Results in Control and Optimization 7 (2022) 100121

Contents lists available at ScienceDirect

Results in Control and Optimization

journal homepage: www.elsevier.com/locate/rico

Newton's method for reinforcement learning and model predictive control

Dimitri Bertsekas[1]

School of Computing, and Augmented Intelligence, Arizona State University, Tempe, AZ, United States of America

ARTICLE INFO

Keywords:
AlphaZero
Off-line training
On-line play
Dynamic programming over an infinite horizon
Reinforcement learning
Model predictive control

ABSTRACT

The purpose of this paper is to propose and develop a new conceptual framework for approximate Dynamic Programming (DP) and Reinforcement Learning (RL). This framework centers around two algorithms, which are designed largely independently of each other and operate in synergy through the powerful mechanism of Newton's method. We call these the *off-line training and the on-line play algorithms*; the names are borrowed from some of the major successes of RL involving games. Primary examples are the recent (2017) AlphaZero program (which plays chess), and the similarly structured and earlier (1990s) TD-Gammon program (which plays backgammon). In these game contexts, the off-line training algorithm is the method used to teach the program how to evaluate positions and to generate good moves at any given position, while the on-line play algorithm is the method used to play in real time against human or computer opponents.

Both AlphaZero and TD-Gammon were trained off-line extensively using neural networks and an approximate version of the fundamental DP algorithm of policy iteration. Yet the AlphaZero player that was obtained off-line is not used directly during on-line play (it is too inaccurate due to approximation errors that are inherent in off-line neural network training). Instead a separate on-line player is used to select moves, based on multistep lookahead minimization and a terminal position evaluator that was trained using experience with the off-line player. The on-line player performs a form of policy improvement, which is not degraded by neural network approximations. As a result, it greatly improves the performance of the off-line player.

Similarly, TD-Gammon performs on-line a policy improvement step using one-step or two-step lookahead minimization, which is not degraded by neural network approximations. To this end it uses an off-line neural network-trained terminal position evaluator, and importantly it also extends its on-line lookahead by rollout (simulation with the one-step lookahead player that is based on the position evaluator).

An important lesson from AlphaZero and TD-Gammon is that the performance of an off-line trained policy can be greatly improved by on-line approximation in value space, with long lookahead (involving minimization or rollout with the off-line policy, or both), and terminal cost approximation that is obtained off-line. This performance enhancement is often dramatic and is due to a simple fact, which is couched on algorithmic mathematics and is the focal point of this work:

(a) Approximation in value space with one-step lookahead minimization amounts to a step of Newton's method for solving Bellman's equation.

E-mail address: dbertsek@asu.edu.

[1] This paper summarizes the principal research ideas of the author's forthcoming book "Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control", Athena Scientific, Belmont, MA, 2022.

https://doi.org/10.1016/j.rico.2022.100121
Received 13 January 2022; Accepted 30 March 2022
Available online 6 April 2022

(b) The starting point for the Newton step is based on the results of off-line training, and may be enhanced by longer lookahead minimization and on-line rollout. Indeed the major determinant of the quality of the on-line policy is the Newton step that is performed on-line, while off-line training plays a secondary role by comparison.

Significantly, the synergy between off-line training and on-line play also underlies Model Predictive Control (MPC), a major control system design methodology that has been extensively developed since the 1980s. This synergy can be understood in terms of abstract models of infinite horizon DP and simple geometrical constructions, and helps to explain the all-important stability issues within the MPC context. In this work we aim to provide insights (often based on visualization), which explain the beneficial effects of on-line decision making on top of off-line training. In the process, we will bring out the strong connections between the artificial intelligence view of RL, and the control theory views of MPC and adaptive control. While we will deemphasize mathematical proofs, there is considerable related analysis, which supports our conclusions and can be found in the author's recent RL books (Bertsekas, 2019; Bertsekas, 2020), and the abstract DP monograph (Bertsekas, 2022).

One of our principal aims is to show, through the algorithmic ideas of Newton's method and the unifying principles of abstract DP, that the AlphaZero/TD-Gammon methodology of approximation in value space and rollout applies very broadly to deterministic and stochastic optimal control problems, involving both discrete and continuous search spaces, as well as finite and infinite horizon.

1. Alphazero, off-line training, and on-line play

Perhaps the most impressive success story in reinforcement learning (RL) is the development of the AlphaZero program by DeepMind Inc; see [1,2]. AlphaZero plays Chess, Go, and other games, and is an improvement in terms of performance and generality over the earlier AlphaGo program [SHM16], which plays the game of Go only. AlphaZero, and other chess programs based on similar principles, play as well or better than all competitor computer programs available in 2021, and much better than all humans. These programs are remarkable in several other ways. In particular, they have learned how to play without human instruction, just data generated by playing against themselves. Moreover, they learned how to play very quickly. In fact, AlphaZero learned how to play chess better than all humans and computer programs within hours (with the help of awesome parallel computation power, it must be said).

We should note also that the principles of the AlphaZero design have much in common with the TD-Gammon program of Tesauro [3–5] that plays backgammon (a game of substantial computational and strategical complexity, which involves a number of states estimated to be in excess of 10^{20}). Tesauro's programs stimulated much interest in RL in the middle 1990s, and exhibit similarly different and better play than human backgammon players. A related program for the (one-player) game of Tetris, based on similar principles, is described by Scherrer et al. [6], together with several antecedents. The backgammon and tetris programs, while dealing with less complex games than chess, are of special interest because they involve significant stochastic uncertainty, and are thus unsuitable for the use of long lookahead minimization, which is widely believed to be one of the major contributors to the success of AlphaZero, and chess programs in general.

Still, for all of their brilliant implementations, these impressive game programs are couched on well established methodology, from optimal and suboptimal control, which is portable to far broader domains of engineering, economics, and other fields. This is the methodology of dynamic programming (DP), policy iteration, limited lookahead minimization, rollout, and related approximations in value space. The aim of this work is to propose a conceptual, somewhat abstract framework, which allows insight into the connections of AlphaZero and TD-Gammon with some of the core problems in decision and control, and suggests potentially far-reaching extensions.

To understand the overall structure of AlphaZero and related programs, and their connections to the DP/RL methodology, it is useful to divide their design into two parts:

(a) *Off-line training*, which is an algorithm that learns how to evaluate chess positions, and how to steer itself towards good positions with a default/base chess player.
(b) *On-line play*, which is an algorithm that generates good moves in real time against a human or computer opponent, using the training it went through off-line.

We will next briefly describe these algorithms, and relate them to DP concepts and principles, focusing on AlphaZero for the most part.

1.1. Off-line training and policy iteration

An off-line training algorithm like the one used in AlphaZero is the part of the program that learns how to play through self-training that takes place before real-time play against any opponent. It is illustrated in Fig. 1, and it generates a sequence of *chess players* and *position evaluators*. A chess player assigns "probabilities" to all possible moves at any given chess position: these may

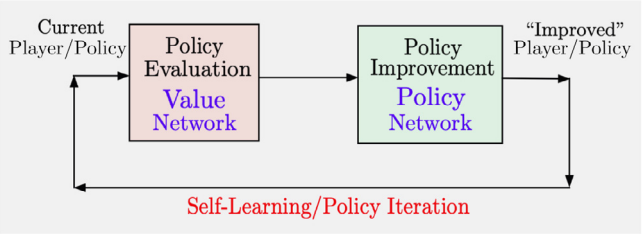

Fig. 1. Illustration of the AlphaZero off-line training algorithm. It generates a sequence of position evaluators and chess players. The position evaluator and the chess player are represented by two neural networks, a value network and a policy network, which accept a chess position and generate a position evaluation and a set of move probabilities, respectively.

be viewed as a measure of "effectiveness" of the corresponding moves. A position evaluator assigns a numerical score to any given chess position, and thus predicts quantitatively the performance of a player starting from any position. The chess player and the position evaluator are represented by neural networks, a *policy network* and a *value network*, which accept as input a chess position and generate a set of move probabilities and a position evaluation, respectively.[2]

The overall training algorithm is a form of *policy iteration*, a DP algorithm that will be of primary interest to us in this paper. Starting from a given player, it repeatedly generates (approximately) improved players, and settles on a final player that is judged empirically to be "best" out of all the players generated. Policy iteration may be separated conceptually into two stages (see Fig. 1).

(a) *Policy evaluation*: Given the current player and a chess position, the outcome of a game played out from the position provides a single data point. Many data points are thus collected, and are used to train a value network, whose output serves as the position evaluator for that player.
(b) *Policy improvement*: Given the current player and its position evaluator, trial move sequences are selected and evaluated for the remainder of the game starting from many positions. An improved player is then generated by adjusting the move probabilities of the current player towards the trial moves that have yielded the best results.

In AlphaZero (as well as AlphaGo Zero, the version that plays the game of Go) the policy evaluation is done by using deep neural networks. The policy improvement uses a complicated algorithm called *Monte Carlo Tree Search* (MCTS for short), a form of randomized multistep lookahead minimization that enhances the efficiency of the multistep lookahead operation, by pruning intelligently the multistep lookahead tree. However, deep neural networks and MCTS, while leading to some performance gains, are not of fundamental importance. The approximation quality that a deep neural network can achieve can also be achieved with a shallow neural network, perhaps with reduced sample efficiency. Similarly MCTS cannot achieve better lookahead accuracy than standard exhaustive search, although it may be more efficient computationally. Indeed, policy improvement can be done more simply without MCTS, as in Tesauro's TD-Gammon program: we try all possible move sequences from a given position, extend forward to some number of moves, and then evaluate the terminal position with the current player's position evaluator. The move evaluations obtained in this way are used to nudge the move probabilities of the current player towards more successful moves, thereby obtaining data that is used to train a policy network that represents the new player.

Regardless of the use of deep neural networks and MCTS, it is important to note that *the final policy and the corresponding policy evaluation produced by approximate policy iteration and neural network training in AlphaZero involve serious inaccuracies, due to the approximations that are inherent in neural network representations*. The AlphaZero on-line player to be discussed next uses approximation in value space with multistep lookahead minimization, and does not involve any neural network, other than the one that has been trained off-line, so it is not subject to such inaccuracies. As a result, it plays much better than the off-line player.

1.2. On-line play and approximation in value space — truncated rollout

Consider the "final" player obtained through the AlphaZero off-line training process. It can play against any opponent by generating move probabilities at any position using its off-line trained policy network, and then simply play the move of highest probability. This player would play very fast on-line, but it would not play good enough chess to beat strong human opponents. The extraordinary strength of AlphaZero is attained only after the player obtained from off-line training is embedded into another algorithm, which we refer to as the "on-line player". In other words *AlphaZero plays on-line much better than the best player it has produced with sophisticated off-line training.* This phenomenon, *policy improvement through on-line play*, is centrally important for our purposes in this paper.

[2] Here the neural networks play the role of *function approximators*. By viewing a player as a function that assigns move probabilities to a position, and a position evaluator as a function that assigns a numerical score to a position, the policy and value networks provide approximations to these functions based on training with data. Actually, AlphaZero uses the same neural network for training both value and policy. Thus there are two outputs of the neural net: value and policy. This is pretty much equivalent to having two separate neural nets and for the purposes of this paper, we prefer to explain the structure as two separate networks. AlphaGo uses two separate value and policy networks. Tesauro's backgammon programs use a single value network, and generate moves when needed by one-step or two-step lookahead minimization, using the value network as terminal position evaluator.

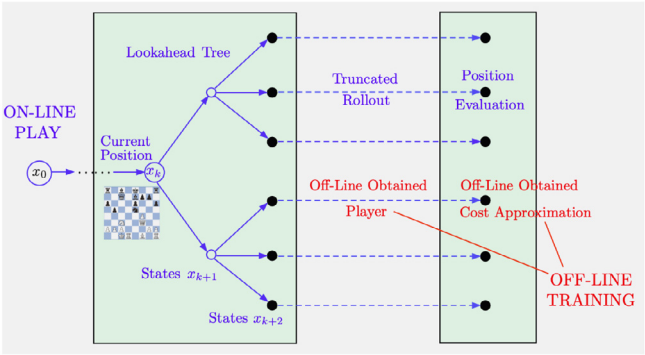

Fig. 2. Illustration of an on-line player such as the one used in AlphaGo, AlphaZero, and Tesauro's backgammon program [5].

Given the policy network/player obtained off-line and its value network/position evaluator, the on-line algorithm plays roughly as follows (see Fig. 2). At a given position, it generates a lookahead tree of all possible multiple move and countermove sequences, up to a given depth. It then runs the off-line obtained player for some more moves, and then evaluates the effect of the remaining moves by using the position evaluator of the value network. The middle portion, called "truncated rollout", may be viewed as *an economical substitute for longer lookahead minimization*. Actually truncated rollout is not used in the published version of AlphaZero [1]; the first portion (multistep lookahead minimization) is very long and implemented efficiently (partly through the use of MCTS), so that the rollout portion is not essential. Rollout has been used in AlphaGo, the AlphaZero predecessor [7]. Moreover, chess and Go programs (including AlphaZero) typically use a well-known limited form of rollout, called "quiescence search", which aims to resolve imminent threats and highly dynamic positions through simulated multi-move piece exchanges, before invoking the position evaluator. Rollout is instrumental in achieving high performance in Tesauro's 1996 backgammon program [5]. The reason is that backgammon involves stochastic uncertainty, so long lookahead minimization is not possible because of rapid expansion of the lookahead tree with every move.[3]

In control system design, similar architectures to the ones of AlphaZero and TD-Gammon are employed in model predictive control (MPC). There, the number of steps in lookahead minimization is called the *control interval*, while the total number of steps in lookahead minimization and truncated rollout is called the *prediction interval*; see e.g., Magni et al. [8].[4] The benefit of truncated rollout in providing an economical substitute for longer lookahead minimization is well known within this context. We will discuss further the structure of MPC and its similarities with the AlphaZero architecture in Section 4.2. The DP ideas with cost function approximations that are similar to the on-line player illustrated in Fig. 2, are also known as *approximate dynamic programming*, or *neuro-dynamic programming*, and will be central for our purposes. They will be generically referred to as *approximation in value space* in this paper.

Note also that in general, off-line training and on-line policy implementation may be designed independently of each other. For example the off-line training portion may be very simple, such as using a known heuristic policy for rollout without truncation, or without terminal cost approximation. Conversely, a sophisticated process may be used for off-line training of a terminal cost function approximation, which is used following the lookahead minimization in a value space approximation scheme.

1.3. Our framework in summary

The AlphaZero and TD-Gammon experiences reinforce an important conclusion that applies more generally to decision and control problems: despite the extensive off-line effort that may have gone into the design of a policy, its performance may be greatly improved by on-line approximation in value space, with extra lookahead involving minimization and/or with rollout using this policy, and terminal cost approximation.

In the following sections, we will aim to amplify on this theme and to focus on the principal characteristics of AlphaZero-like architectures, within a broader context of optimal decision and control. We will make use of intuitive visualization, and the central role of Newton's method for solving Bellman's equation. Briefly, our main point is that *on-line approximation in value space amounts to a step of Newton's method for solving Bellman's equation, while the starting point for the Newton step is based on the results of off-line*

[3] Tesauro's rollout-based backgammon program [5] uses only a value network, called TD-Gammon, which was trained using an approximate policy iteration scheme developed several years earlier [3]. TD-Gammon is used to generate moves for the truncated rollout via a one-step or two-step lookahead minimization. Thus the value network also serves as a substitute for the policy network during the rollout operation. The terminal position evaluation used at the end of the truncated rollout is also provided by the value network. The middle portion of Tesauro's scheme (truncated rollout) is important for achieving a very high quality of play, as it effectively extends the length of lookahead from the current position.

[4] The Matlab toolbox for MPC design explicitly allows the user to set these two intervals.

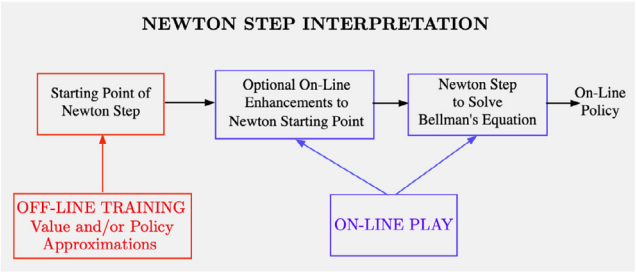

Fig. 3. Schematic illustration of the connections between off-line training, on-line play, and Newton's method for solving Bellman's equation. On-line play is viewed as a Newton step, while off-line training provides the starting point for the Newton step.

training; see Fig. 3. Moreover, *this starting point may be enhanced by several types of on-line operations, including longer lookahead minimization, and on-line rollout with a policy obtained through off-line training, or heuristic approximations*.

This interpretation will be the basis for powerful insights into issues of stability, performance, and robustness of the on-line generated policy. In particular, we will aim to show that feedback control, based on approximation in value space and the underlying off-line training/on-line play structure, offers benefits that go well beyond the conventional wisdom that "feedback corrects for uncertainty, and modeling errors". The reason is that by overlaying on-line play on top of off-line training, we gain significantly in performance, by correcting (through the Newton step) for the errors that are inherent in off-line training with approximation architectures such as neural networks.

Our mathematical framework is couched on unifying principles of abstract DP, including abstract forms of Bellman's equation, and the value and policy iteration algorithms (see the author's books [9,10]). However, in this work, we will deemphasize mathematical proofs. There is considerable related analysis, which supports our conclusions and can be found in the author's recent RL books [11,12].

In summary, our discussion will aim to highlight the following:

(a) Approximation in value space is a single exact step of Newton's method for solving Bellman's equation. The Newton step may be preceded by enhancements of its starting point based on preliminary on-line adjustments and/or value iterations.

(b) The starting point for the Newton step of (a) is obtained by some unspecified off-line methodology, which may involve the solution of a related but simpler problem, and/or training with data that makes use of neural networks or feature-based architectures.

(c) The on-line play and off-line training parts of the AlphaZero/TD-Gammon design structure correspond to (a) and (b) above, respectively.

(d) The on-line player of AphaZero plays much better than its deep neural network-trained player for the same reason that the Newton step (a) improves substantially on its starting point (b), namely the underlying superlinear convergence property that is typical of Newton's method.

(e) ℓ-step lookahead minimization can be viewed as one-step lookahead minimization where $\ell - 1$ value iterations are used to enhance the starting point of the Newton step of (a) above.

(f) The algorithmic processes for (a) and (b) above can be designed by a variety of methods, and independently of each other. For example:

  (1) The implementation of the Newton step (a) may or may not involve any of the following: truncated rollout, on-line Monte Carlo simulation, MCTS or other efficient tree search techniques, forms of continuous space optimization, on-line policy iteration, etc.

  (2) The computation of the starting point (b) may or may not involve any of the following: Q-learning, approximate policy iteration based on temporal differences or aggregation, neural networks, feature-based function approximation, policies trained off-line by approximation in policy space, including policy gradient methods or policy random search, etc. Moreover, the details of this computation may vary broadly without affecting significantly the effectiveness of the overall scheme, which is primarily determined by the Newton step (a).

(g) An efficient implementation of the Newton step (a) is often critical in order to meet real-time constraints for generating controls, and to allow longer lookahead minimization, which typically enhances the starting point of the Newton step and its performance. By contrast, off-line training algorithms used for (b) have much less stringent real-time constraints, and the issues of sample efficiency and fine tuned performance, while important, are not critical.

(h) The efficient implementation of the Newton step may benefit from the use of distributed computation and other simplifications. A case in point is multiagent problems, which we will discuss later (see Section 3.6).

(i) Approximation in value space addresses effectively issues of robustness and on-line replanning for problems with changing parameters. The mechanism is similar to the one of indirect adaptive control: changing problem parameters are estimated on-line and a Newton step is used in place of an expensive full reoptimization of the controller. In the presence of changing

parameters, the Bellman equation changes, but the Newton step itself remains powerful and aims at the optimal solution that corresponds to the estimated system parameters.

(j) Model predictive control (MPC) has a conceptually similar structure to the AlphaZero-like programs, and entails an on-line play component involving multistep lookahead minimization, forms of truncated rollout, and an off-line training component to construct terminal cost approximations, and "safe" state space regions or reachability tubes to deal with state constraints. The success of MPC may be attributed to these similarities and to its resilience to changing problem parameters as per (i) above.

(k) On-line rollout with a stable policy yields a favorable starting point for the Newton step (a): it improves the stability properties of the policy obtained by approximation in value space, and it often provides an economical substitute for long lookahead minimization.

(l) Because the ideas outlined above are couched on principles of DP that often hold for arbitrary state and control spaces, they are valid within very general contexts: continuous-spaces control systems, discrete-spaces Markovian decision problems, control of hybrid systems, decision making in multiagent systems, and discrete and combinatorial optimization.

The preceding points are meant to highlight the essence of the connections between AlphaZero and TD-Gammon, approximation in value space, and decision and control. Naturally in practice there are exceptions and modifications, which need to be worked out in the context of particular applications, under appropriate assumptions. Moreover, while some results and elaborations are available through the research that has been done on approximate DP and on MPC, several of the results suggested by the analysis and insights of the present work remain to be rigorously established and enhanced within the context of specific problems.

The paper is structured as follows. In Section 2 we review the theory of classical infinite horizon optimal control problems, in order to provide some orientation and an analytical platform for what follows in subsequent sections. In Section 3, we introduce an abstract DP framework that will set the stage for the conceptual and visual interpretations of approximation in value space in terms of Newton's method. In this section, we also discuss on-line policy iteration, which aims to improve the on-line approximation in value space algorithm by using training data that is collected on-line. In Section 4, we discuss various issues of changing problem parameters, adaptive control, and MPC. Finally, we provide an appendix that deals with the convergence properties of Newton's method as it applies to Bellman equations that are nondifferentiable and commonly arise in DP.

2. Deterministic and stochastic dynamic programming

In this section we will review a classical optimal control framework, which we will use as a principal example for a more abstract DP framework to be introduced in Section 3. Let us consider a familiar class of stochastic optimal control problems over an infinite horizon. We have a stationary system of the form

$$x_{k+1} = f(x_k, u_k, w_k), \qquad k = 0, 1, \dots,$$

where x_k is an element of some state space X and the control u_k is an element of some control space U. The system includes a random "disturbance" w_k with a probability distribution $P(\cdot \mid x_k, u_k)$ that may depend explicitly on x_k and u_k, but not on values of prior disturbances $w_{k-1}, \dots, w_0$. The control u_k is constrained to take values in a given subset $U(x_k) \subset U$, which depends on the current state x_k. We are interested in policies $\pi = \{\mu_0, \mu_1, \dots\}$, such that each function μ_k maps states into controls, and satisfies $\mu_k(x_k) \in U(x_k)$ for all k. A stationary policy of the form $\{\mu, \mu, \dots\}$ will also be referred to as "policy μ". We make no assumptions on the state, control, and disturbances, and indeed for most of the discussion of this work, these spaces can be arbitrary.

We aim to minimize the expected total cost over an infinite number of stages, given by

$$J_\pi(x_0) = \lim_{N\to\infty} E\left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}, \tag{1}$$

where $\alpha^k g(x_k, u_k, w_k)$ is the cost of stage k, and $\alpha \in (0, 1]$ is a discount factor. If $\alpha = 1$ we refer to the problem as undiscounted. The expected value in Eq. (1) is taken with respect to the random disturbances w_k, $k = 0, 1, \dots$. Here, $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a policy $\pi = \{\mu_0, \mu_1, \dots\}$. The cost function of a stationary policy μ is denoted by J_μ. The optimal cost starting at state x, $\inf_\pi J_\pi(x)$, is denoted by $J^*(x)$, and the function J^* is referred to as the *optimal cost function*.

Here are some special cases, which will be of primary interest to us:

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state, denoted by t; once the system reaches t it remains there at no further cost. Usually, the termination state t represents a goal state that we are trying to reach at minimum cost; these are problems where the cost per stage is nonnegative, and will be of primary interest in this work. In some other types of problems, t may be a state that we are trying to avoid for as long as possible; these are problems where the cost per stage is nonpositive, and will not be specifically discussed in this work.

(b) *Discounted stochastic problems*. Here, $\alpha < 1$ and there need not be a termination state. However, there is a substantial connection between SSP and discounted problems. Aside from the fact that they are both infinite horizon total cost optimization problems, a discounted problem can be readily converted to an SSP problem. This can be done by introducing an artificial termination state to which the system moves with probability $1 - \alpha$ at every state and stage, thus making termination inevitable. Thus SSP and discounted problems share qualitative similarities in their respective theories.

(c) *Deterministic nonnegative cost problems*. Here, the disturbance w_k takes a single known value. Equivalently, there is no disturbance in the system equation and the cost expression, which now take the form

$$x_{k+1} = f(x_k, u_k), \qquad k = 0, 1, \ldots, \tag{2}$$

and

$$J_\pi(x_0) = \lim_{N\to\infty} \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k)\big). \tag{3}$$

We assume further that there is a cost-free and absorbing termination state t, and we have

$$g(x, u) \geq 0, \qquad \text{for all } x \neq t,\ u \in U(x), \tag{4}$$

and $g(t, u) = 0$ for all $u \in U(t)$. This type of structure expresses the objective to reach or approach t at minimum cost, a classical control problem. An extensive analysis of the undiscounted version of this problem was given in the author's paper [13].

Infinite horizon methodology

Many of the analytical and computational issues regarding infinite horizon problems revolve around the relation between the optimal cost function J^* of the problem and the optimal cost function of the corresponding N-stage problem. In particular, let $J_N(x)$ denote the optimal cost of the problem involving N stages, initial state x, cost per stage $g(x, u, w)$, and zero terminal cost. This cost is generated after N steps of the *value iteration* algorithm (VI for short)

$$J_{k+1}(x) = \min_{u\in U(x)} E\Big\{ g(x, u, w) + \alpha J_k\big(f(x, u, w)\big) \Big\}, \qquad k = 0, 1, \ldots, \tag{5}$$

starting from $J_0(x) \equiv 0$ (see Appendix). It is natural to speculate the following three basic properties[5]:

(1) The optimal infinite horizon cost is the limit of the corresponding N-stage optimal costs as $N \to \infty$:

$$J^*(x) = \lim_{N\to\infty} J_N(x) \tag{6}$$

for all states x.

(2) Bellman's equation holds:

$$J^*(x) = \min_{u\in U(x)} E\Big\{ g(x, u, w) + \alpha J^*\big(f(x, u, w)\big) \Big\}, \qquad \text{for all } x. \tag{7}$$

This equation can be viewed as the limit as $k \to \infty$ of the VI algorithm (5), assuming property (1) above holds and guarantees that $J_k(x) \to J^*(x)$ for all x. There is also a Bellman equation for each stationary policy μ. It is given by

$$J_\mu(x) = E\Big\{ g\big(x, \mu(x), w\big) + \alpha J_\mu\big(f(x, \mu(x), w)\big) \Big\}, \qquad \text{for all } x, \tag{8}$$

where J_μ is the cost function of μ.

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman Eq. (7) for each x, then the stationary policy μ should be optimal.

All three of the preceding results hold for discounted problems, provided the expected cost per stage $E\{g(x, u, w)\}$ is bounded over the set of possible values of (x, u, w) (see [9], Chapter 1). They also hold for finite-state SSP problems under reasonable assumptions. For deterministic problems with possibly infinite state and control spaces, there is substantial analysis that provides assumptions under which the results (1)-(3) above hold (see e.g., the DP book [9]).

The VI algorithm is also typically valid, in the sense that $J_k \to J^*$, even if the initial function J_0 is nonzero. The motivation for a different choice of J_0 is faster convergence to J^*; generally the convergence is faster as J_0 is chosen closer to J^*. The intuitive interpretation of the Bellman Eq. (7) is that it is the limit as $k \to \infty$ of the VI algorithm (5) assuming that $J_k \to J^*$. The optimality condition (3) indicates that optimal and near optimal policies can be obtained from within the class of stationary policies, something that is generally true for the problems that we discuss in this work, and that we will implicitly assume in what follows.

Aside from the VI algorithm, another fundamental algorithm is policy iteration (PI for short), which will be discussed in Section 3.3. It can be viewed as Newton's method for solving Bellman's equation, and is central for our purposes.

The author's paper [13], and also the abstract DP book [14], provide a detailed analysis of the undiscounted special case of the problem (2)–(4), where there is a cost-free and absorbing termination state t, the cost function is strictly positive for all other states, as in Eq. (4), and the objective is to reach or asymptotically approach the termination state. This analysis covers the preceding four properties, as well as the issue of convergence of PI, for the case of general state and control spaces (continuous or discrete or a mixture thereof). It delineates conditions under which favorable properties can be guaranteed.

[5] Throughout this work, we will be using "min" instead of the more formal "inf", even if we are not sure that the minimum is attained.

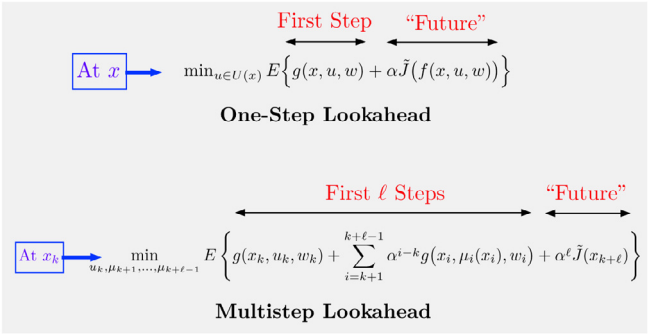

Fig. 4. Schematic illustration of approximation in value space with one-step and ℓ-step lookahead minimization. In the former case, the minimization yields at state x a control $\tilde{u}$, which defines the one-step lookahead policy $\tilde{\mu}$ via $\tilde{\mu}(x)=\tilde{u}$. In the latter case, the minimization yields a control $\tilde{u}_k$ and a sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$. The control $\tilde{u}_k$ is applied at x_k, and defines the ℓ-step lookahead policy $\tilde{\mu}$ via $\tilde{\mu}(x_k)=\tilde{u}_k$. The sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$ is discarded.

2.1. Approximation in value space – one-step and multistep lookahead – stability

A principal RL approach to deal with the often intractable exact computation of J^* is *approximation in value space*. Here in place of J^*, we use an approximation $\tilde{J}$, and generate at any state x, a control $\tilde{\mu}(x)$ by the *one-step lookahead minimization*

$$\tilde{\mu}(x)\in\arg\min_{u\in U(x)} E\Big\{g(x,u,w)+\alpha\tilde{J}\big(f(x,u,w)\big)\Big\}; \tag{9}$$

(we implicitly assume that the minimum above is attained for all x).[6] This minimization yields a stationary policy $\{\tilde{\mu},\tilde{\mu},\ldots\}$, with cost function denoted $J_{\tilde{\mu}}$ [i.e., $J_{\tilde{\mu}}(x)$ is the total infinite horizon discounted cost obtained when using $\tilde{\mu}$ starting at state x]. In the next section, *the change from $\tilde{J}$ to $J_{\tilde{\mu}}$ will be interpreted as a step of Newton's method for solving Bellman's equation*. Among others, this will suggest that $J_{\tilde{\mu}}$ is close to J^* and obeys a superlinear convergence relation

$$\lim_{\tilde{J}\to J^*}\frac{J_{\tilde{\mu}}(x)-J^*(x)}{\tilde{J}(x)-J^*(x)}=0,$$

for all states x. For specific types of problems, this relation represents a plausible result, which likely holds under appropriate conditions. This is similar to the use of Newton's method in numerical analysis, where its global or local convergence is guaranteed only under some assumptions. Within our context of approximate DP, however, *there is an important underlying structure, which is favorable and enhances the convergence properties of Newton's method, namely the monotonicity and concavity properties of Bellman's equation*, which we will discuss in the next section.

While it is desirable that $J_{\tilde{\mu}}$ is close to J^* in some sense, for classical control problems involving control to a goal state (e.g., problems with a cost-free and absorbing terminal state, and positive cost for all other states), stability of $\tilde{\mu}$ may be a principal objective. For the purposes of this work, we will focus on stability issues for just this one class of problems, and *we will consider the policy $\tilde{\mu}$ to be stable if $J_{\tilde{\mu}}$ is real-valued*, i.e.,

$$J_{\tilde{\mu}}(x)<\infty,\qquad \text{for all } x\in X.$$

Selecting $\tilde{J}$ so that $\tilde{\mu}$ is stable is a question of major interest, and will be addressed in Section 3.

ℓ-Step lookahead

An important extension of one-step lookahead minimization is *ℓ-step lookahead*, whereby at a state x_k we minimize the cost of the first $\ell>1$ stages with the future costs approximated by a function $\tilde{J}$ (see Fig. 4). This minimization yields a control $\tilde{u}_k$ and a sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$. The control $\tilde{u}_k$ is applied at x_k, and defines the ℓ-step lookahead policy $\tilde{\mu}$ via $\tilde{\mu}(x_k)=\tilde{u}_k$. The sequence $\tilde{\mu}_{k+1},\ldots,\tilde{\mu}_{k+\ell-1}$ is discarded. Actually, we may view ℓ-step lookahead minimization as the special case of its one-step counterpart where the lookahead function is the optimal cost function of an $(\ell-1)$-stage DP problem with a terminal cost $\tilde{J}(x_{k+\ell})$ on the state $x_{k+\ell}$ obtained after $\ell-1$ stages. In the next section, this will be interpreted as a step of Newton's method for solving Bellman's equation, starting from a function $\hat{J}$, which is an "improvement" over $\tilde{J}$. In particular, $\hat{J}$ is obtained from $\tilde{J}$ by applying $\ell-1$ successive value iterations.

The motivation for ℓ-step lookahead minimization is that *by increasing the value of ℓ, we may require a less accurate approximation $\tilde{J}$ to obtain good performance*. Otherwise expressed, for the same quality of cost function approximation, better performance may be obtained as ℓ becomes larger. This will be explained visually in the next section, and is also supported by error bounds, given for

[6] Note that the general theory of abstract DP is developed with the use of extended real-valued functions, and without the attainment of minimum assumption; see [10].

example in the books [11,12]. In particular, for AlphaZero chess, long multistep lookahead is critical for good on-line performance. *Another motivation for multistep lookahead is to enhance the stability properties of the generated on-line policy*. On the other hand, solving the multistep lookahead optimization problem, instead of the one-step lookahead counterpart of Eq. (9), is more time consuming.

A major issue in value space approximation is the construction of suitable approximate cost functions $\tilde{J}$. This can be done in many different ways, giving rise to some of the principal RL methods. For example, $\tilde{J}$ may be constructed with a sophisticated off-line training method, as discussed in Section 1, in connection with chess and backgammon. Alternatively, the approximate values $\tilde{J}(x)$ are obtained on-line as needed with truncated rollout, by running an off-line obtained policy for a suitably large number of steps, starting from x, and supplementing it with a suitable terminal cost approximation.

We refer to the neurodynamic programming book by Bertsekas and Tsitsiklis [15], the RL book by Sutton and Barto [16], as well as the large number of subsequent RL and approximate DP books, including the author's [11,12], which provide specific examples of cost function approximation methods and associated training algorithms.

3. An abstract view of reinforcement learning

In this section we will use geometric constructions to obtain insight into Bellman's equation, the value and policy iteration algorithms, approximation in value space, and some of the properties of the corresponding one-step or multistep lookahead policy $\tilde{\mu}$. To understand these constructions, we need an abstract notational framework. In particular, we denote by TJ the function of x that appears in the right-hand side of Bellman's equation. Its value at state x is given by[7]

$$(TJ)(x) = \min_{u\in U(x)} E\Big\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x. \tag{10}$$

Also for each policy μ, we introduce the corresponding function $T_\mu J$, which has value at x given by

$$(T_\mu J)(x) = E\Big\{ g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \tag{11}$$

Thus T and T_μ can be viewed as operators (broadly referred to as the *Bellman operators*), which map functions J to other functions (TJ or $T_\mu J$, respectively).[8]

An important property of the operators T and T_μ is that they are *monotone*, in the sense that if J and J' are two functions of x such that

$$J(x) \geq J'(x), \qquad \text{for all } x,$$

then we have

$$(TJ)(x) \geq (TJ')(x), \qquad (T_\mu J)(x) \geq (T_\mu J')(x), \qquad \text{for all } x \text{ and } \mu. \tag{12}$$

This is evident from Eqs. (10) and (11).

Another important property is that the Bellman operator T_μ is *linear*, in the sense that it has the form $T_\mu J = G + A_\mu J$, where $G \in R(X)$ is some function and $A_\mu : R(X) \mapsto R(X)$ is an operator such that for any functions J_1, J_2, and scalars γ_1, γ_2, we have[9]

$$A_\mu(\gamma_1 J_1 + \gamma_2 J_2) = \gamma_1 A_\mu J_1 + \gamma_2 A_\mu J_2.$$

Moreover, from the definitions (10) and (11), we have

$$(TJ)(x) = \min_{\mu\in\mathcal{M}} (T_\mu J)(x), \qquad \text{for all } x,$$

where $\mathcal{M}$ is the set of stationary policies. This is true because for any policy μ, there is no coupling constraint between the controls $\mu(x)$ and $\mu(x')$ that correspond to two different states x and x'. It follows that $(TJ)(x)$ *is a concave function of* J *for every* x, something that will be important for our interpretation of one-step and multistep lookahead as a Newton iteration for solving the Bellman equation $J = TJ$.

Example 1 (*A Two-State and Two-Control Example*)**.** Assume that there are two states 1 and 2, and two controls u and v. Consider the policy μ that applies control u at state 1 and control v at state 2. Then the operator T_μ takes the form

$$(T_\mu J)(1) = \sum_{y=1}^{2} p_{1y}(u)\big(g(1,u,y) + \alpha J(y)\big), \tag{13}$$

$$(T_\mu J)(2) = \sum_{y=1}^{2} p_{2y}(v)\big(g(2,v,y) + \alpha J(y)\big), \tag{14}$$

[7] Recall here our convention that we will be using "min" instead of the more formal "inf", even we are not sure that minimum is attained.

[8] Within the context of this work, the functions J on which T and T_μ operate will be real-valued functions of x. We will assume throughout that the expected values in Eqs. (10) and (11) are well-defined and finite when J is real-valued. This implies that $T_\mu J$ will also be real-valued functions of x. On the other hand $(TJ)(x)$ may take the value $-\infty$ because of the minimization in Eq. (10). We allow this possibility, although our illustrations will primarily depict the case where TJ is real-valued. Note that the general theory of abstract DP is developed with the use of extended real-valued functions; see [10].

[9] An operator T_μ with this property is often called "affine", but in this work we just call it "linear". Also we use abbreviated notation to express pointwise equalities and inequalities, so that we write $J = J'$ or $J \geq J'$ to express the fact that $J(x) = J'(x)$ or $J(x) \geq J'(x)$, for all x, respectively.

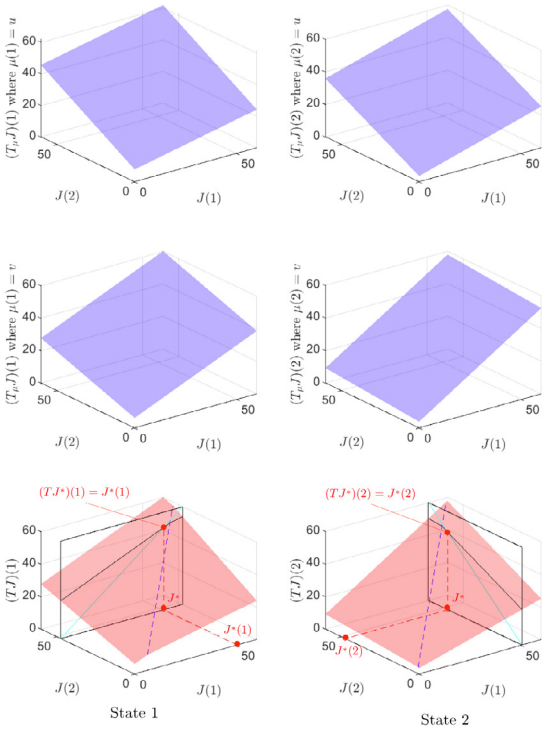

Fig. 5. Geometric illustrations of the Bellman operators T_μ and T for states 1 and 2 in Example 1; cf. Eqs. (13)–(16). The problem's transition probabilities are: $p_{11}(u) = 0.3$, $p_{12}(u) = 0.7$, $p_{21}(u) = 0.4$, $p_{22}(u) = 0.6$, $p_{11}(v) = 0.6$, $p_{12}(v) = 0.4$, $p_{21}(v) = 0.9$, $p_{22}(v) = 0.1$. The stage costs are $g(1, u, 1) = 3$, $g(1, u, 2) = 10$, $g(2, u, 1) = 0$, $g(2, u, 2) = 6$, $g(1, v, 1) = 7$, $g(1, v, 2) = 5$, $g(2, v, 1) = 3$, $g(2, v, 2) = 12$. The discount factor is $\alpha = 0.9$, and the optimal costs are $J^*(1) = 50.59$ and $J^*(2) = 47.41$. The optimal policy is $\mu^*(1) = v$ and $\mu^*(2) = u$. The figure also shows two one-dimensional slices of T that are parallel to the $J(1)$ and $J(2)$ axes and pass through J^*.

where $p_{xy}(u)$ and $p_{xy}(v)$ are the probabilities that the next state will be y, when the current state is x, and the control is u or v, respectively. Clearly, $(T_\mu J)(1)$ and $(T_\mu J)(2)$ are linear functions of J. Also the operator T of the Bellman equation $J = TJ$ takes the form

$$(TJ)(1) = \min\left[\sum_{y=1}^{2} p_{1y}(u)\big(g(1,u,y) + \alpha J(y)\big), \sum_{y=1}^{2} p_{1y}(v)\big(g(1,v,y) + \alpha J(y)\big)\right], \tag{15}$$

$$(TJ)(2) = \min\left[\sum_{y=1}^{2} p_{2y}(u)\big(g(2,u,y) + \alpha J(y)\big), \sum_{y=1}^{2} p_{2y}(v)\big(g(2,v,y) + \alpha J(y)\big)\right]. \tag{16}$$

Thus, $(TJ)(1)$ and $(TJ)(2)$ are concave and piecewise linear as functions of the two-dimensional vector J (with two pieces; more generally, as many linear pieces as the number of controls). This concavity property holds in general since $(TJ)(x)$ is the minimum of a collection of linear functions of J, one for each $u \in U(x)$. Fig. 5 illustrates $(T_\mu J)(1)$ for the cases where $\mu(1) = u$ and $\mu(1) = v$, $(T_\mu J)(2)$ for the cases where $\mu(2) = u$ and $\mu(2) = v$, $(TJ)(1)$, and $(TJ)(2)$, as functions of $J = \big(J(1), J(2)\big)$.

Critical properties from the DP point of view are whether T and T_μ have fixed points; equivalently, whether the Bellman equations $J = TJ$ and $J = T_\mu J$ have solutions within the class of real-valued functions, and whether the set of solutions includes J^* and J_μ, respectively. It may thus be important to verify that T or T_μ are contraction mappings. This is true for example in the benign case of discounted problems with bounded cost per stage. However, for undiscounted problems, asserting the contraction property of T or T_μ may be more complicated, and even impossible; the abstract DP book [10] deals extensively with such questions, and related issues regarding the solution sets of the Bellman equations.

Geometrical interpretations

We will now interpret the Bellman operators geometrically, starting with T_μ. Fig. 6 illustrates its form. Note here that the functions J and $T_\mu J$ are multidimensional. They have as many scalar components $J(x)$ and $(T_\mu J)(x)$, respectively, as there are states x, but they can only be shown projected onto one dimension. The function $T_\mu J$ for each policy μ is linear. The cost function J_μ satisfies $J_\mu = T_\mu J_\mu$, so it is obtained from the intersection of the graph of $T_\mu J$ and the 45 degree line, when J_μ is real-valued. Later we will interpret the case where J_μ is not real-valued as the system being unstable under μ [we have $J_\mu(x) = \infty$ for some initial states x].

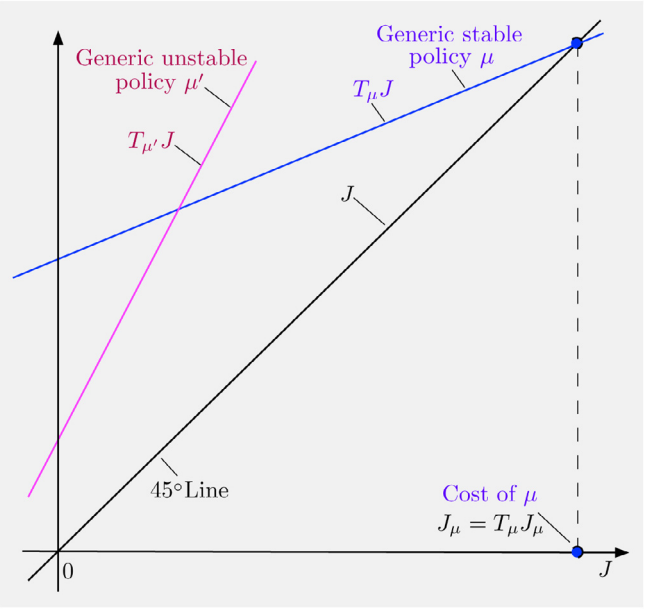

Fig. 6. Geometric interpretation of the linear Bellman operator T_μ and the corresponding Bellman equation. The graph of T_μ is a linear manifold in the space $R(X) \times R(X)$, and when projected on a one-dimensional plane that corresponds to a single state and passes through J_μ, it becomes a line. Then there are three cases: [(a)] The line has slope less than 45 degrees, so it intersects the 45-degree line at a unique point, which is equal to J_μ, the solution of the Bellman equation $J = T_\mu J$. This is true if T_μ is a contraction mapping, as is the case for discounted problems with bounded cost per stage. [(b)] The line has slope greater than 45 degrees. Then it intersects the 45-degree line at a unique point, which is a solution of the Bellman equation $J = T_\mu J$, but is not equal to J_μ. Then J_μ is not real-valued; we will call such μ *unstable* under μ in Section 3.2. [(c)] The line has slope exactly equal to 45 degrees. This is an exceptional case where the Bellman equation $J = T_\mu J$ has an infinite number of real-valued solutions or no real-valued solution at all.

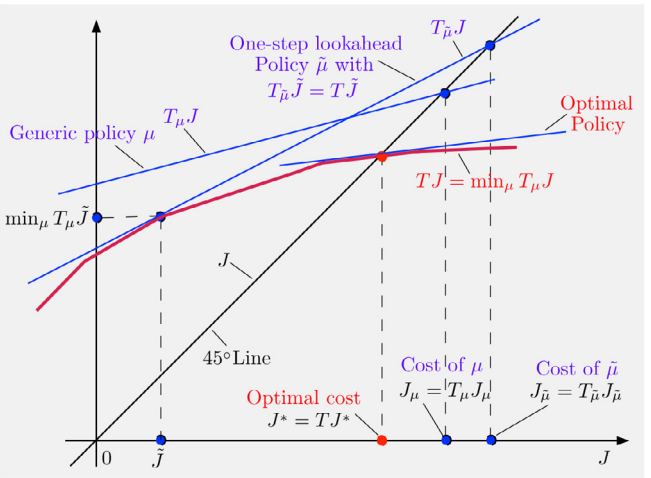

Fig. 7. Geometric interpretation of the Bellman operator T, and the corresponding Bellman equation. For a fixed x, the function $(TJ)(x)$ can be written as $\min_\mu (T_\mu J)(x)$, so it is concave as a function of J. The optimal cost function J^* satisfies $J^* = TJ^*$, so it is obtained from the intersection of the graph of TJ and the 45 degree line shown, assuming J^* is real-valued. Note that the graph of T lies below the graph of every operator T_μ, and is in fact obtained as the lower envelope of the graphs of T_μ as μ ranges over the set of policies $\mathcal{M}$. In particular, for any given function $\tilde J$, for every x, the value $(T\tilde J)(x)$ is obtained by finding a support hyperplane/subgradient of the graph of the concave function $(TJ)(x)$ at $J = \tilde J$, as shown in the figure. This support hyperplane is defined by the control $\mu(x)$ of a policy $\tilde\mu$ that attains the minimum of $(T_\mu \tilde J)(x)$ over μ: $\tilde\mu(x) \in \arg\min_{\mu\in\mathcal{M}} (T_\mu \tilde J)(x)$ (there may be multiple policies attaining this minimum, defining multiple support hyperplanes). This construction also shows how the minimization $(T\tilde J)(x) = \min_{\mu\in\mathcal{M}} (T_\mu \tilde J)(x)$ corresponds to a linearization of the mapping T at the point $\tilde J$.

The form of the Bellman operator T is illustrated in Fig. 7. Again the functions J, J^*, TJ, $T_\mu J$, etc, are multidimensional, but they are shown projected onto one dimension (alternatively they are illustrated for a system with a single state, plus possibly a

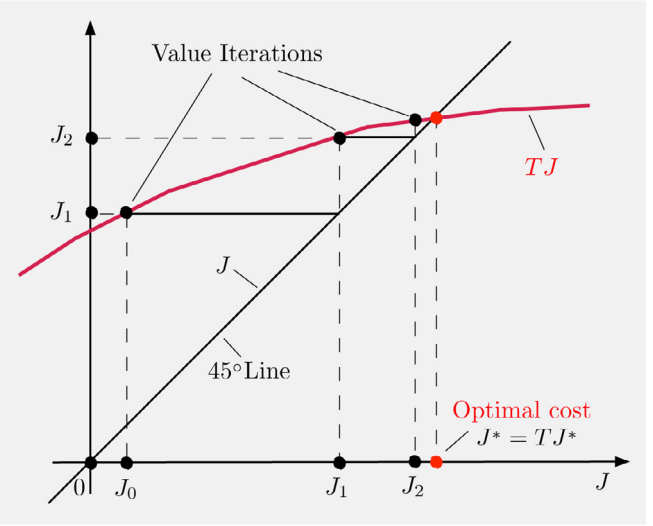

Fig. 8. Geometric interpretation of the VI algorithm $J_{k+1} = TJ_k$, starting from some initial function J_0. Successive iterates are obtained through the staircase construction shown in the figure. The VI algorithm $J_{k+1} = T_\mu J_k$ for a given policy μ can be similarly interpreted, except that the graph of the function $T_\mu J$ is linear.

termination state). The Bellman equation $J = TJ$ may have one or many real-valued solutions. It may also have no real-valued solution in exceptional situations. The figure assumes a unique real-valued solution of the Bellman equations $J = TJ$ and $J = T_\mu J$, which is true if T and T_μ are contraction mappings, as is the case for discounted problems with bounded cost per stage. Otherwise, these equations may have no solution or multiple solutions within the class of real-valued functions. The equation $J = TJ$ typically has J^* as a solution, but may have more than one solution in cases where either $\alpha = 1$, or $\alpha < 1$ and the cost per stage is unbounded.

Visualization of value iteration

The operator notation simplifies algorithmic descriptions, derivations, and proofs related to DP. For example, we can write the VI algorithm in the compact form

$$J_{k+1} = TJ_k, \qquad k = 0, 1, \ldots,$$

as illustrated in Fig. 8. Moreover, the VI algorithm for a given policy μ can be written as

$$J_{k+1} = T_\mu J_k, \qquad k = 0, 1, \ldots,$$

and it can be similarly interpreted, except that the graph of the function $T_\mu J$ is linear. Also we will see shortly that there is a similarly compact description for the policy iteration algorithm.

To keep the presentation simple, we will focus our attention on the abstract DP framework as it applies to the optimal control problems of Section 2.1. In particular, we will assume without further mention that *T and T_μ have the monotonicity property (12), that $T_\mu J$ is linear for all μ, and that (as a consequence) the component $(TJ)(x)$ is concave as a function of J for every state x*. We note, however, that the abstract notation facilitates the extension of the infinite horizon DP theory to models beyond the ones that we discuss in this work. Such models include semi-Markov problems, minimax control problems, risk sensitive problems, Markov games, and others (see the DP textbook [9], and the abstract DP monograph [10]).

3.1. Approximation in value space and Newton's method

Let us now consider approximation in value space and an abstract geometric interpretation, first provided in the author's book [12]. By using the operators T and T_μ, for a given $\tilde{J}$, a one-step lookahead policy $\tilde{\mu}$ is characterized by the equation

$$T_{\tilde{\mu}}\tilde{J} = T\tilde{J},$$

or equivalently

$$T_{\tilde{\mu}}\tilde{J}(x) = T\tilde{J}(x), \qquad \text{for all } x,$$

as in Fig. 9 [cf. Eq. (9)]. In mathematical terms, for each state $x \in X$ the hyperplane $H_{\tilde{\mu}}(x) \in R(X) \times \Re$

$$H_{\tilde{\mu}}(x) = \Big\{ \big(J, \xi\big) \mid (T_{\tilde{\mu}}J)(x) = \xi \Big\}, \tag{17}$$

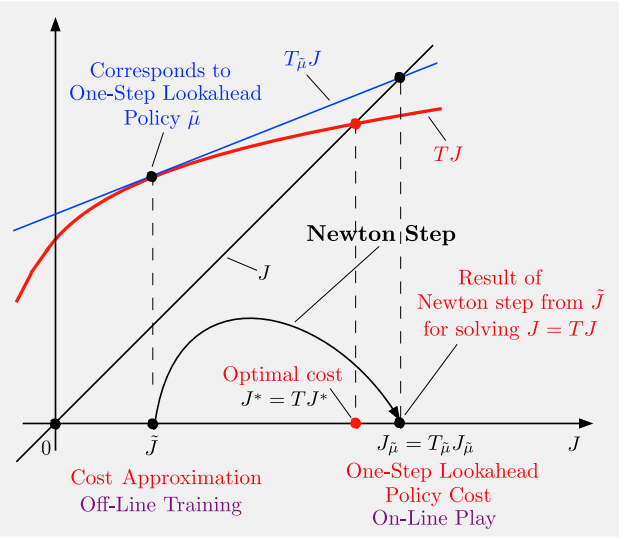

Fig. 9. Geometric interpretation of approximation in value space and the one-step lookahead policy $\tilde{\mu}$ as a step of Newton's method [cf. Eq. (9)]. Given $\tilde{J}$, we find a policy $\tilde{\mu}$ that attains the minimum in the relation $T\tilde{J} = \min_{\mu} T_{\mu}\tilde{J}$. This policy satisfies $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$, so the graph of TJ and $T_{\tilde{\mu}}J$ touch at $\tilde{J}$, as shown. It may not be unique. Because TJ has concave components, the equation $J = T_{\tilde{\mu}}J$ is the linearization of the equation $J = TJ$ at $\tilde{J}$ [at each x, the hyperplane $H_{\tilde{\mu}}(x)$ of Eq. (17) defines a subgradient of (T J)(x) at $\tilde{J}$]. The linearized equation is solved at the typical step of Newton's method to provide the next iterate, which is just $J_{\tilde{\mu}}$.

supports from above the hypograph of the concave function $(TJ)(x)$, i.e., the convex set

$$\Big\{ \big(J, \xi\big) \mid (TJ)(x) \geq \xi \Big\}.$$

The point of support is $(\tilde{J}, (T_{\tilde{\mu}}\tilde{J})(x))$, and relates the function $\tilde{J}$ with the corresponding one-step lookahead minimization policy $\tilde{\mu}$, the one that satisfies $T_{\tilde{\mu}}\tilde{J} = T\tilde{J}$. The hyperplane $H_{\tilde{\mu}}(x)$ of Eq. (17) defines a subgradient of $(TJ)(x)$ at $\tilde{J}$. at the point $\big(\tilde{J}(x), (T\tilde{J})(x)\big)$ and defines a subgradient of $(TJ)(x)$ at $\tilde{J}$. Note that the one-step lookahead policy $\tilde{\mu}$ need not be unique, since T need not be differentiable, so there may be multiple hyperplanes of support at $\tilde{J}$. Still this construction shows that the linear operator $T_{\tilde{\mu}}$ is a linearization of the operator T at the point $\tilde{J}$ (pointwise for each x).

Equivalently, for every $x \in X$, the linear scalar equation $J(x) = (T_{\tilde{\mu}}J)(x)$ is a linearization of the nonlinear equation $J(x) = (TJ)(x)$ at the point $\tilde{J}$. Consequently, the linear operator equation $J = T_{\tilde{\mu}}J$ is a linearization of the equation $J = TJ$ at $\tilde{J}$, and its solution, $J_{\tilde{\mu}}$, can be viewed as the result of a Newton iteration at the point $\tilde{J}$ (here we adopt an expanded view of the Newton iteration that applies to possibly nondifferentiable fixed point equations; see the Appendix). In summary, *the Newton iterate at $\tilde{J}$ is $J_{\tilde{\mu}}$*, the solution of the linearized equation $J = T_{\tilde{\mu}}J$.[10]

[10] The classical Newton's method for solving a fixed point problem of the form $y = G(y)$, where y is an n-dimensional vector, operates as follows: At the current iterate y_k, we linearize G and find the solution y_{k+1} of the corresponding linear fixed point problem. Assuming G is differentiable, the linearization is obtained by using a first order Taylor expansion:

$$y_{k+1} = G(y_k) + \frac{\partial G(y_k)}{\partial y}(y_{k+1} - y_k),$$

where $\partial G(y_k)/\partial y$ is the $n \times n$ Jacobian matrix of G evaluated at the vector y_k. The most commonly given convergence rate property of Newton's method is *quadratic convergence*. It states that near the solution y^*, we have

$$\|y_{k+1} - y^*\| = O\big(\|y_k - y^*\|^2\big),$$

where $\|\cdot\|$ is the Euclidean norm, and holds assuming the Jacobian matrix exists, is invertible, and is Lipschitz continuous (see the books by Ortega and Rheinboldt [17], and by the author [18], Section 1.4). There are well studied extensions of Newton's method that are based on solving a linearized system at the current iterate, but relax the differentiability requirement through alternative requirements of piecewise differentiability, B-differentiability, and semi-smoothness, while maintaining the superlinear convergence property of the method. Relevant works include Josephy [19], Robinson [20], [21], [22], Kojima and Shindo [23], Kummer [24], [25], Pang [26], Qi and Sun [27], [28], Facchinei and Pang [29], Ito and Kunisch [30], Bolte, Daniilidis, and Lewis [31], Dontchev and Rockafellar [32], and additional references cited therein. The Newton method extensions in these works have strong relevance to our context. In particular, the quadratic rate of convergence result for differentiable G of Prop. 1.4.1 of the book [18] admits a straightforward and intuitive extension to piecewise differentiable G, given in the paper [23]; see the Appendix. The structure of the Bellman operators Eq. (10) and Eq. (11), with their monotonicity and concavity properties, tends to enhance the convergence and rate of convergence properties of Newton's method, even in the absence of differentiability, as evidenced by the convergence analysis of PI, and the extensive favorable experience with rollout, PI, and MPC. In fact, the role of monotonicity and concavity in affecting the convergence properties of Newton's method has been addressed in the mathematical literature; see the papers by Ortega and Rheinboldt [33], and Vandergraft [34], the books by Ortega and Rheinboldt [17], and Argyros [35], and the references cited there. In this connection, it is worth noting that in the case of Markov games, where the concavity property does not hold, the PI method may oscillate, as shown by Pollatschek and Avi-Itzhak [36], and needs to be modified to restore its global convergence; see the author's paper [37], and the references cited there.

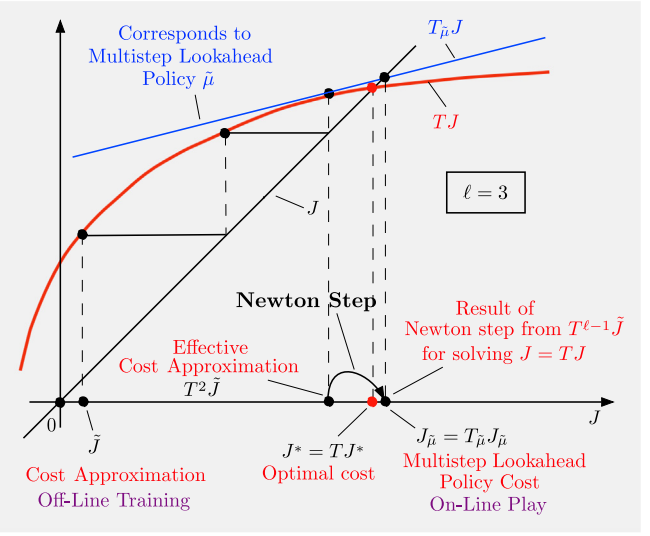

Fig. 10. Geometric interpretation of approximation in value space with ℓ-step lookahead (in this figure $\ell = 3$). It is the same as approximation in value space with one-step lookahead using $T^{\ell-1}\tilde{J}$ as cost approximation. It can be viewed as a Newton step at the point $T^{\ell-1}\tilde{J}$, the result of $\ell - 1$ value iterations applied to $\tilde{J}$. Note that as ℓ increases the cost function $J_{\tilde{\mu}}$ of the ℓ-step lookahead policy $\tilde{\mu}$ approaches more closely the optimal J^*, and that $\lim_{\ell\to\infty} J_{\tilde{\mu}} = J^*$.

As noted earlier, approximation in value space with ℓ-step lookahead using $\tilde{J}$ is the same as approximation in value space with one-step lookahead using the $(\ell - 1)$-fold operation of T on $\tilde{J}$, $T^{\ell-1}\tilde{J}$. Thus it can be interpreted as a Newton step starting from $T^{\ell-1}\tilde{J}$, the result of $\ell - 1$ value iterations applied to $\tilde{J}$. This is illustrated in Fig. 10. In this connection, we note that several variants of Newton's method that involve combinations of first-order iterative methods, such as the Gauss–Seidel and Jacobi algorithms, and Newton's method, are well-known in numerical analysis. They belong to the general family of *Newton-SOR methods* (SOR stands for "successive over-relaxation"); see the classic book by Ortega and Rheinboldt [17] (Section 13.4). Their convergence rate is superlinear, similar to Newton's method, as long as they involve a pure Newton step, along with the first-order steps.

Let us also note that ℓ-step lookahead minimization involves ℓ successive VI iterations, but only the first of these iterations has a Newton step interpretation. As an example, consider two-step lookahead minimization with a terminal cost approximation $\tilde{J}$. The second step minimization is a VI that starts from $\tilde{J}$ to produce $T\tilde{J}$. The first step minimization is a VI that starts from $T\tilde{J}$ to produce $T^2\tilde{J}$, but it also does something else that is more significant: It produces a two-step lookahead minimization policy $\tilde{\mu}$ through $T_{\tilde{\mu}}(T\tilde{J}) = T(T\tilde{J})$, and the step from $T\tilde{J}$ to $J_{\tilde{\mu}}$ (the cost function of $\tilde{\mu}$) is the Newton step. Thus, there is only one policy produced (i.e., $\tilde{\mu}$) and only one Newton step (from $T\tilde{J}$ to $J_{\tilde{\mu}}$). In the case of one-step lookahead minimization, the Newton step starts from $\tilde{J}$ and ends at $J_{\tilde{\mu}}$. Similarly, in the case of ℓ-step lookahead minimization, the first step of the lookahead is the Newton step (from $T^{\ell-1}\tilde{J}$ to $J_{\tilde{\mu}}$), and whatever follows the first step of the lookahead is preparation for the Newton step.

3.2. Region of stability

For any control system design method, the stability of the policy obtained is of paramount importance. It is thus essential to investigate and verify the stability of controllers obtained through approximation in value space schemes. Historically, there have been several proposed definitions of stability in control theory. Within the context of this work, our focus on stability issues will be for problems with a termination state t, which is cost-free, and with a cost per stage that is positive outside the termination state, such as the undiscounted positive cost deterministic problem introduced earlier [cf. Eqs. (2)–(4)]. Moreover, it is best for our purposes to adopt an optimization-based definition. In particular, we say that a policy μ is *unstable* if $J_\mu(x) = \infty$ for some states x. Equivalently, we say that the policy μ *stable* if $J_\mu(x) < \infty$ for all states x. This definition has the advantage that it applies to general state and control spaces. Naturally, it can be made more specific in particular problem instances.[11]

In the context of approximation in value space we are interested in the *region of stability*, which is the set of cost function approximations $\tilde{J} \in R(X)$ for which the corresponding one-step or multistep lookahead policies $\tilde{\mu}$ are stable. For discounted problems with bounded cost per stage, all policies have real-valued cost functions, so questions of stability do not arise. In general, however, the region of stability may be a strict subset of the set of real-valued functions;.

An interesting observation from Fig. 11 is that if $\tilde{J}$ does not belong to the region of stability and $\tilde{\mu}$ is a corresponding one-step lookahead unstable policy, the Bellman equation $J = T_{\tilde{\mu}}J$ may have real-valued solutions. However, these solutions will not be

[11] For the undiscounted positive cost deterministic problem introduced earlier [cf. Eqs. (2)–(4)], it can be shown that if a policy μ is stable, then J_μ is the "smallest" solution of the Bellman equation $J = T_\mu J$ within the class of nonnegative real-valued functions, and under mild assumptions it is the unique solution of $J = T_\mu J$ within the class of nonnegative real-valued functions J with $J(t) = 0$; see the author's paper [13]. Moreover, if μ is unstable, then the Bellman equation $J = T_\mu J$ has no solution within the class of nonnegative real-valued functions.

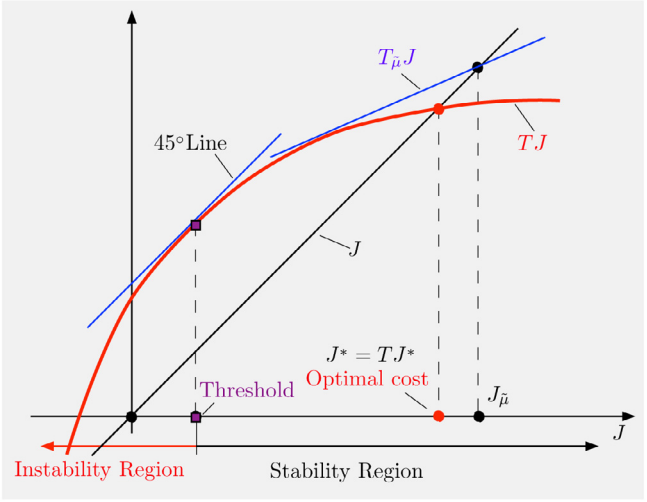

Fig. 11. Illustration of the regions of stability and instability for approximation in value space with one-step lookahead. The stability region is the set of all $\tilde{J}$ such that the policy $\tilde{\mu}$ with $T\tilde{J} = T_{\tilde{\mu}}\tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x.

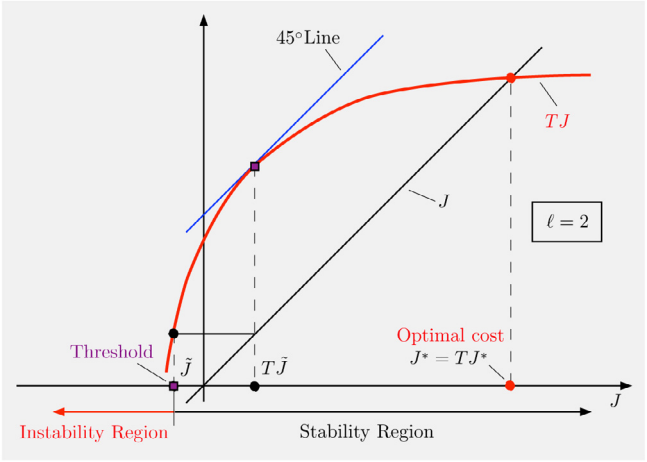

Fig. 12. Illustration of the regions of stability and instability for approximation in value space with multistep lookahead. The stability region is the set of all $\tilde{J}$ for which the policy $\tilde{\mu}$ such that $T^\ell\tilde{J} = T_{\tilde{\mu}}T^{\ell-1}\tilde{J}$ satisfies $J_{\tilde{\mu}}(x) < \infty$ for all x (the figure shows the case $\ell = 2$). The region of instability tends to be reduced as ℓ increases.

equal to $J_{\tilde{\mu}}$, as this would violate the definition of region of stability. Generally, if T_μ is not a contraction mapping, T_μ may have real-valued fixed points, none of which is equal to J_μ.

Fig. 12 illustrates the region of stability for the case of multistep lookahead. The insights from this figure are similar to the one-step lookahead case of Fig. 11. However, the figure indicates that *the region of stability of the ℓ-step lookahead controller $\tilde{\mu}$ depends on ℓ, and tends to become larger as ℓ increases*. The reason is that ℓ-step lookahead with terminal cost $\tilde{J}$ is equivalent to one-step lookahead with terminal cost $T^{\ell-1}\tilde{J}$, which tends to be closer to the optimal cost function J^* than $\tilde{J}$ (assuming convergence of the VI method).

How can we obtain function approximations $\tilde{J}$ within the region of stability?

Naturally, identifying and obtaining cost function approximations $\tilde{J}$ that lie within the region of stability with either one-step or multistep lookahead is very important within our context. We will focus on this question for the special case where the expected cost per stage is nonnegative

$$E\{g(x,u,w)\} \geq 0, \qquad \text{for all } x,\ u \in U(x),$$

and assume that J^* is real-valued. This is the case of most interest in model predictive control, but also arises in other problems of interest, including stochastic shortest path problems that involve a termination state.

From Fig. 12 it can be conjectured that if the sequence $\{T^k\tilde{J}\}$ generated by the VI algorithm converges to J^* for all $\tilde{J}$ such that $0 \le \tilde{J} \le J^*$ (which is true under very general conditions; see [9,14]), then $T^{\ell-1}\tilde{J}$ belongs to the region of stability for sufficiently large ℓ. Related ideas have been discussed in the adaptive DP literature by Liu and his collaborators [38–40], and by Heydari [41,42], who provide extensive references; see also Winnicki et al. [43]. This conjecture is generally true, but requires that, in addition to J^*, all functions $\tilde{J}$ within a neighborhood of J^* belong to the region of stability. Methods to address this difficulty are discussed in the book [10].

An important fact in our context is that *the region of stability includes all real-valued nonnegative functions $\tilde{J}$ such that*

$$T\tilde{J} \le \tilde{J}. \tag{18}$$

Indeed if $\tilde{\mu}$ is the corresponding one-step lookahead policy, we have

$$T_{\tilde{\mu}}\tilde{J} = T\tilde{J} \le \tilde{J},$$

and from a well-known result on nonnegative cost infinite horizon problems [see [9], Prop. 4.1.4(a)], it follows that

$$J_{\tilde{\mu}} \le \tilde{J};$$

(the proof argument is that if $T_{\tilde{\mu}}\tilde{J} \le \tilde{J}$ then $T_{\tilde{\mu}}^{k+1}\tilde{J} \le T_{\tilde{\mu}}^{k}\tilde{J}$ for all k, so, using also the fact $0 \le \tilde{J}$, the limit of $T_{\tilde{\mu}}^{k}\tilde{J}$, call it J_∞, satisfies $J_{\tilde{\mu}} \le J_\infty \le \tilde{J}$). Thus if $\tilde{J}$ is nonnegative and real-valued, $J_{\tilde{\mu}}$ is also real-valued, so $\tilde{\mu}$ is stable. It follows that $\tilde{J}$ belongs to the region of stability. This is a known result in specific contexts, such as MPC (see the book by Rawlings, Mayne, and Diehl [44], Section 2.4, which contains extensive references to prior work on stability issues).

An important special case where the condition $T\tilde{J} \le \tilde{J}$ is satisfied is when $\tilde{J}$ is the cost function of a stable policy, i.e., $\tilde{J} = J_\mu$. Then we have that J_μ is real-valued and satisfies $T_\mu J_\mu = J_\mu$, so it follows that $TJ_\mu \le J_\mu$. This case relates to the rollout algorithm and shows that *rollout with a stable policy yields a stable lookahead policy*. It also suggests that if μ is stable, then $T_\mu^m\tilde{J}$ belongs to the region of stability for sufficiently large m.

3.3. Policy iteration, rollout, and Newton's method

Another major class of infinite horizon algorithms is based on *policy iteration* (PI for short), which involves the repeated use of policy improvement, in analogy with the AlphaZero/TD-Gammon off-line training algorithms, described in Section 1. Each iteration of the PI algorithm starts with a stable policy (which we call *current* or *base* policy), and generates another stable policy (which we call *new* or *rollout* policy, respectively). For the infinite horizon problem of Section 2.1, given the base policy μ, the iteration consists of two phases (see Fig. 13):

(a) *Policy evaluation*, which computes the cost function J_μ. One possibility is to solve the corresponding Bellman equation

$$J_\mu(x) = E\Big\{g\big(x,\mu(x),w\big) + \alpha J_\mu\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x. \tag{19}$$

However, the value $J_\mu(x)$ for any x can also be computed by Monte Carlo simulation, by averaging over many randomly generated trajectories the cost of the policy starting from x. Other, more sophisticated possibilities include the use of specialized simulation-based methods, such as *temporal difference methods*, for which there is extensive literature (see e.g., the books [9,15,16]).

(b) *Policy improvement*, which computes the rollout policy $\tilde{\mu}$ using the one-step lookahead minimization

$$\tilde{\mu}(x) \in \arg\min_{u\in U(x)} E\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x. \tag{20}$$

It is generally expected (and can be proved under mild conditions) that the rollout policy is improved in the sense that $J_{\tilde{\mu}}(x) \le J_\mu(x)$ for all x. Proofs of this fact in a variety of contexts can be found in most DP books, including the author's [9–12,14].

Thus PI generates a sequence of stable policies $\{\mu^k\}$, by obtaining μ^{k+1} through a policy improvement operation using J_{μ^k} in place of J_μ in Eq. (20), which is obtained through policy evaluation of the preceding policy μ^k using Eq. (19). It is well known that (exact) PI has solid convergence properties; see the DP textbooks cited earlier, as well as the author's RL book [11]. These properties hold even when the method is implemented (with appropriate modifications) in unconventional computing environments, involving asynchronous distributed computation, as shown in a series of papers by Bertsekas and Yu [45–47].

In terms of our abstract notation, the PI algorithm can be written in a compact form. For the generated policy sequence $\{\mu^k\}$, the policy evaluation phase obtains J_{μ^k} from the equation

$$J_{\mu^k} = T_{\mu^k}J_{\mu^k}, \tag{21}$$

while the policy improvement phase obtains μ^{k+1} through the equation

$$T_{\mu^{k+1}}J_{\mu^k} = TJ_{\mu^k}. \tag{22}$$

As Fig. 14 illustrates, PI can be viewed as Newton's method for solving the Bellman equation in the function space of cost functions J. In particular, *the policy improvement Eq. (22) is the Newton step starting from J_{μ^k}, and yields μ^{k+1} as the corresponding one-step lookahead/rollout policy*. Fig. 15 illustrates the rollout algorithm, which is just the first iteration of PI.

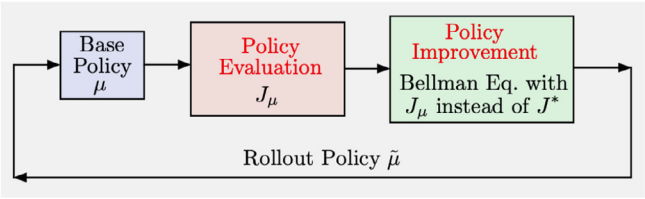

Fig. 13. Schematic illustration of PI as repeated rollout. It generates a sequence of policies, with each policy μ in the sequence being the base policy that generates the next policy $\tilde{\mu}$ in the sequence as the corresponding rollout policy.

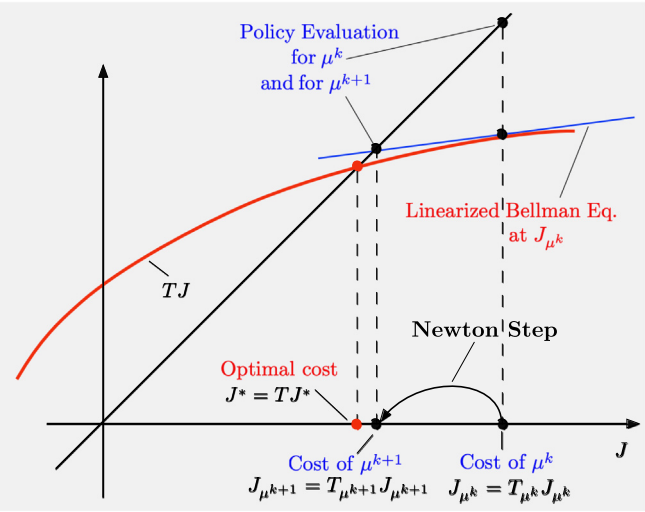

Fig. 14. Geometric interpretation of a policy iteration. Starting from the stable current policy μ^k, it evaluates the corresponding cost function J_{μ^k}, and computes the next policy μ^{k+1} according to $T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}$. The corresponding cost function $J_{\mu^{k+1}}$ is obtained as the solution of the linearized equation $J = T_{\mu^{k+1}} J$, so it is the result of a Newton step for solving the Bellman equation $J = TJ$, starting from J_{μ^k}. Note that in policy iteration, the Newton step always starts at a function J_μ, which satisfies $J_\mu \geq J^*$ as well as $TJ_\mu \leq J_\mu$ (cf. our discussion on stability in Section 3.2).

In contrast to approximation in value space, the interpretation of PI in terms of Newton's method has a long history. We refer to the original works for linear quadratic problems by Kleinman [48],[12] and for finite-state infinite horizon discounted and Markov game problems by Pollatschek and Avi-Itzhak [36] (who also showed that the method may oscillate in the game case). Subsequent works, which discuss algorithmic variations and approximations, include Hewer [52], Puterman and Brumelle [53,54], Saridis and Lee [55] (following Rekasius [56]), Beard [57], Beard, Saridis, and Wen [58], Santos and Rust [59], Bokanowski, Maroso, and Zidani [60], Hylla [61], Magirou, Vassalos, and Barakitis [62], Bertsekas [37], and Kundu and Kunitsch [63]. Some of these papers address broader classes of problems (such as continuous-time optimal control, minimax problems, and Markov games), include superlinear convergence rate results, as well as extensive references to related works. For a recent proof of quadratic convergence for linear discrete-time quadratic problems, see Lopez, Alsalti, and Muller [64].

Rollout

Generally, rollout with a stable base policy μ can be viewed as a single iteration of Newton's method starting from J_μ, as applied to the solution of the Bellman equation (see Fig. 15). Note that rollout/policy improvement is applied just at the current state during real-time operation of the system. This makes the on-line implementation possible, even for problems with very large state space, provided that the policy evaluation of the base policy can be done on-line as needed. For this we often need on-line deterministic or stochastic simulation from each of the states x_k generated by the system in real time.

As Fig. 15 illustrates, the cost function of the rollout policy $J_{\tilde{\mu}}$ is obtained by constructing a linearized version of Bellman's equation at J_μ (its linear approximation at J_μ), and then solving it. If the function TJ is nearly linear (i.e., has small "curvature") the rollout policy performance $J_{\tilde{\mu}}(x)$ is very close to the optimal $J^*(x)$, even if the base policy μ is far from optimal. This explains the large cost improvements that are typically observed in practice with the rollout algorithm.

[12] This was part of Kleinman's Ph.D. thesis [49] at M.I.T., supervised by M. Athans. Kleinman gives credit for the one-dimensional version of his results to Bellman and Kalaba [50]. Note also that the first proposal of the PI method was given by Bellman in his classic book [51], under the name "approximation in policy space".

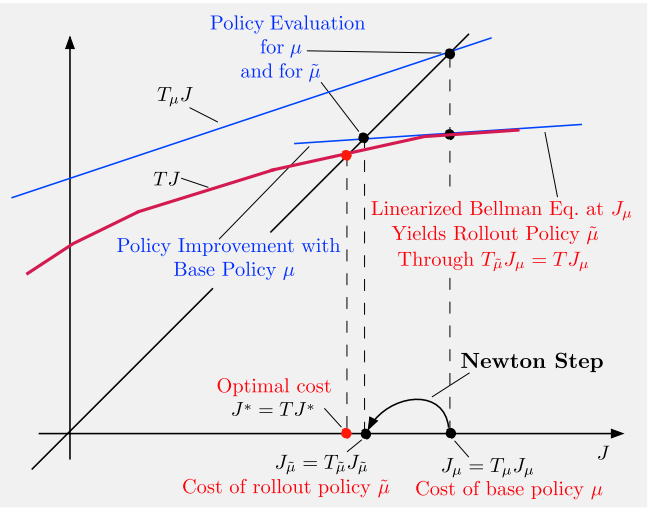

Fig. 15. Geometric interpretation of rollout. Each policy μ defines the linear function $T_\mu J$ of J, given by Eq. (11), and TJ is the function given by Eq. (10), which can also be written as $TJ = \min_\mu T_\mu J$. The figure shows a policy iteration starting from a base policy μ. It computes J_μ by policy evaluation (by solving the linear equation $J = T_\mu J$ as shown). It then performs a policy improvement using μ as the base policy to produce the rollout policy $\tilde{\mu}$ as shown: the cost function of the rollout policy, $J_{\tilde{\mu}}$, is obtained by solving the version of Bellman's equation that is linearized at the point J_μ, as in Newton's method.

An interesting question is how to compare the rollout performance $J_{\tilde{\mu}}(x)$ for a given initial state x, with the base policy performance $J_\mu(x)$. Clearly, we would like $J_\mu(x) - J_{\tilde{\mu}}(x)$ to be large, but this is not the right way to look at cost improvement. The reason is that $J_\mu(x) - J_{\tilde{\mu}}(x)$ will be small if its upper bound, $J_\mu(x) - J^*(x)$, is small, i.e., if the base policy is close to optimal. What is important is that the error ratio

$$\frac{J_{\tilde{\mu}}(x) - J^*(x)}{J_\mu(x) - J^*(x)} \tag{23}$$

is small. Indeed, this ratio becomes smaller as $J_\mu(x) - J^*(x)$ approaches 0 because of the superlinear convergence rate of Newton's method that underlies the rollout algorithm (cf. Fig. 15). Unfortunately, it is hard to evaluate this ratio, since we do not know $J^*(x)$. On the other hand, we should not be underwhelmed if we observe a small performance improvement $J_\mu(x) - J_{\tilde{\mu}}(x)$: the reason may be that the base policy is already near-optimal, and in fact we are likely doing very well in terms of the ratio (23).

Truncated rollout

Variants of rollout may involve multistep lookahead, truncation, and terminal cost function approximation, as in the case of AlphaZero/TD-Gammon, cf. Section 1. These variants admit geometric interpretations that are similar to the ones given earlier; see Fig. 16. Truncated rollout uses m VIs with the base policy μ and a terminal cost function approximation $\tilde{J}$ to approximate the cost function J_μ.

In the case of one-step lookahead, the truncated rollout policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T_\mu^m \tilde{J}) = T(T_\mu^m \tilde{J}),$$

i.e., $\tilde{\mu}$ attains the minimum when the Bellman operator T is applied to the function $T_\mu^m \tilde{J}$ (the cost obtained by using the base policy μ for m steps followed by terminal cost approximation $\tilde{J}$); see Fig. 16. In the case of ℓ-step lookahead, the truncated rollout policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T^{\ell-1} T_\mu^m \tilde{J}) = T(T^{\ell-1} T_\mu^m \tilde{J}).$$

Truncated rollout is related to a variant of PI called *optimistic*. This variant approximates the policy evaluation step by using m value iterations using the base policy μ; see [9,11,15] for a more detailed discussion of this relation.

As noted earlier, variants of Newton's method that involve multiple fixed point iterations, before and after each Newton step, but without truncated rollout, i.e.,

$$T_{\tilde{\mu}}(T^{\ell-1} \tilde{J}) = T(T^{\ell-1} \tilde{J}), \tag{24}$$

are well-known. The classical numerical analysis book by Ortega and Rheinboldt [17] (Sections 13.3 and 13.4) provides various convergence results, under assumptions that include differentiability and convexity of the components of T, and nonnegativity of the inverse Jacobian of T. These assumptions, particularly differentiability, may not be satisfied within our DP context. Moreover, for methods of the form (24), the initial point must satisfy an additional assumption, which ensures that the convergence to J^*

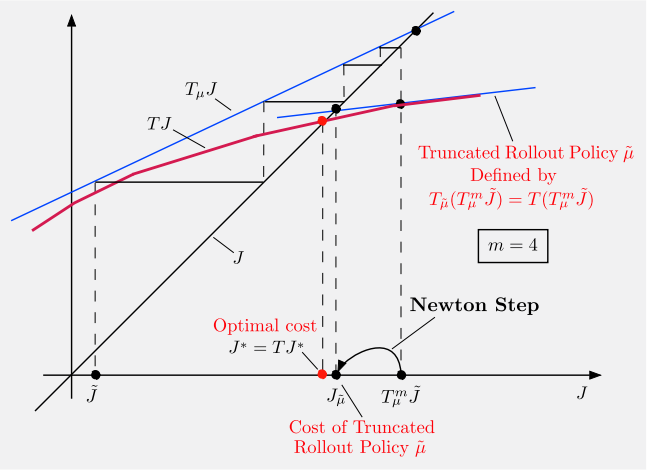

Fig. 16. Geometric interpretation of truncated rollout with one-step lookahead minimization, m value iterations with the base policy μ, and a terminal cost function approximation $\tilde{J}$ (here $m = 4$).

is monotonic from above (in this case, if in addition the Jacobian of T is isotone, an auxiliary sequence can be constructed that converges to J^* monotonically from below; see [17], 13.3.4, 13.4.2). This is similar to existing convergence results for the optimistic PI method in DP; see e.g., [9,15].

Geometrical interpretations such as the ones of Fig. 16 suggest, among others, that:

(a) The cost improvement $J_\mu - J_{\tilde{\mu}}$, from base to rollout policy, tends to become larger as the length ℓ of the lookahead increases.

(b) Truncated rollout with ℓ-step lookahead minimization, followed by m steps of a base policy μ, and then followed by terminal cost function approximation $\tilde{J}$ may be viewed, under certain conditions, as an economic alternative to $(\ell + m)$-step lookahead minimization using $\tilde{J}$ as terminal cost function approximation.

We next discuss the issues of selection of ℓ and m.

Lookahead length issues in truncated rollout

A question of practical interest is how to choose the lookahead lengths ℓ and m in truncated rollout schemes. It is clear that large values ℓ for lookahead minimization are beneficial (in the sense of producing improved lookahead policy cost functions $J_{\tilde{\mu}}$), since additional VI iterations bring closer to J^* the starting point $T^{\ell-1}\tilde{J}$ of the Newton step. On the other hand, large values m for truncated rollout bring the starting point for the Newton step closer to J_μ, and not necessarily closer to J^*. Indeed computational experiments suggest that *increasing values for m may be counterproductive beyond some threshold*, and that this threshold generally depends on the problem and the terminal cost approximation $\tilde{J}$. This is also consistent with long standing experience with optimistic policy iteration, which is closely connected with truncated rollout, as noted earlier. Unfortunately, however, there is no analysis that can illuminate this issue, and the available error bounds for truncated rollout (see [11,12]) are conservative and provide limited guidance in this regard.

An interesting property, which holds in some generality, is that *truncated rollout with a stable policy has a beneficial effect on the stability properties of the lookahead policy*. The reason is that the cost function J_μ of the base policy μ lies well inside the region of stability, as noted in Section 3.2. Moreover value iterations with μ (i.e., truncated rollout) tend to push the starting point of the Newton step towards J_μ. Thus a sufficient number of these value iterations will bring the starting point of the Newton step within the region of stability.

Another important fact to keep in mind is that the truncated rollout steps are much less demanding computationally than the lookahead minimization steps. In particular, large values of m may be computationally tolerable, but even relatively small values of m can be computationally prohibitive. This is especially true for stochastic problems where the width of the lookahead tree tends to grow quickly. Thus, with other concerns weighing equally, it is computationally preferable to use large values of m rather than large values of ℓ (this was the underlying motivation for truncated rollout in Tesauro's TD-Gammon [5]).

The preceding discussion suggests the following qualitative question: *is lookahead by rollout an economic substitute for lookahead by minimization?* The answer to this seems to be a qualified yes: for a given computational budget, judiciously balancing the values of m and ℓ tends to give better lookahead policy performance than simply increasing ℓ as much as possible, while setting $m = 0$ (which corresponds to no rollout). This is consistent with intuition obtained through geometric constructions such as Fig. 16, but it is difficult to establish conclusively; see also the discussion on linear quadratic problems in [10].

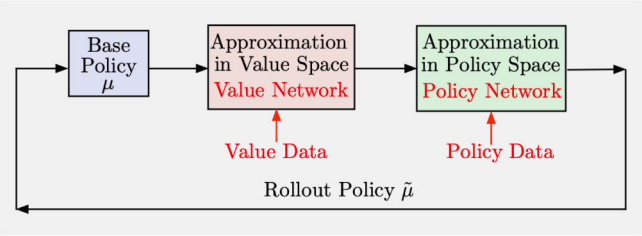

Fig. 17. Schematic illustration of approximate PI. Either the policy evaluation and policy improvement phases (or both) are approximated with a value or a policy network, respectively. These could be neural networks, which are trained with (state, cost function value) data that is generated using the current base policy μ, and with (state, rollout policy control) data that is generated using the rollout policy $\tilde{\mu}$. Note that there are three different types of approximate implementation involving: (1) a value network but no policy network (here the value network defines a policy via one-step or multistep lookahead), or (2) a policy network but no value network (here the policy network has a corresponding value function that can be computed by rollout), or (3) both a policy and a value network (the approximation architecture of AlphaZero is a case in point).

3.4. How sensitive is on-line play to the off-line training process?

An important issue to consider in approximation in value space is errors in the one-step or multistep minimization, or in the choice of terminal cost approximation $\tilde{J}$. Such errors are often unavoidable because the control constraint set $U(x)$ is infinite, or because the minimization is simplified for reasons of computational expediency (see our subsequent discussion of multiagent problems). Moreover, to these errors, we may add the effect of errors due to rollout truncation, and errors due to changes in problem parameters, which are reflected in changes in Bellman's equation (see our subsequent discussion of robust and adaptive control).

Under these circumstances the linearization of the Bellman equation at the point $\tilde{J}$ in Fig. 16 is perturbed, and the corresponding point $T_\mu^m \tilde{J}$ in Fig. 16 is also perturbed. However, the effect of these perturbations tends to be mitigated by the Newton step that produces the policy $\tilde{\mu}$ and the corresponding cost function $J_{\tilde{\mu}}$. The Newton step has a superlinear convergence property, so for an $O(\epsilon)$-order error [i.e., $O(\epsilon)/\epsilon$ stays bounded as $\epsilon \to 0$] in the calculation of $T_\mu^m \tilde{J}$, the error in $J_{\tilde{\mu}}$ will be of the much smaller order $o(\epsilon)$ [i.e., $o(\epsilon)/\epsilon \to 0$ as $\epsilon \to 0$], when $J_{\tilde{\mu}}$ is near J^*.[13] This is a significant insight, as it suggests that *extreme accuracy and fine-tuning of the choice of $\tilde{J}$ may not produce significant effects in the resulting performance of the one-step and particularly a multistep lookahead policy*.

Approximate policy iteration and implementation errors

Both policy evaluation and policy improvement can be approximated, possibly by using training with data and approximation architectures, such as neural networks; see Fig. 17. Other approximations include simulation-based methods such as truncated rollout, and temporal difference methods for policy evaluation, which involve the use of basis functions. Moreover, multistep lookahead may be used in place of one-step lookahead, and simplified minimization, based for example on multiagent rollout, may also be used. Let us also mention the possibility of a combined rollout and PI algorithm, whereby we use PI for on-line policy improvement of the base policy, by using data collected during the rollout process; see the discussion in [10] and the author's paper [65].

Long-standing practical experience with approximate PI is consistent with the view of the effect of implementation errors outlined above, and suggests that substantial changes in the policy evaluation and policy improvement operations often have small but largely unpredictable effects on the performance of the policies generated. For example, when TD(λ)-type methods are used for policy evaluation, the choice of λ has a large effect on the generated policy cost function approximations, but often has little and unpredictable effect on the performance of the generated policies. A plausible conjecture here is that the superlinear convergence property of the exact Newton step "smooths out" the effect of off-line approximation errors.

3.5. Why not just train a policy network and use it without on-line play?

This is a sensible and common question, which stems from the mindset that neural networks have extraordinary function approximation properties. In other words, why go through the arduous on-line process of lookahead minimization, if we can do the same thing off-line and represent the lookahead policy with a trained policy network? More generally, it is possible to use *approximation in policy space*, a major alternative approach to approximation in value space, whereby we select the policy from a suitably restricted class of policies, such as a parametric class of the form $\mu(x, r)$, where r is a parameter vector. We may then estimate r using some type of off-line training process. There are quite a few methods for performing this type of training, such as policy gradient and random search methods (see the books [11,16] for an overview). Alternatively, some approximate DP or classical control system design method may be used.

[13] A rigorous proof of this requires differentiability of T at $\tilde{J}$. Since T is differentiable at almost all points J, the sensitivity property just stated, will likely hold in practice even if T is not differentiable.

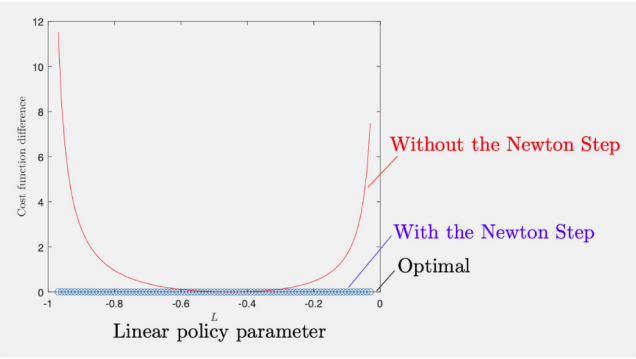

Fig. 18. Illustration of the performance enhancement obtained by rollout with an off-line trained base policy for a linear quadratic problem. Here the system equation is linear of the form $x_{k+1} = ax_k + bu_k$, and the cost per stage is positive definite quadratic of the form $qx^2 + ru^2$. The figure shows an instance where the optimal policy is $\mu^*(x) = L^*x$ with $L^* \approx -0.4$, and the optimal cost function is $J^*(x) = K^*x^2$, where $K^* \approx 1.1$. We consider policies of the form $\mu(x) = Lx$, where L is the parameter, with cost function of the form $J_\mu(x) = K_L x^2$. The figure shows the quadratic cost coefficient differences $K_L - K^*$ and $K_{\tilde{L}} - K^*$ as a function of L, where K_L and $K_{\tilde{L}}$ are the quadratic cost coefficients of μ (without one-step lookahead/Newton step) and the corresponding one-step lookahead policy $\tilde{\mu}$ (with one-step lookahead/Newton step).

An important advantage of approximation in policy space is that once the parametrized policy is obtained, the on-line computation of controls $\mu(x, r)$ is often much faster compared with on-line lookahead minimization. For this reason, approximation in policy space can be used to provide an approximate implementation of a known policy (no matter how obtained) for the purpose of convenient use. On the negative side, because parametrized approximations often involve substantial calculations, they are not well suited for on-line replanning, which is necessary in the context of adaptive control (see Section 4).

From our point of view in this paper, there is another important reason why approximation in value space is needed on top of approximation in policy space: *the off-line trained policy may not perform nearly as well as the corresponding one-step or multistep lookahead/rollout policy*, because it lacks the extra power of the associated exact Newton step (cf. our discussion of AlphaZero and TD-Gammon in Section 1). Fig. 18 illustrates this fact with a one-dimensional linear-quadratic example, and compares the performance of a linear policy, defined by a scalar parameter, with its corresponding one-step lookahead policy.

3.6. Multiagent problems and multiagent rollout

A major difficulty in the implementation of value space approximation with one-step lookahead is the minimization operation over $U(x_k)$. When $U(x_k)$ is infinite, or even when it is finite but has a very large number of elements, the minimization may become prohibitively time consuming. In the case of multistep lookahead the computational difficulty becomes even more acute. In this section we discuss how to deal with this difficulty when the control u consists of m components, $u = (u_1, \ldots, u_m)$, with a separable control constraint for each component, $u_\ell \in U_\ell(x)$, $\ell = 1, \ldots, m$. Thus the control constraint set is the Cartesian product

$$U(x) = U_1(x) \times \cdots \times U_m(x), \tag{25}$$

where the sets $U_\ell(x)$ are given. This structure is inspired by applications involving distributed decision making by multiple agents with communication and coordination between the agents; see Fig. 19.

To illustrate our approach, let us consider the discounted infinite horizon problem, and for the sake of the following discussion, assume that each set $U_\ell(x)$ is finite. Then the one-step lookahead minimization of the standard rollout scheme with base policy μ is given by

$$\tilde{u} \in \arg\min_{u \in U(x)} E\Big\{ g(x, u, w) + \alpha J_\mu\big(f(x, u, w)\big) \Big\}, \tag{26}$$

and involves as many as n^m terms, where n is the maximum number of elements of the sets $U_\ell(x)$ [so that n^m is an upper bound to the number of controls in $U(x)$, in view of its Cartesian product structure (25)]. Thus the standard rollout algorithm requires an exponential [order $O(n^m)$] number of computations per stage, which can be overwhelming even for moderate values of m.

This potentially large computational overhead motivates a far more computationally efficient rollout algorithm, whereby the one-step lookahead minimization (26) is replaced by a sequence of m successive minimizations, *one-agent-at-a-time*, with the results incorporated into the subsequent minimizations. In particular, at state x we perform the sequence of minimizations

$$\begin{aligned}
\tilde{\mu}_1(x) \in \arg\min_{u_1 \in U_1(x)} E_w \Big\{ & g(x, u_1, \mu_2(x), \ldots, \mu_m(x), w) \\
& + \alpha J_\mu\big(f(x, u_1, \mu_2(x), \ldots, \mu_m(x), w)\big) \Big\}, \\
\tilde{\mu}_2(x) \in \arg\min_{u_2 \in U_2(x)} E_w \Big\{ & g(x, \tilde{\mu}_1(x), u_2, \mu_3(x) \ldots, \mu_m(x), w) \\
& + \alpha J_\mu\big(f(x, \tilde{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x), w)\big) \Big\},
\end{aligned}$$

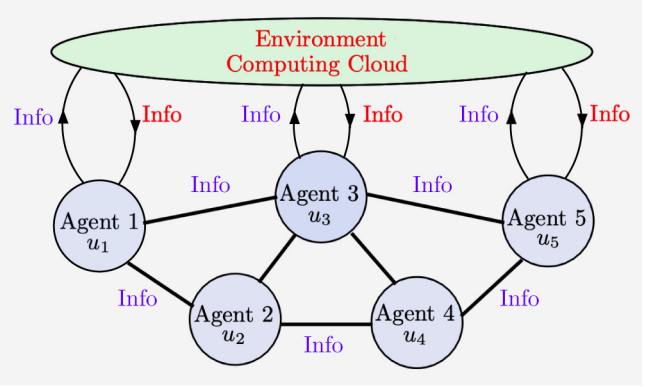

Fig. 19. Schematic illustration of a multiagent problem. There are multiple "agents", and each agent $\ell = 1, \dots, m$, controls its own decision variable u_ℓ. At each stage, agents exchange new information and also exchange information with the "environment", and then select their decision variables for the stage.

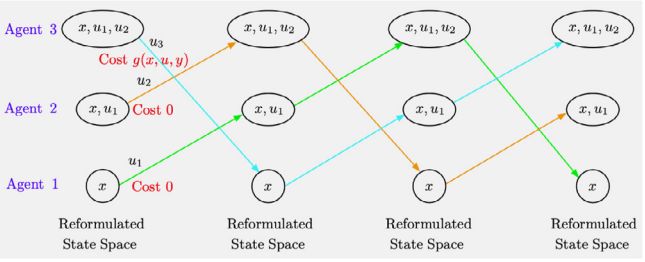

Fig. 20. Equivalent formulation of the stochastic optimal control problem for the case where the control u consists of m components $u_1, u_2, \dots, u_m$: $u = (u_1, \dots, u_m) \in U_1(x_k) \times \cdots \times U_m(x_k)$. The figure depicts the kth stage transitions. Starting from state x, we generate the intermediate states $(x, u_1), (x_k, u_1, u_2), \dots, (x, u_1, \dots, u_{m-1})$, using the respective controls $u_1, \dots, u_{m-1}$. The final control u_m leads from $(x, u_1, \dots, u_{m-1})$ to $\bar{x} = f(x, u, w)$, and the random cost $g(x, u, w)$ is incurred.

$$\cdots$$

$$\tilde{\mu}_m(x) \in \arg \min_{u_m \in U_m(x)} E_w \Big\{ g(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \dots, \tilde{\mu}_{m-1}(x), u_m, w) + \alpha J_\mu \big(f(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \dots, \tilde{\mu}_{m-1}(x), u_m, w) \big) \Big\}.$$

Thus each agent component u_ℓ is obtained by a minimization with the preceding agent components $u_1, \dots, u_{\ell-1}$ fixed at the previously computed values of the rollout policy, and the following agent components $u_{\ell+1}, \dots, u_m$ fixed at the values given by the base policy. This algorithm requires order $O(nm)$ computations per stage, a potentially huge computational saving over the order $O(n^m)$ computations required by standard rollout.

A key idea here is that the computational requirements of the rollout one-step minimization (26) are proportional to the number of controls in the set $U_k(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the book [15], Section 6.1.4, whereby control space complexity is traded off with state space complexity, by "unfolding" the control u_k into its m components, which are applied *one-agent-at-a-time* rather than all-agents-at-once.

In particular, we can reformulate the problem by breaking down the collective decision u_k into m sequential component decisions, thereby reducing the complexity of the control space while increasing the complexity of the state space. The potential advantage is that the extra state space complexity does not affect the computational requirements of some RL algorithms, including rollout.

To this end, we introduce a modified but equivalent problem, involving one-at-a-time agent control selection. At the generic state x, we break down the control u into the sequence of the m controls $u_1, u_2, \dots, u_m$, and between x and the next state $\bar{x} = f(x, u, w)$, we introduce artificial intermediate "states" $(x, u_1), (x, u_1, u_2), \dots, (x, u_1, \dots, u_{m-1})$, and corresponding transitions. The choice of the last control component u_m at "state" $(x, u_1, \dots, u_{m-1})$ marks the transition to the next state $\bar{x} = f(x, u, w)$ according to the system equation, while incurring cost $g(x, u, w)$; see Fig. 20.

It is evident that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, while the cost structure of the two problems is the same. In particular, every policy $\big(\mu_1(x), \dots, \mu_m(x)\big)$ of the original problem, is admissible for the reformulated problem, and has the same cost function for the original as well as the reformulated problem. Reversely, every policy for the reformulated problem can be converted into a policy for the original problem that produces the same state and control trajectories and has the same cost function.

The motivation for the reformulated problem is that the control space is simplified at the expense of introducing $m-1$ additional layers of states, and the corresponding $m-1$ cost functions

$$J^1(x, u_1), J^2(x, u_1, u_2), \dots, J^{m-1}(x, u_1, \dots, u_{m-1}).$$

The increase in size of the state space does not adversely affect the operation of rollout, since the minimization (26) is performed for just one state at each stage.

A major fact that follows from the preceding reformulation is that multiagent rollout still *achieves cost improvement*:

$$J_{\tilde{\mu}}(x) \leq J_{\mu}(x), \qquad \text{for all } x,$$

where $J_{\mu}(x)$ is the cost function of the base policy μ, and $J_{\tilde{\mu}}(x)$ is the cost function of the rollout policy $\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)$, starting from state x. Furthermore, this cost improvement property can be extended to multiagent PI schemes that involve one-agent-at-a-time policy improvement operations, and have sound convergence properties (see the book [12], Chapters 3 and 5, as well as the author's papers [66–69], and [70]).

Another fact that follows from the preceding reformulation is that *multiagent rollout may be viewed as a Newton step applied to the Bellman equation that corresponds to the reformulated problem*. This is very important for our purposes. In particular, *the superlinear cost improvement of the Newton step can still be obtained through multiagent rollout*, even though the amount of computation for the lookahead minimization is dramatically reduced through one-agent-at-a-time minimization. This explains experimental results given in the paper by Bhattacharya et al. [70], which have shown comparable performance for multiagent and standard rollout in the context of a large-scale multi-robot POMDP application.

We finally note that multiagent rollout can become the starting point for various related PI schemes that are well suited for distributed operation in important practical contexts involving multiple autonomous decision makers (see the book [12], Section 5.3.4, and the paper [69]).

4. Robustness, adaptive control, and model predictive control

Our discussion so far dealt with problems with a known and unchanging mathematical model, i.e., one where the system equation, cost function, control constraints, and probability distributions of disturbances are perfectly known. The mathematical model may be available through explicit mathematical formulas and assumptions, or through a computer program that can emulate all of the mathematical operations involved in the model, including Monte Carlo simulation for the calculation of expected values. From our point of view, it makes no difference whether the mathematical model is available through closed form mathematical expressions or through a computer simulator: the methods that we discuss are valid either way, only their suitability for a given problem may be affected by the availability of mathematical formulas.

In practice, however, it is common that the system involves parameters that are either not known exactly or may change over time. In such cases it is important to design controllers that take the parameter changes into account. The methodology for doing so is generally known as *adaptive control*, an intricate and multifaceted subject, with many and diverse applications, and a long history.

We should note also that unknown problem environments are an integral part of the artificial intelligence view of RL. In particular, to quote from the book by Sutton and Barto [16], "learning from interaction with the environment is a foundational idea underlying nearly all theories of learning and intelligence". The idea of interaction with the environment is typically connected with the idea of exploring the environment to identify its characteristics. In control theory this is often viewed as part of the *system identification* methodology, which aims to construct mathematical models of dynamic systems. The system identification process is often combined with the control process to deal with unknown or changing problem parameters. This is one of the most challenging areas of stochastic optimal and suboptimal control, and has been studied extensively since the early 1960s.

In what follows in this section, we will briefly review some of the principal types of adaptive control methods. We will then focus on schemes that are based on on-line replanning, including the use of rollout.

Robust and PID control

Given a controller design that has been obtained assuming a nominal DP problem model, one possibility is to simply ignore changes in problem parameters. We may then try to design a controller that is adequate for the entire range of the changing parameters. This is sometimes called a *robust controller*. A robust controller makes no effort to keep track of changing problem parameters. It is just designed so that it is resilient to parameter changes, and in practice, it often tends to be biased towards addressing the worst case.

An important time-honored robust control approach for continuous-state problems is the *PID (Proportional-Integral-Derivative) controller*; see e.g., the books by Aström and Hagglund [71,72]. In particular, PID control aims to maintain the output of a single-input single-output dynamic system around a set point or to follow a given trajectory, as the system parameters change within a relatively broad range. In its simplest form, the PID controller is parametrized by three scalar parameters, which may be determined by a variety of methods, some of them manual/heuristic. PID control is used widely and with success, although its range of application is mainly restricted to single-input, single-output continuous-state control systems.

Dealing with unknown parameters by system identification and on-line replanning

In robust control schemes, such as PID control, no attempt is made to maintain a mathematical model and to track unknown model parameters as they change. Alternatively we may introduce into the controller a mechanism for measuring or estimating the unknown or changing system parameters, and employ suitable controller reoptimization in response. In the adaptive control literature, schemes of this type are called *indirect*, while schemes that do not involve parameter estimation (like PID control) are

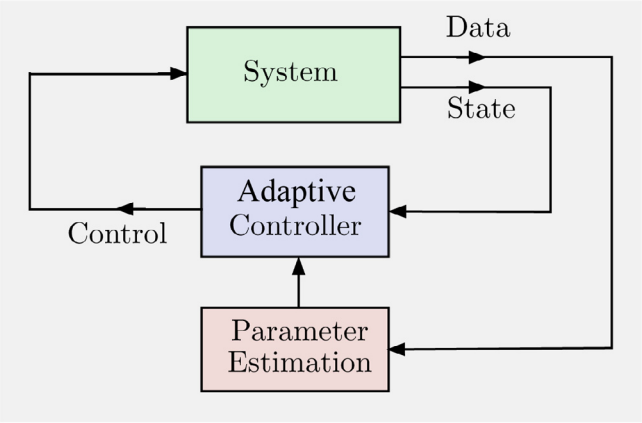

Fig. 21. Schematic illustration of concurrent parameter estimation and system control. The system parameters are estimated on-line and the estimates are passed on to the controller whenever this is desirable (e.g., after the estimates change substantially). This structure is also known as indirect adaptive control.

called *direct*. The textbook literature on the subkect is extensive; see Aström and Wittenmark [73], Bodson [74], Goodwin and Sin [75], Ioannou and Sun [76], Jiang and Jiang [77], Krstic, Kanellakopoulos, and Kokotovic [78], Kokotovic [79], Kumar and Varaiya [80], Liu, et al. [81], Lavretsky and Wise [82], Narendra and Annaswamy [83], Sastry and Bodson [84], Slotine and Li [85], and Vrabie, Vamvoudakis, and Lewis [86].

In what follows in this section (including our discussion of MPC in Section 4.2), we will assume that there is a mechanism to learn (perhaps imperfectly and by some unspecified procedure) the model of the system as it evolves over time. We will loosely refer to this learning process with the classical name *system identification*, but we will not go into specific identification methods, keeping in mind that such methods could be imprecise and challenging, but could also be fast and simple, depending on the problem at hand.

An apparently reasonable scheme is to separate the control process into two phases, a *system identification phase* and a *control phase*. In the first phase the unknown parameters are estimated, while the control takes no account of the interim results of estimation. The final parameter estimates from the first phase are then used to implement an optimal or suboptimal policy in the second phase.

This alternation of estimation and control phases may be repeated several times during the system's operation in order to take into account subsequent changes of the parameters. Note that it is not necessary to introduce a hard separation between the identification and the control phases. They may be going on simultaneously, with new parameter estimates being generated in the background, and introduced into the control process, whenever this is thought to be desirable; see Fig. 21.

One drawback of this approach is that it is not always easy to determine when to terminate one phase and start the other. A second difficulty, of a more fundamental nature, is that the control process may make some of the unknown parameters invisible to the estimation process. This is known as the problem of *parameter identifiability*, which is discussed in the context of adaptive control in several sources. On-line parameter estimation algorithms, which address among others the issue of identifiability, have been discussed extensively in the control theory literature, but the corresponding methodology is complex and beyond our scope in this book. However, assuming that we can make the estimation phase work somehow, we are free to reoptimize the controller using the newly estimated parameters, in a form of on-line replanning process.

Unfortunately, there is still another difficulty with this type of on-line replanning: it may be hard to recompute an optimal or near-optimal policy on-line, using a newly identified system model. In particular, it may be impossible to use time-consuming and/or data-intensive methods that involve for example the training of a neural network, or discrete/integer control constraints. A simpler possibility is to use rollout, which we discuss in the next section.

4.1. Approximation in value space, rollout, and adaptive control

We will now consider an approach for dealing with unknown or changing parameters, which is based on rollout and on-line replanning. We have already noted this approach in Section 1, where we stressed the importance of fast on-line policy improvement.

Let us assume that some problem parameters change over time and the controller becomes aware of the changes, perhaps after a suitable delay for data collection and estimation. The method by which the problem parameters are recalculated or become known is immaterial for the purposes of the following discussion. It may involve a limited form of parameter estimation, whereby the unknown parameters are "tracked" by data collection over a few time stages, with due attention paid to issues of parameter identifiability; or it may involve new features of the control environment, such as a changing number of servers and/or tasks in a service system.

We thus assume away/ignore the detailed issues of parameter estimation, and focus on revising the controller by on-line replanning based on the newly obtained parameters. This revision may be based on any suboptimal method, but rollout with some base policy is particularly attractive. The base policy may be either a fixed robust controller (such as some form of PID control) or

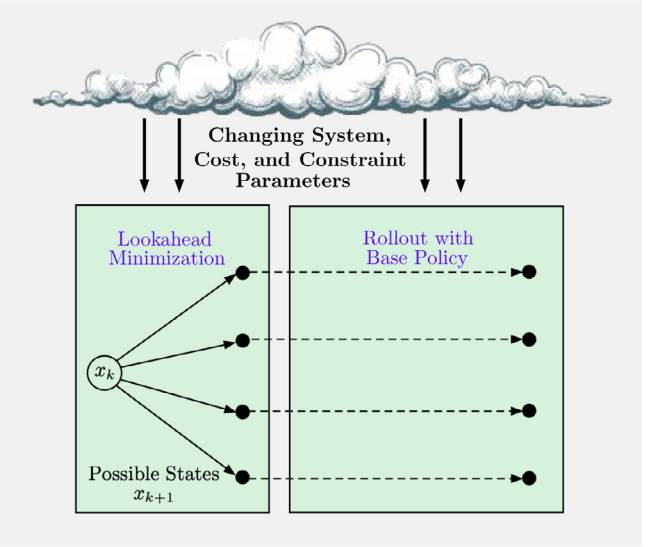

Fig. 22. Schematic illustration of adaptive control by on-line replanning based on rollout. One-step lookahead minimization is followed by simulation with the base policy, which stays fixed. The system, cost, and constraint parameters are changing over time, and the most recent estimates of their values are incorporated into the lookahead minimization and rollout operations. Truncated rollout with multistep lookahead minimization and terminal cost approximation is also possible. The base policy may also be revised based on various criteria. For the discussion of this section, we may assume that all the changing parameter information is provided by some computation and sensor "cloud" that is beyond our control.

it may be updated over time (in the background, on the basis of some unspecified rationale), in which case the rollout policy will be revised both in response to the changed base policy and in response to the changing parameters.

Here the advantage of rollout is that it is simple, reliable, and relatively fast. In particular, it does not require a complicated training procedure, based for example on the use of neural networks or other approximation architectures, so *no new policy is explicitly computed in response to the parameter changes*. Instead the available controls at the current state are compared through a one-step or multistep minimization, with cost function approximation provided by the base policy (cf. Fig. 22).

Another issue to consider is the stability and robustness properties of the rollout policy. In this connection, it can be generally proved, under mild conditions, that *if the base policy is stable within a range of parameter values, the same is true for the rollout policy*; this can also be inferred from Fig. 15. Related ideas have a long history in the control theory literature; see Beard [57], Beard, Saridis, and Wen [58], Jiang and Jiang [77], Kalise, Kundu, Kunisch [87].

The principal requirement for using rollout in an adaptive control context is that the rollout control computation should be fast enough to be performed between stages. In this connection, we note that accelerated/truncated or simplified versions of rollout, as well as parallel computation, can be used to meet this time constraint.

Generally, adaptive control by rollout and on-line replanning makes sense in situations where the calculation of the rollout controls for a given set of problem parameters is faster and/or more convenient than the calculation of the optimal controls for the same set of parameter values. These problems include cases involving nonlinear systems and/or difficult (e.g., integer) constraints.

The following example illustrates on-line replanning with the use of rollout in the context of the simple one-dimensional linear quadratic problem that we discussed earlier. The purpose of the example is to show analytically how rollout with a base policy that is optimal for a nominal set of problem parameters works well when the parameters change from their nominal values. This property is not practically useful in linear quadratic problems because when the parameter change, it is possible to calculate the new optimal policy in closed form, but it is indicative of the performance robustness of rollout in other contexts; for example linear quadratic problems with constraints.

Example 2 (*On-Line Replanning for Linear Quadratic Problems Based on Rollout*)**.** Consider a deterministic undiscounted infinite horizon linear quadratic problem involving the linear system

$$x_{k+1} = x_k + bu_k,$$

and the quadratic cost function

$$\lim_{N\to\infty} \sum_{k=0}^{N-1} (x_k^2 + ru_k^2).$$

As is well-known, the optimal cost function is given by

$$J^*(x) = K^* x^2,$$

where K^* is the unique positive solution of the Riccati equation

$$K = \frac{rK}{r + b^2 K} + 1. \tag{27}$$

The optimal policy has the form

$$\mu^*(x) = L^* x, \tag{28}$$

where

$$L^* = -\frac{bK^*}{r + b^2 K^*}. \tag{29}$$

As an example, consider the optimal policy that corresponds to the nominal problem parameters $b = 2$ and $r = 0.5$: this is the policy (28)–(29), with K computed as the positive solution of the quadratic Riccati for $b = 2$ and $r = 0.5$. For these nominal parameter values, we have

$$K = \frac{2 + \sqrt{6}}{4}.$$

From Eq. (29) we then also obtain

$$L = -\frac{2 + \sqrt{6}}{5 + 2\sqrt{6}}. \tag{30}$$

We will now consider changes of the values of b and r while keeping L constant to the preceding value, and we will compare the quadratic cost coefficient of the following three cost functions as b and r vary:

(a) The optimal cost function $K^* x^2$, where K^* is given by the positive solution of the Riccati Eq. (27).

(b) The cost function $K_L x^2$ that corresponds to the base policy

$$\mu_L(x) = Lx,$$

where L is given by Eq. (30). Here, we have

$$K_L = \frac{1 + rL^2}{1 - (1 + bL)^2}. \tag{31}$$

(c) The cost function $\tilde{K}_L x^2$ that corresponds to the rollout policy

$$\tilde{\mu}_L(x) = \tilde{L}x,$$

obtained by using the policy μ_L as base policy. Using the formulas derived earlier, we have [cf. Eq. (31)]

$$\tilde{L} = -\frac{bK_L}{r + b^2 K_L},$$

and

$$\tilde{K}_L = \frac{1 + r\tilde{L}^2}{1 - (1 + b\tilde{L})^2}.$$

Fig. 23 shows the coefficients K^*, K_L, and $\tilde{K}_L$ for a range of values of r and b. We have

$$K^* \le \tilde{K}_L \le K_L.$$

The difference $K_L - K^*$ is indicative of the robustness of the policy μ_L, i.e., the performance loss incurred by ignoring the changes in the values of b and r, and continuing to use the policy μ_L, which is optimal for the nominal values $b = 2$ and $r = 0.5$, but suboptimal for other values of b and r. The difference $\tilde{K}_L - K^*$ is indicative of the performance loss due to using on-line replanning by rollout rather than using optimal replanning. Finally, the difference $K_L - \tilde{K}_L$ is indicative of the performance improvement due to on-line replanning using rollout rather than keeping the policy μ_L unchanged.

Note that Fig. 23 illustrates the behavior of the error ratio $\frac{\tilde{J} - J^*}{J - J^*}$, where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance. This ratio approaches 0 as $J - J^*$ becomes smaller because of the superlinear/quadratic convergence rate of Newton's method that underlies the rollout algorithm.

4.2. Approximation in value space, rollout, and model predictive control

In this section, we briefly discuss the MPC methodology, with a view towards its connection with approximation in value space and the rollout algorithm. We will focus on the undiscounted infinite horizon deterministic problem, which involves the system

$$x_{k+1} = f(x_k, u_k),$$

whose state x_k and control u_k are finite-dimensional vectors. The cost per stage is assumed nonnegative

$$g(x_k, u_k) \ge 0, \qquad \text{for all } (x_k, u_k),$$

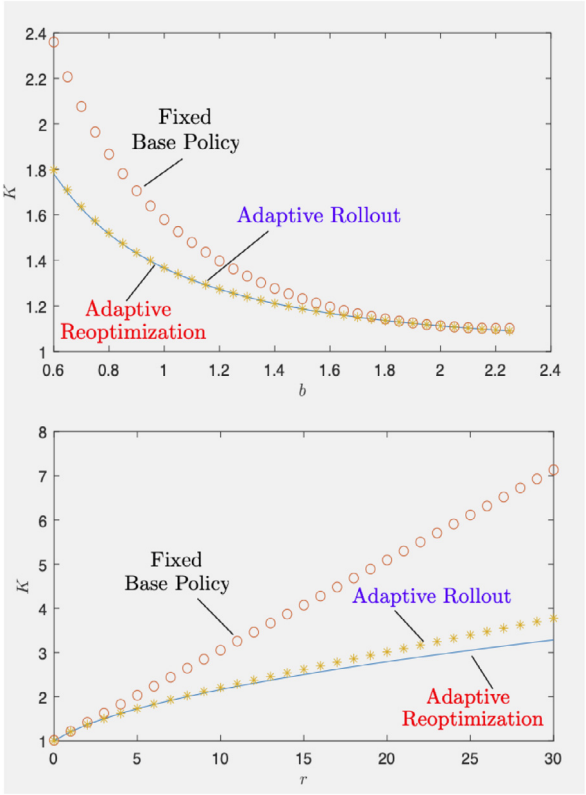

Fig. 23. Illustration of control by rollout under changing problem parameters. The quadratic cost coefficients K^* (optimal, denoted by solid line), K_L (base policy, denoted by circles), and $\tilde{K}_L$ (rollout policy, denoted by asterisks) for the two cases where $r = 0.5$ and b varies, and $b = 2$ and r varies. The value of L is fixed at the value that is optimal for $b = 2$ and $r = 0.5$ [cf. Eq. (30)]. The rollout policy performance is very close to optimal, even when the base policy is far from optimal. Note that, as the figure illustrates, we have $\lim_{J \to J^*} \frac{\tilde{J}-J^*}{J-J^*} = 0$, where for a given initial state, $\tilde{J}$ is the rollout performance, J^* is the optimal performance, and J is the base policy performance. This is a consequence of the superlinear/quadratic convergence rate of Newton's method that underlies rollout, and guarantees that the rollout performance approaches the optimal much faster than the base policy performance does.

(e.g., a positive definite quadratic cost). There are control constraints $u_k \in U(x_k)$, and to simplify the following discussion, we will initially consider no state constraints. We assume that the system can be kept at the origin at zero cost, i.e.,

$$f(0, \bar{u}_k) = 0, \quad g(0, \bar{u}_k) = 0 \qquad \text{for some control } \bar{u}_k \in U(0).$$

For a given initial state x_0, we want to obtain a sequence $\{u_0, u_1, \ldots\}$ that satisfies the control constraints, while minimizing the total cost.

This is a classical problem in control system design, known as the *regulation problem*, where the aim is to keep the state of the system near the origin (or more generally some desired set point), in the face of disturbances and/or parameter changes. In an important variant of the problem, there are additional state constraints of the form $x_k \in X$, and there arises the issue of maintaining the state within X, not just at the present time but also in future times. We will address this issue later in this section.

The classical form of MPC - view as a rollout algorithm

The scope of the MPC algorithm has grown over time to include problem and algorithm variations and extensions, and the literature on MPC is voluminous. For detailed accounts, we refer to the textbooks by Maciejowski [88], Goodwin, Seron, and De Dona [89], Camacho and Bordons [90], Kouvaritakis and Cannon [91], Borrelli, Bemporad, and Morari [92], and Rawlings, Mayne, and Diehl [44]. We will first focus on the original form of the MPC algorithm proposed in the form given here by Keerthi and Gilbert [93].

In this algorithm, at each encountered state x_k, we apply a control $\tilde{u}_k$ that is computed as follows; see Fig. 24:

(a) We solve an ℓ-stage optimal control problem involving the same cost function and the requirement that the state after ℓ steps is driven to 0, i.e., $x_{k+\ell} = 0$. This is the problem

$$\min_{u_t, t=k,\ldots,k+\ell-1} \sum_{t=k}^{k+\ell-1} g(x_t, u_t), \tag{32}$$

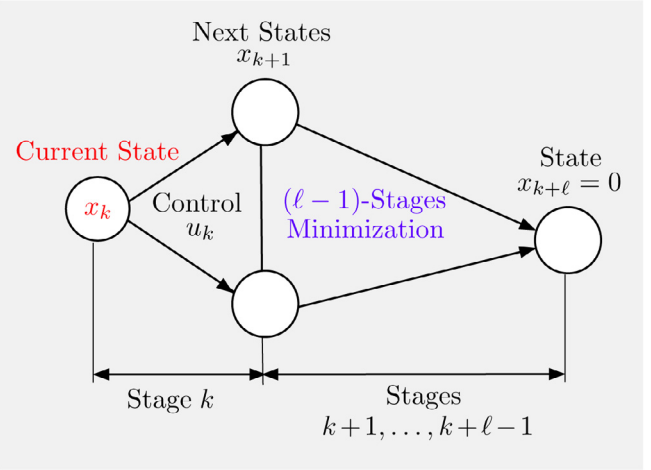

Fig. 24. Illustration of the problem solved by a classical form of MPC at state x_k. We minimize the cost function over the next ℓ stages while imposing the requirement that $x_{k+\ell}=0$. We then apply the first control of the optimizing sequence. In the context of rollout, the minimization over u_k is the one-step lookahead, while the minimization over $u_{k+1},\ldots,u_{k+\ell-1}$ that drives $x_{k+\ell}$ to 0 is the base heuristic.

subject to the system equation constraints

$$x_{t+1}=f(x_t,u_t),\qquad t=k,\ldots,k+\ell-1, \tag{33}$$

the control constraints

$$u_t\in U(x_t),\qquad t=k,\ldots,k+\ell-1, \tag{34}$$

and the terminal state constraint

$$x_{k+\ell}=0. \tag{35}$$

Here ℓ is an integer with $\ell>1$, which is chosen in some largely empirical way.

(b) If $\{\tilde u_k,\ldots,\tilde u_{k+\ell-1}\}$ is the optimal control sequence of this problem, we apply $\tilde u_k$ and we discard the other controls $\tilde u_{k+1},\ldots,\tilde u_{k+\ell-1}$.

(c) At the next stage, we repeat this process, once the next state x_{k+1} is revealed.

To make the connection of the preceding MPC algorithm with rollout, we note that *the one-step lookahead function $\tilde J$ implicitly used by MPC [cf. Eq. (32)] is the cost function of a certain stable base policy*. This is the policy that drives to 0 the state after $\ell-1$ stages (*not ℓ stages*) and keeps the state at 0 thereafter, while observing the state and control constraints, and minimizing the associated $(\ell-1)$-stages cost. This rollout view of MPC was first discussed in the author's paper [94]. It is useful for making a connection with the approximate DP/RL, rollout, and its interpretation in terms of Newton's method. In particular, an important consequence is that *the MPC policy is stable*, since rollout with a stable base policy yields a stable policy, as we have discussed in Section 3.2.

We may also equivalently view the preceding MPC algorithm as rollout with $\bar\ell$-step lookahead, where $1<\bar\ell<\ell$, with the base policy that drives to 0 the state after $\ell-\bar\ell$ stages and keeps the state at 0 thereafter. This suggests variations of MPC that involve truncated rollout with terminal cost function approximation, which we will discuss shortly.

Note also that when faced with changing problem parameters, it is natural to consider on-line replanning as per our earlier discussion. In particular, once new estimates of system and/or cost function parameters become available, MPC can adapt accordingly by introducing the new parameter estimates into the ℓ-stage optimization problem in (a) above.

Variants of MPC - terminal cost approximation

In a common variant of MPC, the requirement of driving the system state to 0 in ℓ steps in the ℓ-stage MPC problem (32), is replaced by a terminal cost $G(x_{k+\ell})$. Thus at state x_k, we solve the problem

$$\min_{u_t,\,t=k,\ldots,k+\ell-1}\left[G(x_{k+\ell})+\sum_{t=k}^{k+\ell-1}g(x_t,u_t)\right], \tag{36}$$

instead of problem (32) where we require that $x_{k+\ell}=0$. This variant can also be viewed as rollout with one-step lookahead, and a base policy, which at state x_{k+1} applies the first control $\tilde u_{k+1}$ of the sequence $\{\tilde u_{k+1},\ldots,\tilde u_{k+\ell-1}\}$ that minimizes

$$G(x_{k+\ell})+\sum_{t=k+1}^{k+\ell-1}g(x_t,u_t).$$

It can also be viewed as approximation in value space with ℓ-step lookahead and terminal cost approximation given by G. Thus our discussion of Section 3 relating to the region of stability of such schemes applies, and relates to results that are known within the MPC framework under various conditions (see the paper by Mayne at al. [95], the MPC book [44], and the author's book [12], Section 3.1.2).

Note that the preceding MPC controller may outperform substantially its base policy (in relative terms), particularly if the base policy is close to optimal, in the sense that

$$G(x_{k+\ell}) \approx J^*(x_{k+\ell}) + \text{ a constant.}$$

This is due to the superlinear/quadratic convergence rate of Newton's method that underlies approximation in value space, as we have discussed in Section 3.

An important question is to choose the terminal cost approximation so that the resulting MPC controller is stable. Among other possibilities, a common approach for linear systems and quadratic cost is to introduce truncated rollout with a stable base policy; see Magni et al. [8], and subsequent works. Our earlier discussion of stability is relevant within this context.

Another variant of MPC involves the use of truncated rollout, whereby the values $G(x_{k+\ell})$ in Eq. (36) are computed by running some base policy for a number of steps m followed by some terminal cost function approximation. This is quite similar to standard truncated rollout, except that the computational solution of the lookahead minimization problem (36) may become complicated when the control space is infinite. Moreover, the base policy may be used to address state constraints; see the papers by Rosolia and Borelli [96,97], by Li et al. [98], and the discussion in the author's RL book [12].

State constraints, target tubes, and off-line training

Our discussion so far has skirted a major issue in MPC, which is that there may be additional state constraints of the form $x_k \in X$, for all k, where X is some subset of the true state space. Indeed much of the original work on MPC was motivated by control problems with state constraints, imposed by the physics of the problem, which could not be handled effectively with the nice unconstrained framework of the linear quadratic problem.

The treatment of state constraints is connected to the theory of reachability of target tubes, first studied by the author in his Ph.D. thesis [99], and subsequent papers [100,101]; see the books [11,12,102] for a discussion that is consistent with the viewpoint of this section. Target tubes consist of subsets $\tilde{X}$ of the state constraint set X, within which the state can be kept indefinitely with feasible control choices, assuming that the initial state belongs to $\tilde{X}$. In other words, the problem (36) may not be feasible for every $x_k \in X$, once the constraint $x_t \in X$ for all $t = k+1, k+2, \ldots$, is added in problem (36). However, a suitable target tube is one specified by a subset $\tilde{X} \subset X$ such that the problem (36) is feasible under the constraint $x_t \in \tilde{X}$ for all $t = k+1, k+2, \ldots$, provided $x_k \in \tilde{X}$.

There are several ways to compute sets $\tilde{X}$ with this property, for which we refer to the aforementioned author's work and the MPC literature; see e.g., the book by Rawlings, Mayne, and Diehl [44], and the survey by Mayne [103]. The important point here is that the computation of a target tube must be done off-line with one of several available algorithmic approaches, so it becomes part of the off-line training (in addition to the terminal cost function G).

Given an off-line training process, which provides a target tube constraint $x_k \in \tilde{X}$ for all k, a terminal cost function G, and possibly one or more base policies for truncated rollout, MPC becomes an on-line play algorithm for which our earlier discussion applies. Significantly, MPC is an effective way to deal with indirect adaptive control contexts, where a model is estimated on-line as it is changing; see Section 4.1.

5. Concluding remarks

While the ideas of approximation in value space, rollout, and PI have a long history, their significance has been highlighted by the success of AlphaZero, and the earlier but just as impressive TD-Gammon program. Both programs were trained off-line extensively using sophisticated approximate PI algorithms and neural networks. Yet the players obtained off-line were greatly improved by on-line play, as we have discussed.

We have argued that this performance enhancement by on-line play defines a new paradigm for decision and control, which is couched on the AlphaZero/TD-Gammon design principles: on-line decision making, using approximation in value space with multistep lookahead, and rollout. There is an additional benefit of policy improvement by approximation in value space, not observed in the context of games (which have stable rules and environment). It is well-suited for on-line replanning and changing problem parameters, as in the context of indirect adaptive control, and also MPC, which in fact embodies several of the AlphaZero/TD-Gammon design ideas.

In this paper, we have aimed to provide the mathematical framework, analysis, and insights (often based on visualization), which facilitate the use of on-line decision making on top of off-line training. In particular, through a unified abstract DP analysis, we have shown that the principal ideas of approximation in value space and rollout apply very broadly to deterministic and stochastic optimal control problems, involving both discrete and continuous search spaces. These ideas can be effectively integrated with adaptive control, MPC, and other important methodologies such as decentralized and multiagent control, discrete and Bayesian optimization, neural network-based value and policy approximations, and heuristic algorithms for discrete optimization, as we have discussed in greater detail in the books [11,12]. We have also emphasized that while the ideas of on-line play and off-line training are implicit in

several decision and control contexts, and particularly MPC, much remains to be gained by a more systematic view of the dichotomy between off-line training and on-line play, and by its incorporation into control system design contexts.

A key idea of this paper is the interpretation of approximation in value space with one-step lookahead as a step of Newton's method. This idea has been known for a long time within the more restrictive contexts of policy iteration and rollout, where the cost function approximation $\tilde{J}$ is restricted to be the cost function of some policy. The extensions of this idea, including more general cost function approximations, multistep lookahead, truncated rollout, and discrete and multiagent optimization, which are provided in this work, are new (following their introduction in the book [12]), and aim to promote the view that Newton's method and other classical algorithms, such as Newton-SOR, are central conceptual elements of the RL methodology.

A major supplementary idea is our interpretation of off-line training of policies and cost approximations as means for enhancement of the initial condition of the Newton step. Among others, this interpretation supports the view that the Newton step/on-line player is the key determinant of the overall scheme's performance, and that the initial condition adjustment/off-line training plays a subsidiary role.

Finally, we have noted that while our focus in this work has been on infinite horizon problems, approximation in value space and rollout can be applied to finite horizon problems as well, and can be similarly interpreted in terms of Newton's method. One way to approach finite horizon problems analytically is to convert them to infinite horizon stochastic shortest path problems with a termination state that corresponds to the end of the horizon. Once this is done, the conceptual framework of the present work can be applied to provide insight on the connections between approximation in value space, rollout, and Newton's method. In particular, our ideas find application beyond the infinite horizon DP context, and apply to the solution of classical discrete and combinatorial optimization problems, which can be transformed to sequential finite horizon optimization problems; this was the basis for the original proposal of the use of rollout for discrete and combinatorial optimization problems in the paper [104]. Rollout has been used with great success for such problems in the past, and is discussed at length in the books [10,12].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Newton's method for nondifferentiable fixed point problems

In this appendix, we first develop the classical theory of Newton's method for solving a fixed point problem of the form $y = G(y)$, where y is an n-dimensional vector, and $G : \Re^n \mapsto \Re^n$ is a continuously differentiable mapping. We then extend the results to the case where G is nondifferentiable because it is obtained as the minimum of continuously differentiable mappings, as in the case of the Bellman operator (cf. Section 3).

The convergence analysis relates to the solution of Bellman's equation $J = TJ$, for the case where J is an n-dimensional (there are n states), TJ is real-valued for all real-valued J, and T is either differentiable or involves minimization over a finite number of controls. However, the analysis illuminates the mechanism by which Newton's method works for more general problems, involving for example infinite spaces problems. Moreover, the analysis does not use the concavity and monotonicity properties of T that hold in the DP contexts that we have discussed (discounted, SSP, and nonnegative-cost deterministic problems), but may not hold in other DP-related contexts.

Newton's method is an iterative algorithm that generates a sequence $\{y_k\}$, starting from some initial vector y_0. It aims to obtain asymptotically a fixed point of G, i.e., $y_k \to y^*$, where y^* is such that $y^* = G(y^*)$. Newton's method is usually analyzed in the context of solving systems of equations. In particular, by introducing the mapping $H : \Re^n \mapsto \Re^n$, given by

$$H(y) = G(y) - y, \qquad y \in \Re^n,$$

the fixed point problem is transformed to solving the equation $H(y) = 0$. We view $H(y)$ as a column vector in $\Re^n$, with its n components denoted by $H_1(y), \ldots, H_n(y)$:

$$H(y) = \begin{pmatrix} H_1(y) \\ \vdots \\ H_n(y) \end{pmatrix}.$$

Each of the functions H_i is given by

$$H_i(y) = G_i(y) - y_i,$$

where y_i is the i-th component of y and G_i is the i-th component of G.

Assuming that H is differentiable, Newton's method takes the form

$$y_{k+1} = y_k - \big(\nabla H(y_k)'\big)^{-1} H(y_k), \tag{37}$$

where ∇H is the $n \times n$ matrix whose columns are the gradients $\nabla H_1, \ldots, \nabla H_n$ of the n components $H_1, \ldots, H_n$, viewed as column vectors:

$$\nabla H(y) = \big(\nabla H_1(y), \ldots, \nabla H_n(y)\big),$$

and $\nabla H(y_k)'$ denotes the transpose of $\nabla H(y_k)$ [i.e., $\nabla H(y_k)'$ is the Jacobian of H at y_k]. The algorithm (37) is the *classical form of Newton's method*, and it assumes that $\nabla H(y_k)$ is invertible for every k. An intuitive view of the Newton iteration is that it first linearizes H at the current point y_k via a first order Taylor series expansion,

$$H(y) \approx H(y_k) + \nabla H(y_k)'(y - y_k),$$

and then computes y_{k+1} as the solution of the linearized system

$$H(y_k) + \nabla H(y_k)'(y - y_k) = 0.$$

Equivalently, in terms of the original fixed point problem $y = G(y)$, Newton's method first linearizes G at the current point y_k via a first order Taylor series expansion,

$$G(y) \approx G(y_k) + \nabla G(y_k)'(y - y_k),$$

where $\nabla G(y_k)'$ is the Jacobian matrix of G evaluated at y_k. It then computes y_{k+1} as the solution of the linearized fixed point problem

$$y = G(y_k) + \nabla G(y_k)'(y - y_k).$$

Thus,

$$y_{k+1} = G(y_k) + \nabla G(y_k)'(y_{k+1} - y_k),$$

or, by subtracting y_k from both sides and collecting terms,

$$(I - \nabla G(y_k)')(y_{k+1} - y_k) = G(y_k) - y_k.$$

By using $H(y) = G(y) - y$, this equation can be written in the form of the Newton iteration (37).

Its analysis has two principal aspects:

(a) *Local convergence*, dealing with the behavior near a nonsingular solution y^*, i.e., one where $H(y^*) = 0$ and the matrix $\nabla H(y^*)$ is invertible.
(b) *Global convergence*, addressing the modifications that are necessary to ensure that the method is valid and is likely to converge to a solution when started far from all solutions.

We will only consider the first of these issues in this appendix, and focus on a single nonsingular solution, i.e., a vector y^* such that $H(y^*) = 0$ and $\nabla H(y^*)$ is invertible. The principal result is that the Newton method (37) converges superlinearly when started close enough to y^*. It will be subsequently used to address the nondifferentiable case.

Proposition 1. *Consider a function $H : \Re^n \mapsto \Re^n$, and a vector y^* such that $H(y^*) = 0$. For any $\delta > 0$, we denote by $S_{\bar{\delta}}$ the open sphere $\{x \mid \|y - y^*\| < \delta\}$, where $\|\cdot\|$ denotes Euclidean norm. Assume that within some sphere $S_{\bar{\delta}}$, H is continuously differentiable, $\nabla H(y^*)$ is invertible, and $\left\|\left(\nabla H(y)'\right)^{-1}\right\|$ is bounded by some scalar $B > 0$:*

$$\left\|\left(\nabla H(y')\right)^{-1}\right\| \le B, \qquad \text{for all } y \in S_{\bar{\delta}}.$$

Assume also that for some $L > 0$,

$$\|\nabla H(x) - \nabla H(y)\| \le L\|x - y\|, \qquad \text{for all } x, y \in S_{\bar{\delta}}, \tag{38}$$

Then there exists $\delta \in (0, \bar{\delta}]$ such that if $y_0 \in S_\delta$, the sequence $\{y_k\}$ generated by the iteration

$$y_{k+1} = y_k - \left(\nabla H(y_k)'\right)^{-1} H(y_k)$$

belongs to S_δ, and converges monotonically to y^*, i.e.,

$$\|y_k - y^*\| \to 0, \qquad \|y_{k+1} - y^*\| \le \|y_k - y^*\|, \quad k = 0, 1, \ldots. \tag{39}$$

Moreover, we have

$$\|y_{k+1} - y^*\| \le \frac{LB}{2}\|y_k - y^*\|^2, \qquad k = 0, 1, \ldots. \tag{40}$$

Proof. We first note that if $y_k \in S_{\bar{\delta}}$, by using the relation

$$H(y_k) = \int_0^1 \nabla H\left(y^* + t(y_k - y^*)\right)' dt (y_k - y^*),$$

we have

$$\begin{aligned}\|y_{k+1} - y^*\| &= \left\|y_k - y^* - \left(\nabla H(y_k)'\right)^{-1} H(y_k)\right\| \\ &= \left\|\left(\nabla H(y_k)'\right)^{-1}\left(\nabla H(y_k)'(y_k - y^*) - H(y_k)\right)\right\|\end{aligned}$$

$$
\begin{aligned}
&= \left\| \left(\nabla H(y_k)'\right)^{-1} \left(\nabla H(y_k)' - \int_0^1 \nabla H\big(y^* + t(y_k - y^*)\big)' dt \right) (y_k - y^*) \right\| \\
&= \left\| \left(\nabla H(y_k)'\right)^{-1} \left(\int_0^1 \Big[\nabla H(y_k)' - \nabla H\big(y^* + t(y_k - y^*)\big)' \Big] dt \right) (y_k - y^*) \right\| \\
&\leq B \left(\int_0^1 \Big\| \nabla H(y_k) - \nabla H\big(y^* + t(y_k - y^*)\big) \Big\| dt \right) \|y_k - y^*\| \\
&\leq B \left(\int_0^1 Lt \|y_k - y^*\| dt \right) \|y_k - y^*\| \\
&= \frac{LB}{2} \|y_k - y^*\|^2,
\end{aligned}
$$

thus showing Eq. (40). Assume that $y_0 \in S_{\bar{\delta}}$. By continuity of ∇H, we can take $\delta \in (0, \bar{\delta}]$ such that $LB\delta < 1$, so if $y_0 \in S_\delta$, from the preceding relation we obtain $\|y_1 - y^*\| \leq \frac{1}{2}\|y_0 - y^*\| < \frac{\delta}{2}$. By repeating this argument with y_1 in place of y_0, we obtain $\|y_2 - y^*\| \leq \frac{1}{2}\|y_1 - y^*\| < \frac{\delta}{4}$, and similarly

$$\|y_{k+1} - y^*\| \leq \frac{1}{2}\|y_k - y^*\| < \frac{\delta}{2^{k+1}}, \qquad k = 0, 1, \ldots .$$

The monotonic convergence property (39) follows. □

Newton's method without differentiability of the Bellman operator

There is considerable literature on extensions of Newton's method that relax the differentiability requirement on T by using alternative notions from nonsmooth analysis, as noted in Section 3. In particular, the preceding proof can be simply extended to the case where G is nondifferentiable and has the minimization structure of the Bellman operator (assuming a finite control space). The idea is that when the kth iterate y_k is sufficiently close to the fixed point y^* of G, the kth Newton iteration can be viewed as a Newton iteration for *some* continuously differentiable mapping $\hat{G}_k$, which also has y^* as fixed point, and is obtained from G by a minimization operation. The preceding Prop. 1, applied to $\hat{G}_k$, shows then that the distance $\|y_k - y^*\|$ decreases monotonically at a quadratic rate.

In the nondifferentiable case, we focus on solution of the equation $H(y) = 0$ where the mapping $H : \Re^n \mapsto \Re^n$ again has real-valued n components, denoted by $H_1(y), \ldots, H_n(y)$:

$$H(y) = \begin{pmatrix} H_1(y) \\ \vdots \\ H_n(y) \end{pmatrix}.$$

The component mappings $H_1, \ldots, H_n : \Re^n \to \Re$, involve minimization over a parameter u (as the notation suggests, the parameter corresponds to control in the DP context), and have the form

$$H_i(y) = \min_{u=1,\ldots,m} H_{i,u}(y), \qquad i = 1, \ldots, n. \tag{41}$$

The mappings $H_{i,u} : \Re^n \to \Re$ are given by

$$H_{i,u}(y) = G_{i,u}(y) - y_i, \qquad i = 1, \ldots, n, \ u = 1, \ldots, m,$$

where $G_{i,u} : \Re^n \mapsto \Re$ is a given real-valued function for each i and u, and y_i is the ith component of y. Given a vector y^* such that $H(y^*) = 0$, we denote by $U^*(i) \subset \{1, \ldots, m\}$ the set of indexes that attain the minimum in Eq. (41) when $y = y^*$:

$$U^*(i) = \arg \min_{u=1,\ldots,m} H_{i,u}(y^*), \qquad i = 1, \ldots, n.$$

We assume that within some sphere $S_{\bar{\delta}}$ centered at y^* with radius $\bar{\delta}$, the mappings $H_{i,u}(\cdot)$ are continuously differentiable for all i and $u \in U^*(i)$, while all the $n \times n$ matrices with columns

$$\nabla H_{1,u_1}(y), \ldots, \nabla H_{n,u_n}(y),$$

are invertible, where for every i, u_i can take any value from the set $U^*(i)$. Thus all the Jacobian matrices of the mappings, which correspond to $(u_1, \ldots, u_n)$ that are "active" at y^* [i.e., $u_i \in U^*(i)$ for all i], are assumed invertible.

Given the iterate y_k, Newton's method operates as follows: It finds for each $i = 1, \ldots, n$, the set of indexes $U(i, k) \subset \{1, \ldots, m\}$ that attain the minimum in Eq. (41) when $y = y_k$:

$$U(i, k) = \arg \min_{u=1,\ldots,m} H_{i,u}(y_k).$$

Then it generates the next iterate y_{k+1} with the following three steps:

(a) It selects arbitrarily an index $u(i, k)$ from within $U(i, k)$, for each $i = 1, \ldots, n$.

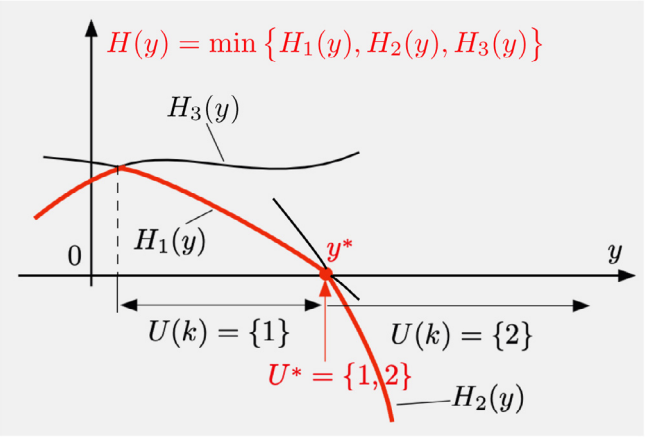

Fig. 25. Illustration of a one-dimensional nondifferentiable equation $H(y) = 0$, where H is obtained by minimization of three differentiable mappings H_1, H_2, H_3: $H(y) = \min\{H_1(y),\ H_2(y),\ H_3(y)\}, y \in \Re$. Here $m = 3$ and $n = 1$ [so the index i is omitted from $U^*(i)$ and $U(i,k)$]. At the solution y^*, the index set U^* consists of $u = 1$ and $u = 2$, and for $y_k \neq y^*$, we have $U(k) = \{1\}$ or $U(k) = \{2\}$. Newton's method applied to H consists of a Newton iteration applied to H_1 when $y < y^*$ and $|y - y^*|$ is small enough, and consists of a Newton iteration applied to H_2 when $y > y^*$. Since at y^* we have $H_1(y^*) = H_2(y^*) = 0$, both iterations, at $y < y^*$ and at $y > y^*$, approach y^* superlinearly.

(b) It forms the $n \times n$ matrix

$$M_k = \big(\nabla H_{1,u(1,k)}(y_k) \ \ldots \ \nabla H_{n,u(n,k)}(y_k)\big),$$

that has columns $\nabla H_{1,u(1,k)}(y_k), \ldots, \nabla H_{n,u(n,k)}(y_k)$, and the column vector G_k with components $H_{1,u(1,k)}(y_k), \ldots, H_{n,u(n,k)}(y_k)$:

$$G_k = \begin{pmatrix} H_{1,u(1,k)}(y_k) \\ \vdots \\ H_{n,u(n,k)}(y_k) \end{pmatrix}.$$

(c) It sets

$$y_{k+1} = y_k - (M_k')^{-1} G_k. \tag{42}$$

For our convergence proof, we argue as follows: When the iterate y_k is sufficiently near to y^*, the index set $U(i,k)$ is a subset of $U^*(i)$ for each $i = 1, \ldots, n$; see Fig. 25 for a case where $n = 1$ and $m = 2$. The reason is that there exists an $\epsilon > 0$ such that for all $i = 1, \ldots, n$ and $u = 1, \ldots, m$,

$$u \notin U^*(i) \qquad \Longrightarrow \qquad H_{i,u}(y^*) \geq \epsilon.$$

Therefore, there is a sphere centered at y^* such that for all y_k within that sphere and all i, we have

$$u \notin U^*(i) \qquad \Longrightarrow \qquad H_{i,u}(y_k) \geq \epsilon/2,$$

and

$$u \in U^*(i) \qquad \Longrightarrow \qquad H_{i,u}(y_k) < \epsilon/2,$$

which implies that if $u \notin U^*(i)$ then $u \notin U(i,k)$, or equivalently $U(i,k) \subset U^*(i)$. It follows that *the iteration (42) can be viewed as a Newton iteration applied to a system of differentiable equations that has y^* as its solution*. This system is

$$H_{i,u(i,k)}(y) = 0, \qquad i = 1, \ldots, n,$$

and corresponds to the set of indexes

$$u(1,k), \ldots, u(n,k).$$

Thus, near y^*, by Proposition 1, the iteration (42) is attracted at a quadratic rate to y^* regardless of which indexes $u(i,k) \in U(i,k)$ are selected for iteration k. Finally, note that while the sphere within which $U(i,k) \subset U^*(i)$ for all i was constructed for a single iteration k, we can take the sphere small enough to ensure that the distance from the current iterate to y^* is reduced for all subsequent iterations, similar to the proof of Proposition 1. We can thus conclude that after y_k gets close enough to y^*, each subsequent iteration is a Newton step applied to *one* of a finite number of differentiable systems of equations that have y^* as their common solution, and for which the convergence properties of Proposition 1 hold.

References

[1] Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 2017, arXiv preprint arXiv:1712.01815.
[2] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Mastering the game of go without human knowledge. Nature 2017;550:354–9.
[3] Tesauro GJ. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput 1994;6:215–9.
[4] Tesauro GJ. Temporal difference learning and TD-Gammon. Commun ACM 1995;38:58–68.
[5] Tesauro G, Galperin GR. On-line policy improvement using monte carlo search. CO: NIPS, Denver; 1996.
[6] Scherrer B, Ghavamzadeh M, Gabillon V, Lesner B, Geist M. Approximate modified policy iteration and its application to the game of tetris. J Mach Learn Res 2015;16:1629–76.
[7] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, et al. Mastering the game of go with deep neural networks and tree search. Nature 2016;529:484–9.
[8] Magni L, De Nicolao G, Magnani L, Scattolini R. A stabilizing model-based predictive control algorithm for nonlinear systems. Automatica 2001;37:1351–62.
[9] Bertsekas DP. Dynamic programming and optimal control, II. 4th Ed.. Belmont, MA: Athena Scientific; 2012.
[10] Bertsekas DP. Lessons from Alphazero for optimal, model predictive, and adaptive control. Belmont, MA: Athena Scientific; 2022.
[11] Bertsekas DP. Reinforcement learning and optimal control. Belmont, MA: Athena Scientific; 2019.
[12] Bertsekas DP. Rollout, policy iteration, and distributed reinforcement learning. Belmont, MA: Athena Scientific; 2020.
[13] Bertsekas DP. Value and policy iteration in deterministic optimal control and adaptive dynamic programming. IEEE Trans Neural Netw Learn Syst 2017;28:500–9.
[14] Bertsekas DP. Abstract dynamic programming. 3rd Ed.. Belmont, MA: Athena Scientific; 2022, (can be downloaded from the author's website).
[15] Bertsekas DP, Tsitsiklis JN. Neuro-dynamic programming. Belmont, MA: Athena Scientific; 1996.
[16] Sutton R, Barto AG. Reinforcement learning. 2nd Ed.. Cambridge, MA: MIT Press; 2018.
[17] Ortega JM, Rheinboldt WC. Iterative solution of nonlinear equations in several variables. Soc Ind Appl Math 2000.
[18] Bertsekas Dimitri P. Nonlinear Programming. Belmont, MA: Athena Scientific; 2016.
[19] Josephy NH. Newton's method for generalized equations. Mathematics research center report no. 1965, Wisconsin Univ-Madison; 1979.
[20] Robinson SM. Strongly regular generalized equations. Math Oper Res 1980;5:43–62.
[21] Robinson SM. Newton's method for a class of nonsmooth functions, Vol. 2. Industrial Engineering Working Paper, University of Wisconsin; also in Set-Valued Analysis; 1988, p. 291–305, 1994.
[22] Robinson SM. A point-of-attraction result for Newton's method with point-based approximations. Optimization 2011;60:89–99.
[23] Kojima M, Shindo S. Extension of Newton and quasi-Newton methods to systems of PC^1 equations. J Oper Res Soc Japan 1986;29:352–75.
[24] Kummer B. Newton's method for non-differentiable functions. Math Res 1988;45:114–25.
[25] Kummer B. Generalized Newton and NCP-methods: Convergence, regularity, actions, discussiones mathematicae, differential inclusions. Control Optim 2000;2:209–44.
[26] Pang JS. Newton's method for B-differentiable equations. Math Oper Res 1990;15:311–41.
[27] Qi L. Convergence analysis of some algorithms for solving nonsmooth equations. Math Oper Res 1993;18:227–44.
[28] Qi L, Sun J. A nonsmooth version of Newton's method. Math Program 1993;58:353–67.
[29] Facchinei F, Pang J-S. Finite-dimensional variational inequalities and complementarity problems, Vol. I and II. N. Y: Springer; 2003.
[30] Ito K, Kunisch K. Semi-smooth Newton methods for variational inequalities of the first kind. Math Model Num Anal 2003;37:41–62.
[31] Bolte J, Daniilidis A, Lewis A. Tame functions are semismooth. Math Program 2009;117:5–19.
[32] Dontchev AL, Rockafellar RT. Implicit functions and solution mappings. second ed.. N. Y: Springer; 2014.
[33] Ortega James M, Rheinboldt Werner C. Monotone iterations for nonlinear equations with application to gauss-seidel methods. SIAM J Num Anal 1967;4(2):171–90.
[34] Vandergraft James S. Newton's method for convex operators in partially ordered spaces. SIAM J Num Anal 1967;4(3):406–32.
[35] Argyros Ioannis K. Convergence and applications of Newton-type iterations. N.Y: Springer Science & Business Media; 2008.
[36] Pollatschek M, Avi-Itzhak B. Algorithms for stochastic games with geometrical interpretation. Manage Sci 1969;15:399–413.
[37] Bertsekas DP. Distributed asynchronous policy iteration for sequential zero-sum games and minimax control. 2021, 2021, arXiv preprint arXiv:2107.10406.
[38] Ha M, Wang D, Liu D. Offline and online adaptive critic control designs with stability guarantee through value iteration. IEEE Trans Cybern 2021.
[39] Liu D, Xue S, Zhao B, Luo B, Wei Q. Adaptive dynamic programming for control: A survey and recent advances. IEEE Trans Syst Man Cybern 2021;51:142–60.
[40] Wei Q, Liu D, Lin H. Value iteration adaptive dynamic programming for optimal control of discrete-time nonlinear systems. IEEE Trans Cybern 2016;46:840–53.
[41] Heydari A. Stability analysis of optimal adaptive control under value iteration using a stabilizing initial policy. IEEE Trans Neural Netw Learn Syst 2017;29:4522–7.
[42] Heydari A. Stability analysis of optimal adaptive control using value iteration with approximation errors. IEEE Trans Automat Control 2018;63:3119–26.
[43] Winnicki A, Lubars J, Livesay M, Srikant R. The role of lookahead and approximate policy evaluation in policy iteration with linear value function approximation. 2021, arXiv preprint arXiv:2109.13419.
[44] Rawlings JB, Mayne DQ, Diehl MM. Model predictive control: theory, computation, and design. 2nd Ed.. Nob Hill Publishing; 2017, (updated in 2019 and 2020).
[45] Bertsekas DP, Yu H. Asynchronous distributed policy iteration in dynamic programming. Proc Allerton Conf Commun Control Comput Allerton Park, Ill 2010;136:8–1374.
[46] Bertsekas DP, Yu H. Q-learning and enhanced policy iteration in discounted dynamic programming. Math Oper Res 2012;37:66–94.
[47] Yu H, Bertsekas DP. Q-learning and policy iteration algorithms for stochastic shortest path problems. Ann Oper Res 2013;208:95–132.
[48] Kleinman DL. On an iterative technique for riccati equation computations. IEEE Trans Automatic Control, AC- 1968;13:114–5.
[49] Kleinman DL. Suboptimal design of linear regulator systems subject to computer storage limitations Doctoral dissertation, M.I.T. Electronic Systems Lab. Rept.; 1967, p. 297.
[50] Bellman R, Kalaba RE. Quasilinearization and nonlinear boundary-value problems. N.Y: Elsevier; 1965.
[51] Bellman R. Dynamic programming. Princeton, N. J: Princeton University Press; 1957.
[52] Hewer G. An iterative technique for the computation of the steady state gains for the discrete optimal regulator. IEEE Trans Automat Control 1971;16:382–4.
[53] Puterman ML, Brumelle SL. The analytic theory of policy iteration. In: Puterman ML, editor. Dynamic programming and its applications. N. Y: Academic Press; 1978.
[54] Puterman ML, Brumelle SL. On the convergence of policy iteration in stationary dynamic programming. Math Oper Res 1979;4:60–9.
[55] Saridis GN, Lee C-SG. An approximation theory of optimal control for trainable manipulators. IEEE Trans Syst Man Cybern 1979;9:152–9.
[56] Rekasius ZV. Suboptimal design of intentionally nonlinear controllers. IEEE Trans Automat Control 1964;9:380–6.
[57] Beard RW. Improving the closed-loop performance of nonlinear systems. (Ph.D. thesis), Rensselaer Polytechnic Institute; 1995.

[58] Beard RW, Saridis GN, Wen JT. Approximate solutions to the time-invariant Hamilton–Jacobi-Bellman equation. J Optim Theory Appl 1998;96:589–626.
[59] Santos MS, Rust J. Convergence properties of policy iteration. SIAM J Control Optim 2004;42:2094–115.
[60] Bokanowski O, Maroso S, Zidani H. Some convergence results for Howard's algorithm. SIAM J Num Anal 2009;47:3001–26.
[61] Hylla T. Extension of inexact kleinman-newton methods to a general monotonicity preserving convergence theory. (Ph.D. thesis), Univ. of Trier; 2011.
[62] Magirou EF, Vassalos P, Barakitis N. A policy iteration algorithm for the American put option and free boundary control problems. J Comput Appl Math 2020;373:112544.
[63] Kundu S, Kunisch K. Policy iteration for hamilton–Jacobi-Bellman equations with control constraints. Comput Optim Appl 2021;1–25.
[64] Lopez VG, Alsalti M, Muller MA. Efficient off-policy Q-learning for data-based discrete-time LQR problems. 2021, arXiv preprint arXiv:2105.07761.
[65] Bertsekas DP. On-line policy iteration for infinite horizon dynamic programming. 2021, arXiv preprint arXiv:2106.00746.
[66] Bertsekas DP. Multiagent rollout algorithms and reinforcement learning. 2019, arXiv preprint arXiv:1910.00120.
[67] Bertsekas DP. Constrained multiagent rollout and multidimensional assignment with the auction algorithm. 2019, arXiv preprint, arxiv:2002.07407.
[68] Bertsekas DP. Multiagent value iteration algorithms in dynamic programming and reinforcement learning. Results Control Optim J 2020;1. arXiv preprint, arxiv:2005.01627.
[69] Bertsekas DP. Multiagent reinforcement learning: Rollout and policy iteration. IEEE/ CAA J Automatica Sinica 2021;8:249–71.
[70] Bhattacharya S, Kailas S, Badyal S, Gil S, Bertsekas DP. Multiagent rollout and policy iteration for POMDP with application to multi-robot repair problems. In: Proc. of conference on robot learning. 2020, also arXiv preprint, arXiv:2011.04222.
[71] Aström KJ, Hagglund T. PID controllers: theory, design, and tuning, instrument society of america. NC: Research Triangle Park; 1995.
[72] Aström KJ, Hagglund T. Advanced PID control. N. C: Instrument Society of America, Research Triangle Park; 2006.
[73] Aström KJ, Wittenmark B. Adaptive control. N. J: Dover Books; also Prentice-Hall, Englewood Cliffs; 2008, 1994.
[74] Bodson M. Adaptive estimation and control. Independently Published; 2020.
[75] Goodwin GC, Sin KSS. Adaptive filtering, prediction, and control. N. J: Prentice-Hall, Englewood Cliffs; 1984.
[76] Ioannou PA, Sun J. Robust adaptive control. N. J: Prentice-Hall, Englewood Cliffs; 1996.
[77] Jiang Y, Jiang ZP. Robust adaptive dynamic programming. N. Y: J. Wiley; 2017.
[78] Krstic M, Kanellakopoulos I, Kokotovic P. Nonlinear and adaptive control design. N. Y: J. Wiley; 1995.
[79] Kokotovic PV, editor. Foundations of adaptive control. Springer; 1991.
[80] Kumar PR, Varaiya PP. Stochastic systems: estimation, identification, and adaptive control. N. J: Prentice-Hall, Englewood Cliffs; 1986.
[81] Liu D, Wei Q, Wang D, Yang X, Li H. Adaptive dynamic programming with applications in optimal control. Berlin: Springer; 2017.
[82] Lavretsky E, Wise K. Robust and adaptive control with aerospace applications. Springer; 2013.
[83] Narendra KS, Annaswamy AM. Stable adaptive systems. Courier Corporation; 2012.
[84] Sastry S, Bodson M. Adaptive control: stability, convergence and robustness. Courier Corporation; 2011.
[85] Slotine J-JE, Li W. Applied nonlinear control. N. J.: Prentice-Hall, Englewood Cliffs.
[86] Vrabie D, Vamvoudakis KG, Lewis FL. Optimal adaptive control and differential games by reinforcement learning principles. London: The Institution of Engineering and Technology; 2013.
[87] Kalise D, Kundu S, Kunisch K. Robust feedback control of nonlinear PDEs by numerical approximation of high-dimensional Hamilton–Jacobi-Isaacs equations. SIAM J Appl Dyn Syst 2020;19:1496–524.
[88] Maciejowski JM. Predictive control with constraints. MA: Addison-Wesley, Reading; 2002.
[89] Goodwin G, Seron MM, De Dona JA. Constrained control and estimation: an optimisation approach. N. Y: Springer; 2006.
[90] Camacho EF, Bordons C. Model predictive control. 2nd Ed.. New York, N. Y: Springer; 2007.
[91] Kouvaritakis B, Cannon M. Model predictive control: classical, robust and stochastic. N. Y: Springer; 2016.
[92] Borrelli F, Bemporad A, Morari M. Predictive control for linear and hybrid systems. Cambridge, UK: Cambridge Univ. Press; 2017.
[93] Keerthi SS, Gilbert EG. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. J Optim Theory Appl 1988;57:265–93.
[94] Bertsekas DP. Dynamic programming and suboptimal control: A survey from ADP to MPC. Eur J Control 2005;11:310–34.
[95] Mayne D, Rawlings JB, Rao CV, Scokaert POM. Constrained model predictive control: Stability and optimality. Automatica 2000;36:789–814.
[96] Rosolia U, Borrelli F. Learning model predictive control for iterative tasks. a data-driven control framework. IEEE Trans Automat Control 2017;63:1883–96.
[97] Rosolia U, Borrelli F. Sample-based learning model predictive control for linear uncertain systems. In: 58th conference on decision and control, Vol. 270. 2019, p. 2–2707.
[98] Li Y, Johansson KH, Martensson J, Bertsekas DP. Data-driven rollout for deterministic optimal control. 2021, arXiv preprint arXiv:2105.03116.
[99] Bertsekas DP. Control of uncertain systems with a set-membership description of the uncertainty. (Ph.D. thesis), Cambridge, MA: Massachusetts Institute of Technology; 1971, (can be downloaded from the author's website).
[100] Bertsekas DP, Rhodes IB. On the minimax reachability of target sets and target tubes. Automatica 1971;7:233–47.
[101] Bertsekas DP. Infinite time reachability of state space regions by using feedback control. IEEE Trans Autom Control, AC- 1972;17:604–13.
[102] Bertsekas DP. Dynamic programming and optimal control, I. 4th Ed.. Belmont, MA: Athena Scientific; 2017.
[103] Mayne DQ. Model predictive control: Recent developments and future promise. Automatica 2014;50:2967–86.
[104] Bertsekas DP, Tsitsiklis JN, Wu C. Rollout algorithms for combinatorial optimization. Heuristics 1997;3:245–62.