Multiagent Reinforcement Learning: 
Rollout and Policy Iteration

Dimitri Bertsekas†

Abstract
We discuss the solution of multistage decision problems using methods that are based on the idea of policy iteration (PI for short), i.e., start from some base policy and generate an improved policy. Rollout is the simplest method of this type, where just one improved policy is generated. We can view PI as repeated application of rollout, where the rollout policy at each iteration serves as the base policy for the next iteration. In contrast with PI, rollout can be applied on-line and is suitable for on-line replanning. Moreover, rollout can use as base policy one of the policies produced by PI, thereby improving on that policy. This is the type of scheme underlying the prominently successful AlphaZero chess program.

In this paper we focus on rollout and PI-like methods for multiagent problems, where the control consists of multiple components each selected by a separate agent. We discuss an approach, whereby at every stage, the agents sequentially (one-at-a-time) execute a local rollout algorithm that uses a base policy, together with some coordinating information from the other agents. The amount of total computation required at every stage grows linearly with the number of agents. By contrast, in the standard rollout algorithm, the amount of total computation grows exponentially with the number of agents. Despite the dramatic reduction in required computation, we show that our multiagent rollout algorithm has the fundamental cost improvement property of standard rollout: it guarantees an improved performance relative to the base policy.

We first develop our agent-by-agent policy improvement approach for finite horizon problems, and then we extend it to exact and approximate PI for discounted and other infinite horizon problems. We prove that the cost improvement property steers the algorithm towards convergence to an agent-by-agent optimal policy, thus establishing a connection with the theory of teams. We also discuss autonomous multiagent rollout schemes that allow the agents to make decisions autonomously through the use of precomputed signaling information, which is sufficient to maintain the cost improvement property, without any on-line coordination of control selection between the agents.

1. INTRODUCTION

In this paper we discuss the solution of large and challenging multistage decision problems, which can be solved in principle by dynamic programming (DP for short), but are addressed in practice using methods

† Fulton Professor of Computational Decision Making, ASU, Tempe, AZ, and McAfee Professor of Engineering, MIT, Cambridge, MA.
of reinforcement learning (RL for short), also referred to by other names such as approximate dynamic programming and neuro-dynamic programming. We will focus on methods that involve various forms of PI, i.e., starting from some policy and generating one or more improved policies.

If just one improved policy is generated, this is called rollout, with the initial policy called base policy and the improved policy called rollout policy. Based on broad and consistent computational experience, rollout appears to be one of the simplest and most reliable of all RL methods (we refer to the author’s textbooks [Ber17], [Ber19a], [Ber20a] for an extensive list of research contributions and case studies on the use of rollout). Rollout is also well-suited for on-line model-free implementation and on-line replanning.

Approximate PI is one of the most prominent types of RL methods. It can be viewed as repeated application of rollout, and can provide (off-line) the base policy for use in a rollout scheme. It can be implemented using data generated by the system itself, and value and policy approximations. Approximate forms of PI, which are based on the use of approximation architectures, such as value and policy neural networks, have most prominently been used in the spectacularly successful AlphaZero chess program; see Silver et al. [SHS17]. In particular, in the AlphaZero architecture a policy is constructed via an approximate PI scheme that is based on the use of deep neural networks. This policy is used as a base policy to generate chess moves on-line though an approximate multistep lookahead scheme that applies Monte Carlo tree search with an approximate evaluation of the base policy used as a terminal cost function approximation. Detailed descriptions of approximate PI schemes can be found in most of the RL textbooks, including the author’s [Ber19a], which shares the notation and point of view of the present paper.

The purpose of this paper is to survey variants of rollout and PI for DP problems with a multiagent structure, where at each stage, the control \( u \) consists of multiple components \( u_1, \ldots, u_m \), each associated with a separate agent, i.e.,

\[
u = (u_1, \ldots, u_m),
\]

where \( u_i \) is the control applied by the \( i \)th agent and is constrained to lie within a constraint set \( U_i \). Thus, the overall constraint set \( U \) is the Cartesian product

\[
U = U_1 \times \cdots \times U_m.
\]

It is assumed that the state \( x \) becomes perfectly known by all the agents at each stage.† Given the current policy \( \mu \) [a function that maps the current state \( x \) to a control \( \mu(x) \), also referred to as the base policy], the policy improvement operation portion of a PI involves at each state \( x \), a one-step lookahead minimization of

---

† Partial state information problems can be converted to problems involving perfect state information by using a belief state. Our assumption then amounts to perfect knowledge of the belief state by all agents. For example, we may think of a central processing computational “cloud” that collects and processes state information, and broadcasts a belief state to all agents at each stage.
the general form

$$\min_{u \in U} H(x, u, J_\mu),$$  \hspace{1cm} (1.1)

where $J_\mu$ is the cost function of policy $\mu$, and $H$ is a problem-dependent Bellman operator. This minimization defines a new policy $\tilde{\mu}$ (also referred to as the rollout policy), whereby the control $\tilde{\mu}(x)$ to be applied at $x$ is the one attaining the minimum above. The key property for the success of the rollout and PI algorithms is the policy improvement property

$$J_{\tilde{\mu}}(x) \leq J_\mu(x), \quad \text{for all states } x.$$  \hspace{1cm} (1.2)

Assuming that each set $U_i$ is finite (as we do in this paper), there is a two-fold difficulty with the lookahead minimization (1.1):

(a) The cardinality of the Cartesian product $U$ grows exponentially with the number $m$ of agents, thus creating formidable difficulties in the minimization over $u \in U$ when $m$ is large.

(b) To implement the minimization (1.1), the agents need to coordinate their choices of controls, something that may be difficult in systems involving autonomous agents with limited intercommunication capabilities.

To deal with the challenges just described, we introduce a form of sequential agent-by-agent one-step lookahead minimization, called multiagent rollout. It mitigates dramatically the computational difficulties due to (a) above. In particular, the amount of computation required at each stage grows linearly with the number of agents $m$, rather than exponentially. Despite the dramatic reduction in required computation, we show that our multiagent rollout algorithm has the fundamental cost improvement property (1.2): it guarantees an improved performance of the rollout policy relative to the base policy.

Multiagent rollout in the form just described involves coordination of the control selections of the different agents. In particular, it requires that the agents select their controls sequentially in a prespecified order, with each agent communicating its control selection to the other agents. To deal with the case of on-line autonomous control selection by the agents [cf. (b) above], we suggest to implement multiagent rollout with the use of a precomputed signaling policy that embodies agent coordination. One possibility is to approximately compute off-line the multiagent rollout policy through approximation in policy space, i.e., training an approximation architecture such as a neural network to learn the rollout policy. This scheme, called autonomous multiagent rollout, allows the use of autonomous, and distributed and asynchronous on-line control selection by the agents, with a potential sacrifice of performance, which depends on the quality of the policy space approximation.

It is worth emphasizing that our multiagent schemes require the agents to have access to the exact system state (or a belief state in the case of a system with partial state observations). Among others,
Figure 1.1  Illustration of the conceptual structure of our multiagent system. The “cloud” collects information from the environment and from the agents on-line, and broadcasts the state (and possibly other information) to the agents at each stage, who then perform local computations to apply their controls as functions of the state information obtained from the cloud. In the case of a problem with partial state observation, the cloud computes the current belief state (rather than the state) and broadcasts it to the agents.

this gives rise to a problem with a so called “classical information pattern,” whereby all agents share the same information, and which can be addressed with the DP formalism. Conceptually, we may think of a computational “cloud” that collects information from the agents on-line, computes the system state, and passes it on to the agents, who then perform local computations to apply their controls; see Fig. 1.1. However, our schemes are also well suited as a starting point for approximations where the state information made available to the agents is replaced by appropriate estimates, which are then treated by the agents as if they were exact. While such schemes appear reasonable, their analysis is beyond the scope of the present paper, and is left as a subject for further research.

We note that multiagent rollout has a strong connection with a well-developed body of research with a long history: the theory of teams and decentralized control, and the notion of person-by-person optimality (see Marschak [Mar55], Radner [Rad62], Witsenhausen [Wit71a], [Wit71b], Ho [Ho80]; for more recent works, see Nayyar, Mahajan, and Teneketzis [NMT13], Nayyar and Teneketzis [NaT19], Li et al. [LTZ19], Qu and Li [QuL19], Gupta [Gup20], the book by Zoppoli, Sanguineti, Gnecco, and Parisini [ZSG20], and the references quoted there). In particular, we develop an infinite horizon DP methodology, which includes value iteration and PI methods that converge to a person-by-person optimal policy. However, in contrast with the present paper, a large portion of the work on team theory and decentralized control allows a “nonclassical information pattern,” whereby the agents to not share the same state information.
In addition to the aforementioned works on team theory and decentralized control, there has been considerable work on multiagent sequential decision making from an RL perspective. Works of this type are surveyed by Busoniu, Babuska, and De Schutter [BBD08], [BBD10b], who list many other references. These works, however, have not considered the idea of agent-by-agent optimization as a means of addressing the exponential growth of the size of the control space as the number of agents increases.

We also mention that distributed DP algorithms have been considered in a number of contexts that involve partitioning of the state space into subsets, with a DP algorithm executed in parallel within each subset. For example distributed value iteration has been investigated in the author’s papers [Ber82], [Ber83], and the book [BeT89]. Also asynchronous PI algorithms have been discussed in a series of papers of the author and H. Yu [BeY10], [BeY12], [YuB13], as well as the books [Ber12], [Ber18a]. The algorithmic ideas of these works do not directly apply to the multiagent context of this paper. Still, however, one may envision applications where parallelization with state space partitioning is combined with the multiagent parallelization ideas of the present paper. This, however, is beyond our scope and is left as an interesting subject for further research.

The approach to maintain cost improvement through agent-by-agent rollout was introduced recently in the author’s papers [Ber19b], [Ber20cb], [Ber20c], and research monograph [Ber20a], with the aim to alleviate substantially the difficulties (a) and (b) above. The paper [BSB20] applies this approach to a challenging large-scale multirobot routing and repair problem, involving partial state information, and explores some of the attendant implementation issues, including autonomous multiagent rollout, through the use of policy neural networks. The paper [Ber20c] and the monograph [Ber20a] discuss constrained forms of rollout for deterministic problems, including multiagent forms, and an extensive range of applications in discrete/combinatorial optimization and model predictive control. We refer to these sources for more details and example applications. The present paper surveys these developments and provides some additional discussion.

The paper is organized as follows. We first introduce finite horizon stochastic optimal control problems in Section 2, we explain the main idea behind the multiagent rollout algorithm, and we show the cost improvement property. We also discuss variants of the algorithm that are aimed at improving its computational efficiency. In Section 3, we consider the implementation of autonomous multiagent rollout, including schemes that allow the distributed and asynchronous computation of the agents’ control components. In Section 4, we extend the multiagent rollout algorithm to infinite horizon discounted problems, we discuss the cost improvement property, and we provide error bounds for versions of the algorithm involving rollout truncation and simulation. Finally, in Section 5 we discuss autonomous multiagent rollout schemes for infinite horizon discounted problems.
2. MULTIAGENT PROBLEM FORMULATION - FINITE HORIZON PROBLEMS

We consider a standard form of an N-stage DP problem (see [Ber17], [Ber19a]), which involves the discrete-time dynamic system

\[ x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \ldots, N - 1, \]  

(2.1)

where \( x_k \) is an element of some (possibly infinite) state space, the control \( u_k \) is an element of some finite control space, and \( w_k \) is a random disturbance, which is characterized by a probability distribution \( P_k(\cdot | x_k, u_k) \) that may depend explicitly on \( x_k \) and \( u_k \), but not on values of prior disturbances \( w_{k-1}, \ldots, w_0 \). The control \( u_k \) is constrained to take values in a given subset \( U_k(x_k) \), which depends on the current state \( x_k \).

The cost of the kth stage is denoted by \( g_k(x_k, u_k, w_k) \); see Fig. 2.1.

We consider policies of the form

\[ \pi = \{\mu_0, \ldots, \mu_{N-1}\}, \]

where \( \mu_k \) maps states \( x_k \) into controls \( u_k = \mu_k(x_k) \), and satisfies a control constraint of the form \( \mu_k(x_k) \in U_k(x_k) \) for all \( x_k \). Given an initial state \( x_0 \) and a policy \( \pi = \{\mu_0, \ldots, \mu_{N-1}\} \), the expected cost of \( \pi \) starting from \( x_0 \) is

\[ J_\pi(x_0) = E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}, \]

where the expected value operation \( E\{\cdot\} \) is over all the random variables \( w_k \) and \( x_k \). The optimal cost starting from \( x_0 \), is defined by

\[ J^*(x_0) = \min_{\pi \in \Pi} J_\pi(x_0), \]

where \( \Pi \) is the set of all policies, while an optimal policy \( \pi^* \) is one that attains the minimal cost for every \( x_0 \); i.e.,

\[ J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0), \quad \text{for all } x_0. \]
Since the optimal cost function $J^*$ and optimal policy $\pi^*$ are typically hard to obtain by exact DP, we consider approximate DP/RL algorithms for suboptimal solution, and focus on rollout, which we describe next.

**2.1. The Standard Rollout Algorithm and Policy Improvement**

In the standard form of rollout, given a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, called base policy, with cost-to-go from state $x_k$ at stage $k$ denoted by $J_{k,\pi}(x_k)$, $k = 0, \ldots, N$, we obtain an improved policy, i.e., one that achieves cost that is less or equal to $J_{k,\pi}(x_k)$ starting from each $x_k$. The base policy is arbitrary. It may be a simple heuristic policy or a sophisticated policy obtained by off-line training through the use of an approximate PI method or a policy gradient method.

The standard rollout algorithm provides on-line control of the system as follows (see the textbooks [BeT96], [Ber17], [Ber19a], [Ber20a]):

**Standard One-Step Lookahead Rollout Algorithm:**

Given a base policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, start with the initial state $x_0$, and proceed forward generating a trajectory

$$\{x_0, \tilde{u}_0, x_1, \tilde{u}_1, \ldots, x_{N-1}, \tilde{u}_{N-1}, x_N\}$$

according to the system equation (2.1), by applying at each state $x_k$ a control $\tilde{u}_k$ selected by the one-step lookahead minimization

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + J_{k+1,\pi}(f_k(x_k, u_k, w_k))\right\}. \quad (2.2)$$

Throughout this paper we will focus on rollout algorithms that involve one-step lookahead minimization as in Eq. (2.2). The basic ideas extend to multistep lookahead, in which case better performance can be expected at the expense of substantially increased on-line computation. The one-step minimization (2.2), which uses $J_{k+1,\pi}$ in place of the optimal cost function $J^*$, defines a policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, where for all $x_k$ and $k$, $\tilde{\mu}_k(x_k)$ is equal to the control $\tilde{u}_k$ obtained from Eq. (2.2). This policy is referred to as the **rollout policy**. The fundamental cost improvement result here is that the rollout policy improves over the base policy in the sense that

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \quad \forall x_k, k, \quad (2.3)$$

where $J_{k,\tilde{\pi}}(x_k)$, $k = 0, \ldots, N$, is the cost-to-go of the rollout policy starting from state $x_k$ (see, e.g., [Ber17], 7.
Section 6.4, or [Ber19a, Section 2.4.2). In this paper we will prove the cost improvement property as a special case of the corresponding multiagent result in Section 2.4.

The expected value in Eq. (2.2) is the Q-factor of the pair \((x_k, u_k)\) corresponding to the base policy:

\[
Q_k,\pi(x_k, u_k) = E\left\{g_k(x_k, u_k, w_k) + J_{k+1,\pi}(f_k(x_k, u_k, w_k))\right\}.
\] (2.4)

In the “standard” implementation of rollout, at each encountered state \(x_k\), the Q-factor \(Q_k,\pi(x_k, u_k)\) is computed by some algorithm separately for each control \(u_k \in U_k(x_k)\) (often by Monte Carlo simulation). Despite the inherent parallelization possibility of this computation, in the multiagent context to be discussed shortly, the number of controls in \(U_k(x_k)\), and the attendant computation of Q-factors, grow rapidly with the number of agents, and can become very large. We will next introduce a modified rollout algorithm for the multiagent case, which requires much less computation while maintaining the cost improvement property (2.3).

2.2. The Multiagent Case

Let us assume a special structure of the control space, corresponding to a multiagent version of the problem.† In particular, we assume that the control \(u_k\) consists of \(m\) components \(u^1_k, \ldots, u^m_k\),

\[u_k = (u^1_k, \ldots, u^m_k),\]

with the component \(u^\ell_k\), \(\ell = 1, \ldots, m\), chosen by agent \(\ell\) at stage \(k\), from within a given set \(U^\ell_k(x_k)\). Thus the control constraint set is the Cartesian product‡

\[U_k(x_k) = U^1_k(x_k) \times \cdots \times U^m_k(x_k).\] (2.5)

Then the minimization (2.2) involves as many as \(q^m\) Q-factors, where \(q\) is the maximum number of elements of the sets \(U^\ell_k(x_k)\) [so that \(q^m\) is an upper bound to the number of controls in \(U_k(x_k)\), in view of its Cartesian product structure (2.5)]. Thus the computation required by the standard rollout algorithm is of order \(O(sq^m)\) per stage.

We propose an alternative rollout algorithm that achieves the cost improvement property (2.3) at much smaller computational cost, namely of order \(O(qm)\) per stage. A key idea here is that the computational

† The term “multiagent” is used widely in the literature, and serves as a useful conceptual metaphor. Our methodology, however, applies to any problem where the control \(u_k\) consists of \(m\) components, \(u_k = (u^1_k, \ldots, u^m_k)\), independently of the associated practical context.

‡ The Cartesian product structure of the constraint set is adopted here for simplicity of exposition, particularly when arguing about computational complexity. The idea of trading off control space complexity and state space complexity (cf. Section 1.3), on which this paper rests, does not depend on a Cartesian product constraint structure. Of course when this structure is present, it simplifies the computations of the methods of this paper.
We noted that a major issue in rollout is the minimization over $u_{k}^{2}$. Trading off Control Space Complexity with State Space Complexity

requirements of the rollout one-step minimization (2.2) 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 neuro-dynamic programming book [BeT96], 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. We discuss this idea next within the multiagent context.

2.3. Trading off Control Space Complexity with State Space Complexity

We noted that a major issue in rollout is the minimization over $u_{k} \in U_{k}(x_{k})$ in Eq. (2.2), which may be very time-consuming when the size of the control constraint set is large. In particular, in the multiagent case where $u_{k} = (u_{k}^{1}, \ldots, u_{k}^{m})$, the time to perform this minimization is typically exponential in $m$. In this case, we can reformulate the problem by breaking down the collective decision $u_{k}$ into $m$ individual 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-agent-at-a-time control selection. At the generic state $x_{k}$, we break down the control $u_{k}$ into the sequence of the $m$ controls $u_{k}^{1}, u_{k}^{2}, \ldots, u_{k}^{m}$, and between $x_{k}$ and the next state $x_{k+1} = f_{k}(x_{k}, u_{k}, w_{k})$, we introduce artificial intermediate “states” $(x_{k}, u_{k}^{1}, u_{k}^{2}), (x_{k}, u_{k}^{1}, u_{k}^{2}), \ldots, (x_{k}, u_{k}^{1}, \ldots, u_{k}^{m-1})$, and corresponding transitions. The choice of the last control component $u_{k}^{m}$ at “state” $(x_{k}, u_{k}^{1}, \ldots, u_{k}^{m-1})$ marks the transition to the next state $x_{k+1} = f_{k}(x_{k}, u_{k}, w_{k})$ according to the system equation, while incurring cost $g_{k}(x_{k}, u_{k}, w_{k})$; see Fig. 2.2.
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

$$\pi = \{(\mu_1^k, \ldots, \mu_m^k) | k = 0, \ldots, N - 1\}$$

of the original problem, including a base policy in the context of rollout, is admissible for the reformulated problem, and has the same cost function for the original as well as the reformulated problem.

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 corresponding \(m - 1\) cost-to-go functions \(J_k^1(x_k, u_k^1), J_k^2(x_k, u_k^2), \ldots, J_k^{m-1}(x_k, u_k^{m-1})\), in addition to \(J_k(x_k)\). On the other hand, the increase in size of the state space does not adversely affect the operation of rollout, since the Q-factor minimization (2.2) is performed for just one state at each stage. Moreover, in a different context, the increase in size of the state space can be dealt with by using function approximation, i.e., with the introduction of cost-to-go approximations

$$\tilde{J}_k^1(x_k, u_k^1, r_k^1), \tilde{J}_k^2(x_k, u_k^2, r_k^2), \ldots, \tilde{J}_k^{m-1}(x_k, u_k^{m-1}, r_k^{m-1}),$$

in addition to \(\tilde{J}_k(x_k, r_k)\), where \(r_k, r_k^1, \ldots, r_k^{m-1}\) are parameters of corresponding approximation architectures (such as feature-based architectures and neural networks).

### 2.4. Multiagent Rollout and Cost Improvement

Consider now the standard rollout algorithm applied to the reformulated problem shown in Fig. 2.2, with a given base policy \(\pi = \{\mu_0, \ldots, \mu_{N-1}\}\), which is also a policy of the original problem [so that \(\mu_k = (\mu_1^k, \ldots, \mu_m^k)\), with each \(\mu_k^\ell, \ell = 1, \ldots, m\), being a function of just \(x_k\)]. The algorithm generates a rollout policy \(\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}\), where for each stage \(k\), \(\tilde{\mu}_k\) consists of \(m\) components \(\tilde{\mu}_k^\ell\), i.e., \(\tilde{\mu}_k = (\tilde{\mu}_k^1, \ldots, \tilde{\mu}_k^m)\), and is obtained for all \(x_k\) according to the sequential one-step lookahead minimizations

$$\tilde{\mu}_k^1(x_k) \in \arg \min_{u_k^1 \in U_1^k(x_k)} E\left\{g_k(x_k, u_k^1, \mu_k^1(x_k), \ldots, \mu_k^m(x_k), w_k) + J_{k+1, \pi} \left(f_k(x_k, u_k^1, \mu_k^1(x_k), \ldots, \mu_k^m(x_k), w_k)\right)\right\},$$

$$\tilde{\mu}_k^2(x_k) \in \arg \min_{u_k^2 \in U_2^k(x_k)} E\left\{g_k(x_k, \tilde{\mu}_k^1(x_k), u_k^2, \ldots, \mu_k^m(x_k), w_k) + J_{k+1, \pi} \left(f_k(x_k, \tilde{\mu}_k^1(x_k), u_k^2, \ldots, \mu_k^m(x_k), w_k)\right)\right\},$$

$$\tilde{\mu}_k^m(x_k) \in \arg \min_{u_k^m \in U_m^k(x_k)} E\left\{g_k(x_k, \tilde{\mu}_k^{m-1}(x_k), \tilde{\mu}_k^m(x_k), w_k) + J_{k+1, \pi} \left(f_k(x_k, \tilde{\mu}_k^{m-1}(x_k), \tilde{\mu}_k^m(x_k), w_k)\right)\right\}.$$ (2.6)

Thus, when applied on-line, at \(x_k\), the algorithm generates the control \(\tilde{\mu}_k(x_k) = (\tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^m(x_k))\) via a sequence of \(m\) minimizations, once over each of the agent controls \(u_k^1, \ldots, u_k^m\), with the past controls
determined by the rollout policy, and the future controls determined by the base policy; cf. Eq. (2.6). Assuming a maximum of \( q \) elements in the constraint sets \( U_k^i(x_k) \), the computation required at each stage \( k \) is of order \( O(q) \) for each of the “states” \( x_k, (x_k, u_k^1), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1}) \), for a total of order \( O(qm) \) computation.

In the “standard” implementation of the algorithm, at each \( (x_k, u_k^1, \ldots, u_k^{\ell-1}) \) with \( \ell \leq m \), and for each of the controls \( u_k^\ell \), we generate by simulation a number of system trajectories up to stage \( N \), with all future controls determined by the base policy. We average the costs of these trajectories, thereby obtaining the Q-factor corresponding to \( (x_k, u_k^1, \ldots, u_k^{\ell-1}, u_k^\ell) \). We then select the control \( u_k^\ell \) that corresponds to the minimal Q-factor, with the controls \( u_k^1, \ldots, u_k^{\ell-1} \) held fixed at the values computed earlier.

Prerequisite assumptions for the preceding algorithm to work in an on-line multiagent setting are:

(a) All agents have access to the current state \( x_k \).

(b) There is an order in which agents compute and apply their local controls.

(c) There is intercommunication between agents, so agent \( \ell \) knows the local controls \( u_k^1, \ldots, u_k^{\ell-1} \) computed by the predecessor agents \( 1, \ldots, \ell - 1 \) in the given order.

In Sections 3 and 5, we will aim to relax Assumptions (b) and (c), through the use of autonomous multiagent rollout. Assumption (a) is satisfied if there is a central computation center (a “cloud”) that collects all the information available from the agents and from other sources, obtains the state (or a belief state in the case of partial state information problem), and broadcasts it to the agents as needed. To relax this assumption, one may assume that the agents use an estimate of the state in place of the unavailable true state in all computations. However, this possibility has not been investigated and is beyond the scope of the present paper.

Note that the rollout policy (2.6), obtained from the reformulated problem is different from the rollout policy obtained from the original problem [cf. Eq. (2.2)]. Generally, it is unclear how the two rollout policies perform relative to each other in terms of attained cost. On the other hand, both rollout policies perform no worse than the base policy, since the performance of the base policy is identical for both the reformulated problem and for the original problem. This is shown formally in the following proposition.

**Proposition 2.1:** Let \( \pi \) be a base policy and let \( \tilde{\pi} \) be a corresponding rollout policy generated by the multiagent rollout algorithm (2.6). We have

\[
J_{k, \tilde{\pi}}(x_k) \leq J_{k, \pi}(x_k), \quad \text{for all } x_k \text{ and } k. \tag{2.7}
\]
Proof: We will show Eq. (2.7) by induction, and for simplicity, we will give the proof for the case of just two agents, i.e., \( m = 2 \). Clearly Eq. (2.7) holds for \( k = N \), since \( J_{N,\bar{\pi}} = J_{N,\pi} = g_N \). Assuming that it holds for index \( k + 1 \), i.e., \( J_{k+1,\bar{\pi}} \leq J_{k+1,\pi} \), we have for all \( x_k \),

\[
J_{k,\bar{\pi}}(x_k) = E\left\{ g_k(x_k, \mu_k^1(x_k), \mu_k^2(x_k), \omega_k) + J_{k+1,\bar{\pi}}\left( f_k(x_k, \mu_k^1(x_k), \mu_k^2(x_k), \omega_k) \right) \right\}
\leq E\left\{ g_k(x_k, \mu_k^1(x_k), \mu_k^2(x_k), \omega_k) + J_{k+1,\pi}\left( f_k(x_k, \mu_k^1(x_k), \mu_k^2(x_k), \omega_k) \right) \right\}
= \min_{u_k^2 \in U_k^2(x_k)} E\left\{ g_k(x_k, \mu_k^1(x_k), u_k^2(x_k), \omega_k) + J_{k+1,\pi}\left( f_k(x_k, \mu_k^1(x_k), u_k^2(x_k), \omega_k) \right) \right\}
\leq E\left\{ g_k(x_k, \mu_k^1(x_k), u_k^2(x_k), \omega_k) + J_{k+1,\pi}\left( f_k(x_k, \mu_k^1(x_k), u_k^2(x_k), \omega_k) \right) \right\}
= J_{k,\pi}(x_k),
\]

where in the preceding relation:

(a) The first equality is the DP equation for the rollout policy \( \bar{\pi} \).

(b) The first inequality holds by the induction hypothesis.

(c) The second equality holds by the definition of the rollout algorithm as it pertains to agent 2.

(d) The third equality holds by the definition of the rollout algorithm as it pertains to agent 1.

(e) The last equality is the DP equation for the base policy \( \pi \).

The induction proof of the cost improvement property (2.7) is thus complete for the case \( m = 2 \). The proof for an arbitrary number of agents \( m \) is entirely similar. \( \text{Q.E.D.} \)

Note that there is are cases where the all-agents-at-once standard rollout algorithm can improve strictly the base policy but the one-agent-at-a-time algorithm will not. This possibility arises when the base policy is “agent-by-agent-optimal,” i.e., each agent’s control component is optimal, assuming that the control components of all other agents are kept fixed at some known values.\(^\dagger\) Such a policy may not be optimal, except under special conditions (we give an example in the next section). Thus if the base policy is agent-by-agent-optimal, multiagent rollout will be unable to improve strictly the cost function, even if this base policy

\[^{\dagger}\] This is a concept that has received much attention in the theory of team optimization, where it is known as person-by-person optimality. It has been studied in the context of somewhat different problems, which involve imperfect state information that may not be shared by all the agents; see Marschak [Mar55], Radner [Rad62], the survey by Ho [Ho80], the recent book by Zoppoli, Sanguineti, Gnecce, and Parisini [ZPB20], and the references quoted there.
Figure 2.3  Illustration of the 2-dimensional spiders-and-fly problem. The state is the vector of distances between spiders and fly. At each time period, each spider moves to a neighboring location or stays where it is. The spiders make moves with perfect knowledge of the locations or each other and of the fly. The fly moves randomly, regardless of the position of the spiders.

is strictly suboptimal. However, we speculate that a situation where a base policy is agent-by-agent-optimal is unlikely to occur in rollout practice, since ordinarily a base policy must be reasonably simple, readily available, and easily simulated.

Let us provide an example that illustrates how the size of the control space may become intractable for even moderate values of the number of agents $m$.

Example 2.1 (Spiders and Fly)

Here there are $m$ spiders and one fly moving on a 2-dimensional grid. During each time period the fly moves to a some other position according to a given state-dependent probability distribution. The spiders, working as a team, aim to catch the fly at minimum cost (thus the one-stage cost is equal to 1, until reaching the state where the fly is caught, at which time the one-stage cost becomes 0). Each spider learns the current state (the vector of spiders and fly locations) at the beginning of each time period, and either moves to a neighboring location or stays where it is. Thus each spider $i$ has as many as five choices at each time period (with each move possibly incurring a different location-dependent cost). The control vector is $u = (u^1, \ldots, u^m)$, where $u^i$ is the choice of the $i$th spider, so there are about $5^m$ possible values of $u$. However, if we view this as a multiagent problem, as per the reformulation of Fig. 2.2, the size of the control space is reduced to $\leq 5$ moves per spider.

To apply multiagent rollout, we need a base policy. A simple possibility is to use the policy that directs each spider to move on the path of minimum distance to the current fly position. According to the multiagent
rollout formalism, the spiders choose their moves in a given order, taking into account the current state, and assuming that future moves will be chosen according to the base policy. This is a tractable computation, particularly if the rollout with the base policy is truncated after some stage, and the cost of the remaining stages is approximated using a certainty equivalence approximation in order to reduce the cost of the Monte Carlo simulation.

Sample computations with this example indicate that the multiagent rollout algorithm of this section performs about as well as the standard rollout algorithm. Both algorithms perform much better than the base policy, and exhibit some “intelligence” that the base policy does not possess. In particular, in the rollout algorithms the spiders attempt to “encircle” the fly for faster capture, rather than moving straight towards the fly along a shortest path.

The following example is similar to the preceding one, but involves two flies and two spiders moving along a line, and admits an exact analytical solution. It illustrates how the multiagent rollout policy may exhibit intelligence and agent coordination that is totally lacking from the base policy. In this example, the base policy is a poor greedy heuristic, while both the standard rollout and the multiagent rollout policy are optimal.

**Example 2.2 (Spiders and Flies)**

This is a spiders-and-flies problem that admits an analytical solution. There are two spiders and two flies moving along integer locations on a straight line. For simplicity we will assume that the flies’ positions are fixed at some integer locations, although the problem is qualitatively similar when the flies move randomly. The spiders have the option of moving either left or right by one unit; see Fig. 2.4. The objective is to minimize the time to capture both flies (thus the one-stage cost is equal to 1, until reaching the state where both flies are captured, at which time the one-stage cost becomes 0). The problem has essentially a finite horizon since the spiders can force the capture of the flies within a known number of steps.

Here the optimal policy is to move the two spiders towards different flies, the ones that are initially closest to them (with ties broken arbitrarily). The minimal time to capture is the maximum of the two initial distances of the two optimal spider-fly pairings.

Let us apply multiagent rollout with the base policy that directs each spider to move one unit towards the closest fly position (and in case of a tie, move towards the fly that lies to the right). The base policy is poor because it may unnecessarily move both spiders in the same direction, when in fact only one is needed to capture the fly. This limitation is due to the lack of coordination between the spiders: each acts selfishly, ignoring the presence of the other. We will see that rollout restores a significant degree of coordination between the spiders through an optimization that takes into account the long-term consequences of the spider moves.

According to the multiagent rollout mechanism, the spiders choose their moves one-at-a-time, optimizing over the two Q-factors corresponding to the right and left moves, while assuming that future moves will be chosen according to the base policy. Let us consider a stage, where the two flies are alive while the spiders
Multiagent rollout with the given base policy starts with spider 1 at location \( n \), and calculates the two Q-factors that correspond to moving to locations \( n-1 \) and \( n+1 \), assuming that the remaining moves of the two spiders will be made using the go-towards-the-nearest-fly base policy. The Q-factor of going to \( n-1 \) is smallest because it saves in unnecessary moves of spider 1 towards fly 2, so spider 1 will move towards fly 1. The trajectory generated by multiagent rollout is to move continuously spiders 1 and 2 towards flies 1 and 2, respectively. Thus multiagent rollout generates the optimal policy.

are at different locations as in Fig. 2.4. Then the rollout algorithm will start with spider 1 and calculate two Q-factors corresponding to the right and left moves, while using the base policy to obtain the next move of spider 2, as well as the remaining moves of the two spiders. Depending on the values of the two Q-factors, spider 1 will move to the right or to the left, and it can be seen that it will choose to move away from spider 2 even if doing so increases its distance to its closest fly contrary to what the base policy will do; see Fig. 2.4. Then spider 2 will act similarly and the process will continue. Intuitively, spider 1 moves away from spider 2 and fly 2, because it recognizes that spider 2 will capture earlier fly 2, so it might as well move towards the other fly.

Thus the multiagent rollout algorithm induces implicit move coordination, i.e., each spider moves in a way that takes into account future moves of the other spider. In fact it can be verified that the algorithm will produce an optimal sequence of moves starting from any initial state. It can also be seen that ordinary rollout (both flies move at once) will also produce an optimal move sequence. Moreover, the example admits a two-dimensional generalization, whereby the two spiders, starting from the same position, will separate under the rollout policy, with each moving towards a different spider, while they will move in unison in the base policy whereby they move along the shortest path to the closest surviving fly. Again this will typically happen for both standard and multiagent rollout.

The preceding example illustrates how a poor base policy can produce a much better rollout policy, something that can be observed in many other problems. Intuitively, the key fact is that rollout is “far-sighted” in the sense that it can benefit from control calculations that reach far into future stages. The qualitative behavior described in the example has been supported by computational experiments with larger two-dimensional problems of the type described in Example 2.1.
2.5. Optimizing the Agent Order in Agent-by-Agent Rollout

In the multiagent rollout algorithm described so far, the agents optimize the control components sequentially in a fixed order. It is possible to improve performance by trying to optimize at each stage \( k \) the order of the agents.

An efficient way to do this is to first optimize over all single agent Q-factors, by solving the \( m \) minimization problems that correspond to each of the agents \( \ell = 1, \ldots, m \) being first in the multiagent rollout order. If \( \ell_1 \) is the agent that produces the minimal Q-factor, we fix \( \ell_1 \) to be the first agent in the multiagent rollout order and record the corresponding control component. Then we optimize over all single agent Q-factors, by solving the \( m - 1 \) minimization problems that correspond to each of the agents \( \ell \neq \ell_1 \) being second in the multiagent rollout order. Let \( \ell_2 \) be the agent that produces the minimal Q-factor, fix \( \ell_2 \) to be the second agent in the multiagent rollout order, record the corresponding control, and continue in the same manner. In the end, after

\[
m + (m - 1) + \cdots + 1 = \frac{m(m + 1)}{2}
\] (2.9)

minimizations, we obtain an agent order \( \ell_1, \ldots, \ell_m \) that produces a potentially much reduced Q-factor value, as well as the corresponding rollout control component selections.

The method just described likely produces better performance, and eliminates the need for guessing a good agent order, but it increases the number of Q-factor calculations needed per stage roughly by a factor \((m+1)/2\). Still this is much better than the all-agents-at-once approach, which requires an exponential number of Q-factor calculations. Moreover, the Q-factor minimizations of the above process can be parallelized, so with \( m \) parallel processors, we can perform the number of \( m(m + 1)/2 \) minimizations derived above in just \( m \) batches of parallel minimizations, which require about the same time as in the case where the agents are selected for Q-factor minimization in a fixed order. We finally note that our earlier cost improvement proof goes through again by induction, when the order of agent selection is variable at each stage \( k \).

2.6. Truncated Rollout with Terminal Cost Function Approximation

An important variation of both the standard and the multiagent rollout algorithms is truncated rollout with terminal cost function approximation. Here the rollout trajectories are obtained by running the base policy from the leaf nodes of the lookahead tree, but they are truncated after a given number of steps, while a terminal cost approximation is added to the heuristic cost to compensate for the resulting error. This is important for problems with a large number of stages, and it is also essential for infinite horizon problems where the rollout trajectories have infinite length. One possibility that works well for many problems is to simply set the terminal cost approximation to zero. Alternatively, the terminal cost function approximation may be
obtained by using some sophisticated off-line training process that may involve an approximation architecture such as a neural network or by using some heuristic calculation based on a simplified version of the problem.

3. ASYNCHRONOUS AND AUTONOMOUS ROLLOUT

In this section we consider multiagent rollout algorithms that are asynchronous in the sense that the agents may compute their rollout controls in parallel or in some irregular order rather than in sequence. An example of such an algorithm is obtained when at a given stage, agent $\ell$ computes the rollout control $\tilde{u}_k^\ell$ before knowing the rollout controls of some of the agents $1, \ldots, \ell - 1$, and uses the controls $\mu_k^1(x_k), \ldots, \mu_k^{\ell-1}(x_k)$ of the base policy in their place.

This algorithm may work well for some problems, but it does not possess the cost improvement property, and may not work well for some problems. In fact we can construct a simple example involving a single state, two agents, and two controls per agent, where the second agent does not take into account the control applied by the first agent, and as a result the rollout policy performs worse than the base policy.

Example 3.1 (Cost Deterioration in the Absence of Adequate Agent Coordination)

Consider a problem with two agents ($m = 2$) and a single state. Thus the state does not change and the costs of different stages are decoupled (the problem is essentially static). Each of the two agents has two controls: $u_k^1 \in \{0, 1\}$ and $u_k^2 \in \{0, 1\}$. The cost per stage $g_k$ is equal to 0 if $u_k^1 \neq u_k^2$, is equal to 1 if $u_k^1 = u_k^2 = 0$, and is equal to 2 if $u_k^1 = u_k^2 = 1$. Suppose that the base policy applies $u_k^1 = u_k^2 = 0$. Then it can be seen that when executing rollout, the first agent applies $u_k^1 = 1$, and in the absence of knowledge of this choice, the second agent also applies $u_k^2 = 1$ (thinking that the first agent will use the base policy control $u_k^1 = 0$). Thus the cost of the rollout policy is 2 per stage, while the cost of the base policy is 1 per stage. By contrast the rollout algorithm that takes into account the first agent’s control when selecting the second agent’s control applies $u_k^1 = 1$ and $u_k^2 = 0$, thus resulting in a rollout policy with the optimal cost of 0 per stage.

The difficulty here is inadequate coordination between the two agents. In particular, each agent uses rollout to compute the local control, each thinking that the other will use the base policy control. If instead the two agents were to coordinate their control choices, they would have applied an optimal policy.

The simplicity of the preceding example raises serious questions as to whether the cost improvement property (2.7) can be easily maintained by a distributed rollout algorithm where the agents do not know the controls applied by the preceding agents in the given order of local control selection, and use instead the controls of the base policy. One may speculate that if the agents are naturally “weakly coupled” in the sense that their choice of control has little impact in the desirability of various controls of other agents, then a more flexible inter-agent communication pattern may be sufficient for cost improvement.† An important

† In particular, one may divide the agents in “coupled” groups, and require coordination of control selection only
question is whether and to what extent agent coordination is essential. In what follows in this section, we will discuss a distributed asynchronous multiagent rollout scheme, which is based on the use of a signaling policy that provides estimates of coordinating information once the current state is known.

**Autonomous Multiagent Rollout**

An interesting possibility for autonomous control selection by the agents is to use a distributed rollout algorithm, which is augmented by a precomputed signaling policy that embodies agent coordination. The idea is to assume that the agents do not communicate their computed rollout control components to the subsequent agents in the given order of local control selection. Instead, once the agents know the state, they use precomputed approximations to the control components of the preceding agents, and apply their own control components asynchronously. We call this algorithm *autonomous multiagent rollout*. Similar to Section 2, the algorithm at the $k$th stage uses a base policy $\mu_k = \{\mu^1_k, \ldots, \mu^{m-1}_k\}$, but it also uses a second policy $\hat{\mu}_k = \{\hat{\mu}^1_k, \ldots, \hat{\mu}^{m-1}_k\}$, called the signaling policy, which is computed off-line, is known to all the agents for off-line use, and is designed to play an agent coordination role. Intuitively, $\hat{\mu}^l_k(x_k)$ embodies information about what agent $\ell$ will do at state $x_k$. This is used in turn by all other agents $i \neq \ell$ to compute asynchronously their own rollout control components on-line.

More precisely, the autonomous multiagent rollout algorithm generates a policy $\hat{\pi} = \{\hat{\mu}^0, \ldots, \hat{\mu}^{N-1}\}$ as follows. At stage $k$ and state $x_k$, $\hat{\mu}_k(x_k) = (\hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^m_k(x_k))$, is obtained according to

$$
\hat{\mu}^1_k(x_k) \in \arg \min_{u^1_k \in U^1_k(x_k)} E \left\{ g_k(x_k, u^1_k, \hat{\mu}^2_k(x_k), \ldots, \hat{\mu}^m_k(x_k), w_k) + J_{k+1,x} \left( f_k(x_k, u_k, \hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^m_k(x_k), w_k) \right) \right\},
$$

$$
\hat{\mu}^2_k(x_k) \in \arg \min_{u^2_k \in U^2_k(x_k)} E \left\{ g_k(x_k, \hat{\mu}^1_k(x_k), u^2_k, \ldots, \hat{\mu}^m_k(x_k), w_k) + J_{k+1,x} \left( f_k(x_k, \hat{\mu}^1_k(x_k), u^2_k, \ldots, \hat{\mu}^m_k(x_k), w_k) \right) \right\},
$$

and

$$
\hat{\mu}^m_k(x_k) \in \arg \min_{u^m_k \in U^m_k(x_k)} E \left\{ g_k(x_k, \hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^{m-1}_k(x_k), u^m_k, w_k) + J_{k+1,x} \left( f_k(x_k, \hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^{m-1}_k(x_k), u^m_k, w_k) \right) \right\}.
$$

(3.1)

Note that the preceding computation of the controls $\hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^m_k(x_k)$ can be done asynchronously and in parallel, and without direct agent coordination, since the signaling policy values $\hat{\mu}^1_k(x_k), \ldots, \hat{\mu}^{m-1}_k(x_k)$ are precomputed and are known to all the agents.

The simplest choice is to use as signaling policy $\hat{\pi}$ the base policy $\mu$, a possibility that we have discussed earlier in this section. However, this choice does not guarantee policy improvement as evidenced by Example within each group, while the computation of different groups may proceed in parallel. For example, in applications where the agents’ locations are distributed within some geographical area, it may make sense to form agent groups on the basis of geographic proximity, i.e., one may require that agents that are geographically near each other (and hence are more coupled) coordinate their control selections, while agents that are geographically far apart (and hence are less coupled) forego any coordination.
3.1 (see also Example 5.1 in Section 5). In fact performance deterioration with this choice is not uncommon, and has also been observed in more complicated examples.

An alternative idea is to choose the signaling policy $\hat{\mu}_k$ to approximate the multiagent rollout policy of Section 2.4 [cf. Eq. (2.6)], which is known to embody coordination between the agents. In particular, one possibility is to obtain the policy $\hat{\mu}_1^k, \ldots, \hat{\mu}_{m-1}^k, k = 0, \ldots, N - 1$, by off-line training a neural network with training samples obtained through the rollout policy of Eq. (2.6); i.e., use as signaling policy $\hat{\mu}_k$ a neural network representation of the rollout policy $\hat{\mu}_k$ of Eq. (2.6). Note that if the neural network representation were perfect, the policy defined by Eq. (3.1) would be the same as the rollout policy of Eq. (2.6). Thus we intuitively expect that if the neural network provides a good approximation of the rollout policy (2.6), the policy defined by Eq. (3.1) would have better performance than both the base policy and the signaling policy. This expectation was confirmed in the context of a challenging multirobot repair application in the paper [BKB20]. The advantage of autonomous multiagent rollout with neural network approximations is that it allows approximate policy improvement (to the extent that the functions $\hat{\mu}_i^k$ are good approximations to $\tilde{\mu}_i^k$), while at the same time allowing asynchronous distributed agent operation without on-line agent coordination through communication of their rollout control values (but still assuming knowledge of the exact state by all agents). We will return to this algorithm and provide more details in Section 5, in the context of infinite horizon problems.

4. MULTIAGENT PROBLEM FORMULATION - INFINITE HORIZON DISCOUNTED PROBLEMS

The multiagent rollout ideas that we have discussed so far can be modified and generalized to apply to infinite horizon problems. In this context, we will also consider multiagent versions of PI algorithms. We focus on discounted problems with finite number of states and controls, so that the Bellman operator is a contraction mapping, and the strongest version of the available theory applies (the solution of Bellman’s equation is unique, and strong convergence results hold for PI); see [Ber12], Chapters 1 and 2, [Ber18a], Chapter 2, or [Ber19a], Chapter 4. However, a qualitatively similar methodology can be applied to undiscounted problems involving a termination state (e.g., stochastic shortest path problems, see [BeT96], Chapter 2, [Ber12], Chapter 3, and [Ber18a], Chapters 3 and 4).

In particular, we consider a standard Markovian decision problem (MDP for short) infinite horizon discounted version of the finite horizon $m$-agent problem of Section 1.2, where $m > 1$. We assume a finite number of states $x = 1, \ldots, n$, and a control $u$ that consists of $m$ components $u_\ell, \ell = 1, \ldots, m$,

$$ u = (u_1, \ldots, u_m), $$

(for the MDP notation adopted for this section, we switch for convenience to subscript indexing for control components, and reserve superscript indexing for policy iterates). At state $x$ and stage $k$, a control $u$ is
Figure 4.1 Illustration of how to transform an \( m \)-agent infinite horizon problem into a stationary infinite horizon problem with fewer control choices available at each state (in this figure \( m = 3 \)).

At the typical stage only one agent selects a control. For example, at state \( x \), the first agent chooses \( u_1 \) at no cost leading to state \( (x, u_1) \). Then the second agent applies \( u_2 \) at no cost leading to state \( (x, u_1, u_2) \). Finally, the third agent applies \( u_3 \) leading to some state \( y \) at cost \( g(x, u, y) \), where \( u \) is the combined control of the three agents, \( u = (u_1, u_2, u_3) \). The figure shows the first three transitions of the trajectories that start from the states \( x, (x, u_1), \) and \( (x, u_1, u_2) \), respectively. Note that the state space of the transformed problem is well suited for the use of state space partitioned PI algorithms; cf. the book [Ber20a], and the papers [BeY10], [BeY12], [YuB13], [BBD10].

applied, and the system transitions to a next state \( y \) with transition probabilities \( p_{xy}(y) \) and cost \( g(x, u, y) \). When at stage \( k \) the transition cost is discounted by \( \alpha^k \), where \( \alpha \in (0, 1) \) is the discount factor. Each control component \( u_\ell \) is separately constrained to lie in a given finite set \( U_\ell(x) \) when the system is at state \( x \). Thus the control constraint is \( u \in U(x) \), where \( U(x) \) is the finite Cartesian product set

\[
U(x) = U_1(x) \times \cdots \times U_m(x).
\]

The cost function of a stationary policy \( \mu \) that applies control \( \mu(x) \in U(x) \) at state \( x \) is denoted by \( J_\mu(x) \), and the optimal cost [the minimum over \( \mu \) of \( J_\mu(x) \)] is denoted \( J^*(x) \).

An equivalent version of the problem, involving a reformulated/expanded state space is depicted in Fig. 4.1 for the case \( m = 3 \). The state space of the reformulated problem consists of

\[
x, (x, u_1), \ldots, (x, u_1, \ldots, u_m),
\]

where \( x \) ranges over the original state space, and each \( u_\ell, \ell = 1, \ldots, m \), ranges over the corresponding constraint set \( U_\ell(x) \). At each stage, the agents choose their controls sequentially in a fixed order: from state \( x \) agent 1 applies \( u_1 \in U_1(x) \) to go to state \( (x, u_1) \), then agent 2 applies \( u_2 \in U_2(x) \) to go to state \( (x, u_1, u_2) \), and so on, until finally at state \( (x, u_1, \ldots, u_m) \), agent \( m \) applies \( u_m \in U_m(x) \), completing the choice of control \( u = (u_1, \ldots, u_m) \), and effecting the transition to state \( y \) at a cost \( g(x, u, y) \), appropriately discounted.

Note that this reformulation involves the type of tradeoff between control space complexity and state
space complexity that we discussed in Section 2.3. The reformulated problem involves \( m \) cost-to-go functions

\[
J^0(x), J^1(x, u_1), \ldots, J^{m-1}(x, u_1, \ldots, u_{m-1}),
\]

with corresponding sets of Bellman equations, but a much smaller control space. Moreover, the existing analysis of rollout algorithms, including implementations, variations, and error bounds, applies to the reformulated problem; see Section 5.1 of the author’s RL textbook [Ber19a]. Similar to the finite horizon case, our implementation of the rollout algorithm, which is described next, involves one-agent-at-a-time policy improvement, while maintaining the basic cost improvement and error bound properties of rollout, since these apply to the reformulated problem.

4.1. Multiagent Policy Iteration

The policies generated by the standard PI algorithm for the reformulated problem of Fig. 4.1 are defined over the larger space and have the form

\[
\mu_0(x), \mu_1(x, u_1), \ldots, \mu_{m-1}(x, u_1, \ldots, u_{m-1}).
\]

However, policies of the preceding form can also be represented in the simpler form

\[
\mu(x) = (\mu_0(x), \mu_1(x), \ldots, \mu_{m-1}(x)),
\]

i.e., as policies for the original infinite horizon problem. We will thus consider a multiagent PI algorithm that operates over the latter class of policies.

Consistent with the multiagent rollout algorithm of Section 2.4, we introduce a one-agent-at-a-time PI algorithm that uses a modified form of policy improvement, whereby the control \( u = (u_1, \ldots, u_m) \) is optimized one-component-at-a-time, with the preceding components computed according to the improved policy, and the subsequent components computed according to the current policy. In particular, given the current policy \( \mu^k \), the next policy is obtained as

\[
\mu^{k+1} \in \tilde{M}_{\mu^k}(J_{\mu^k}),
\]

where for given \( \mu \) and \( J \), we denote by \( \tilde{M}_{\mu}(J) \) the set of policies

\[
\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)
\]

satisfying for all states \( x = 1, \ldots, n \),

\[
\tilde{\mu}_1(x) \in \arg \min_{u_1 \in U_1(x)} \sum_{y=1}^n p_{xy}(u_1, \mu_2(x), \ldots, \mu_m(x)) \left( g(x, u_1, \mu_2(x), \ldots, \mu_m(x), y) + \alpha J(y) \right),
\]

21
\[
\hat{\mu}_1(x) \in \arg\min_{u_2 \in U_2(x)} \sum_{y=1}^{n} p_{xy}(\hat{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x)) \left( g(x, \hat{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x), y) + \alpha J(y) \right),
\]

\[
\hat{\mu}_m(x) \in \arg\min_{u_m \in U_m(x)} \sum_{y=1}^{n} p_{xy}(\hat{\mu}_1(x), \hat{\mu}_2(x), \ldots, \hat{\mu}_{m-1}(x), u_m) \left( g(x, \hat{\mu}_1(x), \hat{\mu}_2(x), \ldots, \hat{\mu}_{m-1}(x), u_m, y) + \alpha J(y) \right).
\]

Note that \( \hat{\mathcal{M}}_\mu(J) \) may not consist of a single policy, since there may be multiple controls attaining the minima in the preceding equations.

Note that each of the \( m \) minimizations (4.3) can be performed for each state \( x \) independently, i.e., the computations for state \( x \) do not depend on the computations for other states, thus allowing the use of parallel computation over the different states. On the other hand, the computations corresponding to individual agent components must be performed in sequence (in the absence of special structure related to coupling of the control components through the transition probabilities and the cost per stage). It will also be clear from the subsequent analysis that for convergence purposes, the ordering of the components is not important, and it may change from one policy improvement operation to the next. In fact there are versions of the algorithm, which aim to optimize over multiple component orders, and are amenable to parallelization as discussed in Section 2.5.

Similar to the finite horizon case of Section 2, the salient feature of the one-agent-at-a-time policy improvement operation (4.3) is that it is far more economical than the standard policy improvement: it requires a sequence of \( m \) minimizations, once over each of the control components \( u_1, \ldots, u_m \). In particular, for the minimization over the typical component \( u_\ell \), the preceding components \( u_1, \ldots, u_{\ell-1} \) have been computed earlier by the minimization that yielded the policy components \( \hat{\mu}_1, \ldots, \hat{\mu}_{\ell-1} \), while the following controls \( u_{\ell+1}, \ldots, u_m \) are determined by the current policy components \( \mu_{\ell+1}, \ldots, \mu_m \). Thus, if the number of controls within each component constraint set \( U_\ell(x) \) is bounded by a number \( q \), the one-agent-at-a-time operation (4.3) requires at most \( qm \) Q-factor calculations.

By contrast, since the number of elements in the constraint set \( U(x) \) is bounded by \( q^m \), the corresponding number of Q-factor calculations in the standard policy improvement operation is bounded by \( q^m \). Thus in the one-agent-at-a-time policy improvement the number of Q-factors grows linearly with \( m \), as compared to the standard policy improvement, where the number of Q-factor calculations grows exponentially with \( m \).

4.2. Convergence to an Agent-by-Agent Optimal Policy

An important fact is that multiagent PI need not converge to an optimal policy. Instead we will show convergence to a different type of optimal policy, which we will now define.
We say that a policy \( \mu = \{ \mu_1, \ldots, \mu_m \} \) is agent-by-agent optimal if \( \mu \in \tilde{M}_\mu(J_\mu) \), or equivalently [cf. Eq. (4.3)], if for all \( x = 1, \ldots, n \), and \( \ell = 1, \ldots, m \), we have

\[
\sum_{y=1}^{n} p_{xy}(\mu_1(x), \ldots, \mu_m(x)) \left( g(x, \mu_1(x), \ldots, \mu_m(x), y) + \alpha J_\mu(y) \right) = \min_{u_\ell \in U_\ell(x)} \sum_{y=1}^{n} p_{xy}(\mu_1(x), \ldots, \mu_{\ell-1}(x), u_\ell, \mu_{\ell+1}(x), \ldots, \mu_m(x)) \cdot \left( g(x, \mu_1(x), \ldots, \mu_{\ell-1}(x), u_\ell, \mu_{\ell+1}(x), \ldots, \mu_m(x), y) + \alpha J_\mu(y) \right).
\]

To interpret this definition, let a policy \( \mu = \{ \mu_1, \ldots, \mu_m \} \) be given, and consider for every \( \ell \in \{1, \ldots, m\} \) the single agent DP problem where for all \( i \neq \ell \) the \( i \)th policy components are fixed at \( \mu_i \), while the \( \ell \)th policy component is subject to optimization. Then by viewing the preceding definition as the optimality condition for all the single agent problems, we can conclude that \( \mu \) is agent-by-agent optimal if each component \( \mu_\ell \) is optimal for the \( \ell \)th single agent problem; in other words by using \( \mu_\ell \), each agent \( \ell \) acts optimally, assuming all other agents \( i \neq \ell \) do not deviate from the policy components \( \mu_i \).

Note that an (overall) optimal policy is agent-by-agent optimal, but the reverse is not true as the following example shows.

**Example 4.1 (Counterexample for Agent-by-Agent Optimality)**

Consider an infinite horizon problem, which involves two agents \( (m = 2) \) and a single state \( x \). Thus the state does not change and the costs of different stages are decoupled (the problem is essentially static). Each of the two agents chooses between the two controls 0 and 1: \( u_1 \in \{0, 1\} \) and \( u_2 \in \{0, 1\} \). The cost per stage \( g \) is equal to 2 if \( u_1 \neq u_2 \), is equal to 1 if \( u_1 = u_2 = 0 \), and is equal to 0 if \( u_1 = u_2 = 1 \). The unique optimal policy is to apply \( \mu_1(x) = 1 \) and \( \mu_2(x) = 1 \). However, it can be seen that the suboptimal policy that applies \( \mu_1(x) = 0 \) and \( \mu_2(x) = 0 \) is agent-by-agent optimal.

The preceding example is representative of an entire class of DP problems where an agent-by-agent optimal policy is not overall optimal. Any static (single step) multivariable optimization problem where there are nonoptimal solutions that cannot be improved upon by a coordinate descent operation (sequential component minimizations, one-component-at-a-time) can be turned into an infinite horizon DP example where these nonoptimal solutions define agent-by-agent optimal policies that are not overall optimal. Conversely, one may search for problem classes where an agent-by-agent optimal policy is guaranteed to be (overall) optimal among the type of multivariable optimization problems where coordinate descent is guaranteed to converge to an optimal solution. For example positive definite quadratic problems or problems involving differentiable strictly convex functions (see [Ber16], Section 3.7). Generally, agent-by-agent optimality may be viewed as an acceptable form of optimality for many types of problems.

Our main result is that the one-agent-at-a-time PI algorithm generates a sequence of policies that
converges in a finite number of iterations to a policy that is agent-by-agent optimal. However, we will show that even if the final policy produced by one-agent-at-a-time PI is not optimal, each generated policy is no worse than its predecessor. In the presence of approximations, which are necessary for large problems, it appears that the policies produced by multiagent PI are often of sufficient quality for practical purposes, and not substantially worse than the ones produced by (far more computationally intensive) approximate PI methods that are based on all-agents-at-once lookahead minimization.

For the proof of our convergence result, we will use a special rule for breaking ties in the policy improvement operation in favor of the current policy component. This rule is easy to enforce, and guarantees that the algorithm cannot cycle between policies. Without this tie-breaking rule, the following proof shows that while the generated policies may cycle, the corresponding cost function values converge to a cost function value of some agent-by-agent optimal policy.

In the following proof and later all vector inequalities are meant to be componentwise, i.e., for any two vectors $J$ and $J'$, we write

$$J \leq J' \quad \text{if} \quad J(x) \leq J'(x) \text{ for all } x.$$  

For notational convenience, we also introduce the Bellman operator $T_\mu$ that maps a function of the state $J$ to the function of the state $T_\mu J$ given by

$$(T_\mu J)(x) = \sum_{y=1}^{n} p_{xy}(\mu(x)) \left( g(x, \mu(x), y) + \alpha J_\mu(y) \right), \quad x = 1, \ldots, n.$$  

**Proposition 4.1:** Let $\{\mu^k\}$ be a sequence generated by the one-agent-at-a-time PI algorithm (4.2) assuming that ties in the policy improvement operation of Eq. (4.3) are broken as follows: If for any $\ell = 1, \ldots, m$ and $x$, the control component $\mu_\ell(x)$ attains the minimum in Eq. (4.3), we choose

$$\tilde{\mu}_\ell(x) = \mu_\ell(x)$$

[even if there are other control components within $U_\ell(x)$ that attain the minimum in addition to $\mu_\ell(x)$]. Then for all $x$ and $k$, we have

$$J_{\mu^k}(x) \leq J_{\mu^{k+1}}(x),$$

and after a finite number of iterations, we have $\mu^{k+1} = \mu^k$, in which case the policies $\mu^{k+1}$ and $\mu^k$ are agent-by-agent optimal.

**Proof:** We recall that for given $\mu$ and $J$, we denote by $\tilde{M}_\mu(J)$ the set of policies $\tilde{\mu}$ satisfying Eq. (4.3).
The critical step of the proof is the following monotone decrease inequality:

\[
T_{\tilde{\mu}} J \leq T_{\mu} J \leq J, \quad \text{for all } J \text{ with } T_{\mu} J \leq J \text{ and } \tilde{\mu} \in \widetilde{M}_\mu(J),
\]

which yields as a special case \(T_{\tilde{\mu}} J_{\mu} \leq J_{\mu}\), since \(T_{\mu} J_{\mu} = J_{\mu}\). This parallels a key inequality for standard PI, namely that \(T_{\tilde{\mu}} J_{\mu} \leq J_{\mu}\), for all \(\tilde{\mu}\) such that \(T_{\tilde{\mu}} J_{\mu} = T J_{\mu}\), which lies at the heart of its convergence proof. Once Eq. (4.4) is shown, the monotonicity of the operator \(T_{\tilde{\mu}}\) implies the cost improvement property \(J_{\tilde{\mu}} \leq J_{\mu}\), and by using the finiteness of the set of policies, the finite convergence of the algorithm will follow.

We will give the proof of the monotone decrease inequality (4.4) for the case \(m = 2\). The proof for an arbitrary number of components \(m > 2\) is entirely similar. Indeed, if \(T_{\tilde{\mu}} J \leq J\) and \(\tilde{\mu} \in \widetilde{M}_\mu(J)\), we have for all states \(x\),

\[
(T_{\tilde{\mu}} J)(x) = \sum_{y=1}^{n} p_{xy}(\tilde{\mu}_1(x), \tilde{\mu}_2(x)) \left( g(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), y) + \alpha J(y) \right)
\]

\[
= \min_{u_2 \in U_2(x)} \sum_{y=1}^{n} p_{xy}(\tilde{\mu}_1(x), u_2) \left( g(x, \tilde{\mu}_1(x), u_2, y) + \alpha J(y) \right)
\]

\[
\leq \sum_{y=1}^{n} p_{xy}(\tilde{\mu}_1(x), \mu_2(x)) \left( g(x, \tilde{\mu}_1(x), \mu_2(x), y) + \alpha J(y) \right)
\]

\[
= \min_{u_1 \in U_1(x)} \sum_{y=1}^{n} p_{xy}(u_1, \mu_2(x)) \left( g(x, u_1, \mu_2(x), y) + \alpha J(y) \right)
\]

\[
\leq \sum_{y=1}^{n} p_{xy}(\mu_1(x), \mu_2(x)) \left( g(x, \mu_1(x), \mu_2(x), y) + \alpha J(y) \right)
\]

\[
= (T_{\mu} J)(x)
\]

\[
\leq J(x),
\]

where:

(1) The first equality uses the definition of the Bellman operator for \(\tilde{\mu}\).

(2) The first two inequalities hold by the definition of policies \(\tilde{\mu} \in \widetilde{M}_\mu(J)\).

(3) The last equality is the definition of the Bellman operator for \(\mu\).

(4) The last inequality is the assumption \(T_{\tilde{\mu}} J \leq J\).

By letting \(J = J_{\mu_k}\) in the monotone decrease inequality (4.4), we have \(T_{\mu_{k+1}} J_{\mu_k} \leq J_{\mu_k}\). In view of the monotonicity of \(T_{\mu_{k+1}}\), we also have \(T_{\mu_{k+1}} J_{\mu_k} \leq T_{\mu_{k+1}} J_{\mu_{k+1}}\) for all \(\ell \geq 1\), so that

\[
J_{\mu_{k+1}} = \lim_{\ell \to \infty} T_{\mu_{k+1}} J_{\mu_k} \leq T_{\mu_{k+1}} J_{\mu_{k+1}} \leq J_{\mu_k}.
\]

It follows that either \(J_{\mu_{k+1}} = J_{\mu_k}\), or else we have strict policy improvement, i.e., \(J_{\mu_{k+1}}(x) < J_{\mu_k}(x)\) for at least one state \(x\). As long as strict improvement occurs, no generated policy can be repeated by the
algorithm. Since there are only finitely many policies, it follows that within a finite number of iterations, we will have \( J_{\mu^{k+1}} = J_{\mu^k} \). Once this happens, equality will hold throughout in Eq. (4.5). This implies, using also the preceding proof, that

\[
\sum_{y=1}^{n} p_{xy}(\mu_1^{k+1}(x), \mu_2^{k+1}(x)) \left( g(x, \mu_1^{k+1}(x), \mu_2^{k+1}(x), y) + \alpha J_{\mu^k}(y) \right)
= \min_{u_2 \in U_2(x)} \sum_{y=1}^{n} p_{xy}(\mu_1^{k+1}(x), u_2) \left( g(x, \mu_1^{k+1}(x), u_2, y) + \alpha J_{\mu^k}(y) \right)
= \sum_{y=1}^{n} p_{xy}(\mu_1^{k+1}(x), \mu_2^{k}(x)) \left( g(x, \mu_1^{k+1}(x), \mu_2^{k}(x), y) + \alpha J_{\mu^k}(y) \right),
\]

and

\[
\sum_{y=1}^{n} p_{xy}(\mu_1^{k+1}(x), \mu_2^{k}(x)) \left( g(x, \mu_1^{k+1}(x), \mu_2^{k}(x), y) + \alpha J_{\mu^k}(y) \right)
= \min_{u_1 \in U_1(x)} \sum_{y=1}^{n} p_{xy}(u_1, \mu_2^{k}(x)) \left( g(x, u_1, \mu_2^{k}(x), y) + \alpha J_{\mu^k}(y) \right)
= \sum_{y=1}^{n} p_{xy}(\mu_1^{k}(x), \mu_2^{k}(x)) \left( g(x, \mu_1^{k}(x), \mu_2^{k}(x), y) + \alpha J_{\mu^k}(y) \right).
\]

In view of our tie breaking rule, this equation implies that \( \mu_1^{k+1} = \mu_1^k \), and then Eq. (4.6) implies that \( \mu_2^{k+1} = \mu_2^k \). Thus we have \( \mu^{k+1} = \mu^k \), and from the preceding two equations, it follows that \( \mu^{k+1} \) and \( \mu^k \) are agent-by-agent optimal. \( \text{Q.E.D.} \)

### 4.3. Variants and Approximations

An important variant of multiagent PI is an optimistic version, whereby policy evaluation is performed by using a finite number of one-agent-at-a-time value iterations. This type of method together with a theoretical convergence analysis of multiagent value iteration is given in the paper [Ber20b] and in the monograph [Ber20a] (Sections 5.4-5.6). It is outside the scope of this paper.

As Example 4.1 shows, there may be multiple agent-by-agent optimal policies, with different cost functions. This illustrates that the policy obtained by the multiagent PI algorithm may depend on the starting policy. It turns out that the same example can be used to show that the policy obtained by the algorithm depends also on the order in which the agents select their controls.

**Example 4.2 (Dependence of the Final Policy on the Agent Iteration Order)**

Consider the problem of Example 4.1. In this problem there are two agent-by-agent optimal policies: the optimal policy \( \mu^* \) where \( \mu_1^*(x) = 1 \) and \( \mu_2^*(x) = 1 \), and the suboptimal policy \( \hat{\mu} \) where \( \hat{\mu}_1(x) = 0 \) and \( \hat{\mu}_2(x) = 0 \).

Let the starting policy be \( \mu^0 \) where \( \mu_1^0(x) = 1 \) and \( \mu_2^0(x) = 0 \). Then if agent 1 iterates first, the algorithm will
Approximation $\hat{\mu}$ to Multiagent Rollout Policy $\tilde{\mu}$

**Figure 4.2** Approximate multiagent PI with value and policy networks. The value network provides a trained approximation to the current base policy $\mu$. The policy network provides a trained approximation $\hat{\mu}$ to the corresponding multiagent rollout policy $\tilde{\mu}$. The policy network may consist of $m$ separately trained policy networks, one for each of the agent policies $\hat{\mu}_1, \ldots, \hat{\mu}_m$.

terminate with the suboptimal policy, $\mu^1 = \hat{\mu}$, while if agent 2 iterates first, the algorithm will terminate with the optimal policy, $\mu^1 = \mu^\ast$.

As noted in Section 2.5, it is possible to try to optimize the agent order at each iteration. In particular, first optimize over all single agent Q-factors, by solving the $m$ minimization problems that correspond to each of the agents $\ell = 1, \ldots, m$ being first in the multiagent rollout order. If $\ell_1$ is the agent that produces the minimal Q-factor, we fix $\ell_1$ to be the first agent in the multiagent rollout order. Then we optimize over all single agent Q-factors, by solving the $m - 1$ minimization problems that correspond to each of the agents $\ell \neq \ell_1$ being second in the multiagent rollout order, etc.

**Value and Policy Neural Network Approximations**

There are also several possible versions for approximate one-agent-at-a-time PI, including the use of value and policy neural networks. In particular, the multiagent policy improvement operation (4.3) may be performed at a sample set of states $x^s$, $s = 1, \ldots, q$, thus yielding a training set of state-rollout control pairs $(x^s, \tilde{\mu}(x^s))$, $s = 1, \ldots, q$, which can be used to train a (policy) neural network to generate an approximation $\hat{\mu}$ to the policy $\tilde{\mu}$. The policy $\hat{\mu}$ becomes the new base policy and can be used in turn to train a (value) neural network that approximates its cost function value $J_{\hat{\mu}}$. The approximate multiagent PI cycle can thus be continued (cf. Fig. 4.2). Note that the training of the agent policies $\hat{\mu}_1, \ldots, \hat{\mu}_m$ may be done separately for each agent, with $m$ separate neural networks. With this scheme, the difficulty with a large control space is overcome by one-agent-at-a-time policy improvement, while the difficulty with a large state space is overcome by training value and policy networks.

The RL books [Ber19a] and [Ber20a] provide a lot of details relating to the structure and the training of value and policy networks in various contexts, some of which apply to the algorithms of the present paper. These include the use of distributed asynchronous algorithms that are based on partitioning of the state space
and training different networks on different sets of the state space partition; see also the paper [BBW20], which applies partitioning to the solution of a challenging class of partial state information problems.

**Value and Policy Approximations with Aggregation**

One of the possibilities for value and policy approximations in multiagent rollout arises in the context of aggregation; see the books [Ber12] and [Ber19a], and the references quoted there. In particular, let us consider the aggregation with representative features framework of [Ber19a], Section 6.2 (see also [Ber12], Section 6.5). The construction of the features may be done with sophisticated methods, including the use of a deep neural network as discussed in the paper [Ber18b]. Briefly, in this framework we introduce an expanded DP problem involving a finite number of additional states $i = 1, \ldots, n$, called aggregate states. Each aggregate state $i$ is associated with a subset $X_i$ of the system’s state space $X$. We assume that the sets $X_i, i = 1, \ldots, n$, are nonempty and disjoint, and collectively include every state of $X$. We also introduce aggregation probabilities mapping an aggregate state $i$ to the subset $X_i$, and disaggregation probabilities $\phi_{yj}$ mapping system states $y$ to subsets of aggregate states $X_j$.

A base policy $\mu$ defines a set of aggregate state costs $r_\mu(j), j = 1, \ldots, n$, which can be computed by simulation involving an “aggregate” Markov chain (see [Ber12], [Ber19a]). The aggregate costs $r_\mu(j)$ define an approximation $\hat{J}_\mu$ of the cost function $J_\mu$ of the base policy, through the equation

$$\hat{J}_\mu(y) = \sum_{j=1}^{n} \phi_{yj} r_\mu(j), \quad y \in X.$$  

Then an (approximate) multiagent rollout policy $\hat{\mu}$ can be defined by one-step lookahead using $\hat{J}_\mu$ in place of $J_\mu$, i.e., $\hat{\mu} \in \hat{\mathcal{M}}_\mu(\hat{J}_\mu)$, where the set $\hat{\mathcal{M}}_\mu(J)$ is defined for any $\mu$ and $J$ by Eq. (4.3). In other words, the multiagent rollout algorithm with aggregation is defined by $\hat{\mu} \in \hat{\mathcal{M}}_\mu(\hat{J}_\mu)$ instead of its counterpart without aggregation, which is defined by $\hat{\mu} \in \hat{\mathcal{M}}_\mu(J_\mu)$.

Note that using an approximation architecture based on aggregation has a significant advantage over a neural network architecture because aggregation induces a DP structure that facilitates PI convergence and improves associated error bounds (see [Ber12] and [Ber19a]). In particular, a multiagent PI algorithm based on aggregation admits a convergence result like the one of Prop. 4.1, except that this result asserts convergence to an agent-by-agent optimal policy for the associated aggregate problem.

**4.4. Truncated Multiagent Rollout and Error Bound**

Another approximation possibility, which may also be combined with value and policy network approximations is truncated rollout, which operates similar to the finite horizon case described in Section 2.5. Here, we use multiagent one-step lookahead, we then apply rollout with base policy $\mu$ for a limited number of steps, and finally we approximate the cost of the remaining steps using some terminal cost function approximation
In truncated rollout schemes, $J'$ may be heuristic, based on problem approximation, or based on a more systematic simulation methodology. For example, the values $J_\mu(x)$ may be computed by simulation for all $x$ in a subset of representative states, and $J'$ may be selected from a parametric class of functions through training, e.g., a least squares regression of the computed values. This approximation may be performed off-line, outside the time-sensitive restrictions of a real-time implementation, and the result $\tilde{J}$ may be used on-line in place of $J_\mu$ as a terminal cost function approximation.

We have the following performance bound the proof of which are given in [Ber20a] (Prop. 5.2.6).

**Proposition 4.2: (Performance Bounds for Multiagent Truncated Rollout)** Let $m$ be a positive integer, let $\mu$ be a policy, and let $J$ be a function of the state. Consider the multiagent rollout scheme that consists of one-step lookahead, followed by rollout with a policy $\mu$ for a given number of steps, followed by a terminal cost function approximation $J'$. Let $\tilde{\mu}$ be the generated rollout policy.

(a) We have

$$J_{\tilde{\mu}}(x) \leq \tilde{J}(x) + \frac{c}{1 - \alpha}, \quad x = 1, \ldots, n,$$

where

$$c = \max_{x=1,\ldots,n} \left( (T_{\mu}J')(x) - J'(x) \right).$$

(b) We have

$$J_{\tilde{\mu}}(x) \leq J_\mu(x) + \frac{2}{1 - \alpha} \|J' - J_\mu\|, \quad x = 1, \ldots, n.$$
similar to Section 3, this algorithm uses a base policy \( \mu = (\mu_1, \ldots, \mu_m) \), but it also uses a signaling policy \( \hat{\mu} = (\hat{\mu}_1, \ldots, \hat{\mu}_m) \), which is computed off-line.

In particular, given a base policy \( \mu \) and a signaling policy \( \hat{\mu} \), the autonomous multiagent rollout algorithm generates a policy \( \hat{\mu} \) as follows. At state \( x \), \( \hat{\mu}(x) = (\hat{\mu}_1(x), \ldots, \hat{\mu}_m(x)) \), is obtained according to

\[
\hat{\mu}_1(x) \in \arg \min_{u_1 \in U_1(x)} E\left\{g(x, u_1, u_2(x), \ldots, u_m(x), w) + J_{\mu}\left(f(x, u_1, u_2(x), \ldots, u_m(x), w)\right)\right\},
\]

\[
\hat{\mu}_2(x) \in \arg \min_{u_2 \in U_2(x)} E\left\{g(x, \hat{\mu}_1(x), u_2, \ldots, u_m(x), w) + J_{\mu}\left(f(x, \hat{\mu}_1(x), u_2, \ldots, u_m(x), w)\right)\right\},
\]

\[
\vdots \quad \quad \quad \quad \vdots \quad \quad \quad \quad \vdots
\]

\[
\hat{\mu}_m(x) \in \arg \min_{u_m \in U_m(x)} E\left\{g(x, \hat{\mu}_1(x), \ldots, \hat{\mu}_{m-1}(x), u_m, w) + J_{\mu}\left(f(x, \hat{\mu}_1(x), \ldots, \hat{\mu}_{m-1}(x), u_m, w)\right)\right\}. \quad (5.1)
\]

Note that the preceding computation of the controls \( \hat{\mu}_1(x), \ldots, \hat{\mu}_m(x) \) can be done asynchronously and in parallel, and without agent intercommunication of their computed controls, since the signaling policy values \( \hat{\mu}_1(x), \ldots, \hat{\mu}_{m-1}(x) \) and the base policy values \( \mu_1(x), \ldots, \mu_{m-1}(x) \) are available to all the agents.

There is no restriction on the signaling policy, but of course its choice affects the performance of the corresponding autonomous multiagent rollout algorithm. The simplest possibility is to use as signaling policy the base policy; i.e., \( \hat{\mu} = \mu \). However, this choice does not guarantee policy improvement as evidenced by Example 3.1. Still, using the base policy as signaling policy can be an attractive possibility, which one may wish to try, in view of its simplicity and its parallelization potential. On the other hand, if the signaling policy \( \hat{\mu} \) is taken to be the (nonautonomous) multiagent rollout policy \( \tilde{\mu} \) defined by

\[
\tilde{\mu}_1(x) \in \arg \min_{u_1 \in U_1(x)} E\left\{g(x, \tilde{\mu}_1(x), u_1, \ldots, u_m(x), w) + J_{\mu}\left(f(x, \tilde{\mu}_1(x), u_1, \ldots, u_m(x), w)\right)\right\},
\]

\[
\tilde{\mu}_2(x) \in \arg \min_{u_2 \in U_2(x)} E\left\{g(x, \tilde{\mu}_1(x), u_2, \ldots, u_m(x), w) + J_{\mu}\left(f(x, \tilde{\mu}_1(x), u_2, \ldots, u_m(x), w)\right)\right\},
\]

\[
\vdots \quad \quad \quad \quad \vdots
\]

\[
\tilde{\mu}_m(x) \in \arg \min_{u_m \in U_m(x)} E\left\{g(x, \tilde{\mu}_1(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, w) + J_{\mu}\left(f(x, \tilde{\mu}_1(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, w)\right)\right\}. \quad (5.2)
\]

the autonomous and nonautonomous multiagent rollout policies coincide, so nothing is gained from the use of this signaling policy.

A related interesting possibility is to choose the signaling policy \( \hat{\mu} \) to approximate the multiagent rollout policy of Eq. (5.2). In particular, we may obtain the policy \( \hat{\mu}_1, \ldots, \hat{\mu}_{m-1} \), by off-line training and approximation in policy space using a neural network, with a training set generated by the multiagent rollout policy (5.2); cf. Section 4.3 and Fig. 4.2. If the neural network were to provide perfect approximation of the rollout policy, the policy defined by Eq. (5.1) would be the same as the rollout policy, as noted earlier. Thus, intuitively, if the neural network provides a good approximation of the rollout policy (2.6), the policy
defined by Eq. (5.1) will have better performance than both the base policy and the signaling policy. This was confirmed in the paper [BKB20], in the context of a multirobot repair application. The advantage of autonomous multiagent rollout with neural network approximations is that it allows approximate policy improvement (to the extent that the functions $\hat{\mu}_i$ are good approximations to $\tilde{\mu}_i$), while at the same time allowing autonomous agent operation without on-line agent coordination through communication of their rollout control values, as in the multiagent PI algorithm of Section 4.1. The following example aims to illustrate these ideas.

**Example 5.1 (Autonomous Spiders and Flies)**

Let us return to the two-spiders-and-two-flies Example 2.2, and use it as a test of the autonomous multiagent rollout algorithm. Formally, we view the problem as an MDP of the stochastic shortest path type. Recall that here the base policy is poor and moves each spider selfishly towards the closest surviving fly with no coordination with the other spider, while both the standard and the multiagent rollout algorithm are optimal, as discussed in Example 2.2.

We will now apply autonomous multiagent rollout with a signaling policy that is arbitrary. This also includes the case where the signaling policy is an error-corrupted version of the standard (nonautonomous) multiagent rollout policy; cf. the preceding discussion. The errors can be viewed as the result of the approximation introduced by a policy network that aims to represent the multiagent rollout policy (which is optimal as discussed in Example 2.2). Then it can be verified that the autonomous multiagent rollout policy with arbitrary signaling policy acts optimally as long as the spiders are separated on the line by more than one unit. What is happening here is that the Q-factors that are minimized in Eq. (5.1) involve a first stage cost (which is fixed at 1 and is independent of the signaling policy), and the cost the base policy $J_{\mu}(y)$ starting from the next state $y$, which is not sufficiently affected by the signaling policy $\hat{\mu}$ to change the outcome of the Q-factor minimizations (5.1). For a counterpart to this situation, suppose that the two spiders start at the same position, and we use as signaling policy the base policy. Then the algorithm gets locked onto an oscillation where the spiders keep moving together back and forth, and never capture the flies. Thus, using the base policy as the signaling policy can lead to very poor performance when starting from some of the states/spider positions.

The preceding example illustrates a situation in which approximation errors in the calculation of the signaling policy matter little. This is the case where at the current state the agents are sufficiently decoupled so that there is a dominant Q-factor in the minimization (5.1) whose dominance is not affected much by the choice of the signaling policy. As noted in Section 3, one may exploit this type of structure by dividing the agents in “coupled” groups, and require coordination of the rollout control selections only within each group, while the computation within different groups may proceed in parallel with a signaling policy such as the base policy. Then the computation time/overhead for selecting rollout controls one-agent-at-a-time will be proportional to the size of the largest group rather than proportional to the number of agents.

Analysis that quantifies the sensitivity of the performance of the autonomous multiagent rollout policy
with respect to problem structure is an interesting direction for further research. The importance of such an analysis is magnified by the significant implementation advantages of autonomous versus nonautonomous rollout schemes: agent computations of their respective controls can proceed asynchronously and in parallel without the need of interagent coordination.

6. CONCLUDING REMARKS

We have shown that in the context of multiagent problems, an agent-by-agent version of the rollout algorithm has greatly reduced computational requirements, while still maintaining the fundamental cost improvement property of the standard rollout algorithm. There are several variations of rollout algorithms for multiagent problems, which deserve attention. Moreover, additional computational tests in some practical multiagent settings will be helpful in comparatively evaluating some of these variations.

We have primarily focused on the cost improvement property, and the important fact that it can be achieved at a much reduced computational cost. However, it is useful to keep in mind that the agent-by-agent rollout algorithm is simply the standard all-agents-at-once rollout algorithm applied to the (equivalent) reformulated problem of Fig. 2.2 (or Fig. 4.1 in the infinite horizon case). As a result, all known insights, results, error bounds, and approximation techniques for standard rollout apply in suitably reformulated form. The fact that multiagent rollout cannot improve strictly over a (possibly suboptimal) policy that is agent-by-agent optimal is a theoretical limitation, which, however, for many problems does not seem to prevent the method from performing comparably to the far more computationally expensive (and even intractable for problems with a modest number of agents) standard rollout algorithm.

In this paper, we have assumed that the control constraint set is finite in order to argue about the computational efficiency of the agent-by-agent rollout algorithm. The rollout algorithm itself and its cost improvement property are valid even in the case where the control constraint set is infinite, including the model predictive control context (cf. Section 2.5 of the RL book [Ber19a]), and linear-quadratic problems. However, it is as yet unclear whether agent-by-agent rollout offers an advantage in the infinite control space case, especially if the one-step lookahead minimization in the policy improvement operation is not done by discretization of the control constraint set, and exhaustive enumeration and comparison of the associated Q-factors.

We have also discussed an agent-by-agent version of PI for discounted infinite horizon problems, which uses one-component-at-a-time policy improvement. While this algorithm may terminate with a suboptimal policy that is agent-by-agent optimal, it may produce comparable performance to the standard PI algorithm, which however is computationally intractable even for a moderate number of agents. Moreover, our multiagent PI convergence result of Prop. 4.1 can be extended beyond the finite-state discounted context to more general infinite horizon DP contexts, where the PI algorithm is well-suited for algorithmic solution. Other
extensions include agent-by-agent variants of value iteration, optimistic PI, and other related methods. The analysis of such extensions is reported separately; see [Ber20a] and [Ber20b].

We have also proposed new autonomous multiagent rollout schemes for both finite and infinite horizon problems. The idea is to use a precomputed signaling policy, which embodies sufficient agent coordination to obviate the need for interagent communication during the on-line implementation of the algorithm. In this way the agents may apply their control components asynchronously and in parallel. We have still assumed, however, that the agents share perfect state information (or belief state information in the context of partial state observation problems). Intuitively, for many problems it should be possible to implement effective autonomous multiagent rollout schemes that use state estimates in place of exact states. Computational experimentation with such schemes should be very useful and may lead to improved understanding of their properties.

We finally mention that the idea of agent-by-agent rollout also applies within the context of challenging deterministic discrete/combinatorial optimization problems, which involve constraints that couple the controls of different stages. While we have not touched upon this subject in the present paper, we have discussed the corresponding constrained multiagent rollout algorithms separately in the paper [Ber20c].

Several unresolved questions remain regarding algorithmic variations and conditions that guarantee that our algorithms obtain an optimal policy rather than one that is agent-by-agent optimal (the paper [Ber20c] and monograph [Ber20a] provide relevant discussions). Moreover, approximate versions of our algorithms that use value and policy network approximations are of great practical interest, and are a subject for further investigation. Finally, the basic idea of our approach, namely simplifying the one-step lookahead minimization defining the Bellman operator while maintaining some form of cost improvement or convergence guarantee, can be extended in other directions to exploit special problem structures.

7. REFERENCES


