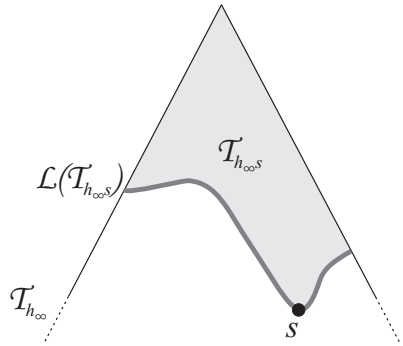

Figure 4.2: An infinite policy tree $\mathscr{T}_{h_\infty}$, and a subtree $\mathscr{T}_{h_\infty s}$ for some s.

Define for each ε the set of nodes:

$$S_\varepsilon = \left\{ s \in \mathscr{T}_\infty \middle| \text{(i) } \alpha(s) \geq \varepsilon \text{ and (ii) } \exists h_\infty \ni s, V^*(x_0) - V^{h_\infty}(x_0) \leq \frac{N}{\gamma}\alpha(s) \right\} \quad (4.4)$$

Condition (i) requires the node to have a sizable contribution in terms of α, and (ii) that the node belongs to a near-optimal policy.

Definition 4.1 *The* near-optimality exponent *is the smallest constant* $\psi \geq 0$ *so that* $\exists a > 0, b \geq 0$ *for which:*

$$|S_\varepsilon| = \tilde{O}(\varepsilon^{-\psi}), \ \textit{i.e.} \ |S_\varepsilon| \leq a \left(\log \frac{1}{\varepsilon} \right)^b \varepsilon^{-\psi} \quad (4.5)$$

Note that if $\psi = 0$, then $b > 0$, because $|S_\varepsilon|$ cannot remain constant as ε shrinks. A policy h is called ε-optimal if $V^*(x_0) - v(h) \leq \varepsilon$. Let $\varepsilon = V^*(x_0) - l(h^*)$ where h^* is the policy returned by OPMDP, which immediately means that h^* is ε-optimal, since $v(h^*) \geq l(h^*)$. The following main result holds.

Theorem 4.2 *The near-optimality of the tree policy chosen by OPMDP after n node expansions satisfies for large n:*

$$\varepsilon_n = \begin{cases} \tilde{O}(n^{-\frac{1}{\psi}}) & \text{if } \psi > 0 \\ O(\exp[-(\frac{n}{a})^{\frac{1}{b}}]) & \text{if } \psi = 0 \end{cases}$$

It is important to note that OPMDP does not require knowledge of the value of ψ, and yet, as the result shows, it automatically adapts to this value.

The measure ψ is connected to other complexity measures from optimistic optimization (bandits) and planning, such as the zooming dimension (Kleinberg et al., 2008), the near-optimality dimension (Bubeck et al., 2009), or the branching factor of near-optimal action sequences for OPD (Hren and Munos, 2008a) and open-loop optimistic planning (OLOP) (Bubeck and Munos, 2010). Characterizing the complexity of the planning problem by using these measures would essentially look at the individual tree policies as the elementary entity, and the analysis could only interpret the algorithm as a hierarchical search in the space of such policies. OPMDP has this hierarchical component, which is realized by refining the optimistic policy $h_t^\dagger$. However, OPMDP does more than just this: since the chosen node s_t belongs to all the policies that reach it with a positive probability, expanding it refines all these classes, not just the optimistic one. This subtler, global effect must be captured in the definition of problem complexity, and this is achieved by introducing the quantity $\alpha(s)$ to describe the *global impact* of a node on the policies it belongs to, and by defining S_ε and ψ in terms of individual nodes and not directly policies.

Therefore, the analysis intuitively says that OPMDP finds a near-optimal policy by only expanding those nodes that have a large impact on near-optimal policies. As the number of such nodes decreases, the sets S_ε grow more slowly, which is characterized by a smaller ψ, and the problem becomes easier. In particular, $\psi = 0$ means S_ε grows logarithmically instead of polynomially. The definition (4.4) of S_ε highlights two ways in which a problem can be easier: the transition probabilities are less uniform, leading to fewer nodes with large α; or the rewards are concentrated on a few actions, leading to fewer near-optimal policies. To formalize these intuitions, the next section will provide values of ψ for several representative types of MDPs, exhibiting varying degrees of structure in the rewards and probabilities.

Regarding the time complexity of OPMDP, at the expense of some extra memory, each iteration can be brought down to $O(d(s_t))$, the depth of the expanded node. Depths generally depend on ψ, but to obtain a range notice they are between $O(\log n)$ when the tree is developed uniformly and $O(n)$ when a single path is developed (see also next section). So the overall complexity is between $O(n \log n)$ and $O(n^2)$. Furthermore, while here we focus on the near-optimality perspective, our result can also be interpreted the other way around: to achieve ε-optimality, a budget n on the order of $\varepsilon^{-\psi}$ should be spent.

To prove Theorem 4.2, it must first be shown that the suboptimality of the algorithm is upper-bounded by the smallest α among expanded nodes, denoted α^*, which will be done in Lemma 4.4. Then, we will show in Lemma 4.5 that the algorithm always does useful work to decrease α, by only expanding nodes in S_{α^*}. Expressing the size of this set using the near-optimality exponent ψ will complete the proof. Before all this, a preliminary result is needed.

Lemma 4.3 *The l-values of the near-optimal policies on the tree increase over iterations: $l(h^*_{t+1}) \geq l(h^*_t)$, where $h^*_t \in \arg\max_{h \in \mathscr{T}_t} l(h)$.*

Proof: Consider first one policy h, split by expanding some leaf node $s \in \mathscr{L}(\mathscr{T}_h)$. One child policy h' is obtained for each action u, and we have $\mathscr{L}(\mathscr{T}_{h'}) = (\mathscr{L}(\mathscr{T}_h) \setminus \{s\}) \cup \mathscr{C}(s,u)$. By easy calculations, since the rewards are positive, the terms that nodes $\mathscr{C}(s,u)$ contribute to $l(h')$ add up to more than the term of s in $l(h)$, and the other terms remain constant. Thus $l(h') \geq l(h)$. Then, among the policies $h_t \in \mathscr{T}_t$, some are split in $\mathscr{T}_{t+1}$ and some remain unchanged. For the children of split classes l-values are larger than their parents'; while l-values of unchanged classes remain constant. Thus, the maximal l-value increases across iterations. ■

Note it can similarly be shown that $b(h^\dagger_{t+1}) \leq b(h^\dagger_t)$.

Lemma 4.4 *Define $\alpha_t = \alpha(s_t)$, the α value of the node expanded at iteration t; and $\alpha^* = \min_{t=0,\ldots,n-1} \alpha_t$. The near-optimality after n expansions satisfies $\varepsilon_n = V^*(x_0) - l(h^*_n) \leq \frac{N}{\gamma}\alpha^*$.*

Proof: We will first bound, individually at each iteration t, the suboptimality of $l(h^*_t)$, by showing:

$$V^*(x_0) - l(h^*_t) \leq \delta(h^\dagger_t) \leq \frac{N}{\gamma}\alpha_t \tag{4.6}$$

To this end, observe that:

$$l(h^\dagger_t) \leq l(h^*_t) \leq V^*(x_0) \leq b(h^\dagger_t) \tag{4.7}$$

The inequality $l(h^*_t) \leq V^*(x_0)$ is true by definition: $l(h^*_t)$ is a lower bound on the value of some policy, itself smaller than $V^*(x_0)$. For the leftmost inequality, h^*_t maximizes the lower bound across all policies compatible with the current tree, so its lower bound is at least as large as that of the optimistic policy $h^\dagger_t$. Similarly, for the rightmost inequality, since $h^\dagger_t$ maximizes the upper bound, its upper bound is immediately larger than the true optimal value. Using this string of inequalities, we get:

$$V^*(x_0) - l(h^*_t) \leq b(h^\dagger_t) - l(h^\dagger_t) = \delta(h^\dagger_t) = \sum_{s \in \mathscr{L}(\mathscr{T}_{h^\dagger_t})} c(s) \tag{4.8}$$

We now investigate the relationship between this diameter and α_t. Consider the subtree $\mathscr{T}_{h^\dagger_t}$ of policy $h^\dagger_t$, exemplified in Figure 4.3 using thicker lines (this subtree

has a branching factor of N). We are thus interested in finding an upper bound for $\sum_{s\in\mathscr{L}(\mathscr{T}_{h_t^\dagger})} c(s)$ as a function of α_t. Consider the tree $\mathscr{T}_{h_\infty s_t}$, as introduced earlier in the definition of $n(s)$, which is included in $\mathscr{T}_{h_t^\dagger}$ and is the same for any $h_\infty \in h_t^\dagger$. To see this, recall that s_t maximizes c among the leaves of $\mathscr{T}_{h_t^\dagger}$. Since additionally c strictly decreases along paths, any node with a contribution larger than $c(s_t)$ must be above these leaves, and this holds for any $h \in h_t^\dagger$.

Denote in this context $\mathscr{T}_{h_\infty s_t}$ more simply by $\mathscr{T}'$, shown in gray in the figure, and its leaves by $\mathscr{L}'$, shown as a gray outline. Denote the *children* of $\mathscr{L}'$ by $\mathscr{L}''$, shown as a dashed line.

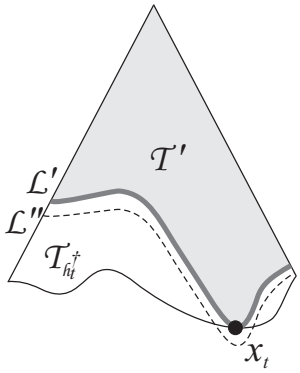

Figure 4.3: Tree of the optimistic policy and various subtrees.

Recall that for any h and $s \in \mathscr{T}_h$, $\sum_{s'\in\mathscr{C}(s,h(s))} c(s') = \gamma c(s)$. This also means the sum of contributions for the leaves of any subtree of $\mathscr{T}_h$ having some s as its root is smaller than $c(s)$. Using these properties, we have:

$$\sum_{s\in\mathscr{L}(\mathscr{T}_{h_t^\dagger})} c(s) \leq \sum_{s'\in\mathscr{L}'} c(s') = \frac{1}{\gamma}\sum_{s''\in\mathscr{L}''} c(s'') \leq \frac{1}{\gamma}\sum_{s''\in\mathscr{L}''} c(s_t)$$
$$\leq \frac{1}{\gamma}N\left|\mathscr{L}'\right| c(s_t) \leq \frac{1}{\gamma}Nn(s_t)c(s_t) = \frac{N}{\gamma}\alpha_t$$

where we additionally exploited the facts that $c(s'') \leq c(s_t)$ (otherwise s'' would have been in $\mathscr{T}'$), that each node in $\mathscr{L}'$ has N children in $\mathscr{L}''$, and that by the definition of $n(s)$ $|\mathscr{L}'| \leq n(s_t)$. From this and also (4.8), the desired intermediate result (4.6) is obtained.

Using now (4.6), as well as the fact that $l(h_n^*) \geq l(h_t^*)\ \forall t < n$ due to Lemma 4.3, the desired conclusion is immediate. ■

Lemma 4.5 *All nodes expanded by the algorithm belong to S_{α^*}, so that $n \leq |S_{\alpha^*}|$.*

Proof: We show first that $s_t \in S_{\alpha_t}$ at any iteration t. Condition (i) in the definition (4.4) of S_{α_t} is immediately true. For condition (ii), an $\frac{N}{\gamma}\alpha_t$-optimal policy h_∞ whose

tree $\mathscr{T}_{h_\infty}$ contains s_t is needed. Choose any h_∞ starting with $h_t^\dagger$, then $s_t \in \mathscr{T}_{h_\infty}$ and:

$$V^*(x_0) - V^{h_\infty}(x_0) \leq b(h_t^\dagger) - l(h_t^\dagger) = \delta(h_t^\dagger) \leq \frac{N}{\gamma}\alpha_t$$

where we used some of the inequalities derived in the proof of Lemma 4.4. Thus $s_t \in S_{\alpha_t}$. Furthermore, $\alpha^* \leq \alpha_t$ implies $S_{\alpha_t} \subseteq S_{\alpha^*}$, and we are done. ■

With these lemmas established, we can now prove Theorem 4.2.

Proof:[Theorem 4.2] Exploiting Lemma 4.5 in combination with (4.5):

- if $\psi > 0$, $n = \tilde{O}(\alpha^{*-\psi})$, thus for large n, $\alpha^* = \tilde{O}(n^{-\frac{1}{\psi}})$;
- if $\psi = 0$, $n \leq a\left(\log \frac{1}{\alpha^*}\right)^b$, thus $\alpha^* \leq \exp[-(\frac{n}{a})^{\frac{1}{b}}]$.

By Lemma 4.4, $\varepsilon_n \leq \frac{N}{\gamma}\alpha^*$ which immediately leads to the desired results. ■

4.1.3 Some interesting values of ψ

To add meaning to the near-optimality exponent ψ, in this section we provide its value for several interesting special cases. We obtain smaller values when the MDP has "more structure", namely when there are non-uniform probabilities or rewards. The earlier OPD bounds (Hren and Munos, 2008a) are recovered in the deterministic case, showing that the OPMDP guarantees encompass those of OPD as a special case.

Uniform rewards and probabilities. In this case, the rewards of all transitions are equal, and for any action, the probability of reaching one of the N next states is $\frac{1}{N}$.

Proposition 4.6 *In the uniform case,* $\psi_{\text{unif}} = \frac{\log NM}{\log 1/\gamma}$ *and* $\varepsilon_n = O(n^{-\frac{\log 1/\gamma}{\log NM}})$.

Proof: We study the size of S_ε. Due to the equal rewards all the policies are optimal, and condition (ii) in (4.4) does not eliminate any nodes. The contribution of a node is $c(s) = P(s)\frac{\gamma^{d(s)}}{1-\gamma} = (\frac{\gamma}{N})^{d(s)}\frac{1}{1-\gamma}$ since the probability of reaching a node at depth $d(s)$ is $(\frac{1}{N})^{d(s)}$. This also means that, for any policy h, the tree $\mathscr{T}_{hs}$ consists of all the nodes s' up to the depth of s. The number of leaves of this tree is $N^{d(s)}$ (recall that a policy tree has only branching factor N), and since this number does not depend on the policy, $n(s)$ is also $N^{d(s)}$. Therefore, $\alpha(s) = n(s)c(s) = \frac{\gamma^{d(s)}}{1-\gamma}$ and condition (i) eliminates nodes with depths larger than $D = \frac{\log \varepsilon(1-\gamma)}{\log \gamma}$. The remaining nodes in the whole tree, with branching factor NM, form S_ε, which is of size:

$$|S_\varepsilon| = O((NM)^D) = O((NM)^{\frac{\log \varepsilon(1-\gamma)}{\log \gamma}}) = O(\varepsilon^{-\frac{\log NM}{\log 1/\gamma}})$$

yielding for ψ the value: $\psi_{\text{unif}} = \frac{\log NM}{\log 1/\gamma}$. So, for large n the near-optimality $\varepsilon_n = \tilde{O}(n^{-\frac{\log 1/\gamma}{\log NM}})$. In fact, as can be easily checked by examining the proof of Theorem 4.2, the logarithmic component disappears in this case and $\varepsilon_n = O(n^{-\frac{\log 1/\gamma}{\log NM}})$. ■

One interpretation of ψ is that the argument of the logarithm at the numerator is an equivalent branching factor of the tree that OPMDP must explore. A branching factor of NM means that OPMDP will have to explore the whole planning tree in a uniform fashion, expanding nodes in the order of their depth. So the uniform MDP is an interesting worst case, where ψ is the largest possible. Notice also that in this case the bounds do not have a logarithmic component, so O is used instead of $\tilde{O}$.

In fact, the near-optimality bound of OPMDP in uniform MDPs is the same as that of a *uniform planning* algorithm, which always expands the nodes in the order of their depth. However, the uniform algorithm can guarantee only this bound for *any* problem, whereas in non-uniform problems, OPMDP adapts to the value of ψ to obtain better guarantees.

This near-optimality is also the smallest achievable in a worst-case sense, which means that for any planning algorithm and value of n, one can construct a problem for which $\varepsilon_n = \Omega(n^{-\frac{\log 1/\gamma}{\log NM}})$. To see this, choose the largest D so that $n \geq \frac{(NM)^D - 1}{NM-1}$, assign uniform probabilities everywhere, rewards of 1 for some arbitrary policy h^* but only starting from level $D+1$ onward, and rewards of 0 everywhere else. Then, both OPMDP and uniform planning have uniformly expanded all nodes up to $D-1$ but none at $D+1$, so they have no information and must make an arbitrary action choice, which may not be optimal, leading to a suboptimality of $\frac{\gamma^{D+1}}{1-\gamma} = \Omega(n^{-\frac{\log 1/\gamma}{\log NM}})$. An algorithm that does not expand uniformly may miss the optimal policy for an even larger number of expansions n, so their suboptimality is at least as large. This fact also shows that OPMDP *behaves correctly* in the uniform case: as long as only uniform rewards and probabilities are observed, the tree must be expanded uniformly, and this behavior is reflected in the bound.

Finally, note that in the deterministic case, when $N = 1$, the bound of OPD for the uniform-reward case is recovered: $\varepsilon_n = O(n^{-\frac{\log 1/\gamma}{\log M}})$.

Structured rewards. In this case, probabilities are uniform but a single policy has maximal rewards (equal to 1) for all transitions, and all other transitions have a reward of 0, see Figure 4.4. So, there is a "maximal" amount of structure in the reward function.

Proposition 4.7 *In the case of structured rewards, if $N > 1$, $\psi_{\text{rew}} = \frac{\log N}{\log 1/\gamma}(1 + \frac{\log M}{\log N/\gamma})$, whereas if $N = 1$, $\psi_{\text{rew}} = 0$ and $\varepsilon_n = O(\exp(-\frac{n}{a}))$ for some constant a.*

Proof: Since $\alpha(s)$ depends only on the probabilities, condition (i) leads to the same $D = \frac{\log \varepsilon(1-\gamma)}{\log \gamma}$ as in the uniform case. However, now condition (ii) becomes

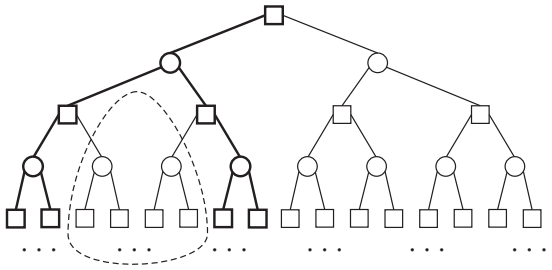

Figure 4.4: Illustration of a planning tree for structured rewards, up to depth 2, for $N = M = 2$. Thick lines: subtree of optimal policy, where each transition is associated with a reward of 1. Thin lines: the rest of the tree, associated with 0 rewards.

important, so to obtain the size of S_ε, we must only count *near-optimal* nodes up to depth D.

Consider the set of nodes in $\mathscr{T}_\infty$ which do not belong to the optimal policy, but lie below nodes that are at depth d' on this policy. An example is enclosed by a dashed line in Figure 4.4, where $d' = 1$. All these nodes are sub-optimal to the extent of the loss incurred by not choosing the optimal action at their parent, namely: $(\frac{\gamma}{N})^{d'}\frac{1}{1-\gamma}$. Note these nodes *do* belong to a policy that is near-optimal to this extent, one which makes the optimal choices everywhere except at their parent. Looking now from the perspective of a given depth d, for any $m \le d$ there are $N^d M^m$ nodes at this depth that are $(\frac{\gamma}{N})^{d-m}\frac{1}{1-\gamma}$-optimal. Condition (ii), written $(\frac{\gamma}{N})^{d-m}\frac{1}{1-\gamma} \le \frac{N}{\gamma}\frac{\gamma^d}{1-\gamma}$, leads to $m \le d\frac{\log N}{\log N/\gamma} + 1$. Then:

$$|S_\varepsilon| \le \sum_{d=0}^{D} N^d M^{d\frac{\log N}{\log N/\gamma}+1} \le M\sum_{d=0}^{D}\left(NM^{\frac{\log N}{\log N/\gamma}}\right)^d$$

If $N > 1$:

$$\begin{aligned}|S_\varepsilon| &= O\Big(\big(NM^{\frac{\log N}{\log N/\gamma}}\big)^D\Big) = O\Big(\big(NM^{\frac{\log N}{\log N/\gamma}}\big)^{\frac{\log \varepsilon(1-\gamma)}{\log\gamma}}\Big)\\ &= O\Big(\varepsilon^{-\frac{\log N}{\log 1/\gamma}(1+\frac{\log M}{\log N/\gamma})}\Big)\end{aligned}$$

yielding the desired value of $\psi_{\text{rew}} = \frac{\log N}{\log 1/\gamma}(1+\frac{\log M}{\log N/\gamma})$.

If $N = 1$ (deterministic case), $\psi_{\text{rew}} = 0$ and:

$$\begin{aligned}|S_\varepsilon| &= \sum_{d=0}^{D} 1\cdot M = (D+1)M = \left(\frac{\log\varepsilon(1-\gamma)}{\log\gamma}+1\right)M\\ &\le a\log 1/\varepsilon\end{aligned}$$

for small ε and some constant a, which is of the form (4.5) for $b = 1$. From Theorem 4.2, the near-optimality is $O(\exp(-\frac{n}{a}))$. ■

The values of ψ_{rew} are smaller than ψ_{unif}, so the guarantee takes advantage of the additional structure introduced in the problem by the reward function.

In the deterministic case, $\psi_{\text{rew}} = 0$, the problem becomes easy and near-optimality is exponential in n, having the form in the second branch of Theorem 4.2 for $b = 1$. This is the same bound that OPD obtains when a single policy is optimal.

Examining the algorithm reveals that it will only explore the optimal policy's subtree, with branching factor N, so the analysis is conservative in this case and the ideal value for ψ is $\frac{\log N}{\log 1/\gamma}$.

Structured probabilities. Finally, we consider problems where the rewards for all transitions are equal, but the transitions have significantly different probabilities. Take for simplicity identical Bernoulli transitions: $N = 2$ and the two successors of any state (and thus state node) have probabilities p and $1-p$, so that p is close to 1.

Proposition 4.8 *In the Bernoulli case, for p close to 1, $\psi_{\text{prob}} = \frac{\log M\eta'}{\log 1/(p\gamma\eta')}$, where $\eta' = \left(\frac{e}{\eta}\right)^{\eta}$ and $\eta = \frac{\log 1/(p\gamma)}{\log 1/(\gamma(1-p))}$*

Proof: We will show that the quantities of nodes with sizable contributions on the subtree of one policy, and respectively on the whole tree, satisfy:

$$n(\theta) = |\{s \in \mathscr{T}_\infty \,|\, c(s) \geq \theta\}| = \tilde{O}(\theta^{-\varphi})$$
$$n_h(\theta) = |\{s \in \mathscr{T}_{h_\infty} \,|\, c(s) \geq \theta\}| = \tilde{O}(\theta^{-\varphi_h})$$

for constants φ_h and φ; and we will find values for these constants. (Note $n_h(\theta)$ is not a function of h, since all policies have the same probability structure.) Then, since condition (ii) always holds and nodes in S_ε only have to satisfy condition (i):

$$\begin{aligned}
|S_\varepsilon| &= |\{s \in \mathscr{T}_\infty \,|\, n(s)c(s) \geq \varepsilon\}| \\
&\leq |\{s \in \mathscr{T}_\infty \,|\, n_h(c(s))c(s) \geq \varepsilon\}| \\
&\leq \left|\left\{s \in \mathscr{T}_\infty \,\middle|\, a[\log 1/c(s)]^b c(s)^{1-\varphi_h} \geq \varepsilon\right\}\right| \\
&= \tilde{O}(\varepsilon^{-\frac{\varphi}{1-\varphi_h}})
\end{aligned}$$

where we used $n(s) \leq n_h(c(s))$ and $n_h(c(s)) = \tilde{O}(c(s)^{-\varphi_h})$. Thus $\psi = \frac{\varphi}{1-\varphi_h}$.

Consider now $n_h(\theta)$. The nodes at each depth d correspond to a binomial distribution with d trials, so there are C_d^m nodes with contribution $c(s) = p^{d-m}(1-p)^m\frac{\gamma^d}{1-\gamma}$, for $m = 0, 1, \ldots, d$. Since these contributions decrease monotonically with d, as well as with m at a certain depth, condition $c(s) \geq \theta$ eliminates all nodes above a certain maximum depth D, as well as at every depth d all nodes above a certain $m(d)$, where:

$$\frac{(p\gamma)^d}{1-\gamma} \geq \theta \quad \Rightarrow \quad d \leq \frac{\log 1/(\theta(1-\gamma))}{\log 1/(p\gamma)} = D$$
$$m \leq \frac{\log 1/(\theta(1-\gamma))}{\log p/(1-p)} - d\frac{\log 1/(p\gamma)}{\log p/(1-p)} = m(d)$$

Note in the condition for D we set $m = 0$ to obtain the largest probability. So, $m(d)$ decreases linearly with d, so that up to some depth m^*, $m(d) \geq d$ and we count all the nodes up to $m = d$; while above m^*, $m(d) < d$ and we count fewer nodes. The depth m^* is obtained by solving $m(d) = d$, leading to $m^* = \frac{\log 1/(\theta(1-\gamma))}{\log 1/(\gamma(1-p))} = \frac{\log 1/(p\gamma)}{\log 1/(\gamma(1-p))} D = \eta D$ with the notation $\eta = \frac{\log 1/(p\gamma)}{\log 1/(\gamma(1-p))}$. The structure of the subtree satisfying $c(s) \geq \theta$ is represented in Figure 4.5.

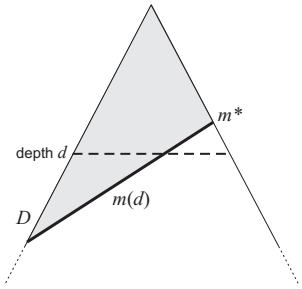

Figure 4.5: Schematic representation of the subtree satisfying $c(s) \geq \theta$, shown in gray. Nodes with larger probabilities are put to the left. The thick line represents the fringe $m(d)$ where nodes stop being counted.

Now:

$$\begin{aligned} n_h(\theta) &= \sum_{d=0}^{D} \sum_{m=0}^{\min\{m(d),d\}} C_d^m \leq \sum_{d=0}^{D} \sum_{m=0}^{\min\{m(d),d\}} \left(\frac{de}{m}\right)^m \\ &\leq \sum_{d=0}^{D} \sum_{m=0}^{m^*} \left(\frac{De}{m^*}\right)^{m^*} = Dm^* \left(\frac{De}{m^*}\right)^{m^*} \\ &= \eta D^2 \left(\frac{e}{\eta}\right)^{\eta D} = \tilde{O}\Big(\left(\frac{e}{\eta}\right)^{\eta D}\Big) \end{aligned}$$

where we used $C_d^m \leq \left(\frac{de}{m}\right)^m$ as well as $\left(\frac{de}{m}\right)^m \leq \left(\frac{De}{m}\right)^m \leq \left(\frac{De}{m^*}\right)^{m^*}$. The latter inequality can be shown by noticing that $\left(\frac{De}{m}\right)^m$, as a function of m, increases up to $m = D$, and $m^* \leq D$ is on the increasing part. Denoting now $\eta' = \left(\frac{e}{\eta}\right)^{\eta}$ and continuing:

$$n_h(\theta) = \tilde{O}(\eta'^D) = \tilde{O}(\eta'^{\frac{\log 1/(\theta(1-\gamma))}{\log 1/(p\gamma)}}) = \tilde{O}(\theta^{-\frac{\log \eta'}{\log 1/(p\gamma)}})$$

leading to the value for $\varphi_h = \frac{\log \eta'}{\log 1/(p\gamma)}$.[1]

[1]The definition of $n(s)$ in fact only requires counting the *leaves* of the subtree corresponding to $n_h(\theta)$ (thick line in Figure 4.5), while we counted all the nodes (gray area). Exploiting this property is unlikely to be helpful, however, since in the upper bound derived for $n_h(\theta)$ the inner term in the sum (corresponding to C_d^m, the number of nodes having a certain probability) is dominant. The fact that the whole tree is taken into account only enters the logarithmic component of the bound.

Similarly, it is shown that $n(\theta) = \tilde{O}(\theta^{-\frac{\log M\eta'}{\log 1/(p\gamma)}})$ and thus $\varphi = \frac{\log M\eta'}{\log 1/(p\gamma)}$, where the extra M comes from the fact we count the nodes corresponding to all M^d policies rather than just one.

The desired result is immediate: $\psi_{\text{prob}} = \frac{\varphi}{1-\varphi_h} = \frac{\log M\eta'}{\log 1/(p\gamma\eta')}$. Note throughout, we silently used the fact that p is close to 1; indeed, this is required for some of the steps to be meaningful, such as having $\log 1/(p\gamma\eta') > 0$. ■

In the deterministic case, when $p \to 1$, $\eta \to \infty$ leading to $\eta' \to 1$, and $\psi_{\text{prob}} \to \frac{\log M}{\log 1/\gamma}$, recovering the uniform-reward bound for $N = 1$. Nearby, when p is large, $\psi \approx \frac{\log M}{\log 1/\gamma}$, and OPMDP expands "almost" only a tree with branching factor M. When the probabilities are uniform, $\eta = 1$ would lead to $\psi_{\text{prob}} = \frac{\log eM}{\log 2/e\gamma}$, and ψ_{unif}, which was developed specifically for that case, is better.

4.1.4 Experimental results

The behavior of OPMDP is studied in the relatively simple inverted pendulum problem, and then in the more challenging problem of HIV infection control.

Inverted pendulum swingup. Using the problem of swinging up an underactuated inverted pendulum, the behavior of OPMDP is studied as a function of the computational budget n provided, and OPMDP is compared to uniform planning and OLOP (Bubeck and Munos, 2010). The system is the same as in Section 3.1.3 and the discount factor remains unchanged at $\gamma = 0.95$, but the reward function weights are changed to $Q_{\text{rew}} = \text{diag}[5, 0.1]$, $R_{\text{rew}} = 1$. The actions are discretized into the set $U = \{-3, 0, 3\}$, so that $M = 3$. Recall that these action magnitudes are insufficient to push up the pendulum in one go. In addition to the deterministic dynamics of the original pendulum, an *unreliable actuator* is modeled that only applies the intended action u with probability 0.6, and applies an action with smaller magnitude, $0.7u$, with probability 0.4 (when the intended action is 0 it remains 0 with probability 1). This corresponds to an MDP from the class handled by OPMDP, with $N = 2$.

For OPMDP and uniform planning, the computational budget n varies in the set $\{100, 200, \ldots, 1000\}$. OLOP has a different computational unit, consisting of simulating a single random transition instead of NM such transitions, so for fairness it is allowed $NM = 6n$ transitions. Note that instead of the theoretical OLOP algorithm of Bubeck and Munos (2010), we use a variant more amenable to practical implementation, which like OPMDP relies on developing planning trees.

To obtain a global performance measure, all algorithms are applied in an offline fashion, to find actions u_0 for the states on the grid:

$$X_0 = \left\{-\pi, \tfrac{-150\pi}{180}, \tfrac{-120\pi}{180}, \ldots, \pi\right\} \times \{-15\pi, -14\pi, \ldots, 15\pi\}$$

The average over this grid of a quantity called *simple regret* is reported. The simple regret $V^*(x_0) - Q^*(x_0, u_0)$, i.e. loss resulting from applying the possibly suboptimal

action u_0 returned by OP and then acting optimally, with respect to acting optimally from the first step and achieving $V^*(x_0)$. This measure of performance isolates the effects of the action returned, and as such is appropriate for an algorithm applied in receding-horizon, as OP methods usually are. Since an exact optimal solution for the inverted pendulum problem is not known, in order to approximate the regret, a near-optimal solution is computed instead. To this end, the fuzzy Q-iteration algorithm (Buşoniu et al., 2010b) is modified to work for the sparsely stochastic systems considered in this section, and applied to the inverted pendulum using a very accurate approximator over the state space.

Figure 4.6, top-left reports the (approximate) regret of the three algorithms, averaged over the set X_0. As expected, OPMDP is better than uniform planning, since it expands the planning trees in a smart way. As Figure 4.6, top-right shows, this results in much deeper trees than for uniform planning. Less expected is that, despite its strong theoretical guarantees, OLOP works poorly, similarly to uniform planning. This happens because the computational budgets considered do not allow OLOP to sufficiently decrease the upper confidence bounds on the returns; any advantage OLOP may have can only manifest for larger budgets. Because the algorithms simulate a similar number of transitions, their execution times are similar (Figure 4.6, bottom). Note that with these execution times the algorithms would not yet be applicable in real-time; a faster implementation than our proof-of-concept Matlab program is needed for that.

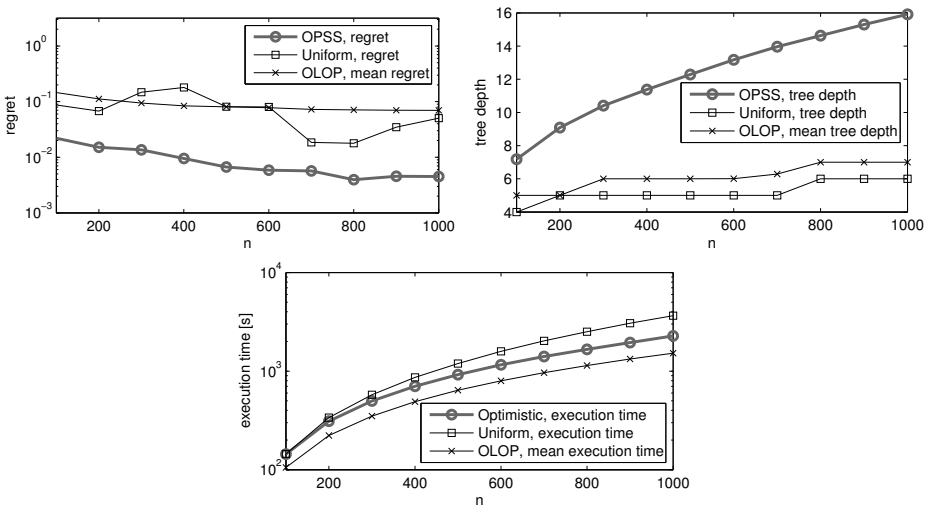

Figure 4.6: Comparison between OPMDP and uniform planning: average regret over X_0 (top-left), average tree depth over X_0 (top-right), execution time (bottom). As the results of OLOP depend on particular realizations of stochastic trajectories, this algorithm is run 10 times and mean results are reported (the 95% confidence regions are too tight to be visible at this scale).

HIV infection control. We use the HIV infection problem of Section 3.2.4, where the two treatment drugs have stochastic effectiveness. Unlike in Section 3.2.4 where this uncertainty was handled conservatively using the minimax algorithm OMS, here we take into account the probability distribution of (3.17) and apply OPMDP.

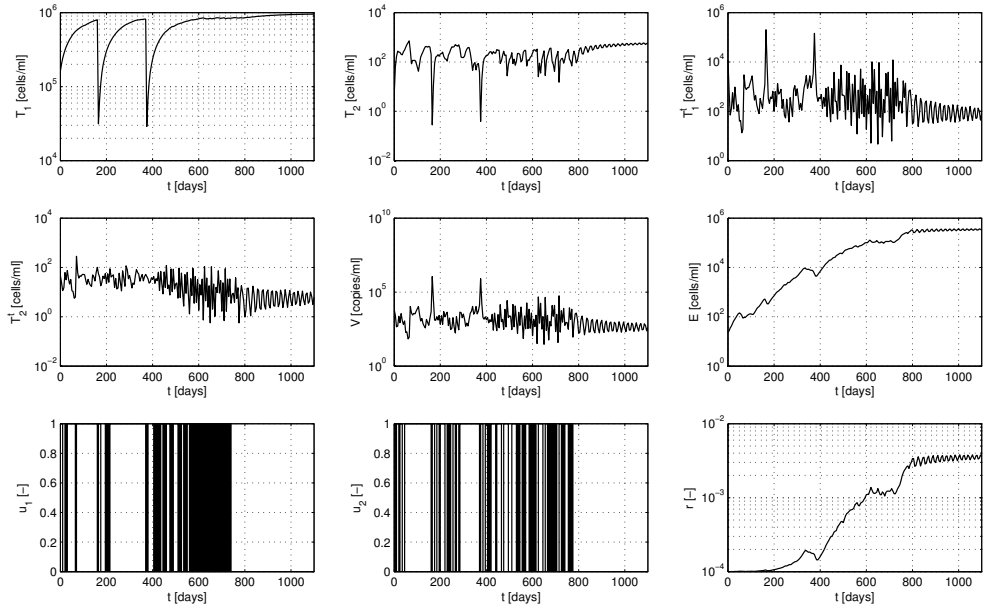

Figure 4.7: Trajectory of HIV system controlled online with OPMDP. The trajectories of the six system states are shown on the top and middle rows, while the two applied actions and the obtained rewards are shown on the bottom row.

Figure 4.7 shows the results when controlling the system online starting from x_{u}, with a computational budget of $n = 3000$ node expansions at each time step. Like OMS, the algorithm eventually stops administering drugs ($u_1 = u_2 = 0$), and the state slowly converges to the desired, healthy equilibrium x_{d}. We also applied uniform planning and OLOP to this problem, with poorer results than OPMDP; graphs are not provided here. The CPU time required by OPMDP to plan an action for each state was around 350 s in our Matlab implementation – significantly smaller than the decision interval of 5 days, which means that the algorithm would easily satisfy the real-time constraints in this problem.

4.2 Optimistic planning with discretization of continuous transition distributions

The OPMDP method discussed in the previous section is limited to transition distributions with finitely many possible next states. Our goal here is to overcome this limitation and extend the method to a class of stochastic systems with *continuous* transition distributions. This type of transitions is essential in practical applications of control, arising e.g. due to continuous disturbances and noise. Some OP techniques can be applied to continuous transition distributions (Bubeck and Munos, 2010; Weinstein and Littman, 2012, 2013) but they only search for open-loop action sequences. Such solutions are suboptimal in the stochastic case, and closed-loop solutions are needed to react to the transition realizations. By expoiting the principles of OPMDP, we will be able here to search for such closed-loop solutions.

The proposed method tackles the continuous transition distributions by discretizing them into *sigma points* using the unscented transform (Julier and Uhlmann, 1997; Ristic et al., 2004, Ch. 2). The method is therefore called sigma-OP. This simple idea allows us to directly apply OPMDP to the discretized version of the model; the solution it returns is then applied to the original system. Building on existing OP guarantees, we show that this solution is near-optimal, where the bound includes a new, constant term due to the error (assumed bounded) made by the unscented transform in approximating the relevant expectations. The method is evaluated in simulations on a linear system and a nonlinear one.

4.2.1 Problem statement and OP with sigma-point discretization

We are in the general case of Section 2.1, where a stochastic system is considered. The probability density function of the next state x' after taking action u in state x is $f(x,u,x')$, and the reward function is $\rho(x,u,x')$. In addition to the standing Assumption 2.1 of reward boundedness, we require discrete actions per Assumption 2.4. However, unlike in Section 4.1 above, we do *not* require discrete next states, since our goal is to work with the *continuous* transition distributions.

It will be useful to consider the dynamic programming backup operator:

$$T(V;x,u) := \mathrm{E}_{x'\sim f(x,u,\cdot)}\left\{\rho(x,u,x') + \gamma V(x')\right\} \tag{4.9}$$

which is parameterized by the value function V and applied at a given state-action pair (x,u). Note already the similarity with the L and B-value updates in OPMDP, see (4.3), which in fact are dynamic programming backups along the tree (this will be detailed later). The optimal value function $V^*(x)$ is the solution of the Bellman equation:

$$V^*(x) = \max_u T(V^*;x,u)$$

Operator T involves expectations over continuous variables, which cannot be computed in general, so it is not possible to directly use this operator to implement a planning algorithm for the original continuous distribution.

Instead, our main idea is simple: exploit the unscented transform to approximate the expectations, by discretizing the random variable x' into a set of *sigma points* (Ristic et al., 2004, Ch. 2). Denote by $\mu(x,u)$ and $\sigma(x,u)$ the mean and covariance of x'; note they can change with the origin (x,u) of the state transition. The means and covariances must be known (true for many forms of f), or otherwise their estimation must be computationally cheap. The sigma points are then:

$$\mathscr{X}_i(x,u) = \mu(x,u) + \begin{cases} 0, & i = 0 \\ [\sqrt{(m+\mathscr{K})\sigma(x,u)}]_i, & i = 1,\dots,m \\ -[\sqrt{(m+\mathscr{K})\sigma(x,u)}]_i, & i = m+1,\dots,2m \end{cases}$$

and their weights:

$$p_0(x,u) = \frac{\mathscr{K}}{m+\mathscr{K}}, p_i(x,u) = \frac{1}{2(m+\mathscr{K})}, i = 1,\dots,2m$$

where $\mathscr{K}$ is a tuning parameter and $[\cdot]_i$ denotes the ith row of the argument matrix.

Then, the expected value in (4.9) is approximated by a weighted summation:

$$\begin{aligned} T(V;x,u) &\approx \sum_{i=0}^{2m} p_i(x,u)[\rho(x,u,\mathscr{X}_i(x,u)) + \gamma V(\mathscr{X}_i(x,u))] \\ &=: \widehat{T}(V;x,u) \end{aligned}$$

which is accurate up to the second order of the Taylor expansion of the function inside the expectation (Ristic et al., 2004).

To be able to apply OPMDP, define a discretized, approximate version of the original continuous-distribution problem, by taking $\widehat{f}(x,u,x') = p_i(x,u)$ when $x' = \mathscr{X}_i(x,u)$, and 0 otherwise. Importantly, the weights p_i are interpreted as probabilities (hence the notation), which is possible as long as $\mathscr{K}$ is chosen positive. The reward function ρ is kept unchanged. Denote by $\widehat{V}$ a generic value function in the discretized problem, and by $\widehat{V}^*$ the optimal value function.

OPMDP (Algorithm 4.2) is then simply applied to this discretized problem, to find an approximately optimal action u_0 which is applied to the real, original system. We call the overall approach sigma-OP, short for "OP with sigma-point discretization".

To get better insight, we explain the algorithm in more detail. It expands a node by only creating children for the sigma points, $N = 2m+1$ children for each state-action pair. To compute B and L-values on the tree, it applies (4.3). At leaves the values are initialized as usual, and at inner node the updates look like, e.g. for the

B-values:

$$B(s) = \max_u \sum_{s' \in \mathscr{C}(s,u)} \widehat{f}(x(s), u, x(s'))[\rho(x(s), u, x(s')) + \gamma B(s')] \tag{4.10}$$

By making the probabilities $\widehat{f}$ and state labels $x(s')$ explicit using the sigma-point discretization, we get:

$$B(s) = \max_u \sum_{i=0}^{2m} p_i(x(s), u)[\rho(x(s), u, \mathscr{X}_i(x(s), u)) + \gamma V(\mathscr{X}_i(x(s), u))] = \max_u \widehat{T}(V; x, u)$$

So, OPMDP in the discretized problem boils down to using the discretized operator $\widehat{T}$ to backup the B and L values.

4.2.2 Analysis

Since the discretized problem is a valid MDP, the guarantees of OPMDP directly hold *for this discretized problem*. Denote $\widehat{v}^* = \widehat{V}^*(x_0)$, and by $\widehat{v}(u_0)$ the value of the truncated policy consisting of the single action u_0, see (4.2). The following near-optimality bound for the discretized MDP is essential:

$$\widehat{v}^* - \widehat{v}(u_0) \leq \varepsilon_n \tag{4.11}$$

where ε_n is the near-optimality of OPMDP after n iterations, see Theorem 4.2, The bound follows directly because u_0 is a subpolicy of h^*, so $\widehat{v}(u_0) \geq l(u_0) \geq l(h^*)$.

The goal is, however, to bound the suboptimality in the *original* problem. We therefore analyze the corresponding quantity $v^* - v(u_0)$ under the original value function, i.e. with $v^* := V^*(x_0)$ and $v(u_0) := T(V^*; x_0, u_0)$. Note that, similarly, $\widehat{v}(u_0) = \widehat{T}(\widehat{V}^*; x_0; u_0)$.

Define an approximate value iteration algorithm:

$$\widehat{V}_{t+1}(x) = \max_u \widehat{T}(\widehat{V}_t; x, u) \tag{4.12}$$

with $\widehat{V}_0$ arbitrarily initialized. These updates are closely related to sigma-OP: when $\widehat{V}_0 = \frac{1}{1-\gamma}$ (or 0) they correspond to B-value (or L-value) updates on the infinite tree $\mathscr{T}_\infty$.

Assumption 4.2 *There exists an $\sigma \geq 0$ and a bounded initial value function $\widehat{V}_0$ so that, for any $\widehat{V}_t$ and any x, u, sigma-point approximation makes an error of at most σ:* $\left|T(\widehat{V}_t; x, u) - \widehat{T}(\widehat{V}_t; x, u)\right| \leq \sigma$, *and the same for V^*:* $\left|T(V^*; x, u) - \widehat{T}(V^*; x, u)\right| \leq \sigma$.

Since the unscented transform guarantees accuracy up to the second order, this assumption essentially requires the terms under the expectations (rewards and value functions) to be well-behaved. For example, if the rewards (and therefore the value functions) are quadratic in x', then $\sigma = 0$. Under Assumption 4.2, the following main result holds.

Theorem 4.9 *The action returned by sigma-OP is near-optimal:* $v^* - v(u_0) \leq \varepsilon_n + 2\frac{\sigma}{1-\gamma}$.

Proof: The assumption implies that for any t and x:

$$\left|\widehat{V}_{t+1}(x) - \max_u T(\widehat{V}_t; x, u)\right| \leq \sigma$$

where the second term is the update *exact* value iteration would apply to $\widehat{V}_t$. In addition, since (4.12) is just value iteration on the discretized MDP, it converges to $\widehat{V}^*$. Given these conditions and the boundedness of V^*, standard results in approximate dynamic programming (Bertsekas and Tsitsiklis, 1996, Sec. 6.5.3)[2] guarantee that:

$$\left|V^*(x) - \widehat{V}^*(x)\right| \leq \frac{\sigma}{1-\gamma}, \quad \forall x \tag{4.13}$$

Now:

$$\begin{aligned}
\widehat{v}(u_0) &= \widehat{T}(\widehat{V}^*; x_0, u_0) \\
&= \widehat{T}(V^*; x_0, u_0) + \gamma \sum_{i=0}^{2m} p_i[\widehat{V}^*(\mathscr{X}_i) - V^*(\mathscr{X}_i)] \\
&= v(u_0) + \widehat{T}(V^*; x_0, u_0) - T(V^*; x_0, u_0) + \gamma \sum_{i=0}^{2m} p_i[\widehat{V}^*(\mathscr{X}_i) - V^*(\mathscr{X}_i)]
\end{aligned}$$

by the definition of $\widehat{v}(u_0)$ and $v(u_0)$, where for readability we skipped the argument (x,u) of p_i and $\mathscr{X}_i$. The first difference in the last formula is bounded in magnitude by σ by Assumption 4.2, and the second by $\gamma\frac{\sigma}{1-\gamma}$ due to (4.13). Therefore, $|\widehat{v}(u_0) - v(u_0)| \leq \sigma + \gamma\frac{\sigma}{1-\gamma} = \frac{\sigma}{1-\gamma}$.

Combining this with $\left|V^*(x_0) - \widehat{V}^*(x_0)\right| \leq \frac{\sigma}{1-\gamma}$ from (4.13), and with (4.11), $\widehat{v}^* - \widehat{v}(u_0) \leq \varepsilon_n$, the final result follows. ■

The theorem says that sigma-OP will make, in addition to the error ε_n made by OP in the discretized problem, (up to) a constant error $\frac{2\sigma}{1-\gamma}$ due to the sigma point approximation. Connecting this to the analysis of OPMDP, in the worst case where the full tree must be expanded in the order of depth, we have $\varepsilon_n = \frac{\gamma^d}{1-\gamma}$ where $n = \frac{NM^d-1}{NM-1}$ expansions are required to reach depth d, so that $v^* - v(u_0) \leq (\gamma^{\frac{\log n}{\log NM}} + 2\sigma)\frac{1}{1-\gamma}$. In typical problems, only a smaller subtree must be expanded, so the ε_n part of the bound decreases faster with n.

Note that the OPMDP analysis closely characterizes the size of this subtree and the dependence of ε_n on n, but only asymptotically, for ε_n around 0 ($n \to \infty$). This is

[2] In fact, sigma-point discretization leads to a classical type of value function approximator called an averager.

not appropriate in sigma-OP, because the constant error $2\frac{\sigma}{1-\gamma}$ dominates asymptotically; instead, in future work it may be useful to investigate the behavior of the tree for ε_n on the order of nonzero constant $\frac{\sigma}{1-\gamma}$.

In any case, the analysis indicates that computation should not be invested to shrink the diameter beyond a constant value, since as n grows the performance will plateau around this constant anyway. This constant is on the order of the error made by sigma-point approximation.

4.2.3 Experimental results

In our experiments we compare sigma-OP with three alternative planning algorithms. The first is uniform planning in the discretized problem: it expands the same tree as sigma-OP, but uniformly, in the order of depth. It is still a correct algorithm (it attains a near-optimal solution given enough computation), but it shrinks diameters slower: it always behaves as sigma-OP would in the worst case. Uniform planning is used as a baseline to confirm that optimistic expansion makes sense despite approximation errors.

The second alternative is OP for deterministic systems (OPD) from Section 2.3 (Hren and Munos, 2008b), which is applied to the nominal, deterministic model, while the actions returned are heuristically executed in the true stochastic system. OPD can be viewed as "certainty-equivalence" planning. The final algorithm is HOLOP (Weinstein and Littman, 2012), which we also used in Section 3.1.3. HOLOP has different characteristics than sigma-OP: it seeks solutions represented as action sequences, which in contrast to closed-loop policies, cannot represent optimal controls in the stochastic case. HOLOP works for continuous actions, while the other three methods require discretized actions. So in effect HOLOP and sigma-OP are solving different optimal control problems, but they can both be applied as approximations to continuous-action continuous-distribution stochastic systems, as we will do below.

DC motor stabilization. The first problem concerns the DC motor introduced in Section 3.1.3, with linear dynamics but now affected by noise z:

$$x_{k+1} = Ax_k + Bu_k + z_k$$

The other variables have the same meaning as before: shaft angle $x_1 \in [-\pi, \pi]$ rad, angular velocity $x_2 \in [-16\pi, 16\pi]$ rad/s, and voltage $u \in [-10, 10]$ V. The noise z is zero-mean Gaussian distributed with covariance $\sigma = 0.1 \cdot \mathrm{diag}(1,1)$, leading to a stochastic component of considerable amplitude. The two states are kept within their bounds by saturation, also when discretizing into sigma points, and the actions are discretized in $\{-10, 0, 10\}$ V, so $M = 3$. The (unnormalized) rewards are quadratic with $Q_{\mathrm{rew}} = \mathrm{diag}(1,0)$, $R_{\mathrm{rew}} = 0.001$, and the discount factor is $\gamma = 0.95$. The DC

motor is chosen because its nominal dynamics are simple, so we can focus on the effects of noise.

The four planning algorithms were applied in receding horizon, from the initial state $[-\pi, 0]^T$ and for a duration of 1 s (100 steps, due to a sampling time of 0.01 s). For each algorithm and parameter setting, 25 independent runs were performed. In sigma-OP and uniform planning, the number of sigma points was $N = 5$ ($m = 2$), and $\mathscr{K}$ was set to 0.001. The algorithms were tested for a range of computational budgets expressed as numbers of transitions simulated during planning at each step: $n_\mathrm{t} = 150, 300, 600, 1200, 2400, 4800$. Since for sigma-OP and uniform planning budgets n are given in terms of node expansions, and each expansion takes $NM = 15$ transitions, n is taken $\lceil n'/15 \rceil$ where $\lceil \cdot \rceil$ denotes ceiling. HOLOP is additionally parameterized by the horizon K over which it searches for action sequences, and for each n_t we tried values $5, 10, 25, 75, 100$ for K; the experiment with the best upper confidence bound on the return is reported (in this problem, $K = 5$ was always the best). Since the problem is linear and quadratic, by disregarding the state and action constraints a continuous-action optimal solution is analytically computed as in (Bertsekas, 2012, Sec. 4.2), and its optimal value is included on the graphs.

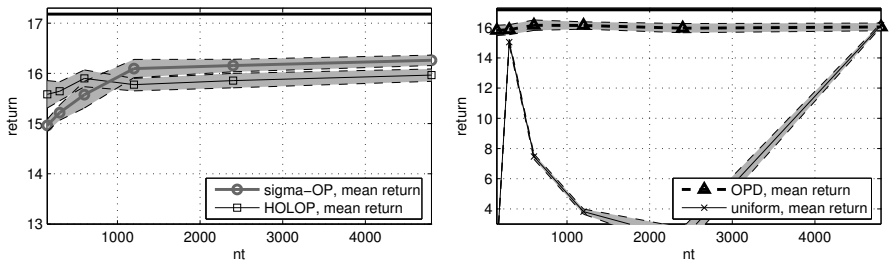

Figure 4.8: Return for the DC motor. Mean performances are shown, with their 95% confidence intervals as a shaded region. The horizontal solid line shows the analytically computed optimal value. For readability, results are shown in two graphs with different vertical scale.

The results are shown in Figure 4.8, where performance is measured by the discounted return obtained in the experiment. For larger budgets sigma-OP is better than HOLOP. Certainty-equivalence OPD on the other hand performs as well as sigma-OP. Note however that using it means the analytical guarantees of Theorem 4.9 are sacrificed, since OPD is only a heuristic in the stochastic problem. Uniform planning has unreliable performance, sometimes reaching good values but then becoming worse again for larger budgets. This indicates a good algorithm must search optimistically. Since it is a correct algorithm, uniform planning would eventually stabilize to a good performance for very large budgets. Because actions are discretized, no algorithm reaches the continuous-action optimal value.

Inverted pendulum swing-up. For the second problem we consider the inverted pendulum from Section 3.1.3, but now with additive zero-mean Gaussian noise, with covariance $\sigma = 0.015 \cdot \mathrm{diag}(1,1)$. The reward weights are $Q_{\mathrm{rew}} = \mathrm{diag}(1,0)$ and $R_{\mathrm{rew}} = 0.3$. The simulation time was chosen to be 5 s with a sampling time of 0.05 s (100 steps), and the initial state was $x_0 = [-\pi, 0]^T$ (pointing down).

The same values as for the DC motor were set for the sigma points parameters, while the budgets were $n_{\mathrm{t}} = 500, 1000, 5000, 10000, 15000$ (and $n = \lceil n'/15 \rceil$). Figure 4.9, left reports the results of a batch of 35 experiments. For the K parameter in HOLOP, values $5, 10, 15, 20, 25, 30, 40, 50, 75, 100$ were attempted, and the best results corresponding to the values of n_{t} above are obtained for, respectively, $K = 10, 5, 10, 15, 10$. Sigma-OP remains overall better than HOLOP, although for $n_{\mathrm{t}} = 500$ and 15000 their performances are not statistically different. OPD this time works better than all other algorithms, confirming that it may be a good choice in practice despite the lack of guarantees. Uniform planning is still unreliable.

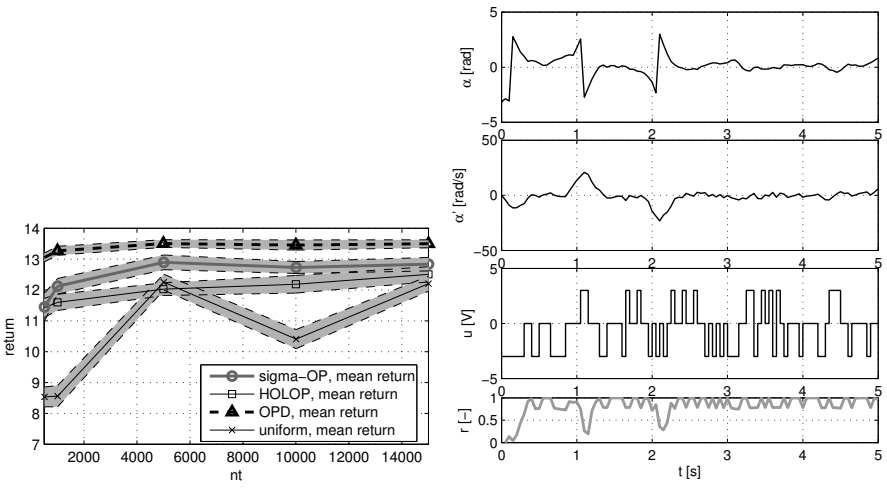

Figure 4.9: Left: Return for the inverted pendulum. Right: A controlled trajectory.

Figure 4.9, right shows a controlled trajectory with sigma-OP, for $n = 667$. Chattering is observed due to controlling to an unstable equilibrium using only discrete actions, and also because sometimes the random transitions move the system away from the equilibrium and it must be brought back. At certain points even a new swing may be necessary.

Regarding computation, it is dominated by the number of simulations, which is set the same for all algorithms. However, small differences arise due to their different internal workings. In our Matlab implementation, HOLOP is more expensive, while the other algorithms are cheaper and their relationships change with the problem.

4.3 Summary and conclusions

In the first part of this chapter, we have studied *optimistic planning for stochastic MDPs* (OPMDP), an algorithm which explores the space of closed-loop planning policies. The core feature of this method is that it adapts to the complexity of the planning problem, encoded in the near-optimality exponent ψ. Specializing the exponent and performance bound for some interesting classes of MDPs illustrated that the algorithm works better when there are fewer near-optimal policies and less uniform transition probabilities. Simulation results showed that OPMDP provides good performance.

In the second part, OPMDP was extended to the case of continuous transition distributions. The extension was performed by discretizing the continuous distribution using sigma points, and obtaining an algorithm we called *optimistic planning with sigma-point discretization*. Assuming the error introduced by the sigma-point discretization is bounded, we analyzed the solution returned, showing that it is near-optimal. In experiments, sigma-OP successfully solved a linear problem and a nonlinear one. A heuristic certainty-equivalence approach that plans using the deterministic model also performed very well empirically.

Chapter 5

Planning: Related topics and outlook

5.1 Related directions

Alongside the main fundamental research line described above, several other topics have been investigated. They are outlined next.

A common feature of the algorithms analyzed in previous chapters is that in general they have exponential complexity: the computational budget needed to achieve a certain near-optimality is exponential in this near-optimality. For example, to achieve ε-optimal solutions we need to invest $n = O(\varepsilon^{-\Psi})$ budget in OPMDP, or $O(\varepsilon^{-\frac{\log \vartheta}{\log 1/\lambda}})$ in OMS (the latter bound is obtained by inverting the relationship in Theorem 3.6). This is an unavoidable consequence of the generality of these algorithms, since they work for any nonlinear dynamics and reward functions. It can, however, be cumbersome in applications where computation time is limited. Therefore, one line of research has focused on designing planning algorithms with sub-exponential complexity. This can only be done by restricting the class of solutions explored – or, equivalently, the class of problems where near-optimal solutions can be found. In particular, we have focused on the specific subclass of control problems where control actions do not change often, such as bang-bang, time-optimal control.

In this context, we have explored two ideas. The first is to refine action sequences similarly to OPD, but in addition to one-step actions, to also add repetitions of each action up to K times. An algorithm called optimistic planning with K identical actions (OKP) is obtained, published in (P7). It was analyzed showing that the a posteriori performance guarantees are similar to those of OPD, improving with the length of the explored sequences. When K is properly tuned, OKP overperforms OPD in some time-optimal control experiments. However, from an analytical point of view the depths of the OKP trees are in most cases smaller than those of the OPD trees, and therefore OKP ensures a weaker regret bound than OPD, which is disappointing.

Motivated by this, we have explored a second, alternative idea of enforcing a limit on the total number of *switches* between different actions in the sequences explored. This algorithm is called optimistic switch-limited planning (OSP) and was published in (P8). We developed analysis showing that the switch constraint leads to polynomial complexity in the search horizon, in contrast to the exponential complexity of state-of-the-art OP; and to a correspondingly faster convergence. The degree of the polynomial varies with the problem and is a meaningful measure for the difficulty of solving it. We studied this degree in two representative, opposite cases. In simulations we first applied OSP to a problem where limited-switch sequences are near-optimal, and then in a networked control setting where the switch constraint must be satisfied in closed loop.

A different research line considers the case where the dynamics of the system are not known. In particular, we developed in (P9) the Bayesian OP (BOP) algorithm, which extends OPMDP to the case where the transition probabilities of the MDP are initially unknown and progressively learned through interactions within the environment. The knowledge about the unknown MDP is represented with a probability distribution over all possible transition models, using a Dirichlet parametrization, and the BOP algorithm plans in the belief-augmented state space constructed by concatenating the original state vector with the current posterior distribution over transition models. Using a straightforward extension of the OPMDP analysis, we showed that BOP becomes Bayesian optimal when the budget parameter increases to infinity. Preliminary empirical validations showed promising performance.

The following publications detail the directions discussed in this section:

(P7) K. Máthé, L. Buşoniu, L. Miclea, "Optimistic Planning with Long Sequences of Identical Actions for Near-Optimal Nonlinear Control". In *Proceedings 2014 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR-14)*, Cluj-Napoca, Romania, 22–24 May 2014.

(P8) K. Máthé, L. Buşoniu, R. Munos, B. De Schutter, "Optimistic Planning with a Limited Number of Action Switches for Near-Optimal Nonlinear Control", *Proceedings 2014 Conference on Decision and Control (CDC-14)*, Los Angeles, USA, 15–17 December 2014.

(P9) R. Fonteneau, L. Buşoniu, R. Munos, "Optimistic Planning for Belief-Augmented Markov Decision Processes", *Proceedings 2013 Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-13)*, Singapore, 15–19 April 2013.

5.2 Open issues and ongoing work

For SOOP, the main open issue is analyzing the performance as a function of the budget n of model calls, and this is an important direction of current work. We have

begun with an analysis of a DOO-like continuous-action algorithm. This already presents significant challenges in deriving an appropriate complexity measure, but we have a significant intermediate result.

In OMS, by requiring the upper and lower bounds, we have implicitly assumed knowledge about the smoothness of the value function. This can easily be obtained e.g. for discounted returns, but in general it may be too strong. Thus, the next big step is to check if it is possible to derive an algorithm that does *not* require knowing these bounds, only that they exist and get closer together as depth increases. This is again similar to the idea in SOO, so similar principles should apply.

The basic form of OPMDP studied above can benefit from many algorithmic improvements, and analyzing their effect on the bounds would be very interesting. For example, since OP is similar to AO*, improvements originally developed for classical AO* can almost directly be applied, such as closing the tree into a graph upon encountering duplicate states (Hansen and Zilberstein, 1999). OPMDP does not assume any knowledge about the complexity of the problem, e.g. it does not require to know β. Deriving algorithms that, when such prior knowledge is available, exploit it to obtain better performance guarantees is another open issue.

The unscented transform used by sigma-OP is just one way to discretize the transition distribution, and other methods could be applied. E.g. one limitation of sigma points is that most of them have equal probabilities, which can lead to a large branching factor in the OP tree, so following the analysis of OMDP we may expect poor performance. An alternative is to use a particle-filter-like discretization, for which the probabilities are less symmetrical, similarly to Silver and Veness (2010). The branching factor may then be too large due to the number of particles, and different tree expansion strategies may be necessary.

Orthogonal to these algorithm-specific developments, another essential idea for online control is *reusing* information across interaction steps. We are actively pursuing this for OPD and OPMDP, but the idea applies to all the algorithms. The basic algorithms discard the planning data at the end of each planning step, and start from scratch at the next step. To avoid this waste, we have developed a method to learn online, from this data, the upper bounds used to guide the planning process. Several approximators for the upper bounds are studied. Our analysis characterizes the influence of the approximation error on the performance, and reveals that for small errors, learning-based planning performs better; while experimental studies are very promising. A journal submission has already been developed presenting our research in this direction. In this context, near-optimality guarantees for approximate value iteration (Szepesvári, 2001; Rust, 1996) could be useful to analyze the actual convergence of the learned b-values across the steps; and then this analysis can be combined with the existing near-optimality guarantees to achieve a complete picture.

Bibliography

Adams, B., Banks, H., Kwon, H.-D., and Tran, H. (2004). Dynamic multidrug therapies for HIV: Optimal and STI control approaches. *Mathematical Biosciences and Engineering*, 1(2):223–241.

Berliner, H. (1979). The B* search algorithm: A best first proof procedure. *Artificial Intelligence*, 12.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4th edition.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Billinton, R. and Allan, R. N. (1996). *Reliability Evaluation of Power Systems*. Springer.

Bubeck, S. and Munos, R. (2010). Open loop optimistic planning. In *Proceedings 23rd Annual Conference on Learning Theory (COLT-10)*, pages 477–489, Haifa, Israel.

Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. (2009). Online optimization in X-armed bandits. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 201–208. MIT Press.

Buşoniu, L., Babuška, R., De Schutter, B., and Ernst, D. (2010a). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Automation and Control Engineering. Taylor & Francis CRC Press.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2010b). Approximate dynamic programming with a fuzzy parameterization. *Automatica*, 46(5):804–814.

Cassandras, C. G. and Lafortune, S. (1999). *Introduction to Discrete-Event Systems*. Kluwer.

Coquelin, P.-A. and Munos, R. (2007). Bandit algorithms for tree search. In *Proceedings 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 67–74, Vancouver, Canada.

Costa, O., Fragoso, M., and Marques, R. (2005). *Discrete-Time Markov Jump Linear Systems*. Springer.

Hansen, E. A. and Zilberstein, S. (1999). A heuristic search algorithm for Markov decision problems. In *Proceedings Bar-Ilan Symposium on the Foundation of Artificial Intelligence*, Ramat Gan, Israel.

Hren, J.-F. (2012). *Planification Optimiste pour Systèmes Déterministes*. PhD thesis, Lille 1 University - Science and Technology.

Hren, J.-F. and Munos, R. (2008a). Optimistic planning of deterministic systems. In *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, pages 151–164, Villeneuve d'Ascq, France.

Hren, J.-F. and Munos, R. (2008b). Optimistic planning of deterministic systems. In Girgin, S., Loth, M., Munos, R., Preux, P., and Ryabko, D., editors, *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*, pages 151–164. Springer.

Julier, S. and Uhlmann, J. (1997). A new extension of the kalman filter to nonlinear systems. In *Proceedings 11th International Symposium on Aerospace/Defense Sensing, Simulations and Controls (AeroSense-97)*.

Kearns, M. J., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208.

Kleinberg, R., Slivkins, A., and Upfal, E. (2008). Multi-armed bandits in metric spaces. In *Proceedings 40th Annual ACM Symposium on Theory of Computing (STOC-08)*, pages 681–690, Victoria, Canada.

Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings 17th European Conference on Machine Learning (ECML-06)*, pages 282–293, Berlin, Germany.

Korf, R. E. (1998). Artificial intelligence search algorithms. In Atallah, M., editor, *Algorithms and Theory of Computation Handbook*, pages 1–20. CRC Press.

Korf, R. E. and Chickering, D. M. (1996). Best-first minimax search. *Artificial Intelligence*, 84(1–2):299–337.

La Valle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Maciejowski, J. M. (2002). *Predictive Control with Constraints*. Prentice Hall.

Mahmoud, M., Jiang, J., and Zhang, Y. (2001). Stochastic stability analysis of fault-tolerant control systems in the presence of noise. *IEEE Transaction on Automatic Control*, 46(11):1810–1815.

Mansley, C., Weinstein, A., and Littman, M. L. (2011). Sample-based planning for continuous action Markov decision processes. In *Proceedings 21st International Conference on Automated Planning and Scheduling*, pages 335–338, Freiburg, Germany.

Meyer, P.-J., Girard, A., and Witrant, E. (2013). Controllability and invariance of monotone systems for robust ventilation automation in buildings. In *Proceedings 52nd IEEE Conference on Decision and Control (CDC-13)*, pages 1289–1294, Firenze, Italy.

Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 783–791.

Munos, R. (2014). The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search. *Foundations and Trends in Machine Learning*, 7(1):1–130.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga Publishing.

Palay, A. J. (1982). The B* tree search algorithm – new results. *Artificial Intelligence*, 19:145–163.

Pearl, J. (1982). The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. *Communications of the ACM*, 25(8):559–564.

Péret, L. and Garcia, F. (2004). On-line search for solving Markov decision processes via heuristic sampling. In *Proceedings 16th European Conference on Artificial Intelligence, ECAI'2004*, pages 530–534, Valencia, Spain.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (1996). Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1–2):255–293.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley.

Ratschan, S. (2002). Search heuristics for box decomposition methods. *Journal of Global Optimization*, 24:35–49.

Ristic, B., Arulampalam, S., and Gordon, N. (2004). *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House.

Rust, J. (1996). Numerical dynamic programming in economics. In Amman, H. M., Kendrick, D. A., and Rust, J., editors, *Handbook of Computational Economics*, volume 1, chapter 14, pages 619–729. Elsevier.

Silver, D. and Veness, J. (2010). Monte-Carlo planning in large POMDPs. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2164–2172. MIT Press.

Szepesvári, Cs. (2001). Efficient approximate planning in continuous space Markovian decision problems. *AI Communications*, 13(3):163–176.

Walsh, T. J., Goschin, S., and Littman, M. L. (2010). Integrating sample-based planning and model-based reinforcement learning. In *Proceedings 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, US.

Wang, Y., Audibert, J.-Y., and Munos, R. (2008). Algorithms for infinitely many-armed bandits. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1729–1736. MIT Press.

Wang, Y. and Gelly, S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *Proceedings 2007 IEEE Symposium on Computational Intelligence and Games (CIG-07) USA, 1-5 April, 2007*, pages 175–182, Honolulu, Hawaii.

Weinstein, A. and Littman, M. L. (2012). Bandit-based planning and learning in continuous-action Markov decision processes. In *Proceedings 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, São Paulo, Brazil.

Weinstein, A. and Littman, M. L. (2013). Open-loop planning in large-scale stochastic domains. In *Proceedings 27th AAAI Conference on Artificial Intelligence (AAAI-13)*, pages 1436–1442, Bellevue, Washington, US.

Part III

Applications to nonlinear networked systems

Introduction and outline

Beyond their fundamental interest in optimal control, optimistic planning methods are also useful to address other challenges in nonlinear control. This is because planning methods can address very general classes of nonlinear systems, whereas more standard techniques are often geared towards some specific classes.

We focus on challenges related to networked systems, motivated by the fact that society increasingly evolves towards a networked world. Communication networks, power and transport grids, decentralized computing networks, and social networks are just some examples illustrating this evolution. The effective, optimal operation of these networks is very important, and the best way to achieve it is active control of the systems connected into the network. We therefore investigate in Part III of this thesis applications of optimistic planning to the control of networked systems, treating these systems from two complementary perspectives. The first perspective tackles the coordinated behavior of multiple, interconnected systems called agents, under the constraints imposed by the interconnection topology (Chapter 6). In this context, we propose first an algorithm based on optimistic optimization of fixed-length action sequences in order to achieve consensus over the agents' state variables, under a fixed communication topology (Section 6.1). Then, in Section 6.2, we exploit optimistic planning over variable-length action sequences for flocking, where the topology is dictated by a proximity relationship between the agents; here our main analytical aim is to preserve the interconnection topology under this constraint.

The second perspective deals with communication constraints induced by a network interposed between a single system and its controller. In Chapter 7, we exploit optimistic planning for deterministic systems, and specifically the fact that it returns long and near-optimal action sequences. We propose two optimal, networked control strategies using planning that reduce the number of transmissions over the network. In the first strategy, action sequences are transmitted to the plant at a fixed period. In the second strategy, the algorithm decides the next transmission instant according to the last state measurement (leading to a self-triggered policy), working with a fixed computation budget.

All these algorithms are thoroughly analyzed, showing that they effectively solve the problems they target. We also evaluate them numerically in example problems. In Chapter 8, closing this part, we discuss open issues and ongoing work into the control

of networked systems, as well as other research directions connected to control.

The material in this part is based on the following publications:

(P10) L. Buşoniu, C. Morarescu, "Consensus for black-box nonlinear agents using optimistic optimization.", *Automatica*, vol. 50, no. 4, pages 1201–1208, 2014 (Section 6.1).

(P11) L. Buşoniu, C. Morarescu, "Consensus for Agents with General Dynamics Using Optimistic Optimization". *Proceedings 2013 IEEE Conference on Decision and Control (CDC-13)*, Florence, Italy, 10-13 December 2013 (Section 6.1).

(P12) L. Buşoniu, C. Morarescu, "Topology-preserving flocking of nonlinear agents using optimistic planning", *Control Theory and Technology*, vol. 13, no. 1, pages 70–81, 2015 (Section 6.2).

(P13) L. Buşoniu, C. Morarescu, "Optimistic planning for consensus", *Proceedings 2013 American Control Conference (ACC-13)*, Washington, US, 18–19 June 2013 (Chapter 6).

(P14) L. Buşoniu, R. Postoyan, J. Daafouz, "Near-optimal strategies for nonlinear networked control systems using optimistic planning", *Proceedings 2013 American Control Conference (ACC-13)*, Washington, US, 18–19 June 2013 (Chapter 7).

Chapter 6

Optimistic optimization and planning for multiagent consensus

Multi-agent systems are a type of interconnected systems with applications in a wide variety of domains, such as robotic teams, energy and telecommunication networks, collaborative decision support systems, data mining, etc. Each agent typically has only a local, limited view of the overall system, which means decentralized control approaches are necessary. Requirements on the coherent behavior of the agents are often expressed in terms of *consensus*, in which the agents must reach agreement on controlled variables of interest (Olfati-Saber et al., 2007; Ren and Beard, 2008). Inspired by the behavior of flocks of birds, researchers also studied the *flocking* variant of consensus, which only requires consensus on velocities while also using position measurements (Olfati-Saber, 2006; Tanner et al., 2007). Flocking is highly relevant in e.g. mobile robot teams (Dong, 2011). Often, the flocking agents must ensure that the network remains connected despite a limited inter-agent communication range.

Classical consensus and flocking algorithms are designed for agents with linear dynamics, for which the behavior is well understood. In this setting, the literature considers fixed and time-varying communication topologies (Ren and Beard, 2005; Moreau, 2005), directed or undirected graphs (Olfati-Saber et al., 2007; Ren and Beard, 2008), synchronous or asynchronous information exchange (Tsitsiklis et al., 1986; Fang et al., 2005), delayed or immediate transmissions (Olfati-Saber and Murray, 2004; Michiels et al., 2009), etc. Approaches also exist for nonlinear agent dynamics, such as second-order systems with nonlinear acceleration dynamics (Su et al., 2011; Zhou et al., 2012), nonholonomic robots (Tanner et al., 2005), and Euler-Lagrange dynamics (Mei et al., 2011). These works usually require an explicit mathematical model of the agents, the form of which is exploited to derive tailored control laws, often via Lyapunov synthesis.

Our goal here is different: generic consensus and flocking methods, applicable without changes to a wide class of nonlinear agents. To achieve this, we propose

approaches that only requires black-box models (Sjöberg et al., 1995; Suykens and Vandewalle, 1998) of the agents, thereby eliminating the dependence on the particular mathematical form of their dynamics. Neural networks are a typical category of black-box models, e.g. (Hunt et al., 1992; Lewis and Liu, 2012), but nonstandard models like computer programs are also allowed; such models are common e.g. in planning (Edelkamp and Schrödl, 2012).

Our two approaches focus, respectively, on consensus via optimistic optimization of fixed-length action sequences (Section 6.1), and on flocking using optimistic planning to find variable-length sequences (Section 6.2). Next, one section is dedicated to each of these two methods.

6.1 Optimistic optimization for consensus

The first method aims to achieve consensus for general nonlinear dynamics, and, for simplicity, on the entire vector of agent states. A first ingredient of the approach is a classical consensus algorithm, used to design reference next states for the agents. The main novelty is the use of optimistic optimization (OO) from Section 2.2 and (Munos, 2011) to solve the control problem of reaching close to these states, formulated as optimal predictive control. OO allows our approach to deal with highly general, black-box dynamics, since it performs global nonconvex optimization by only simulating the dynamics for some inputs. Furthermore, the near-optimality guarantees of OO allow a tight characterization of the relation between closeness to the reference states and computation invested, and in particular guarantee arbitrarily small errors as computation increases. Exploiting this, our main results prove practical consensus, i.e. that the agents converge close to each other, as detailed next.

The main technical requirement is that each agent is controllable in a small number K of steps. The communication graph is connected, possibly directed, and to keep the focus on the challenge of general nonlinear agents, we do not consider in this section time-varying graphs, noise, or network effects such as delays or packet dropouts. Then, under some additional technical assumptions related to optimization, our analysis ensures that the agents converge to a small region and then return to this region once every K steps. Under stronger regularity conditions including invertibility, we further guarantee the agents remain close to each other at *every* step. The near-optimality dimension β from Section 2.2 modulates the relationship between computation and approximation error. We analyze this dimension and build insight into the meaning of the assumptions for the entire class of linear systems, as well as for a nonlinear example; and provide simulation results for both cases.

Note that our approach is related to model-predictive control, which was applied to the consensus of linear agents by Keviczky and Johansson (2008) and Ferrari-Trecate et al. (2009). Like Qu et al. (2008), we obtain an approximately linear overall behavior of the agents, but while they use explicitly the model to perform feedback

linearization, we work with black-box models by exploiting the generality of OO.

6.1.1 Consensus problem statement

We consider a set of m agents with decoupled discrete-time nonlinear dynamics $x_{i,k+1} = f_i(x_{i,k}, u_{i,k})$, $i = 1, \ldots, m$. The dimensionality of x (the order of the dynamical system f), denoted n_x, is the same for every agent, and we have $x_i \in \mathbb{R}^{n_x}$. Similarly, for every agent u_i belongs to a compact set U. Define also the collective state $\boldsymbol{x} = [x_1^\top, \cdots, x_m^\top]^\top \in \mathbb{R}^{mn_x}$, containing the states of all the agents. The ideal goal is to achieve full-state consensus:

$$\lim_{k\to\infty} \left\| x_{i,k} - x_{j,k} \right\| = 0 \quad \forall i, j = 1, \ldots, m$$

where $\|\cdot\|$ is the Euclidean 2-norm, here as well as in the sequel. We employ the more pragmatic requirement of practical consensus, defined as the existence, for any given disagreement $\Delta > 0$, of a finite step k_0 so that for all $k \geq k_0$, $\left\| x_{i,k} - x_{j,k} \right\| \leq \Delta$ for all i, j.

An agent can receive information only from its neighbors on an interconnection graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$. The set of nodes $\mathscr{V} = \{1, \ldots, m\}$ represents the agents, and the edges $\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$ are the communication links. Denote by $\mathscr{N}_i = \{ j \mid (i, j) \in \mathscr{E} \}$ the set of neighbors of node i. The graph may be directed (for some edges $(i, j) \in \mathscr{E}$, $(j, i) \notin \mathscr{E}$), but we require that it is time-invariant and strongly connected. The latter property means a path exists between any pair of nodes i, j: a sequence of nodes $i = i_1, \ldots, i_T = j$ so that $(i_t, i_{t+1}) \in \mathscr{E}, 1 \leq t < T$.

Let $X_0 \subset \mathbb{R}^{n_x}$ be a user-defined, bounded set of interesting initial agent states, from which consensus might be required. Let $\mathbf{B}(x_c, R)$ be the smallest Euclidean ball containing X_0, which has center x_c and radius R. Then, define an enlarged set $\mathscr{X} = \mathbf{B}(x_c, 3R + \Delta^+)$, where $\Delta^+ > 0$ is an arbitrarily small constant. Finally, denote by $\tilde{u}_{i,k} = [u_{i,k}^\top, \ldots, u_{i,k+K-1}^\top]^\top$ a sequence of K actions for agent i starting at step k, and by $\tilde{f}_i(x_{i,k}, \tilde{u}_{i,k})$ the resulting state of the agent after applying $\tilde{u}_{i,k}$ (K steps later). We impose a controllability assumption on f_i in the set $\mathscr{X}$.

Assumption 6.1 (Controllability) *There exists a finite K such that, for any $x, x^* \in \mathscr{X}$ and any agent i, there exists a sequence $\tilde{u}_i \in U^K$ so that:*

$$\tilde{f}_i(x, \tilde{u}_i) = x^*$$

This type of controllability property is studied by e.g. (Jakubczyk and Sontag, 1990), where Lie-algebraic conditions are provided for nonlinear systems, including a characterization of the size of reachable sets. The controllable set is enlarged to $\mathscr{X}$ to deal with the errors made by the algorithm in some special, unfavorable initial conditions, such as when most agents are close together at one edge of X_0 and one agent is

diametrally opposite. In fact, for most initial conditions, we expect the agents to stay close to the convex hull of X_0.

Under this assumption and some other requirements related to the optimization algorithm, stated below, our results guarantee practical consensus at steps that are multiples of K. To guarantee the stronger result of practical consensus at every step, an additional assumption is imposed. Define an *invertibility* property of dynamics $\tilde{f}_i$, which requires that for every pair $(x,x^*) \in \mathscr{X}^2$ the sequence $\tilde{u}_i$ achieving $\tilde{f}_i(x,\tilde{u}_i) = x^*$ is unique. The *inverse* $\tilde{f}_i^{-1} : \mathscr{X}^2 \to U^K$ is defined as the mapping between state pairs and their corresponding action sequences.

Assumption 6.2 (Homogeneity and invertibility) *The multiagent system is homogeneous: $f_i = f_j =: f$, $\forall i,j$ with common dynamics f that are Lipschitz-continuous in $\mathscr{X} \times U$. Furthermore, the K-step dynamics $\tilde{f}$ has a unique inverse $\tilde{f}^{-1}$, which is also Lipschitz continuous in its domain $\mathscr{X}^2$.*

Invertibility conditions are an important topic in nonlinear system theory, and several types of conditions for discrete-time systems are provided e.g. by Grizzle (1993); Zheng and Evans (2002); Fliess (1992). Since the set of states $\mathscr{X}$ and the action space U were already taken compact, the Lipschitz conditions are not significantly more restrictive. Homogeneity (together with invertibility) is needed to guarantee the agents behave similarly for similar desired transitions, see Section 6.1.3 for a simulation illustrating this.

It is essential to note that the dynamics f_i are only available to the agents as black-box, simulation models, the inverse dynamics $\tilde{f}^{-1}$ (if they exist) are not available at all, and the sequences $\tilde{u}_i$ of Assumption 6.1 are unknown; indeed, the main task of the algorithm will be to approximate them.

6.1.2 Consensus approach and analysis

Our approach to the consensus problem in Section 6.1.1 works as follows. Once every K steps, each agent i computes a reference (desired) next state, which is a weighted average of the current states of its neighbors $\mathscr{N}_i$. Then, a sequence of actions is optimized with OO so that after K steps, the reference state is approached as closely as possible. All agents apply their sequences and the whole process is repeated. The main idea is therefore intuitive: by emulating a classical consensus algorithm (Olfati-Saber et al., 2007), an approximately linear overall agent behavior is obtained. The main novelty is using OO to find the controls, which only requires sampling a black-box model, without knowledge about its internal structure. The properties of OO tightly connect computation invested with approximation error. In our analysis, we focus on these computational aspects and on the impact of the error on the consensus guarantees.

Formally, at step k – a multiple of K – each agent i computes a reference state with:

$$\begin{gathered} x^*_{i,k+K} = \sum_{j=1}^{m} \alpha_{i,j} x_{j,k} \\ \text{where } \alpha_{i,j} \in [0,1] \ \forall i,j, \quad \textstyle\sum_{j=1}^{m} \alpha_{i,j} = 1 \ \forall i, \\ \alpha_{i,j} \neq 0 \text{ iff } i = j \text{ or } (i,j) \in \mathscr{E} \end{gathered} \tag{6.1}$$

Since each agent only uses information about its neighbors, these reference states can be computed in a decentralized way. Matrix $\boldsymbol{\alpha} = [\alpha_{i,j}]$ should be freely chosen to impose a desired consensus behavior. Several decentralized strategies to select it are given in (Olfati-Saber and Murray, 2004; Xiao and Boyd, 2004; Olshevsky and Tsitsiklis, 2009).

Then, each agent solves with DOO or SOO the optimal control problem:

$$\begin{gathered} \tilde{u}^*_{i,k} = \arg\max_{\tilde{u}_{i,k}} g_{i,k}(\tilde{u}_{i,k}) \\ \text{where } g_{i,k}(\tilde{u}_{i,k}) := -\left\| \tilde{f}_i(x_{i,k}, \tilde{u}_{i,k}) - x^*_{i,k+K} \right\| \end{gathered} \tag{6.2}$$

(recall the definitions of K-step actions and dynamics). This is a type of dynamic inversion control, see e.g. (Hunt and Meyer, 1997), and similar ideas are used among others in adaptive inverse control (Widrow and Walach, 2008). A near-optimal extended action $\widehat{\tilde{u}}_{i,k}$ is found, which will reach close to the reference state. The objective function $g_{i,k}(\tilde{u}_{i,k})$, together with the partitioning rule for the space $U^K =: \tilde{U}$, must satisfy the optimization Assumptions 2.2 and 2.3. For easy reference, we restate these assumptions in the context of consensus, dropping the subscript i,k for readability.

Assumption 6.3 *For all i and k, the objective function and the partitioning method of OO satisfy the following conditions:*

6.3.i There exists an optimum $\tilde{u}^*$ and a semimetric ℓ so that:

$$g(\tilde{u}^*) - g(\tilde{u}) \leq \ell(\tilde{u}, \tilde{u}^*) \quad \forall \tilde{u} \in \tilde{U} \tag{6.3}$$

6.3.ii There exist $c > 0$ and $\lambda \in (0,1)$ such that for any d, $\delta_{d,j} \leq c\lambda^d$ for all nodes j at depth d, where $\delta_{d,j} := \sup_{\tilde{u} \in \tilde{U}_{d,j}} \ell(\tilde{u}_{d,j}, \tilde{u})$.

6.3.iii There exists a constant μ such that any subset $\tilde{U}_{d,j}$ contains a ball with center $\tilde{u}_{d,j}$ and radius $\mu c\lambda^d$ in the semimetric ℓ.

6.3.iv A finite β exists.

Algorithm 6.1 Optimistic-optimization consensus at agent i

Input: row $\boldsymbol{\alpha}_i$ of $\boldsymbol{\alpha}$, budget n, OO algorithm to use, if DOO: ℓ, if SOO: $d_{\max}(t)$

$k \leftarrow 0$
loop
  measure $x_{i,k}$ and send it to $j \in \mathcal{N}_i$
  receive $x_{j,k}$ from $j \in \mathcal{N}_i$
  compute reference state $x^*_{i,k+K}$ using $\boldsymbol{\alpha}_i$
  $g_{i,k}(\tilde{u}_{i,k}) := -\left\| \tilde{f}_i(x_{i,k}, \tilde{u}_{i,k}) - x^*_{i,k+K} \right\|$
  if using DOO **then**
    $\widehat{\tilde{u}}_{i,k} \leftarrow \mathrm{DOO}(g_{i,k}, \ell, n, \text{partitioning of } U)$
  else if using SOO **then**
    $\widehat{\tilde{u}}_{i,k} \leftarrow \mathrm{SOO}(g_{i,k}, d_{\max}, n, \text{partitioning of } U)$
  end if
  apply $\widehat{\tilde{u}}_{i,k}$ in open loop
  $k \leftarrow k + K$
end loop

Algorithm 6.1 summarizes the overall procedure.

Most of the parameters of OO can usually be chosen in a standard way. The partitioning of U is typically defined as a collection of hyperboxes in a Kn_u-dimensional space, where n_u denotes the number of action variables. A large hyperbox contains the whole action space, and each node expansion corresponds to splitting the parent hyperbox into 2^{Kn_u} subboxes, in half along all dimensions. Another alternative is to split one dimension at a given depth, looping through the dimensions sequentially. In either case, the complexity of splitting is large (exponential in K when all dimensions are split), so a practical requirement of our method is that *K must be small*; the limit depends on the agent's computational power, but to give an idea, say that the product Kn_u should be less than 10.

For DOO, the semimetric ℓ is a less obvious choice. For some problems, its form may be available from prior knowledge. In general, defining ℓ can be difficult, in which case SOO should be used. In SOO, a default choice of $d_{\max}(t)$ is $\sqrt{t}$, expecting that a semimetric that yields a near-optimality dimension of 0 exists; if the optimization is expected to be difficult, a better schedule is t^a with small a, to increase the asymptotic convergence rate, see Proposition 2.1. We provide more insight about the semimetric and near-optimality dimension in the examples.

The main parameter is however the budget n. In particular, since simulating the nonlinear dynamics will usually dominate computation, the time complexity at each agent roughly linear in n.

Therefore, we start our analysis by investigating what is the required value for the budget n to achieve a uniformly small approximation error ε^*. By essentially

inverting the relationships in Proposition 2.1, we find the following budgets at agent i in collective state $\boldsymbol{x}$, where the optimization complexity is characterized by $\beta(i,\boldsymbol{x})$. In DOO, $n = O(\varepsilon^{*-\beta(i,\boldsymbol{x})})$, a power of $1/\varepsilon^*$, when $\beta(i,\boldsymbol{x})$ is positive. When $\beta(i,\boldsymbol{x}) = 0$, $n = O(\log(1/\varepsilon^*))$, logarithmic (much lower). In SOO, taking for simplicity $d_{\max}(t) = \sqrt{t}$, $n = O(\varepsilon^{*-2\beta(i,\boldsymbol{x})})$ when $\beta(i,\boldsymbol{x}) > 0$ and $n = O(\log^2(1/\varepsilon^*))$ when $\beta(i,\boldsymbol{x}) = 0$. So the complexity behaves similarly but is larger – the power of $1/\varepsilon^*$ is double (which can be improved by better choosing $d_{\max}$), while the logarithm is squared.

In practice, the agents will use the same n everywhere. In that case, any desired ε^* can be achieved by taking a sufficiently large n, which is related to the largest β (of course, this large n is conservative where β is smaller, leading to errors smaller than ε^* in those situations). This uniform error property is essential to our main results, given next.

Theorem 6.1 (Consensus every K steps) *Under Assumptions 6.1 and 6.3, for large enough n, Algorithm 6.1 achieves practical consensus at steps multiple of K.*

Proof: Denote by Λ the left eigenvector of $\boldsymbol{\alpha}$ associated with the eigenvalue 1 (which will always exist due to the structure of $\boldsymbol{\alpha}$), $\bar{x} = \Lambda^\top \boldsymbol{x}_0 \in \mathbb{R}^n_x$, and $\bar{\boldsymbol{x}} = [\bar{x},\ldots,\bar{x}]^\top$. We will first show (i) that $\exists t_0, \Delta$ so that for all $t \geq t_0$, $\|x_{i,tK} - \bar{x}\| \leq \Delta$ for all i, while assuming that the agents remain controllable. Then, it is proven that (ii) they actually do remain controllable.

(i) $\boldsymbol{\alpha}$ is a row stochastic matrix associated to a strongly connected graph. By iterating the ideal protocol $x^*_{i,k+K} = \sum_{j=1}^m \alpha_{i,j} x^*_{j,k}$ starting from $\boldsymbol{x}_0$, all agents would converge exponentially to $\bar{x}$. Next, the effect of the deviation of the real states from this ideal trajectory is analyzed. Let e be the largest eigenvalue of $\boldsymbol{\alpha}$ that is different from 1. Then, $0 < |e| < 1$ and:

$$\left\| \boldsymbol{x}^*_{(t+1)K} - \bar{\boldsymbol{x}} \right\| = |e| \cdot \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \tag{6.4}$$

Since each agent reaches its reference state with at most ε^* error:

$$\|\boldsymbol{x}_{tK} - \boldsymbol{x}^*_{tK}\| \leq \sqrt{m}\varepsilon^*, \forall t \tag{6.5}$$

Combining (6.5) and (6.4), for arbitrary $\Delta > 0$ and $\bar{e} \in (|e|, 1)$, Δ-practical consensus with a convergence rate $\bar{e}$ will be ensured. To this end, choose a large enough budget n to guarantee $\varepsilon^* \leq (\bar{e} - |e|)\Delta/\sqrt{m}$.

Assume first that the initial disagreement is larger than Δ. Then, for all $\boldsymbol{x}_{tK}$ such that $\|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| > \Delta$ one has:

$$\begin{aligned}
\left\|\boldsymbol{x}_{(t+1)K} - \bar{\boldsymbol{x}}\right\| &\leq \left\|\boldsymbol{x}_{(t+1)K} - \boldsymbol{x}^*_{(t+1)K}\right\| + \left\|\boldsymbol{x}^*_{(t+1)K} - \bar{\boldsymbol{x}}\right\| \\
&\leq \sqrt{m}\varepsilon^* + |e| \cdot \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \\
&\leq (\bar{e} - |e|)\Delta + |e| \cdot \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \\
&< \bar{e}\,\|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\|
\end{aligned}$$

This exponential decay ensures that after a finite $t_0 = \lceil \log_{\bar{e}} \Delta / \|\boldsymbol{x}_0 - \bar{\boldsymbol{x}}\| \rceil$, the distance to $\bar{x}$ will drop below Δ. Once this is true, i.e., if $\|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| < \Delta$, then:

$$\begin{aligned}\left\|\boldsymbol{x}_{(t+1)K} - \bar{\boldsymbol{x}}\right\| &\leq \left\|\boldsymbol{x}_{(t+1)K} - \boldsymbol{x}^*_{(t+1)K}\right\| + \left\|\boldsymbol{x}^*_{(t+1)K} - \bar{\boldsymbol{x}}\right\| \\ &\leq \sqrt{m}\varepsilon^* + |e| \cdot \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \\ &\leq (\bar{e} - |e|)\Delta + |e|\,\Delta = \bar{e}\Delta < \Delta\end{aligned}$$

so the state will remain at distance Δ from $\bar{x}$. If the initial disagreement is already below Δ, then $t_0 = 0$ and the derivation of the exponential decay is no longer needed.

(ii) To ensure controllability, the states of all the agents must remain inside set $\mathscr{X}$ at each step tK (recall that $X_0 \subseteq \mathbf{B}(x_c, R)$ and $\mathscr{X} = \mathbf{B}(x_c, 3R + \Delta^+)$). Define $\Delta_0 = \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|$, the *collective* initial disagreement, and $\Delta_0^{\mathrm{a}} = \max_i \|x_{i,0} - \bar{x}\|$, the *per-agent* initial disagreement. Take a desired collective disagreement of $\Delta \leq 2R$; part (i) allows imposing any $\Delta > 0$, and since $2R$ upper-bounds the diameter of X_0, a larger value makes little sense.

If $\Delta \geq \Delta_0$, then already, for any $t \geq 0$, $\|x_{i,tK} - \bar{x}\| \leq \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \leq \Delta$. Thus the agents remain in $\mathbf{B}(\bar{x}, \Delta)$ and since $\bar{x} \in \mathbf{B}(x_c, R)$, they remain in $\mathbf{B}(x_c, 3R) \subset \mathscr{X}$. Now if $\Delta < \Delta_0$, then for any t and i we have:

$$\begin{aligned}\left\|x_{i,(t+1)K} - \bar{x}\right\| &\leq \left\|x_{i,(t+1)K} - x^*_{i,(t+1)K}\right\| + \left\|x^*_{i,(t+1)K} - \bar{x}\right\| \\ &\leq \varepsilon^* + \max_j \left\|x_{j,tK} - \bar{x}\right\| \\ &\leq (t+1)\varepsilon^* + \Delta_0^{\mathrm{a}} \qquad (6.6)\end{aligned}$$

where the second inequality implies the third by induction. Furthermore, by part (i), as long as the collective disagreement is larger than Δ it decreases exponentially:

$$\|x_{i,tK} - \bar{x}\| \leq \|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \leq \bar{e}^t \Delta_0$$

This allows taking a finite t_1 so that $\bar{e}^{t_1}\Delta_0 \leq \max\{\Delta, \Delta_0^{\mathrm{a}}\}$, from which $\|x_{i,tK} - \bar{x}\| \leq \max\{\Delta, \Delta_0^{\mathrm{a}}\}\ \forall i, t \geq t_1$. Combining this with (6.6), we have $\|x_{i,tK} - \bar{x}\| \leq \max\{\Delta, t_1\varepsilon^* + \Delta_0^{\mathrm{a}}\}, \forall i, t \geq 0$. Therefore, finally imposing $\varepsilon^* \leq \Delta^+ / t_1$ and noticing that $\Delta_0^{\mathrm{a}} \leq 2R$, the states are guaranteed to remain in $\mathscr{X}$. ■

Theorem 6.2 (Consensus) *Under Assumptions 6.1–6.3, for sufficiently large n Algorithm 6.1 achieves practical consensus at every step.*

Proof: In Theorem 6.1, take $t \geq t_0$. Then $\forall i, j$, $\left\|x_{i,tK} - x_{j,tK}\right\| \leq 2\Delta$, due to the fact that $\|\boldsymbol{x}_{tK} - \bar{\boldsymbol{x}}\| \leq \Delta$ and the triangle inequality. Due to Assumption 6.2, we have $\tilde{u}_{i,tK} = \tilde{f}^{-1}(x_{i,tK}, x_{i,(t+1)K})$, $\tilde{u}_{j,tK} = \tilde{f}^{-1}(x_{j,tK}, x_{j,(t+1)K})$ and denoting the Lipschitz constant of $\tilde{f}^{-1}$ by L_{-1}:

$$\left\|\tilde{u}_{i,tK} - \tilde{u}_{j,tK}\right\| \leq L_{-1}\left(\left\|x_{i,tK} - x_{j,tK}\right\| + \left\|x_{i,(t+1)K} - x_{j,(t+1)K}\right\|\right) \leq 4L_{-1}\Delta$$

Then, at steps $tK+k$ with $k=1,\ldots,K-1$, we have:

$$\left\|u_{i,tK+k}-u_{j,tK+k}\right\| \leq \sqrt{n_u}\left\|\tilde{u}_{i,tK}-\tilde{u}_{j,tK}\right\|_\infty \leq 4\sqrt{n_u}L_{-1}\Delta$$

Denote $L'=4\sqrt{n_u}L_{-1}$, and L the Lipschitz constant of f. Finally, by a straightforward derivation we get:

$$\left\|x_{i,tK+k}-x_{j,tK+k}\right\| \leq (L^k+L'\sum_{k'=1}^{k}L^{k'})\Delta$$

which is the desired result. ■

From the proof of Theorem 6.1, a rather strong type of practical consensus is ensured, where agents exponentially approach and then remain inside a region of size Δ around $\bar{x}$ – but only at steps multiple of K. The size Δ can be controlled by ε^*, and so indirectly by the budget n, as seen in the discussion about the budget above. In-between these steps and above t_0K, Theorem 6.2 ensures, under the additional Assumption 6.2, a weaker practical consensus where agents are only close to each other and not necessarily to $\bar{x}$. Indeed, the region containing the states can travel in the state space, with the constraint that it must always return around $\bar{x}$ at each multiple of K. Note that the region may grow somewhat in-between the multiples of K, but its size remains proportional to Δ. Figure 6.1 illustrates these guarantees.

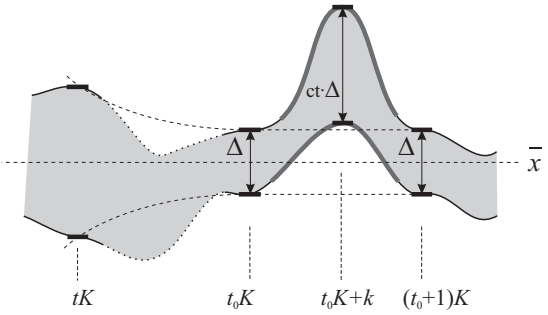

Figure 6.1: Symbolic illustration of consensus guarantees. The agent states will stay within the shaded area. Dotted contour: analysis does not constrain the states; solid black contour: guarantees of Theorem 6.1; thick gray contour: guarantees of Theorem 6.2.

6.1.3 Analysis and simulations in representative cases

We next exemplify linear agents as a special case of our framework, together with a type of nonlinear agents. Although the method targets nonlinear agents, understanding how it works in the familiar, linear case builds useful insight. For each type of agents, we analyze the assumptions and present simulation results. It must be emphasized that these examples are analyzed here in detail for illustrative purposes, using 'white-box', full mathematical models of the agent dynamics. Indeed,

the whole point of our algorithm is to avoid the need to use such models; instead, they are a black box supplied to the algorithm, and it is precisely this which lends the approach its generality. In practice, the user simply chooses a sufficiently large $\tilde{U}$ and—provided of course the assumptions hold—does not need to worry about the other details.

Linear agents. Consider an agent with linear controllable dynamics:

$$x_{k+1} = Ax_k + Bu_k$$

Different agents may have different dynamics if we are only interested in consensus every K steps; otherwise, agents must be homogeneous due to Assumption 6.2. In both cases, we drop the agent index since the development is generally valid for any linear dynamics.

The set of interesting initial states considered is $X_0 = \mathbf{B}(0,R)$. It follows that $\mathscr{X} = \mathbf{B}(0,3R+\Delta^+)$ for an arbitrary $\Delta^+ > 0$. Denote $R_{\mathscr{X}} = 3R+\Delta^+$. For now, K is taken equal to the order n of the dynamics, and the K-step dynamics are:

$$x_{k+K} = \tilde{A}x_k + \tilde{B}\tilde{u}_k = A^K + [A^{K-1}B, A^{K-2}B, \ldots, B]\tilde{u}_k$$

Since the system is controllable $\tilde{B}$ is full row rank, and for any x_k at least one $\tilde{u}_k^*$ exists that reaches an arbitrary x_{k+K}^* (there may be more):

$$\tilde{u}_k^* = \tilde{B}^{\ddagger}(x_{k+K}^* - \tilde{A}x_k) \tag{6.7}$$

where $\ddagger$ denotes left pseudoinverse. Then $\left\|\tilde{u}_k^*\right\| \leq \left\|\tilde{B}^{\ddagger}\right\| (1+\left\|\tilde{A}\right\|) \|x_k\|$, and an action space $\tilde{U}$ including a ball of radius $R_U = \left\|\tilde{B}^{\ddagger}\right\| (1+\left\|\tilde{A}\right\|) R_{\mathscr{X}}$ ensures Assumption 6.1 ($\|\cdot\|$ denotes the induced norm for matrices).

The optimal control at step k must maximize:

$$g(\tilde{u}_k) = -\left\|x_{k+K} - x_{k+K}^*\right\| = -\left\|\tilde{B}\tilde{u}_k - \tilde{B}\tilde{u}_k^*\right\|$$

where $\tilde{u}_k^*$ is the (unknown) action that exactly reaches x_{k+K}^*. We have:

$$g(\tilde{u}_k^*) - g(\tilde{u}_k) = 0 + \left\|\tilde{B}\tilde{u}_k - \tilde{B}\tilde{u}_k^*\right\| \leq \left\|\tilde{B}\right\| \|\tilde{u}_k - \tilde{u}_k^*\|$$

So, a semimetric $\ell(\tilde{u},\tilde{u}') = \overline{\gamma}\|\tilde{u}-\tilde{u}'\|$ with $\overline{\gamma} \geq \left\|\tilde{B}\right\|$ (in fact a metric) ensures the weak-Lipschitz property Assumption 6.3.i. Note that $g(\tilde{u}_k^*) = 0$, and also that $\left\|\tilde{B}\right\|$ plays the role of Lipschitz constant for g. If the prior knowledge that ℓ is a good metric in the linear problem is available, then DOO should be applied with this metric. Otherwise, SOO should be used. Further, it is clear that if we take a hyperbox U^K and create a natural, exponential partitioning by recursively splitting it into smaller hyperboxes along each dimension (2^{Kn_u} new hyperboxes at each step), Assumptions 6.3.ii and 6.3.iii hold.

Next, the near-optimality dimension β is investigated (omitting the proof). If $n_u K > n_x$, there are multiple optimal solutions and β is upper-bounded by $\varphi = n_u K - n_x$, the number of extra degrees of freedom in the solution (6.7). This can intuitively be understood as OO having to uniformly explore the φ-dimensional subspace of equally good near-optimal solutions, and provides yet another incentive for having K as small as possible. Since β is bounded, Assumption 6.3.iv holds.

So far, we only imposed Assumption 6.1. Further requiring Assumption 6.2, invertibility in K steps, boils down to $\operatorname{rank}(\tilde{B}) = n_u K$ (Sain and Massey, 1969) (note that the direct and inverse dynamics are Lipschitz). Since due to controllability $\operatorname{rank}(\tilde{B}) = n_x$, we must have $K = n_x/n_u$, and controllability must hold in $K < n_x$ steps. In effect, all this leads to a square, invertible $\tilde{B}$. So, the condition is significantly more restrictive than Assumption 6.1. The benefit is that due to invertibility, β is now 0, and the optimization problem is easy to solve.

Moving on to our simulation results, consider the classical example of double-integrator agents (Olfati-Saber, 2006):

$$x^{\mathrm{p}}_{i,k+1} = x^{\mathrm{p}}_{i,k} + T_{\mathrm{s}} x^{\mathrm{p}}_{i,k}, \quad x^{\mathrm{v}}_{i,k+1} = x^{\mathrm{v}}_{i,k} + T_{\mathrm{s}} u_{ik}$$

where x^{p} is the position, x^{v} the velocity, the input is u_{ik}, and Euler discretization is used with sampling time $T_{\mathrm{s}} = 0.5\,\mathrm{s}$. The system is controllable in $K = 2$ steps (note that $n = Km$, leading to $\beta = 0$).

Five agents are considered, having the (undirected) connection graph from Figure 6.2 (top), initial positions $0,1,2,3,4$, and initial velocities $-3,-1,0,1,3$. In all our experiments, the neighbor weights are set with the method of Olfati-Saber and Murray (2004): the agents first agree on a value $\alpha' \in (0, \frac{1}{\max_i |\mathcal{N}_i|})$, and then agent i sets $\alpha_{i,i} = 1 - \alpha' |\mathcal{N}_i|$ and $\alpha_{i,j} = \alpha'$, $\forall j \in \mathcal{N}_i$. We set $\alpha' = 0.2$. The action space is taken large enough to ensure controllability (specifically, $\tilde{U} = [-200, 200]^2$). To characterize performance in each experiment, an inter-agent disagreement is computed at every step: $\Delta_k = \sum_{i<j} \left\| x_{i,k} - x_{j,k} \right\|$, and the average of Δ_k over the entire trajectory is reported.

We will study the performance for varying budget $n = 15, 25, 50, 75, 100, 200, \ldots,$ 700, in three cases. First, DOO is applied, using the 'ideal' metric with constant $\overline{\gamma} = \left\| \tilde{B} \right\|$. In the second, more realistic case, $\overline{\gamma}$ is overestimated as $4 \left\| \tilde{B} \right\|$. Third, the metric is no longer assumed known and SOO is applied. As seen in Figure 6.2 (bottom left), performance largely increases with n for all optimization methods, although monotonicity is not guaranteed. Moreover, as expected, DOO with the ideal metric performs best, obtaining near-optimal performance for small budgets. Since it does not use the metric, SOO lags slightly behind, but quickly reaches near-optimal performance as well. DOO with the overestimated metric performs poorly, falling behind SOO. Figure 6.2 (bottom right) shows a trajectory for SOO and $n = 200$. Practical consensus is obtained in under 20 s. Since the desired agent state $x^*_{i,k+2}$ is

designed without knowing the structure of the dynamics, the resulting trajectory is not smooth.

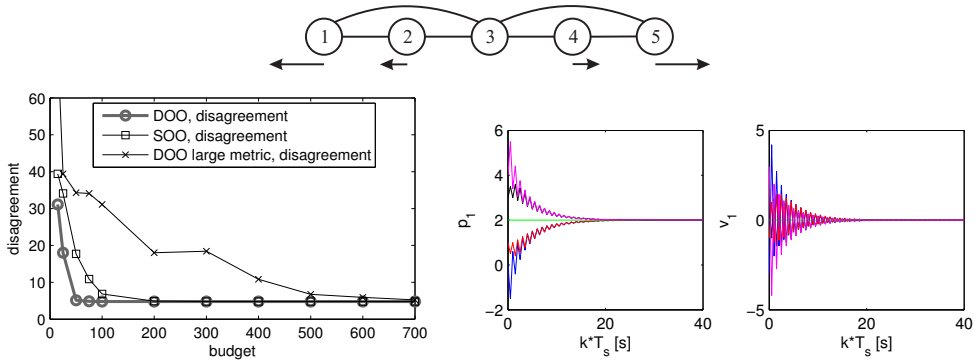

Figure 6.2: Top: Connection graph for double integrators, with initial velocities symbolized by arrows. Bottom left: Results for varying N. Bottom right: A representative trajectory.

Hammerstein nonlinear agents. To illustrate the framework in the nonlinear case, we consider an example where linear dynamics are affected by an algebraic input nonlinearity – a Hammerstein system, taken from the literature (Ding and Chen, 2005). The example is adapted to our framework by transforming it into state-space form and eliminating the noise, leading to:

$$\begin{aligned} x_{k+1} &= Ax_k + Bz_k = \begin{bmatrix} 1.6 & -0.8 \\ 1 & 0 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} z_k \\ z_k &= g(u_k) = u_k + 0.5u_k^2 + 0.25u_k^3 \end{aligned} \tag{6.8}$$

where again we do not include the agent index.

Like before, $X_0 = \mathbf{B}(0,R)$, leading to $\mathscr{X} = \mathbf{B}(0,R_{\mathscr{X}})$ with $R_{\mathscr{X}} = 3R + \Delta^+$. Note that the input nonlinearity g is strictly increasing and bijective. From the 2-step dynamics:

$$x_{k+2} = A^2 x_k + \tilde{B}\tilde{z}_k = A^2 x_k + [AB, B][z_k, z_{k+1}]^\top$$

with a full-rank $\tilde{B}$, the linear system is controllable in $K = 2$ steps, and due to the bijective nonlinearity the same remains true of the nonlinear system. Furthermore, the inverse dynamics are:

$$\tilde{u}_k = [g^{-1}(z_k), g^{-1}(z_{k+1})]^\top, \tilde{z}_k = \tilde{B}^{-1}(x^*_{k+2} - A^2 x_k)$$

From the fact that $0.5|u| \leq |g(u)|$, we have $|u| \leq 2\|\tilde{z}_k\| \leq 2\left\|\tilde{B}^{-1}\right\|(1 + \left\|A^2\right\|)R_{\mathscr{X}}$, where the second part is taken from the linear case above. So, a compact input box

of this size is sufficient to make the system controllable in $\mathscr{X}$. Since the direct and inverse dynamics are also Lipschitz, Assumptions 6.1 and 6.2 have been verified.

The objective function of the optimal control problem is $g(\tilde{u}_k) = -\left\|x_{k+2} - x^*_{k+2}\right\|$. To find a semimetric ℓ satisfying Assumption 6.3.i, note that:

$$\begin{aligned} g(\tilde{u}^*_k) - g(\tilde{u}_k) &= \left\|x_{k+2} - x^*_{k+2}\right\| \leq \left\|\tilde{B}(\tilde{z}_k - \tilde{z}^*_k)\right\| \\ &\leq \left\|\tilde{B}\right\| \left\|[g(u_k) - g(u^*_k), g(u_{k+1}) - g(u^*_{k+1})]^\top\right\|_1 \\ &\leq (1 + R_{\mathscr{X}} + 0.75R^2_{\mathscr{X}}) \left\|\tilde{B}\right\| \|\tilde{u}_k - \tilde{u}^*_k\|_1 \end{aligned} \tag{6.9}$$

where the last step is due to:

$$\begin{aligned} |g(u) - g(u^*)| &= |u - u^* + 0.5(u^2 - u^{*2}) + 0.25(u^3 - u^{*3})| \\ &\leq |u - u^*| \cdot |1 + 0.5(u + u^*) + 0.25(u^2 + u^{*2} + uu^*)| \\ &\leq (1 + R_{\mathscr{X}} + 0.75R^2_{\mathscr{X}}) |u - u^*| \end{aligned}$$

Therefore, an appropriate metric is $\ell(\tilde{u}_k, \tilde{u}^*_k) = \overline{\gamma}\left\|\tilde{u}_k - \tilde{u}^*_k\right\|_1$ with a 'Lipschitz constant' $\overline{\gamma}$ greater than the multiplying factor above. Noticing that under this metric the natural partitioning into squares satisfies the remaining conditions 6.3.ii and 6.3.iii, Assumption 6.3 is validated.

Finally, to find the near-optimality dimension β, we study the sets:

$$\tilde{U}_\varepsilon = \{\tilde{u} \,|\, g(\tilde{u}^*) - g(\tilde{u}) \leq \varepsilon\} \subseteq \left\{\tilde{u} \,\middle|\, \underline{\gamma}\|\tilde{u}_k - \tilde{u}^*_k\|_1 \leq \varepsilon\right\}$$

The inclusion holds due to $g(\tilde{u}^*) - g(\tilde{u}) = \left\|\tilde{B}(\tilde{z}_k - \tilde{z}^*_k)\right\| \geq \sqrt{e_{\min}(\tilde{B}^\top \tilde{B})/2}\left\|\tilde{z}_k - \tilde{z}^*_k\right\|_1$ with $e_{\min}$ denoting the smallest eigenvalue. Further, by the monotonicity of g, around any u^* we have $|z - z^*| = |g(u) - g(u^*)| \geq \alpha |u - u^*|$ with some positive α. Thus, $\tilde{U}_\varepsilon$ is included in a ℓ-ball of radius proportional to ε, and therefore the packing number is constant, leading to a near-optimality dimension of 0: Assumption 6.3.iv is satisfied and the optimization problem is easy to solve.

To illustrate Hammerstein systems in simulation, consider five agents with dynamics (6.8) that are connected with the directed graph shown at the top-left of Figure 6.3. We apply SOO consensus with $n = 200$ and neighbor weight $\alpha = 0.2$, from uniformly random initial states $x_{i,0} \in [-5, 5]^2$ (the action space is then taken sufficiently large, $\tilde{U} = [-300, 300]^2$). Figure 6.3 (bottom) shows the resulting trajectories of the agents. The algorithm easily deals with these nonlinear dynamics. Note that, unlike for the double integrator, here the consensus state is not an equilibrium, so the states no longer reach a *constant* ball around it; instead, they synchronously travel in the state space as predicted by the analysis, see Figure 6.3 (top-right) and again Figure 6.1.

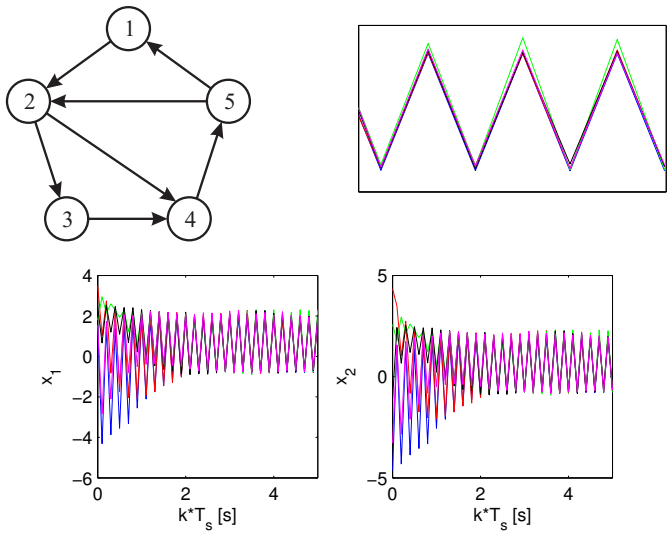

Figure 6.3: Consensus of Hammerstein systems. Top-left: Communication graph. Bottom: State trajectories (the sampling time is taken 0.1 s). Top-right: zoom-in on the last portion of the x_2 trajectory; note the agents are synchronized.

6.2 Optimistic planning for flocking

An important limitation of the OO consensus method above is that it works in increments of a small number K of steps. A natural extension to large, possibly unknown K is to apply long-horizon optimistic *planning*.

We introduce such a method in this section, focusing on a generalized version of the flocking problem, in which agreement is sought for a subset of agent variables while other variables define the interconnection topology between the agents. These two subsets may represent e.g. velocities and positions. The communication connections between agents are based on a proximity relationship, in which a connection is active when the agents are closer than some threshold in terms of the connectivity variables. A controllability property is imposed that, for any connected state, roughly requires the existence of an input sequence which preserves connectivity. We define agent reward functions with separate agreement and connectivity components, and use OP for deterministic systems (OPD, see Section 2.3) at the agents to find control actions. Our main analytical result shows that if the connectivity rewards are sufficiently large, the algorithm will preserve the interconnection topology. In interesting cases, the computational complexity of the flocking problem is not larger than if the agent would solve the agreement-only problem.

The theoretical algorithm is restrictive in still requiring to know, like OO consensus, the length of action sequences satisfying the controllability property. We therefore also provide a practical algorithm variant which does not use this knowledge, and validate it in simulation to nonholonomic agents and robot arms (Mei et al., 2011). In the second problem we illustrate that despite our focus on flocking, the method also works in the full-state consensus case.

The main advantage of the OP approach compared to classical methods is that, like the OO-based method, it is agnostic to the specific agent dynamics, and so it works uniformly for general nonlinear agents. In particular, our analysis shows that when a solution that preserves the topology exists (in a sense that will be formalized later), then irrespective of the details of the dynamics the algorithm will indeed maintain the topology. Existing topology preservation results are focused on specific types of agents, mainly linear (Zavlanos and Pappas, 2008; Fiacchini and Morarescu, 2014; Bullo et al., 2009, Ch. 4), or sometimes nonlinear as in e.g. (Zhu et al., 2013) where the weaker requirement of connectivity is considered. Our practical flocking algorithm exhibits the same generality, whereas existing methods exploit the structure of the specific dynamics targeted to derive predefined control laws, e.g. for linear double integrators (Olfati-Saber, 2006), agents with nonlinear acceleration dynamics (Su et al., 2011; Zhou et al., 2012), or nonholonomic robots (Tanner et al., 2005; Zhu et al., 2013).

6.2.1 Flocking problem statement

The problem we consider is closely related to that in Section 6.1.1, but it is more general in several respects. First, since we deal with flocking, agreement may only be sought on some of the variables, whereas in Section 6.1.1 the entire state vector was always considered. Other variables define the interconnection structure, so that the communication graph is now time-varying, whereas before it was fixed. However, communication links are symmetric so the graph is undirected. Further, our algorithmic tools and analytical goals are different, so we make different assumptions. For completeness, we fully formalize these problem characteristics below.

The m agents have decoupled nonlinear dynamics $x_{i,k+1} = f_i(x_{i,k}, u_{i,k})$, for $i = 1, \ldots, m$. The agents can be heterogeneous: they can have different dynamics and state or input dimensionality. An agent only receives information from its neighbors on a possibly time-varying interconnection graph $\mathscr{G}_k = (\mathscr{V}, \mathscr{E}_k)$, with agents in the vertices $\mathscr{V} = \{1, \ldots, m\}$ and communication links (edges) $\mathscr{E}_k \subseteq \mathscr{V} \times \mathscr{V}$. The time-varying set of neighbors of node i at step k is $\mathscr{N}_{i,k} = \{ j \mid (i,j) \in \mathscr{E}_k \}$.

The ideal objective is to achieve:

$$\lim_{k \to \infty} \left\| x^{\mathrm{a}}_{i,k} - x^{\mathrm{a}}_{j,k} \right\| = 0 \quad \forall i, j = 1, \ldots, m$$

where x^{a} selects only those state variables for which *agreement* is desired, and $\|\cdot\|$ denotes an arbitrary norm. We require of course that the selection produces a vector with the same dimensionality for all agents. Usually, x^{a} and x^{c} do not overlap, being e.g., the agent's velocity and position (Olfati-Saber, 2006), so that velocities must be synchronized under communication constraints dependent on the position. Specifically, we consider the case where a link is active when the *connectivity* states of two agents are close:

$$\mathscr{E}_k = \left\{ (i,j) \;\middle|\; i \neq j, \left\| x^{\mathrm{c}}_{i,k} - x^{\mathrm{c}}_{j,k} \right\| \leq R \right\} \tag{6.10}$$

For example when x^{c} is a position this corresponds to the agents being physically closer than some transmission range R. The full-state consensus problem of Section 6.1 is obtained as a special case of the generalized flocking setting, when all agents have the same state dimension, $x^{\mathrm{a}} = x$, x^{c} is empty, and $\mathscr{E}_k = \mathscr{E}$ (a fixed communication graph). While the OP technique can be applied to this case, as will be illustrated in the experiments, in the analytical development we will focus on the flocking problem, where the communication network varies.

Since the agents will use OPD, which works for discrete actions, we impose a discrete-action requirement for all agents.

Assumption 6.4 *Agent actions are discretized:* $u_i \in \{U_i\}$ *with* $|U_i| = M_i$.

Other authors showed interest in multiagent coordination with discretized actions, e.g. (De Persis and Frasca, 2013).

Denote $\boldsymbol{u}_{i,K} = (u_{i,0}, u_{i,1}, \ldots, u_{i,K-1}) \in U_i^K$ a sequence of K actions of agent i, and $\tilde{f}_i(x_i, \boldsymbol{u}_{i,K})$ the result of applying this sequence: the agent's state after K steps, with $\tilde{f}_i$ the extended dynamics. Note we use the sequence notation $\boldsymbol{u}$ from OPD, rather than $\tilde{u}$ from OO consensus, since we will later use OPD to find the sequences.

Assumption 6.5 *There exists K so that for any agent i, and any states x_i, x_j, $\forall j \in \mathcal{N}_{i,k}$ so that $\left\|x_i^{\mathrm{c}} - x_j^{\mathrm{c}}\right\| \leq R$, there exists some sequence $\boldsymbol{u}_{i,K}$ so that $\left\|\tilde{f}_i^{\mathrm{c}}(x_i, \boldsymbol{u}_{i,K}) - x_j^{\mathrm{c}}\right\| \leq R$, $\forall j \in \mathcal{N}_{i,k}$.*

This is a feasibility assumption: it is difficult to preserve the topology without requiring such a condition. The condition simply means that for any joint state of the system in which an agent is connected to some neighbors, this agent has an action sequence by which it is again connected after K steps, if its neighbors do not move. So if the assumption does not hold and the problem is such that the neighbors do stay still, the agent will indeed lose some connections and topology cannot be preserved. Of course, in general the neighbors will move, but as we will show Assumption 6.5 is nevertheless sufficient to ensure connectivity.

In Section 6.1 our analytical goal was to prove consensus, whereas here we are interested in connectivity. So our requirements were stronger there, requiring that the control is able to move the agent between *any* two arbitrary states in a bounded region, see Assumption 6.1. With a sufficiently fine action discretization, such an assumption would locally imply Assumption 6.5.

When making the assumption, we could also use the following definition for the links:

$$\mathscr{E}_k = \{(i,j) | i \neq j, \left\|x_{i,k}^{\mathrm{c}} - x_{j,k}^{\mathrm{c}}\right\| \leq R, \text{ and if } k > 0, (i,j) \in \mathscr{E}_{k-1}\} \tag{6.11}$$

so that the agents never gain new neighbors, and only need to stay connected to their initial neighbors. The analysis will also hold in this case, which is important because with (6.10), as k grows many or all the agents may become interconnected. For simplicity we use (6.10) in the sequel.

6.2.2 Flocking approach and analysis

The OP-based approach to the flocking problem in Section 6.2.1 works as follows. Similarly to OO consensus, at every time step k, a local optimal control problem is defined for each agent i, using information locally available to it. The goal in this problem is to align the agreement states x^{a} with those of the neighbors $\mathcal{N}_{i,k}$, while maintaining the connection topology by staying close to them in terms of x^{c}. OPD is used to near-optimally solve this control problem, and an initial subsequence of the sequence returned is applied by the agent. Then the system evolves, and the procedure is applied again, for the new states and possibly changed graph.

To construct its optimal control problem, each agent needs the predicted behavior of its neighbors. Here, agents will exchange the predicted state sequences resulting from the near-optimal action sequences returned by OPD. Because the agents must act at the same time, how they exchange predictions is nontrivial. If predictions do not match, a *coordination* problem may arise where mismatching actions are applied. Coordination is a difficult challenge in multi-agent systems and is typically solved in model-predictive control by explicit, iterative negotiation over successive local solutions, e.g. (Negenborn et al., 2008). However, it is unlikely that the agents can computationally afford to repeatedly communicate and reoptimize their solutions at every step. Thus we adopt a sequential communication procedure in which agents optimize once per step, similar to the procedure for distributed MPC in (Liu et al., 2010). We show below that connectivity can be guaranteed despite this one-shot solution.

To implement the sequential procedure, each agent needs to know its index i as well as the indices of its neighbors. One way to ensure this is an initial, centralized assignment of indices to the agents. Agent i waits until the neighbors j with $j < i$ have solved their local optimal control problems and found their predicted state sequences. These agents communicate their predictions to i. For $j > i$, agent i constructs other predictions as described later. Agent i optimizes its own behavior while coordinating with the predictions. It then sends its own, newly computed prediction to neighbors $j > i$.

Next, we formalize the approach. Denote quantities that depend on the time step by superscript k. Then, the planner of some agent i returns at step k an action sequence denoted $\boldsymbol{u}_i^k = (u_{i,0}^k, u_{i,1}^k, ..., u_{i,d-1}^k)$, which leads to predicted state sequence $\boldsymbol{x}_i^k = (x_{i,0}^k, x_{i,1}^k, \ldots, x_{i,d}^k)$. Here $x_{i,0}^k = x_{i,k}$ is the state measurement, and the other states are predictions. Sequences found at different time steps may have different actions at corresponding positions (e.g., $u_{i,1}^k$ may be different from $u_{i,0}^{k+1}$ even though they correspond to the same time index, $k+1 = (k+1)+0$).

Consider now a specific agent i. At every step k, it receives the states $x_{j,k}$ of its neighbors $j \in \mathcal{N}_{i,k}$. For neighbors $j \in \mathcal{N}_{i,k},\ j < i$, it directly receives their prediction at k and uses this as an estimation of their future behavior: $\widehat{\boldsymbol{x}}_j^{i,k} = (\widehat{x}_{j,0}^{i,k}, \widehat{x}_{j,1}^{i,k}, \ldots) = \boldsymbol{x}_j^k$. For $j \in \mathcal{N}_{i,k},\ j > i$, updated predictions are not available, instead a different prediction $\widehat{\boldsymbol{x}}_j^{i,k}$ is formed in a way that we specify later. We add i to the superscript to highlight that the predictions are from the point of view of agent i.

When OPD (Algorithm 2.3) is called, we internally relabel time k to 0, so that indices/depths d in Section 2.3 are relative to k. The local optimal control problem of agent i is defined using the reward function:

$$\rho_{i,d}^k(\boldsymbol{u}_{i,d}) = (1-\Lambda)\Delta_{i,d}^k(\boldsymbol{u}_{i,d}) + \Lambda\Gamma_{i,d}^k(\boldsymbol{u}_{i,d}) \tag{6.12}$$

where $\rho_{i,d}^k : U^d \to [0,1]$ gives the reward after d steps, $\Delta_{i,d}^k : U_i^d \to [0,1]$ rewards the alignment between agreement states, and $\Gamma_{i,d}^k : U_i^d \to [0,1]$ rewards the preservation

of neighbor connections, with $\Lambda \in (0,1)$ weighing the relative importance of these terms. The following value function that must be maximized is therefore:

$$v_i^k(\boldsymbol{u}_{i,\infty}) = \sum_{d=0}^{\infty} \gamma^d \rho_{i,d+1}^k(\boldsymbol{u}_{d+1}) \tag{6.13}$$

and we denote by v^{*k}_i the maximal value. So the rewards are *time-varying*, whereas OPD and its analysis were originally developed for time-invariant reward functions (Hren and Munos, 2008; Munos, 2014), see (2.4). However, this fact is not used in the development and analysis of OPD, which therefore entirely carries over to the time-varying case explained here.

Typically, $\Lambda \geq 1-\Lambda$ so that connectivity is given priority. Both Δ and Γ may use the predictions $\widehat{\boldsymbol{x}}_j^{i,k}$. Note that d may exceed the length of the available predictions; when that happens the predictions are heuristically kept constant at the last value available.

In the implementation, if the agents have their neighbors' models, they could also exchange predicted action sequences instead of states. Since actions are discrete and states usually continuous, this saves some bandwidth at the cost of extra computation to resimulate the neighbor's transitions up to the prediction length. In any case, it should be noted that agents do *not* optimize over the actions of their neighbors, so complexity does not directly scale with the number of neighbors.

So far, we have deliberately left open the specific form of the rewards and predictions for neighbors $j > i$. Next, we instantiate them in a theoretical algorithm for which we guarantee the preservation of the interconnection topology and certain computational properties. However, this theoretical variant has shortcomings, so we will additionally present a different instantiation which is more suitable in practice and which we later show works well in experiments.

A theoretical algorithm with guaranteed topology preservation. Our aim at this point is to exploit Assumption 6.5 to derive an algorithm that preserves the communication connections. We first develop the flocking protocol for each agent, shown as Algorithm 6.2. Our analysis proceeds by showing that, if sequences preserving the connections exist at a given step, the rewards can be designed to ensure that the algorithm will indeed find one such sequence (Lemma 6.3). This property is then used to prove topology preservation in closed loop, in Theorem 6.4. Finally, Theorem 6.5 shows an interesting computational property of the algorithm: under certain conditions the extra connectivity reward does not increase the complexity from the case where only agreement would be required.

Define a prediction for agents $j > i$ held constant to the latest exchanged state, $\widehat{\boldsymbol{x}}_j^{i,k} = (x_{j,k}, x_{j,k}, \dots)$. Then, the connectivity reward for agent i is an indicator function that becomes 0 only if agent i breaks connectivity with some neighbor(s) after K

steps:

$$\Gamma^k_{i,d}(\boldsymbol{u}_{i,d}) = \begin{cases} 0 & \text{if } d = K \text{ and} \\ & \exists j \in \mathscr{N}_{i,k}, \left\| \widehat{x}^{i,k,\mathrm{c}}_{j,d} - x^{k,\mathrm{c}}_{i,d} \right\| > R \\ 1 & \text{otherwise} \end{cases} \tag{6.14}$$

The agreement reward is left general, but to fix ideas, it could be for instance:

$$\Delta^k_{i,d}(\boldsymbol{u}_{i,d}) = 1 - \frac{1}{|\mathscr{N}_{i,k}|} \sum_{j \in \mathscr{N}_{i,k}} \min\{ \left\| \widehat{x}^{i,k,\mathrm{a}}_{j,d} - x^{k,\mathrm{a}}_{i,d} \right\|, 1 \} \tag{6.15}$$

where the distance measure $\|\cdot\|$ (which may be a norm or more general) is properly weighted to be sensitive to the relevant regions of x^{a}. Then, the agents always apply in open loop the first K actions from their computed sequences, after which they close the loop, measure the state, and repeat the procedure, see Algorithm 6.2.

Algorithm 6.2 Optimistic-planning flocking at agent i – theoretical variant.

set initial prediction $\boldsymbol{x}_i^{-1}$ to an empty sequence
for $t = 0, 1, 2, \ldots$ **do**
  current step is $k \leftarrow tK$
  exchange state at k with all neighbors $j \in \mathscr{N}_{i,k}$
  send $\boldsymbol{x}_i^{k-1}$ to $j < i$
  wait to receive new predictions $\widehat{\boldsymbol{x}}_j^{i,k}$ from all $j < i$
  form predictions $\widehat{\boldsymbol{x}}_j^{i,k}$ for $j > i$
  run OPD with (6.12) and (6.14), obtaining $\boldsymbol{u}_i^k$ and $\boldsymbol{x}_i^k$
  send $\boldsymbol{x}_i^k$ to $j > i$
  execute K actions $u^k_{i,0}, \ldots, u^k_{i,K-1}$ in open loop
end for

The reader may wonder why we do not simply redefine the optimal control problem in terms of the multistep dynamics $\tilde{f}_i$. The answer is that this would introduce exponential complexity in K: instead of M_i actions, we would have M_i^K, and this would also be the number of children created with each node expansion in OPD. In contrast, applying OPD directly to the 1-step problem leads to significantly decreased computation – in some cases no more than solving a 1-step problem without connectivity constraints, as shown in Theorem 6.5 below.

Moving on to the analysis now, we first show that when it is possible, each agent preserves connectivity with respect to the predicted states of its neighbors.

Lemma 6.3 *Let Assumption 6.4 hold. Take* $\Lambda \geq \frac{1/(1-\gamma)+\varepsilon}{1/(1-\gamma)+\gamma^{K-1}}$ *for some* $\varepsilon \in (0, \gamma^{K-1})$. *Assume that for any agent i, there exists a sequence* $\boldsymbol{u}_{i,K}$ *that preserves connectivity with the neighbors at step K, i.e.* $\Gamma^k_{i,K}(\boldsymbol{u}_{i,K}) = 1$. *Then, given a sufficiently large budget n, the solution returned by OPD contains at least K actions and does indeed preserve connectivity.*

Proof: The value of a solution that preserves connectivity at step K is at least $v_1 = \frac{\Lambda}{1-\gamma}$, while for a solution that does not it is at most $v_2 = \frac{1}{1-\gamma} - \Lambda\gamma^{K-1}$, since the Λ reward is not received at step K. We have:

$$v_1 - v_2 \geq \frac{\Lambda}{1-\gamma} - \frac{1}{1-\gamma} + \Lambda\gamma^{K-1} \geq \varepsilon$$

obtained by replacing the value of Λ. Therefore, the optimal value satisfies $v^* \geq v_1$, and as soon as the OPD reaches depth $d+1$ for which $\frac{\gamma^d}{1-\gamma} < \varepsilon$, due to Theorem 2.2(i) it will return a solution that is closer than ε to v^* and which therefore preserves connectivity. For sufficiently large n, depth $\max\{d,K\}+1$ is reached which guarantees both that the precision is ensured and that the length of the solution at least K. The proof is complete. ■

Putting the local guarantees together, we have topology preservation for the entire system, as follows.

Theorem 6.4 *Take Λ and n as in Lemma 6.3, then under Assumptions 6.4–6.5 and if the graph is initially connected, Algorithm 6.2 preserves the connections at any step $k = tK$.*

Proof: The intuition is very simple: each agent i will move so as to preserve connectivity with the *previous* state of any neighbor $j > i$, and then in turn j will move while staying connected with the *updated* state of i, which is what is required. However, since Assumption 6.5 requires connectivity to hold globally for all neighbors, the formal proof is somewhat technical.

To make it easier to understand, define relation $\mathscr{R}(i, j_1, \ldots, j_{N_i})$, where indices j_l are all the neighbors $\mathscr{N}_{i,k}$ at k sorted in ascending order, and $N_i = |\mathscr{N}_{i,k}|$. This relation means that i is connected with all j_l, i.e. $\left\| x^{\mathrm{c}}_{i,k} - x^{\mathrm{c}}_{j_l,k} \right\| \leq R$, for $l = 1, \ldots, N_i$. When some agents have superscript '+' in the relation, this means that the relation holds with their *updated* states after K steps.

Assume the agents are connected via edges $\mathscr{E}_k$ at step k, a multiple of K. We will show by induction that $\mathscr{R}(i^+, j_1^+, \ldots, j_{l(i)}^+, j_{l(i)+1}, j_{N_i})$ where $l(i)$ is the last neighbor smaller than i. For the base case $i = 1$, we have $\mathscr{R}(1, j_1, \ldots, j_{N_1})$ by the fact that $(1, j_l) \in \mathscr{E}_k$. Hence the conditions of Assumption 6.5 are satisfied and there exists some $\boldsymbol{u}_{1,K}$ that preserves connectivity with the previous states of all neighbors. By Lemma 6.3 the algorithm finds and applies such a sequence, which implies $\mathscr{R}(1^+, j_1, \ldots, j_{N_1})$. For the general case, we have that $\mathscr{R}(i, j_1^+, \ldots, j_{l(i)}^+, j_{l(i)+1}, \ldots, j_{N_i})$ by simply looking at earlier cases stated where the first argument of relation $\mathscr{R}$ is $m = j_1, \ldots, j_{l(i)}$ (they are earlier cases since $j_{l(i)} < i$). As above, this means the conditions of Assumption 6.5 and therefore Lemma 6.3 are satisfied for the *updated* states of $j_1^+, \ldots, j_{l(i)}^+$, and therefore that $\mathscr{R}(i^+, j_1^+, \ldots, j_{l(i)}^+, j_{l(i)+1}, \ldots, j_{N_i})$ which completes the induction.

Take any $(i,j) \in \mathscr{E}_k$ for which $i > j$, which is sufficient since the graph is undirected. Then, $j \leq j_{l(i)}$ and the relation $\mathscr{R}(i^+, j_1^+, \ldots, j_{l(i)}^+, j_{l(i)+1}, \ldots, j_{N_i})$ (already shown) implies $(i,j) \in \mathscr{E}_{k+K}$. So all the links are preserved, and since the derivation holds for arbitrary k, they are preserved in closed loop. ■

Theorem 6.4 guarantees that the *topology* is preserved when the initial agent states correspond to a connected network. However, this result does not concern the stability of the *agreement*. In practice, we solve the agreement problem by choosing appropriately the rewards Δ, such as in (6.15), so that by maximizing the discounted returns the agents achieve agreement. In Section 6.2.3, we illustrate that this approach performs well in experiments. Note that Theorem 6.4 holds whether the graph is defined with (6.10) or (6.11).

It is also interesting to study the following result about the performance of OPD. Consider some agent i at step k. Since we need to look into the details of OPD for a single agent i at fixed step k, for readability we suppress these indices in the sequel, so that e.g. we write $\rho_d(\boldsymbol{u}_d) = (1-\Lambda)\Delta_d(\boldsymbol{u}_d) + \Lambda\Gamma_d(\boldsymbol{u}_d)$ for reward function (6.12). We define two optimal control problems derived from this reward function. The first removes the connectivity constraint, so that $\rho_{d,\mathrm{u}}(\boldsymbol{u}_d) = (1-\Lambda)\Delta_d(\boldsymbol{u}_d) + \Lambda$. The second is the *agreement* (only) problem with $\rho_{d,\mathrm{a}}(\boldsymbol{u}_d) = \Delta_d(\boldsymbol{u}_d)$, i.e. for $\Lambda = 0$. Denote $v_{\mathrm{u}}^* = \sup_{\boldsymbol{u}_\infty} v_{\mathrm{u}}(\boldsymbol{u}_\infty)$ and $v_{\mathrm{a}}^* = \sup_{\boldsymbol{u}_\infty} v_{\mathrm{a}}(\boldsymbol{u}_\infty)$ where v_{u} and v_{a} are the discounted returns under the new reward functions.

We will compare performance in the original problem with that in the agreement problem.

Theorem 6.5 *Assume $v^* = v_{\mathrm{u}}^*$. For OPD applied to the* original *problem, the near-optimality bounds of Theorem 2.2(ii) hold with the branching factor κ_{a} of the* agreement *problem.*

Proof: We start with a slight modification to the analysis of OPD. For any problem, define the set:

$$\bar{\mathscr{T}} = \{\boldsymbol{u}_d \,|\, d \geq 0, v^* \leq b(\boldsymbol{u}_d)\}$$

Note that $\bar{\mathscr{T}} \subseteq \mathscr{T}^*$ of (2.5), since:

$$v(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma} \geq l(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma} = b(\boldsymbol{u}_d)$$

and so the condition in $\bar{\mathscr{T}}$ implies the one in (2.5). Further, OPD only expands nodes in $\bar{\mathscr{T}}$, since in any tree considered, there always exists some sequence $\boldsymbol{u}$ with $b(\boldsymbol{u}) \geq v^*$ (e.g., the initial subsequence of an optimal sequence), and OPD always expands a sequence that maximizes b.

Denote now $\bar{\mathscr{T}}_{\mathrm{u}}$ and $\bar{\mathscr{T}}_{\mathrm{a}}$ the corresponding sets for the unconstrained and agreement cases. Take a sequence $\boldsymbol{u}_d \in \bar{\mathscr{T}}$, the set in the original problem. By assumption $v^* = v_{\mathrm{u}}^*$, and by construction $b(\boldsymbol{u}_d) \leq b_{\mathrm{u}}(\boldsymbol{u}_d)$, so $v^* \leq b(\boldsymbol{u}_d)$ implies $v_{\mathrm{u}}^* \leq b_{\mathrm{u}}(\boldsymbol{u}_d)$, and

$\bar{\mathscr{T}} \subseteq \bar{\mathscr{T}}_{\mathrm{u}}$. Next, $v_{\mathrm{u}}^* = (1-\Lambda)v_{\mathrm{a}}^* + \frac{\Lambda}{1-\gamma}$ and $b_{\mathrm{u}}(\boldsymbol{u}_d) = l_{\mathrm{u}}(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma} = (1-\Lambda)l_{\mathrm{a}}(\boldsymbol{u}_d) + \frac{\gamma^d}{1-\gamma} = (1-\Lambda)b_{\mathrm{a}}(\boldsymbol{u}_d) + \frac{\Lambda\gamma^d}{1-\gamma}$. Replacing these in condition $v_{\mathrm{u}}^* \leq b_{\mathrm{u}}(\boldsymbol{u}_d)$, we obtain:

$$(1-\Lambda)v_{\mathrm{a}}^* + \Lambda\frac{1-\gamma^d}{1-\gamma} \leq (1-\Lambda)b_{\mathrm{a}}(\boldsymbol{u}_d)$$

which implies $v_{\mathrm{a}}^* \leq b_{\mathrm{a}}(\boldsymbol{u}_d)$, and so $\bar{\mathscr{T}}_{\mathrm{u}} \subseteq \bar{\mathscr{T}}_{\mathrm{a}}$.

Therefore, finally:

$$\bar{\mathscr{T}} \subseteq \bar{\mathscr{T}}_{\mathrm{u}} \subseteq \bar{\mathscr{T}}_{\mathrm{a}} \subseteq \mathscr{T}_{\mathrm{a}}^*$$

Given budget n, the smallest possible depth reached by OPD in the original problem is that obtained by exploring the set $\bar{\mathscr{T}}$ uniformly, in the order of depth. Due to the inclusion chain above, this depth is at least as large as that obtained by exploring $\mathscr{T}_{\mathrm{a}}^*$ uniformly. The latter depth is $\Omega(\log n/\log\kappa_{\mathrm{a}})$ if $\kappa_{\mathrm{a}} > 1$, or else $\Omega(n)$. The bounds follow as in the proof of Theorem 2.2.

■

Theorem 6.5 can be interpreted as follows. If the unconstrained optimal solution would have naturally satisfied connectivity (which is not unreasonable), adding the constraint does not harm the performance of the algorithm, so that flocking is as easy as solving only the agreement problem. This is a nice property to have.

A practical algorithm. Algorithm 6.2 has an important shortcoming in practice: it requires knowing a value of K for which Assumption 6.5 is satisfied. Further, keeping predictions constant for $j > i$ is safe, but conservative, since better predictions are usually available: those made by the neighbors at previous steps, which may be expected to remain valid, e.g. when a steady state is being approached.

Next, we present a more practical variant that does not have these issues. It works in increments of 1 step (rather than K), and at step k, it forms the predictions for neighbors $j > i$ as follows: $\widehat{\boldsymbol{x}}_j^{i,k} = (x_{j,k}, x_{j,2}^{k-1}, ..., x_{j,d}^{k-1})$; Thus for the present step $x_{j,k}$ is used since it was already measured and exchanged, while for future steps the previously communicated trajectories are used.

Since K is unknown, the agent will try preserving connectivity at *every* step, with as many neighbors as possible:

$$\Gamma_{i,d}^k(\boldsymbol{u}_{i,d}) = \frac{1}{|\mathscr{N}_{i,k}|}\sum_{j\in\mathscr{N}_{i,k}}\begin{cases}1 & \text{if } \left\|x_{i,d}^{k,\mathrm{c}} - \widehat{x}_{j,d}^{i,\mathrm{c}}\right\| \leq R\\ 0 & \text{otherwise}\end{cases} \tag{6.16}$$

For the links, definition (6.10) is used, since old neighbors may be lost but the graph may still remain connected due to new neighbors. So the aim here is only connectivity, weaker than topology preservation. For the agreement component, (6.15) is employed. Algorithm 6.3 summarizes the resulting protocol for generic agent i.

Algorithm 6.3 Optimistic-planning flocking at agent i – practical variant.

set initial prediction $\boldsymbol{x}_i^{-1}$ to an empty sequence
for step $k = 0, 1, 2, \ldots$ **do**
    exchange state at k with all neighbors $j \in \mathcal{N}_{i,k}$
    send $\boldsymbol{x}_i^{k-1}$ to $j < i$, receive $\boldsymbol{x}_j^{k-1}$ from $j > i$
    wait to receive new predictions $\widehat{\boldsymbol{x}}_j^{i,k}$ from all $j < i$
    form predictions $\widehat{\boldsymbol{x}}_j^{i,k}$ for $j > i$
    run OPD with (6.12) and (6.16), obtaining $\boldsymbol{u}_i^k$ and $\boldsymbol{x}_i^k$
    send $\boldsymbol{x}_i^k$ to $j > i$
    execute action $u_{i,0}^k$
end for

The main advantage of our approach, for both Algorithm 6.2 and Algorithm 6.3, is the generality of the agent dynamics it can address. This generality comes at the cost of communicating sequences of states and of a relatively computationally involved algorithm. Recall that the time complexity of each individual OPD application is between $O(n \log n)$ and $O(n^2)$ depending on κ. The overall complexity for all agents, if they run OPD in parallel as soon as the necessary neighbor predictions become available, is larger by a factor equal to the length of the longest path from any i to any $j > i$. Depending on the current graph this length may be significantly smaller than the number of agents m.

6.2.3 Experimental results

OP flocking is evaluated in two problems with nonlinear agent dynamics. The first problem concerns flocking for a simple type of nonholonomic agents, where we also study the influence of the tuning parameters of the method. In the second experiment, full-state consensus for two-link robot arms is sought. This experiment illustrates that the algorithm can on the one hand handle rather complicated agent dynamics, and on the other hand that it also works for standard consensus on a fixed graph, even though our analytical focus was placed on the flocking problem.

While both types of agents have continuous-time underlying dynamics, they are controlled in discrete time, as is commonly done in practical computer-controlled systems. The discrete-time dynamics are then the result of integrating the continuous-time dynamics with zero-order-hold inputs. Then, in order for the analysis to hold for Algorithm 6.2, Assumption 6.5 must be satisfied by these discretized dynamics. Note that in practice we apply Algorithm 6.3, and the numerical integration technique introduces model errors that our analysis does not handle.

Flocking of nonholonomic agents. Consider homogeneous agents that evolve on a plane and have the state vector $x = [X, Y, v, \theta]$ with X, Y the position on the plane [m], v the linear velocity [m/s], and θ the orientation [rad]. The control inputs are the rate of change a of the velocity and ω of the orientation. The discrete-time dynamics are:

$$X_{k+1} = X_k + T_s v_k \cos\theta_k, \qquad Y_{k+1} = Y_k + T_s v_k \sin\theta_k$$
$$v_{k+1} = v_k + T_s a_k, \qquad \theta_{k+1} = \theta_k + T_s \omega_k$$

where Euler discretization with sampling time T_s was employed. The aim is to agree on $x^a = [v, \theta]^\top$, which represent the velocity vector of the agent, while maintaining connectivity on the plane by keeping the distances between the connectivity states $x^c = [X, Y]^\top$ below the communication range R.

The specific multiagent system we experiment with consists of 9 agents initially arranged on a grid with diverging initial velocities, see Figure 6.4, top. Their initial communication graph has some redundant links. In the reward function, $\Lambda = 0.5$ so that agreement and connectivity rewards have the same weight, and the agreement reward is (6.15) with the distance measure being a 2-norm weighted so that it saturates to 1 at a distance 5 between the agreement states. The range is $R = 5$. The sampling time is $T_s = 0.25\,\text{s}$.

Figure 6.4 shows that the OP method preserves connectivity while achieving flocking, up to errors due mainly to the discretized actions. The discretized action set was $\{-0.5, 0, 0.5\}$ m/s^2 $\times$ $\{-\pi/3, 0, \pi/3\}$ rad/s, and the planning budget of each agent is $n = 300$ node expansions. For all the experiments, the discount factor γ is set to 0.95, so that long-term rewards are considered with significant weight.

Next, we study the influence of the budget n and a *cutoff length* D for the communicated state predictions, a crucial parameter for the communication requirements of the algorithm. With a finite D, even if OPD provides a longer sequences of predicted states, only the first D values are communicated to the neighbors, and they set subsequent state predictions constant at the last known values. To characterize performance in each experiment with a single number, a mean inter-agent disagreement is computed at every step: $\delta_k = \frac{2}{m(m-1)} \sum_{i<j} \left\| x^a_{i,k} - x^a_{j,k} \right\|$, and the average of δ_k across all steps in the trajectory is reported.

The following budgets are used: $n = 25, 50, 75, 100, 200, \ldots, 600$, and the length of the predictions is not limited. As shown in Figure 6.5, left and as expected from the theoretical guarantees of OPD, disagreement largely decreases with n although the decrease is not monotonic. The influence of the prediction length is studied for fixed $n = 300$, by taking $D = 0, 1, 3, 4$ and then allowing full predictions.[1] Figure 6.5, right indicates that performance is not monotonic in D, and medium-length predictions are better in this experiment. While it is expected that too long predictions will not increase performance since they will rarely be actually be followed, the good results

[1] In effect, predictions with this budget do not exceed length 4 so the last two results will be identical.

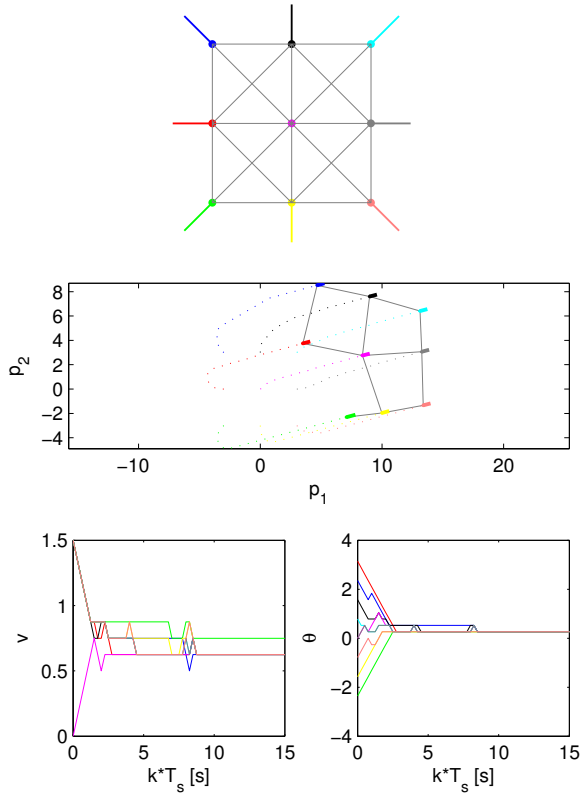

Figure 6.4: Results for nonholonomic agents. Top: initial configuration, with the agents shown as colored dots, their initial velocities and orientations symbolized by the thick lines, and their initial graph with thin gray lines. Middle: trajectories on the plane, also showing the final configuration of the agents. Bottom: evolution of agreement variables.

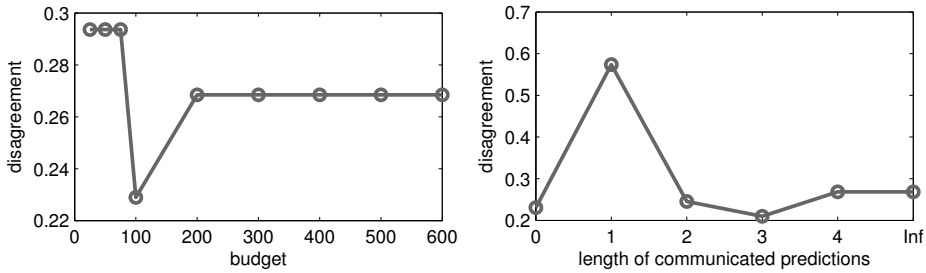

Figure 6.5: Left: Influence of the expansion budget. Right: Influence of the maximal prediction length ("Inf" means it is not limited).

for communicating just the current state without any prediction are more surprising, and need to be studied further.

Consensus of robotic arms. Consider next the full-state consensus of two-link robotic arms operating in a horizontal plane. Each individual arm is the same as in Section 3.1.3. Recall that the state variables for each arm agent are the angles and angular velocities of the two links, $x_i = [\theta_{i,1}, \dot{\theta}_{i,1}, \theta_{i,2}, \dot{\theta}_{i,2}]$, and the actions are the torques of the motors actuating the two links $u_i = [\tau_{i,1}, \tau_{i,2}]$. To attain full-state consensus, the agreement variables comprise the entire state, $x_i^{\mathrm{a}} = x_i$, without a connectivity state or reward component. Applications of this type of consensus problem include decentralized manipulation and teleoperation.

Three robots are connected on a fixed undirected communication graph in which robot 1 communicates with both 2 and 3, but 2 and 3 are not connected. The initial angular positions are taken random with zero initial velocities, see Figure 6.6. The distance measure is the squared Euclidean distance, weighted so that the angular positions are given priority. The discretized actions are $\{-1.5, 0, 1.5\}$ Nm $\times$ $\{-1, 0, 1\}$ Nm, and the budget of each agent is $n = 400$. Consensus is achieved without problems.

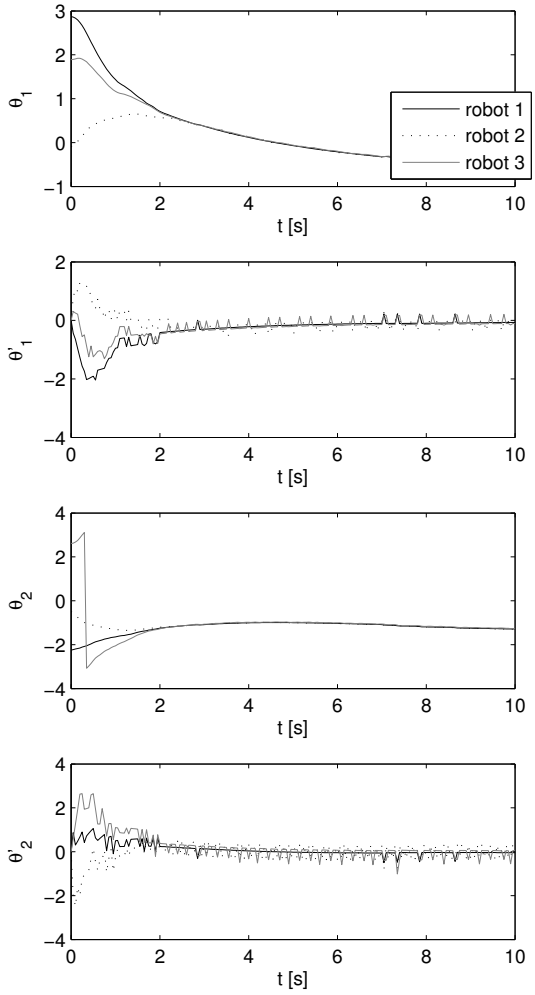

Figure 6.6: Full-state OP consensus of multiple robotic arms: angles and angular velocities for the two links, overimposed for all the robots. Angles wrap around in the interval $[-\pi, \pi)$.

6.3 Summary and conclusions

In this chapter we considered consensus problems in cooperative multiagent systems, and we focused on the challenge of dealing in a generic way with nonlinear agent dynamics, represented as a black box with unknown mathematical form. We devised two approaches. The first designs a reference behavior with a classical consensus method, and uses OO to find controls that closely follow the reference behavior. The first advantage of OO is that it only needs to sample the black-box model of the agent, and so achieves our goal of handling unknown nonlinearities. Secondly, a tight relationship is guaranteed between computation invested and closeness to the reference behavior. Our main results exploited these properties to prove practical consensus. An analysis of representative examples clarified the meaning of the assumptions, showed that in certain problems the optimization is easy to solve, and was complemented by simulations.

For the approach, we considered the generalized flocking variant of consensus, where the agents must drive a subset of their state variables to common values, while communication is constrained by a proximity relationship in terms of another subset of variables. Each agent uses OPD to find an action sequence at each step. By defining the rewards to be optimized in a well-chosen way, the preservation of the interconnection topology was guaranteed, under a controllability assumption. We also gave a practical variant of the algorithm that does not require to know the details of this assumption, and showed that it works well in experiments on nonlinear agents.

Chapter 7

Optimistic planning for networked control systems

So far, in Chapter 6, we have addressed interconnected multiagent systems. In what follows, we consider another type of networked structure, where there is a single system but the controller is implemented over a network in order to reduce installation costs and to facilitate maintenance. This leads to a *networked control system* (NCS). The control law therefore has to share the communication bandwidth with other tasks. This constraint cannot be ignored in general as it may have a serious impact on the system performance. Two main NCS approaches are distinguished based on whether the transmissions are defined by a clock, see e.g. (Branicky et al., 2002; Hespanha et al., 2007), or are triggered depending on the state of the plant, in which case we talk of *event-* or *self-triggered control*, see e.g. (Tabuada, 2007; Heemels et al., 2009; Anta and Tabuada, 2010; Wang and Lemmon, 2009; Henningsson et al., 2008). However, most of these works focus on stabilization or estimation problems, while few address optimal control, and then mostly for linear systems, e.g. (Molin and Hirche, 2009; Blind and Allgöwer, 2011; Rabi et al., 2008; Antunes et al., 2012; Berglind et al., 2012; Henriksson et al., 2012). One interesting exception is (Eqtami et al., 2011), where model-predictive control is used to address nonlinear systems with quadratic costs, focusing on stability. Thus the general problem of optimal control in nonlinear NCS remains largely open.

In this chapter, we propose to use OPD for the near-optimal control of nonlinear NCS with general costs. We focus on the challenge of reducing the number of transmissions over the network, without considering other effects such as delays, packet drop-outs, etc. Crucially, from the analysis of OPD in Section 2.3, (Hren and Munos, 2008) it turns out that it returns a *long sequence* of actions that has *near-optimal* performance. Thus, rather than sending only the first action in the sequence and then rerunning the algorithm, as usually done, we choose to send a longer subsequence. This simple idea allows us to reduce the need for communication (and also computa-

tion) while guaranteeing near-optimality.

We propose two strategies. In the first, communication between the plant and the controller is set to occur at a fixed period which is freely selected. We then investigate the resulting near-optimality and the induced computational complexity. The second strategy enforces a fixed computation budget at every OPD execution, and within this budget generates a sequence of actions from the last measured plant state. As a result, the communication interval adapts to the state, leading to a self-triggered policy, e.g. (Velasco et al., 2003; Anta and Tabuada, 2010; Wang and Lemmon, 2009). We then investigate how sequence length and near-optimality vary with the computation budget. Both strategies allow sending only an initial part of the sequences found, spanning a spectrum from the original method which only applies the first action, to applying the complete sequences. Interestingly, shorter subsequences may do better in some problems, but worse in others, and we provide results and insight about this phenomenon.

7.1 Problem statement

We consider the deterministic optimal control problem of Section 2.3, with system dynamics $x_{k+1} = f(x_k, u_k)$, rewards $r_{k+1} = \rho(x_k, u_k)$, and the objective of minimizing the discounted return $V^{\boldsymbol{u}_\infty}(x) = \sum_{k=0}^{\infty} \gamma^k r_{k+1}$, see (2.4).

We require the standing Assumption 2.1 of reward boundedness, and in addition, to apply OPD, discrete or discretized actions per Assumption 2.4. It is important to note that discretized actions may actually be preferable due to their benefits in NCS: the size of communication packets can be reduced by encoding the discrete actions by their index.

We focus on a networked-control setting, in which actuation and state signals are exchanged over a network that must be efficiently utilized. To this end, the controller should only communicate with the plant when needed. OPD is well equipped to handle this case, since it guarantees that it will return *long and near-optimal sequences* of actions.

We envision the following setup, see Figure 7.1. The sequence of transmission instants is denoted by k_i, $i \in \{0, 1, 2, \ldots\}$, and it will either be fixed by the user or defined by the controller itself. At each k_i, the controller receives the state's measurement and generates a sequence of control actions which is sent as a single packet to the actuators' buffer, like in (Bemporad, 1998; Chaillet and Bicchi, 2008; Quevedo et al., 2011; Quevedo and Nesic, 2012; Henriksson et al., 2012). The actuators then apply the k'-th component of the sequence to the plant at step $k_i + k'$, until the full sequence has been used. Afterwards, the new state's measurement is sent to the controller and the procedure is repeated. The number of transmissions is reduced, since the channel is only used at intervals equal to the sequence lengths.

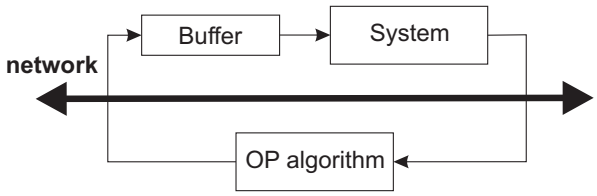

Figure 7.1: NCS architecture in the deterministic case.

7.2 Algorithms and analysis

Algorithm 2.3 and Theorem 2.2 suggest two ways in which OPD could be exploited for NCS. The first possibility is to impose a desired sequence length (planning depth) d at every controller execution step, and then send to the plant either the full sequence or an initial subsequence thereof. Denoting the length of the sent subsequence by $d' \leq d$, this means the communication between the controller and the plant is set to occur with a period d'. Applying OPD in this way is novel. Since length d and the controller execution interval $d' \leq d$ are freely selected, this first strategy is called Clock-triggered OP (COP); it is summarized in Algorithm 7.1.

Algorithm 7.1 Clock-triggered optimistic planning (COP)

Input: initial state x_0, target depth d, subsequence length d'

1: $k \leftarrow 0$
2: **loop**
3: measure current state x_k
4: apply OPD(x_k, d), obtaining a sequence $\boldsymbol{u}_d$
5: send initial subsequence $\boldsymbol{u}_{d'}$ to plant
6: $k \leftarrow k + d'$, wait d' steps
7: **end loop**

The second possibility is to impose the computation budget n, like in the classical application of OPD, and let the algorithm find the longest sequence it can within this budget. Then, different from classical OPD which sends just one action, we send again either the whole sequence or a subsequence. The returned sequence length depends in addition to n also on the current state, through the planning complexity as expressed by branching factor $\kappa(x)$. Therefore, the algorithm is self-triggered and we call it Self-Triggered OP (STOP); it is summarized as Algorithm 7.2. To allow sending subsequences, STOP is parameterized by the fraction $\alpha \in (0,1]$, so that if a sequence of length d is returned by OPD, only the first $\lceil \alpha d \rceil$ actions are actually sent and applied, where $\lceil \cdot \rceil$ denotes the ceiling operator.

We now proceed with the analysis of the two algorithms. An algorithm is called ε-optimal if it applies in closed loop a sequence $\boldsymbol{u}_\infty$ satisfying $V^*(x_0) - V^{\boldsymbol{u}_\infty}(x_0) \leq \varepsilon$.

Algorithm 7.2 Self-triggered optimistic planning (STOP)

Input: initial state x_0, budget n, subsequence fraction α

1: $k \leftarrow 0$
2: **loop**
3: measure current state x_k
4: apply OPD(x_k, n), obtaining a sequence $\boldsymbol{u}_{d(x)}$
5: send initial subsequence $\boldsymbol{u}_{\lceil \alpha d(x) \rceil}$ to plant
6: $k \leftarrow k + \lceil \alpha d(x) \rceil$, wait $\lceil \alpha d(x) \rceil$ steps
7: **end loop**

This property extends the ε-optimality of a single, finite sequence $\boldsymbol{u}_d$, which would be $V^*(x_0) - v_{x_0}(\boldsymbol{u}_d) \leq \varepsilon$. Consider first COP.

Theorem 7.1 *For any d and $d' \leq d$, the following hold. (a) COP is $\frac{\gamma^d}{1-\gamma}$-optimal. (b) For large d, at every state x where it is called, COP requires: • $n(x) = O(\kappa(x)^d)$ expansions if $\kappa(x) > 1$; • $n(x) = O(d)$ expansions if $\kappa(x) = 1$, with $\kappa(x)$ the branching factor of Section 2.3.*

Proof: The second part of the theorem is a consequence of Theorem 2.2(ii). To prove part (a), denote by $\boldsymbol{u}^0$ the sequence returned by OPD when applied at x_0, and recall that $\varepsilon(x_0) = V^*(x_0) - l_{x_0}(\boldsymbol{u}^0)$. If the full sequence is applied, then no matter what actions are taken afterwards at least value $l_{x_0}(\boldsymbol{u}^0)$ is obtained, so COP is $\varepsilon(x_0)$-optimal.

Now, consider applying a subsequence $\boldsymbol{u}'^0$ strictly shorter than $\boldsymbol{u}^0$, and then reexecuting OPD in the resulting state x^1 to obtain $\boldsymbol{u}^1$, see Figure 7.2. Denote by $\boldsymbol{u}''^0$ the leftover subsequence from $\boldsymbol{u}^0$. For arbitrary sequences $\boldsymbol{u}$ and $\tilde{\boldsymbol{u}}$, let $(\boldsymbol{u}, \tilde{\boldsymbol{u}})$ denote their concatenation.

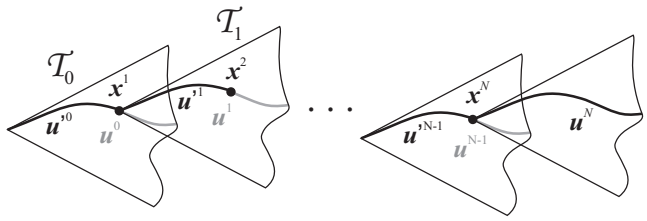

Figure 7.2: Using OPD with subsequences. Different from Figure 2.3, the trees are now oriented horizontally.

When applied from x^1, OPD builds the tree $\mathscr{T}_1$ by expanding nodes in the exactly the same order as it would have expanded, when applied from x_0, nodes in the subtree of $\mathscr{T}_0$ having x^1 at root. That is, for any sequence $\tilde{\boldsymbol{u}}^1$ in $\mathscr{T}_1$, the following b-value relationship holds by definition: $b_{x_0}(\boldsymbol{u}'^0, \tilde{\boldsymbol{u}}^1) = l_{x_0}(\boldsymbol{u}'^0) + \gamma^{d_1} b_{x^1}(\tilde{\boldsymbol{u}}^1)$, where d_1 is the

depth of x^1 in $\mathscr{T}_0$. So, maximizing $b_{x^1}(\tilde{\boldsymbol{u}}^1)$ is the same as maximizing $b_{x_0}(\boldsymbol{u}'^0, \tilde{\boldsymbol{u}}^1)$ with respect to $\tilde{\boldsymbol{u}}^1$. Because OPD is applied with the same setting in x^1 as in x_0, it will expand more nodes and so $\boldsymbol{u}''^0$ is inside $\mathscr{T}_1$. Since $\boldsymbol{u}^1$ maximizes l_{x^1} on $\mathscr{T}_1$, we have $l_{x^1}(\boldsymbol{u}^1) \geq l_{x^1}(\boldsymbol{u}''^0)$, which means the composite sequence satisfies: $l_{x_0}(\boldsymbol{u}'^0, \boldsymbol{u}^1) = l_{x_0}(\boldsymbol{u}'^0) + \gamma^{d_1} l_{x^1}(\boldsymbol{u}^1) \geq l_{x_0}(\boldsymbol{u}'^0) + \gamma^{d_1} l_{x^1}(\boldsymbol{u}''^0) = l_{x_0}(\boldsymbol{u}^0)$ where d_1 is the depth of x^1.

Continuing in a similar fashion, for any N, applying $N-1$ shorter sequences followed by the full Nth sequence achieves $l(\boldsymbol{u}'^0, \boldsymbol{u}'^1, \ldots, \boldsymbol{u}'^{N-1}, \boldsymbol{u}^N) \geq l_{x_0}(\boldsymbol{u}^0)$, see again Figure 7.2. Thus the same is true of the limit as $N \to \infty$, and since this limit is the value $V^{\boldsymbol{u}_\infty}(x_0)$ of the overall closed-loop sequence, we have obtained $V^*(x_0) - V^{\boldsymbol{u}_\infty}(x_0) \leq V^*(x_0) - l_{x_0}(\boldsymbol{u}^0) \leq \varepsilon(x_0)$. To obtain the final result, notice that by Theorem 2.2(i), $\varepsilon(x_0) \leq \frac{\gamma^d}{1-\gamma}$. ■

Thus, the quality of the solution grows with the imposed sequence length d, and the computation requirements to reach this length are bounded and characterized using $\kappa(x)$. Specifically, computation grows exponentially in d, with base $\kappa(x)$ – unless $\kappa(x) = 1$, in which case it grows linearly in d. Next, we move on to STOP.

Theorem 7.2 *Take any large budget n and any $\alpha \in (0,1]$. (a) The near-optimality of STOP is: • $O(n^{-\frac{\log 1/\gamma}{\log \kappa(x_0)}})$ if $\kappa(x_0) > 1$, and • $O(\gamma^{cn})$ if $\kappa(x_0) = 1$. (b) At every state x where it is called, STOP produces a sequence of length: • $d(x) = \Omega(\frac{\log n}{\log \kappa(x)})$ if $\kappa(x) > 1$, and • $d(x) = \Omega(n)$ if $\kappa(x) = 1$.* □

Proof: It directly follows by reapplying the proof of Theorem 7.1(b) that STOP is $\varepsilon(x_0)$-optimal, and using the expressions of $\varepsilon(x_0)$ from Theorem 2.2(iii) completes the first part. The second part follows directly from Theorem 2.2(iii). ■

The performance guarantee of STOP depends only on the planning difficulty at the initial state x_0: it is a negative power of n when $\kappa(x_0) > 1$, and exponential (better) in n when it is $\kappa(x_0) = 1$. The sequence length grows fast, in a way that is characterized using $\kappa(x)$, and which basically 'inverts' the relationship between computation and length in COP.

It must be emphasized that the analysis is performed under the assumption that the model is correct. This is the main reason for which the subsequence length (represented by d' in COP and α in STOP) does not affect the near-optimality guarantee: there is no loss, whether the loop is closed sooner or later. Also, the full initial sequence could be applied and followed by arbitrary actions, while still guaranteeing $\frac{\gamma^d}{1-\gamma}$-optimality. No predictive algorithm can do better in general without increasing the horizon, because the rewards are not assumed to be smooth so they may change unfavorably beyond the horizon explored at the first step. Of course, in practice uncertainty is always present, as model errors or disturbances, which means the sequences cannot be too long and the loop must be closed fairly often.

Even when the model is correct, some nontrivial relations arise between shorter and longer sequences: applying shorter sequences – closing the loop more often –

may achieve better or worse performance, depending on the problem. The following result characterizes this behavior, in a general way that applies to both COP and STOP.

Theorem 7.3 *Let $x \in X$ and denote by $\boldsymbol{u}_d$ the sequence returned by OPD at x. Let $\boldsymbol{u}_{d'}$ be an initial subsequence of $\boldsymbol{u}_d$ and $\boldsymbol{u}_{d_1}$ be obtained by replanning after $\boldsymbol{u}'_d$ (see Figure 7.3). Define similarly $\boldsymbol{u}_{d''}$ and $\boldsymbol{u}_{d_2}$ with $d'' > d'$. Then:*

$$v_x(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1}) \geq v_x(\boldsymbol{u}_{d''}, \boldsymbol{u}_{d_2}) - \frac{\gamma^{d'+d_1}}{1-\gamma}$$

Furthermore, if the budget or target depth of OPD are held constant, then the bound is tight in a worst-case sense. □

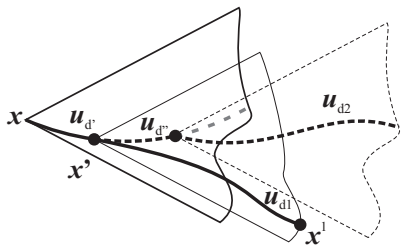

Figure 7.3: Shorter versus longer subsequences.

Proof: Denote by x' and x'' the states reached by $\boldsymbol{u}'_d$ and $\boldsymbol{u}''_d$, respectively. The inequality is shown as follows:

$$\begin{aligned} v_x(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1}) &= l_x(\boldsymbol{u}_{d'}) + \gamma^{d'} v_{x'}(\boldsymbol{u}_{d_1}) = l_x(\boldsymbol{u}_{d'}) + \gamma^{d'} V^*(x') - \gamma^{d'} [V^*(x') - v_{x'}(\boldsymbol{u}_{d_1})] \\ &\geq v_x(\boldsymbol{u}_{d'}) - \frac{\gamma^{d'+d_1}}{1-\gamma} \geq v_x(\boldsymbol{u}_{d''}, \boldsymbol{u}_{d_2}) - \frac{\gamma^{d'+d_1}}{1-\gamma} \end{aligned}$$

where the first step follows from the definition of the v-value, the second just adds and subtracts an extra term, the third follows from Theorem 2.2(i) when applied at x', and finally the last step is true because v cannot increase if more actions are added to the sequence.

To show tightness, a worst-case example is provided where the bound holds with equality. Construct a problem, in the form of a tree, where all rewards are 0 except for one subtree placed below x'' at depth $d' + d_1$, in which they are all 1, see Figure 7.4. Note that due to the zero rewards until $d' + d_1$, up until this depth all trees will be expanded uniformly.When OPD is applied to find u_d and u_{d_1}, it cannot discriminate between sequences since they all have a lower bound l equal to 0, so OPD must choose one arbitrarily. We take the arbitrary sequence u_{d_1} so that it does not contain x'', leading to a value $v_x(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1}) = 0$.

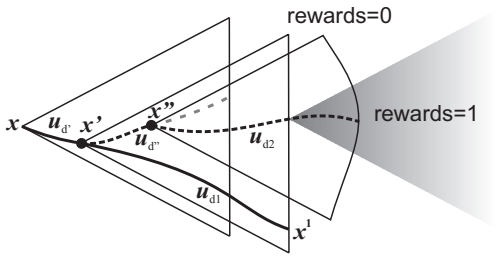

Figure 7.4: Constructing an example where the bound is tight.

When OPD is called at x'', it starts expanding nodes uniformly, and since this state is at depth $d'' > d'$ and OPD has the same budget or target depth as at x', it will expand at least a node at depth $d' + d_1$. We simply place the subtree with rewards of 1 under this node, thereby ensuring that the algorithm discovers it and that the sequence $(\boldsymbol{u}_{d''}, \boldsymbol{u}_{d_2})$ has the optimal value $\frac{\gamma^{d'+d_1}}{1-\gamma}$. So the bound is tight. ■

The theorem says that applying a shorter sequence and then replanning may lose some performance, but not too much: the maximum loss is given by the accuracy of the *entire composite sequence* $(\boldsymbol{u}_{d'}, \boldsymbol{u}_{d_1})$, i.e., $\frac{\gamma^{d'+d_1}}{1-\gamma}$. Further, from the worst-case example it is clear that the same loss can be incurred even if the loop is closed again sooner than d_1 or d_2. The following examples provide more insight into this issue, using COP as it allows to directly control the (sub)sequence length.

Example 7.1 *Shorter sequences can perform better.* Consider an MDP with state space $\{1,2,\ldots,5\}$, two actions $-1,1$ ("left" and "right"), and additive dynamics $x_{k+1} = \max(1, \min(5, x_k + u_k))$. The rewards obtained upon reaching each of the five states are, respectively, $0.8, 0.7, 0.5, 0.8, 0$, and the discount factor is 0.8, see Figure 7.5.

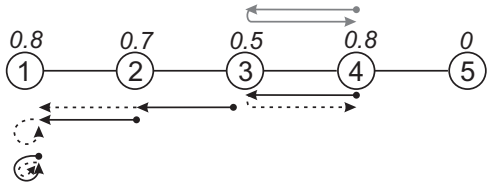

Figure 7.5: A five-state MDP and two COP solutions. States are shown in circles, and rewards in italics above them. The solution from $x_0 = 4$ with $d' = d = 2$ is shown in gray on top of the figure, while the one for $d' = 1$, $d = 2$ is shown in black on the bottom. Solutions are shown as sequences of actions, where the bullets mark the states in which planning is run, and unapplied sequence tails are shown in dashed lines.

When applied from $x_0 = 4$ with $d = 2$ and $d' = 1$, COP replans in $x_1 = 3$, which

allows it to detect the larger rewards to the left. It eventually reaches state 1 and remains there, achieving the optimal return of 3.62. However, when d' is increased to 2, COP exploits the rewards of states 4 and 5 and cycles between these states forever, obtaining a suboptimal return of 3.17. □

Example 7.2 *Longer sequences can perform better.* A similar MDP is taken but now with state space $\{1,2,\ldots,7\}$ and the rewards shown in Figure 7.6. The discount factor is the same.

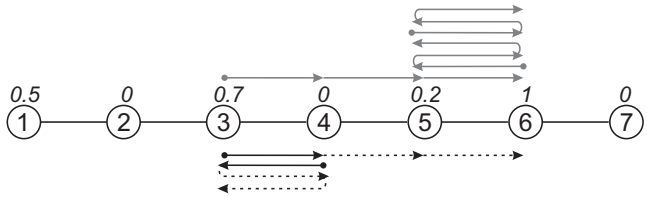

Figure 7.6: A seven-state MDP and two COP solutions, for $d' = d = 3$ (top, gray) and $d' = 1$, $d = 3$ (bottom, black).

Now, when applied from $x_0 = 3$ with $d' = d = 3$, COP discovers the large reward in state 6 and controls towards this state, cycling afterwards between 5 and 6 for a return of 2.22. When $d' = 1$ however, replanning from state 4 misleads the algorithm into a shorter-horizon cycle that focuses on the reward 0.7, achieving only a suboptimal return of 1.56. □

7.3 Experimental results

We study the behavior of COP and STOP in simulations with the linear DC motor and the nonlinear robotic arm.

DC motor. We use the DC motor system from Section 3.1.3, with states $x_1 = \alpha$ the shaft angle, $x_2 = \dot{\alpha}$ the angular velocity, and input (action) u the voltage. The voltage range is increased here to $[30,30]$ V, and the actions are discretized into the set $U = \{-10,-3,0,3,10\}$.. The goal is to stabilize the system and the reward function is quadratic with $Q_{\text{rew}} = \text{diag}(5,0.001)$, $R_{\text{rew}} = 0.01$, and the discount factor is taken $\gamma = 0.9$. State and action saturation ensure bounded rewards, and these rewards are then rescaled into $[0,1]$.

We apply the two algorithms from $x_0 = [2\pi/3,\pi]^\top$, setting $d = 10$ for COP and $n = 300$ for STOP. Figure 7.7 shows the solutions obtained when the complete returned sequences are applied, that is, when $d' = 10$ and respectively $\alpha = 1$. It is interesting to see the evolution of the planning complexity along the trajectory. This

is shown in COP by the changing computation number n of expansions required to reach the desired sequence lengths, where the practical effects of Theorem 7.1(b) are seen; and in STOP by the lengths produced, illustrating Theorem 7.2(b). Complexity is generally smaller in states closer to the equilibrium (fewer expansions/longer sequences), although the evolution is not always monotonic. STOP especially requires only three controller executions and transmissions, thanks to a very long last sequence.

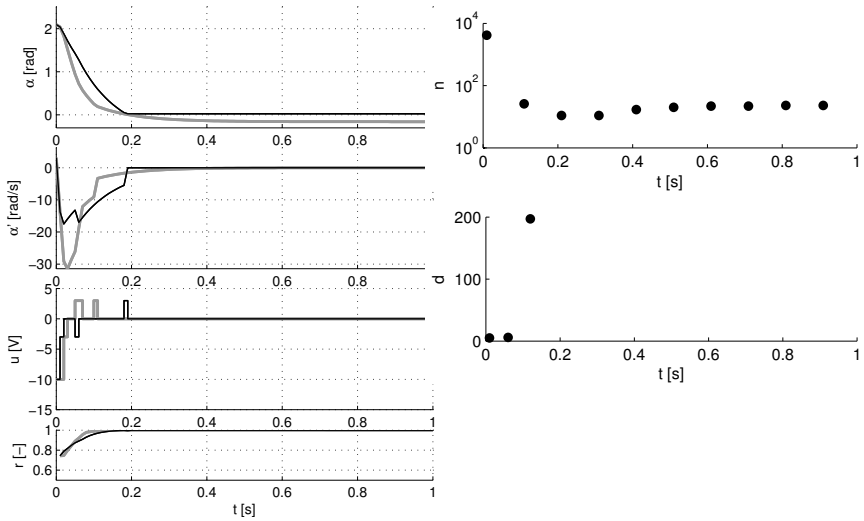

Figure 7.7: Comparison between COP and STOP when applying complete sequences. The left graphs show the controlled trajectories, with COP in gray and STOP in black. The right graphs show, at every controller execution instant: for COP, the computation n spent; and for STOP: the length d of the sequences found. The horizontal coordinates of the points in these graphs are also the transmission times.

To investigate the effect of applying shorter sequences, we vary for COP $d' = 1,2,\ldots,10$ and for STOP $\alpha = 0.1,0.2,\ldots,1$; the returns obtained are shown in Figure 7.8. The suboptimality of each return obeys the upper bounds of Theorem 7.1(a) for COP and Theorem 7.2(a) for STOP; e.g., the COP bound is $0.9^{10}/(1-0.9) \approx 3.49$. Although the loss from Theorem 7.3 is unavoidable in general, in this problem shorter sequences are indeed better, e.g. STOP gains significantly more return when α is below 0.5.

Robot arm. To exemplify our approach in a nonlinear problem, we apply STOP to stabilize the two-link robot arm of Section 3.1.3. Recall that the state variables are the angles θ and angular velocities of the two links, and the actions are the torques of the motors actuating the links. The sampling time is $T_s = 0.05$. The goal of

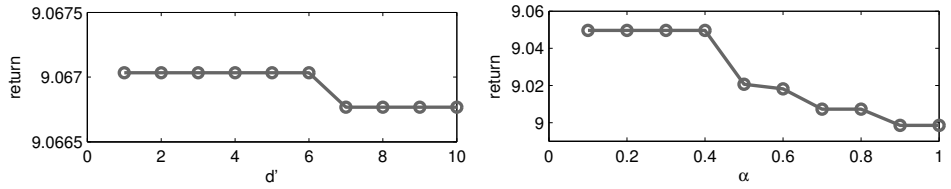

Figure 7.8: Returns obtained by COP (left) and STOP (right) as the length of the applied subsequence varies.

stabilizing in the zero state is modeled by a quadratic reward with weights $Q_{\text{rew}} = \text{diag}[1, 0.05, 1, 0.05]$ and $R_{\text{rew}} = \text{diag}[0.01, 0.01]$. The discount factor is $\gamma = 0.95$, and the discretized action set is $\{-1.5, 0, 1.5\} \times \{-1, 0, 1\}$.

STOP is applied with $n = 1000$ and $\alpha = 0.4$ from the initial state $x_0 = [\pi, 0, \pi, 0]^\top$, and the results are shown in Figure 7.9. A good performance is achieved – subject to the limitations of the action discretization, due to which adjustments have to be made close to the equilibrium state. Note the variation of the sequence lengths with the complexity of the planning problem at different states: in particular, at states further away from the equilibrium sequences are shorter due to higher complexity, whereas at states closer to the equilibrium the complexity decreases and the sequences are longer.

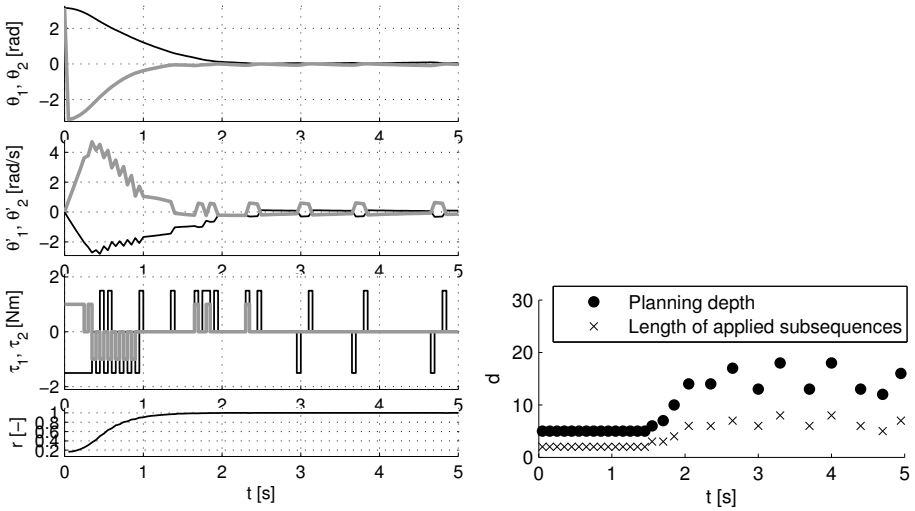

Figure 7.9: Results of STOP for robot arm stabilization: trajectory (top), planning depths and lengths of the applied subsequences (bottom).

7.4 Summary and conclusions

Two novel methods were introduced for the optimal networked control of nonlinear deterministic systems. They both rely on properties of optimistic planning (OP) and guarantee near-optimal performance. *Clock-triggered OP* repeatedly sends action sequences of a fixed length, and bounds the computation required to find them. *Self-triggered OP* works with fixed computation, and adapts the sequence length – and thus the communication interval – to the current state, guaranteeing long sequences. Interestingly, applying shorter subsequences instead of the full sequences may work better or worse, depending on the problem. Examples and simulation experiments illustrated and validated the technique.

Chapter 8

Networked systems: Related topics and outlook

8.1 Related directions

Since optimistic algorithms originate in the planning and machine learning communities, their analysis is geared towards characterizing near-optimality and computation. However, an important concern in the control of physical systems is *stability* – ensuring that the trajectories of the system state are well-behaved, e.g., that they do no diverge. Some particular properties related to stability were addressed in the multiagent control methods of Chapter 6, by explicitly defining the reward functions so that near-optimal trajectories exhibited these desired properties: convergence towards consensus or the preservation of communication links. However, the general problem of stability under the near-optimal control produced by OP was not addressed. A prerequisite step is studying the stability of the fully optimal control law under the discounted criterion (2.1). We have undertaken this step in (P15), starting from some global asymptotic stability properties of the solution under undiscounted returns, with $\gamma = 1$. We then showed that this property is semiglobally and practically preserved in the discounted case, where the adjustable parameter is the discount factor. We then adapted these guarantees to the case of bounded rewards, which is important for practical algorithms such as OP. Finally, we provided sufficient conditions, including boundedness of the stage cost, under which the value function, which serves as a Lyapunov function for the analysis, is continuous.

Another issue in applying OP to control physical systems is obeying real-time constraints. This is challenging for OP algorithms, because their generality comes at the cost of large computational requirements – often growing exponentially with decreasing near-optimality. Therefore, in (P16) we introduced an extension of OP to real-time control, which similarly to OP for networked control systems in Chapter 7 applies open-loop sequences of actions to the system. There is one important differ-

ence: while the current sequence is applied, the *next* sequence is being sought from the *predicted* state at the end of this current sequence. Thus this next sequence will be ready when that state is (approximately) reached, and the overall method can work in real-time. We provided conditions under which the algorithm is provably feasible in real-time, and we analyzed its performance. The algorithm was successful in real and simulated experiments, where the impact of model errors was also studied.

The following publications were discussed in this section:

(P15) R. Postoyan, L. Buşoniu, D. Nešić, J. Daafouz, "Stability of infinite-horizon optimal control with discounted cost", *Proceedings 2014 Conference on Decision and Control (CDC-14)*, Los Angeles, USA, 15–17 December 2014.

(P16) T. Wensveen, L. Buşoniu, R. Babuška, "Real-time Optimistic Planning with Action Sequences", *Proceedings 2015 Conference on Control Systems and Computer Science (CSCS-15)*, Bucharest, Romania, 27–29 May 2015.

8.2 Open issues and ongoing work

An interesting direction for future work into the optimistic multiagent control methods of Chapter 6 is to extend them to other open problems in nonlinear consensus, such as gossiping, where only one communication link can be active at a given time, decentralized formation control of mobile robots, etc. For the specific OP flocking method of Section 6.2, a more immediate step is to develop guarantees also on the agreement component of the state variable, using a line of analysis similar to that of OP consensus.

For the NCS methods of Chapter 7, the most important topic for future work is reducing the dependence on a fully known, deterministic model. This can be done by exploiting stochastic variants of OP in order to handle certain classes of random disturbances. This is a direction of sustained ongoing work, and in particular we have applied OPMDP to handle discrete random disturbances, using them e.g. to model delays in the transmission channel that are multiples of the discrete-time sampling period. Appropriate extensions of the near-optimality and long-sequence guarantees of COP and STOP have been derived, and an article presenting the deterministic case of Chapter 7 together with these new stochastic-case developments is in an advanced stage with a top control journal.

Separately from the approaches presented here, we are investigating the application of OP to another open problem in nonlinear control: the optimal control of the switches in so-called switched systems, which combine continuous dynamics called modes with discontinuous transitions among these modes. We extended the analysis of OP to provide sequences of switches with certification (upper and lower) bounds on the optimal and worst-case costs, and characterized the convergence rate of the gap

between the bounds with increasing computation. Since the controller must often ensure a minimum dwell time (number of steps during which the mode must be kept constant in-between switches), we introduced a new optimistic planning variant that can handle this case, and analyzed its convergence rate. A meaningful related problem arises when the switches are an uncontrolled disturbance, and we are looking for the worst-case cost under any sequence of switches. All the methods and guarantees can be easily applied to this case by essentially negating the reward function. A conference version of this work has been accepted to the flagship 2015 IEEE Conference on Decision and Control, and an extended journal article is under preparation.

Other optimistic planning variants can surely provide advantages in nonlinear control problems, and many interesting directions can be identified in this way. For example, continuous-action planning can be applied to networked control systems, the limited-switch variant of OP may be used to provide guarantees on a notion of average dwell time in switched systems, and so on.

Bibliography

Anta, A. and Tabuada, P. (2010). To sample or not to sample: self-triggered control for nonlinear systems. *IEEE Transactions on Automatic Control*, 55(9):2030–2042.

Antunes, D., Heemels, W., and Tabuada, P. (2012). Dynamic programming formulation of periodic event-triggered control: Performance guarantees and co-design. In *IEEE Conference on Decision and Control, Hawai: U.S.A.*, pages 7212–7217.

Bemporad, A. (1998). Predictive control of teleoperated constrained systems with unbounded communication delays. In *Proceedings 37th Conference on Decision and Control*, pages 2133–2138, Tampa, Florida, USA.

Berglind, J. B., Gommans, T., and Heemels, W. (2012). Self-triggered MPC for constrained linear systems and quadratic costs. In *IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout: The Netherlands*, pages 342–348.

Blind, R. and Allgöwer, F. (2011). On the optimal sending rate for networked control systems with a shared communication medium. In *CDC / ECC (IEEE Conference on Decision and Control and European Control Conference) Orlando, U.S.A., Orlando: U.S.A.*

Branicky, M., Phillips, S., and Zhang, W. (2002). Scheduling and feedback co-design for networked control systems. In *CDC (IEEE Conference on Decision and Control) Las Vegas, U.S.A.*, pages 1211–1217.

Bullo, F., Cortés, J., and Martinez, S. (2009). *Distributed Control of Robotic Networks. A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press.

Chaillet, A. and Bicchi, A. (2008). Delay compensation in packet-switching networked controlled systems. In *CDC (IEEE Conference on Decision and Control), Cancun, Mexico*, pages 3620–3625.

De Persis, C. and Frasca, P. (2013). Robust self-triggered coordination with ternary controllers. *IEEE Transactions on Automatic Control*, 58(12):3024–3038.

Ding, F. and Chen, T. (2005). Identification of hammerstein nonlinear armax systems. *Automatica*, 41(9):1479–1489.

Dong, W. (2011). Flocking of multiple mobile robots based on backstepping. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(2):414–424.

Edelkamp, S. and Schrödl, S. (2012). *Heuristic Search: Theory and Applications*. Morgan Kauffman.

Eqtami, A., Dimarogonas, D., and Kyriakopoulos, K. (2011). Novel event-triggered strategies for model predictive controllers. In *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), Orlando: U.S.A.*, pages 3392–3397.

Fang, L., Antsaklis, P., and Tzimas, A. (2005). Asynchronous consensus protocols: Preliminary results, simulations and open questions. In *Proceedings of the Joint 44th IEEE Conference on Decision and Control, and the European Control Conference*, pages 2194–2199.

Ferrari-Trecate, G., Galbusera, L., Marciandi, M., and Scattolini, R. (2009). Model predictive control schemes for consensus in multi-agent systems with single- and double-integrator dynamics. *IEEE Transactions on Automatic Control*, 54(11):2560–2572.

Fiacchini, M. and Morarescu, I.-C. (2014). Convex conditions on decentralized control for graph topology preservation. *IEEE Transactions on Automatic Control*, 59(6):1640–1645.

Fliess, M. (1992). Reversible linear and nonlinear discrete-time dynamics. *IEEE Transactions on Automatic Control*, 37(8):1144–1153.

Grizzle, J. (1993). A linear algebraic framework for the analysis of discrete-time nonlinear systems. *SIAM Journal of Control and Optimization*, 31(4):1026–1044.

Heemels, W., Sandee, J., and van den Bosch, P. (2009). Analysis of event-driven controllers for linear systems. *International Journal of Control*, 81(4):571–590.

Henningsson, T., Johannesson, E., and Cervin, A. (2008). Sporadic event-based control of first-order linear stochastic systems. *Automatica*, 44:2890–2895.

Henriksson, E., Quevedo, D., Sandberg, H., and Johansson, K. (2012). Self-triggered model predictive control for network scheduling and control. In *IFAC Symposium on Advanced Control of Chemical Processes, Singapore*, pages 432–438.

Hespanha, J., Naghshtabrizi, P., and Xu, Y. (2007). A survey of recent results in networked control systems. *IEEE Special Issue on Technology of Networked Control Systems*, 95(1):138–162.

Hren, J.-F. and Munos, R. (2008). Optimistic planning of deterministic systems. In *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, pages 151–164, Villeneuve d'Ascq, France.

Hunt, K., Sbarbaro, D., Zbikowski, R., and Gawthrop, P. (1992). Neural networks for control systems – a survey. *Automatica*, 28(6):1083–1112.

Hunt, L. and Meyer, G. (1997). Stable inversion for nonlinear systems. *Automatica*, 33(8):1549–1554.

Jakubczyk, B. and Sontag, E. D. (1990). Controllability of nonlinear discrete-time systems: A lie-algebraic approach. *SIAM Journal of Control and Optimization*, 28:1–33.

Keviczky, T. and Johansson, K. (2008). A study on distributed model predictive consensus. In *Proceedings 17th IFAC World Congress (IFAC-08)*, pages 1516–1521, Seoul, Korea.

Lewis, F. and Liu, D., editors (2012). *Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control*. Wiley.

Liu, J., Chen, X., de la Peña, D. M., and Christofides, P. D. (2010). Sequential and iterative architectures for distributed model predictive control of nonlinear process systems. *American Institute of Chemical Engineers (AIChE) Journal*, 56(8):2137–2149.

Mei, J., Ren, W., and Ma, G. (2011). Distributed coordinated tracking with a dynamic leader for multiple Euler-Lagrange systems. *IEEE Transactions on Automatic Control*, 56(6):1415–1421.

Michiels, W., Morarescu, I.-C., and Niculescu, S.-I. (2009). Consensus problems with distributed delays, with application to traffic flow models. *SIAM Journal on Control and Optimization*, 48(1):77–101.

Molin, A. and Hirche, S. (2009). On LQG joint optimal scheduling and control under communication constraints. In *CDC (IEEE Conference on Decision and Control) Shangai: China*.

Moreau, L. (2005). Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50:169–182.

Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 783–791.

Munos, R. (2014). The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search. *Foundations and Trends in Machine Learning*, 7(1):1–130.

Negenborn, R. R., De Schutter, B., and Hellendoorn, H. (2008). Multi-agent model predictive control for transportation networks: Serial versus parallel schemes. *Engineering Applications of Artificial Intelligence*, 21(3):353–366.

Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420.

Olfati-Saber, R., Fax, J. A., and Murray, R. M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233.

Olfati-Saber, R. and Murray, R. M. (2004). Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transaction on Automatic Control*, 49:1520–1533.

Olshevsky, A. and Tsitsiklis, J. (2009). Convergence speed in distributed consensus and averaging. *SIAM Journal of Control and Optimization*, 48(1):33–55.

Qu, Z., Wang, J., and Hull, R. (2008). Cooperative control of dynamical systems with application to autonomous vehicles. *IEEE Transaction on Automatic Control*, 53(4):894–911.

Quevedo, D. E. and Nesic, D. (2012). Robust stability of packetized predictive control of nonlinear systems with disturbances and markovian packet losses. *Automatica*, 48(8):1803–1811.

Quevedo, D. E., Østergaard, J., and Nesic, D. (2011). Packetized predictive control of stochastic systems over bit-rate limited channels with packet loss. *IEEE Transactions on Automatic Control*, 56(12):2854–2868.

Rabi, M., Johansson, K., and Johansson, M. (2008). Optimal stopping event-triggered sensing and actuation. In *CDC (IEEE Conference on Decision and Control) Cancun, Mexico*, pages 3607–3612.

Ren, W. and Beard, R. (2005). Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, pages 655–661.

Ren, W. and Beard, R. W. (2008). *Distributed Consensus in Multi-Vehicle Cooperative Control: Theory and Applications*. Communications and Control Engineering. Springer.

Sain, M. K. and Massey, J. L. (1969). Invertibility of linear time-invariant dynamical systems. *IEEE Transaction on Automatic Control*, 14(2):141–149.

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A. (1995). Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724.

Su, H., Chen, G., Wang, X., and Lin, Z. (2011). Adaptive second-order consensus of networked mobile agents with nonlinear dynamics. *Automatica*, 47(2):368–375.

Suykens, J. and Vandewalle, J., editors (1998). *Nonlinear Modeling: Advanced Black-Box Techniques*. Springer.

Tabuada, P. (2007). Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685.

Tanner, H., Jadbabaie, A., and Pappas, G. (2005). Flocking in teams of nonholonomic agents. In Kumar, V., Leonard, N., and Morse, A., editors, *Cooperative Control*, volume 309 of *Lecture Notes in Control and Information Sciences*, pages 458–460. Springer.

Tanner, H., Jadbabaie, A., and Pappas, G. (2007). Flocking in fixed and switching networks. *IEEE Transactions on Automatic Control*, 52(5):863–868.

Tsitsiklis, J., Bertsekas, D., and Athans, M. (1986). Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31:803–812.

Velasco, M., Fuertes, J., and Marti, P. (2003). The self triggered task model for real-time control systems. *24th IEEE Real-Time Systems Symposium*, pages 67–70.

Wang, X. and Lemmon, M. (2009). Self-triggered feedback control systems with finite-gain $\mathscr{L}_2$ stability. *IEEE Transactions on Automatic Control*, 45:452–467.

Widrow, B. and Walach, E. (2008). *Adaptive Inverse Control: A Signal Processing Approach, Reissue Edition*. Wiley.

Xiao, L. and Boyd, S. (2004). Fast linear iterations for distributed averaging. *System and Control Letters*, 53:65–78.

Zavlanos, M. and Pappas, G. (2008). Distributed connectivity control of mobile networks. *IEEE Transactions on Robotics*, 24(6):1416–1428.

Zheng, Y. and Evans, R. J. (2002). Minimal order discrete-time nonlinear system inversion. In *Proceedings 15th IFAC World Congress*, pages 1119–1125, Barcelona, Spain.

Zhou, J., Wu, X., Yu, W., Small, M., and Lu, J. (2012). Flocking of multi-agent dynamical systems based on pseudo-leader mechanism. *Systems & Control Letters*, 61(1):195–202.

Zhu, J., Lu, J., and Yu, X. (2013). Flocking of multi-agent non-holonomic systems with proximity graphs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):199–210.

Part IV

Other topics and future plans

Chapter 9

Other directions

In addition to my main work thread on optimistic planning and its applications, I have also worked together with students on reinforcement learning offshoot directions from my PhD research. Further, recently I have begun placing emphasis on control applications in robotics: manipulation and unmanned aerial vehicles. These two directions are briefly overviewed next, in Sections 9.1 and 9.2, respectively.

9.1 Reinforcement learning

Optimistic planning is largely a model-based paradigm: it requires a model of the Markov decision process, in the form of the dynamics f and reward function ρ. The field of reinforcement learning (RL) forgoes this requirement and asks the controller to learn an optimal solution for an initially unknown system. This leads to a new set of challenges in addition to near-optimality, crucial among which is the exploration-exploitation dilemma, which asks whether current imperfect knowledge should be acted upon (exploiting it), or novel, possibly better solutions should be attempted (exploration).

A major direction of our research here has been the investigation with PhD student Ivo Grondman of actor-critic RL algorithms, which separately represent and learn on the one hand the value function (returns) $Q(x,u)$ or $V(x)$, and on the other hand the control policy π. In particular, policy gradient based actor-critic algorithms are very popular. Their advantage of being able to search for optimal policies using low-variance gradient estimates has made them useful in several real-life applications, such as robotics, power control and finance. We have therefore begun by surveying the literature on this class of actor-critic algorithms (P17), with a focus on methods that can work in an online setting and use function approximation in order to deal with continuous state and action spaces, which are important in control. In addition to the origins of actor-critic algorithms, we have described the workings of the natural gradient, which has made its way into many actor-critic algorithms in the

past few years. A review of several standard and natural actor-critic algorithms was provided, together with an overview of application areas and a discussion on open issues.

Then, we have delved into the particular research topic of learning *models* of the system in order to make the policy updates in the actor-critic method more effective and thereby increase learning speed. To this end, we have proposed in (P18), (P19), (P20) two such model-learning methods, both using local linear regression to construct approximations of the functions involved. The first algorithm uses a novel model-based update rule for the actor parameters. The second algorithm does not use an explicit actor but learns a reference model which represents a desired behavior, from which desired control actions can be calculated using the inverse of the learned process model. The two novel methods and a standard actorcritic algorithm were applied to the pendulum swing-up problem, in which the novel methods achieve faster learning than the standard algorithm.

A more direct use of models is in imitation learning, which is related to RL but attempts to directly replicate demonstrated behavior rather than learning new behaviors on its own. This is motivated by human learning, where a task is rarely learned from scratch, instead starting from a demonstration by a skilled person. In (P21), we used imitation to quickly generate a rough solution to a robotic task from demonstrations, supplied as a collection of state-space trajectories. Appropriate control actions needed to steer the system along the trajectories are automatically learned in the form of a (nonlinear) state-feedback control law. The learning scheme has two components: a dynamic reference model and an adaptive inverse process model, both based on local linear regression. The reference model infers the desired behavior from the demonstration trajectories, while the inverse process model provides the control actions to achieve this behavior and is improved online using learning. Experimental results with a pendulum swing-up problem and a robotic arm demonstrated the practical usefulness of this approach.

Other contributions in this area include several overviews of approximate reinforcement learning (P22), (P23).

The following publications were discussed in this section:

(P17) I. Grondman, L. Buşoniu, G. Lopes, R. Babuška, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients", *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Cybernetics*, vol. 42, no. 6, pages 1291–1307, 2012.

(P18) I. Grondman, M. Vaandrager, L. Buşoniu, R. Babuška, E. Schuitema, "Efficient Model Learning Methods for Actor-Critic Control", *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 42, no. 3, pages 591–602, 2012.

(P19) I. Grondman, M. Vaandrager, L. Buşoniu, R. Babuška, E. Schuitema, "Actor-Critic Control with Reference Model Learning". *Proceedings 18th IFAC World Congress (IFAC-11)*, Milano, Italy, 22 August–2 September 2011.

(P20) I. Grondman, L. Buşoniu, R. Babuška, "Model-Learning Actor-Critic Algorithms: Performance Evaluation in a Motion Control Task". *Proceedings 51st IEEE Conference on Decision and Control (CDC-12)*, Maui, Hawaii, 10–13 December 2012.

(P21) M. Vaandrager, R. Babuska, L. Busoniu, G. Lopes, "Imitation Learning with Non-Parametric Regression". *Proceedings 2012 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR-12)*, Cluj-Napoca, Romania, 24–27 May 2012.

(P22) L. Buşoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuska, B. De Schutter, "Least-squares methods for policy iteration". In *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Editors, series *Adaptation, Learning, and Optimization*, vol. 12, pages 75–109, Springer, 2012.

(P23) L. Buşoniu, B. De Schutter, and R. Babuška, "Approximate Dynamic Programming and Reinforcement Learning". In *Interactive Collaborative Information Systems*, R. Babuška and F.C.A. Groen, Editors, series *Studies in Computational Intelligence*, vol. 281, pages 3–44. Springer, 2010.

9.2 Robotics applications

Bridging the gap between the RL techniques of the previous section and robotics applications, we have attempted to address an important challenge in such applications: the limited availability of interaction data with the system, which is at least as important in RL as limited computation time. A promising approach to achieve this is experience replay (ER), which quickly learns from a limited amount of data by repeatedly presenting these data to an underlying RL algorithm. Despite the fact that the approach and its benefits are well-known in the literature, ER RL has been studied only sporadically and its applications have largely been confined to simulated systems. Therefore, in (P25) we have evaluated ER RL on real-time control experiments involving a pendulum swing-up problem and the vision-based control of a goalkeeper robot. These real-time experiments were complemented by simulation studies and comparisons with traditional RL.

With students Koppány Máthé and Elöd Páll, I have more recently delved into a different robotics application: lightweight, low-cost unmanned aerial vehicles (UAVs) for civilian uses. To set the foundations for the research, we have overviewed in (P26) the literature on vision and control methods that can be applied to low-cost UAVs. We overviewed among others feature detection and tracking, optical flow and visual

servoing, low-level stabilization, and high-level planning methods. We then listed popular low-cost UAVs, selecting mainly quadrotors. We investigated applications, restricting our focus to the field of infrastructure inspection. Finally, as an example, we formulated two use-cases for railway inspection, a less explored application field, and illustrate usage of the vision and control techniques reviewed by selecting appropriate ones to tackle these use-cases.

One of these use-cases is railway following, and in (P27) we have focused on solving it. The method developed relies on vision-based detection and tracking of the vanishing point of the railway tracks, overhead lines, and other related lines in the image, coupled with a controller that adjusts the yaw so as to keep the vanishing point in the center of the image. Simulation results illustrated the method is effective, and were complemented by vanishing-point tracking results on real image.

This work was presented in the following publications:

(P25) S. Adam, L. Buşoniu, and R. Babuška, "Experience Replay for Real-Time Reinforcement Learning Control", *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 42, no. 2, pages 201–212, 2012.

(P26) K. Máthé, L. Buşoniu. "Vision and Control for UAVs: A Survey of General Methods and of Inexpensive Platforms for Infrastructure Inspection", *Sensors* vol. 15 no. 7, pages 14887–14916, 2015.

(P27) E. Páll, K. Máthé, L. Tamás, L. Buşoniu, "Railway Track Following with the AR.Drone Using Vanishing Point Detection". In *Proceedings 2014 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR-14)*, Cluj-Napoca, Romania, 22–24 May 2014.

An ongoing direction is the application of OPMDP in an assistive robotics scenario, where a robotic arm on a mobile platform manipulates objects such as switches in the environment of an elderly or disabled person. The scenario is modeled as partially-observable Markov decision process, where not all the state variables are directly measurable, and OPMDP is applied to derive an optimal control. In this context, OPMDP is known as the AEMS2 algorithm, but a detailed analysis such as the one in Section 4.1 is not available, and our second goal is to develop one. The primary goal is to demonstrate that planning algorithms can solve a socially relevant, state-of-the-art robotics problem.

Chapter 10

Overall plans for the future

10.1 Introduction and main objective

Automated systems are increasingly expected to perform well in complex, open, unstructured and unpredictable environments, such as in the everyday surroundings of human beings. Even in more traditional industrial or lab environments, many systems cannot be accurately modeled, for instance because they are insufficiently understood or the modeling process is too expensive. *Learning* to deal with the unknown, combined system-environment dynamics is an important capability of their control algorithms. Another essential component is generality, in order to handle complex systems in a uniform way; *planning* methods can supply this generality. Therefore, my main research objective is

> *To develop an algorithmic framework for the learning and planning control of complex systems*

where a complex system is defined by its highly nonlinear, (partly) unknown, possibly stochastic and high-dimensional behavior. To succeed, such a framework must combine planning and learning with control techniques. Therefore, I will rely on my extensive expertise in optimistic planning and reinforcement learning, and I will cross-fertilize it with ideas and methods from advanced control, among which nonlinear, predictive, networked, and adaptive control.

The end result will be a comprehensive set of algorithmic tools for the learning and planning control of a wide array of systems, accompanied by analytical performance and safety guarantees, as well as by practical applications. I will focus on applications in robotics, but I will also explore more general decision-making applications in areas such as logistics or medicine. These results will serve as a solid platform from which to explore new directions in decision and control on the one hand, and machine learning and artificial intelligence on the other.

10.2 Research plan

As a general theme for the upcoming three to five years, I will start from the optimistic planning and control research presented in this thesis, and I will integrate novel control insights, together with machine and reinforcement learning ideas, in order to approach the overall objective stated above. The following major topics will be addressed:

- **Integration of planning and learning**
  Starting from the early ideas on learning the value function and unknown transition probabilities presented in Chapter 5, a set of integrated learning and planning algorithms will be developed. This work will first consider simple, deterministic problems, and then progress towards stochastic and partially-observable versions. The algorithms will on the one hand plan ahead action sequences at each step of interaction with the system, like optimistic planning methods but now using a learned, approximate model instead of the exact one; and also learn the optimal solution and model of the system across the interaction trajectories, like reinforcement learning. The strengths of the two techniques will be combined, with guarantees on the rate of approach towards the optimal solution at each step of planning, as well as across the entire experiment.
- **Stability guarantees**
  A crucial requirement before a control algorithm can be applied in the human environment or the industry is guaranteeing safety: the controller must not damage the system or endanger its users. Safety is a complementary property to the performance guarantees that form the focus of this thesis, and is a challenging open problem in planning and reinforcement learning. To address this problem, building on the initial stability result for optimal solutions outlined in Chapter 8, we will develop guarantees for the *near*-optimal solutions produced by planning. The interaction with reinforcement learning is additionally challenging, since the algorithm must discover good behavior by explore a range of options that, when left unchecked, may turn out to be unsafe. Thus exploration must be balanced with stability. Useful results can be identified in the areas of predictive and adaptive nonlinear control, which often employ Lyapunov stability analysis.
- **Open problems in nonlinear control**
  Starting from the existing work in Part III, I will continue to deepen existing work in networked systems with the aim of handling issues such as packet losses, delays, bandwidth limits, etc. Beyond this, I will explore the relationship of planning and learning with other, novel problems in nonlinear control, difficult to tackle with classical methods, such as switched systems, extremum-seeking control, economic model-predictive control, etc.

- **Applications to real case studies**
  I will validate the fundamental and algorithmic contributions described above in real-life case studies. Cooperation with industrial partners will be sought, with the longer-term goal of commercial deployment.

  I have already been involved in several lab robotics projects, and I strongly believe that the quality of human life can be improved by using and interacting with robots in everyday life. In particular, a subarea that can greatly benefit society is assistive robotics, concerned with providing assistance to the disabled and to the ever-increasing elderly population. Since the environment of assistive robots is highly complex, unstructured, and unpredictable, these robots require planning and learning control to be successful. The simple demonstration task of assistive object manipulation outlined in Section 9.2 will serve as a first step.

10.3 Long-term research goals

Over the longer term, I will develop the research group I am currently leading and strive to propel it to the forefront of international research. To this end, I will continue attracting and recruiting top undergraduate students, and I will exploit public funding opportunities at the national, European, and international levels, as well as industrial funding with local and international companies. Our research will have developed a unified algorithmic framework for the planning and learning control of a wide range of complex systems, accompanied by theoretical guarantees and applications to domains such as robotics, transportation, medical treatment, etc.

I strongly believe that a symbiosis between artificial intelligence and advanced control theory is instrumental to achieving these goals. Beyond these fields, my results will serve as a strong platform upon which to explore novel directions in artificial intelligence and computer science on the one hand, and in control and decision making on the other. I will also investigate inter-disciplinary connections with operations research and medicine.