Multiagent Reinforcement Learning: Rollout and Policy Iteration

Dimitri Bertsekas

***Abstract*—We discuss the solution of complex multistage decision problems using methods that are based on the idea of policy iteration (PI), 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 has a robustness property: it 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 problems where the control consists of multiple components each selected (conceptually) by a separate agent. This is the class of multiagent problems where the agents have a shared objective function, and a shared and perfect state information. Based on a problem reformulation that trades off control space complexity with state space complexity, we develop 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 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.

For discounted and other infinite horizon problems, we also consider exact and approximate PI algorithms involving a new type of one-agent-at-a-time policy improvement operation. For one of our PI algorithms, we prove convergence to an agent-by-agent optimal policy, thus establishing a connection with the theory of teams. For another PI algorithm, which is executed over a more complex state space, we prove convergence to an optimal policy. Approximate forms of these algorithms are also given, based on the use of policy and value neural networks. These PI algorithms, in both their exact and their approximate form are strictly off-line methods, but they can be used to provide a base policy for use in an on-line multiagent rollout scheme.

***Index Terms*—Dynamic programming, multiagent problems, neuro-dynamic programming, policy iteration, reinforcement learning, rollout.**

Manuscript received September 23, 2020; revised October 28, 2020; accepted October 30, 2020. Recommended by Associate Editor Qinglai Wei.

For a video lecture and slides based on this paper, see the web pages of the books [2], [3] at the author's web site: http://web.mit.edu/dimitrib/www/RLbook.html.

Citation: D. Bertsekas, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 2, pp. 249−272, Feb. 2021.

D. Bertsekas is with the Arizona State University (ASU), Tempe, AZ 85281 USA, and also with Massachusetts Institute of Technology (MIT), Cambridge, MA 02139 USA (e-mail: dimitrib@mit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JAS.2021.1003814

I. Introduction

In this paper we discuss the solution of large and challenging multistage decision and control problems, which involve controls with multiple components, each associated with a different decision maker or agent. We focus on problems that can be solved in principle by dynamic programming (DP), but are addressed in practice using methods of reinforcement learning (RL), also referred to by names such as approximate dynamic programming and neuro-dynamic programming. We will discuss methods that involve various forms of the classical method of policy iteration (PI), which starts from some policy and generates 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 [1]−[3] 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.* [4]. 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 through 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 [2], [3], which share the notation and point of view of the present paper.

1) Our Multiagent Structure

The purpose of this paper is to survey variants of rollout and PI for DP problems involving a control u at each stage that consists of multiple components $u_1, \ldots, u_m$, i.e.,

$$u = (u_1, \ldots, u_m), \tag{1}$$

where the components u_ℓ are selected independently from within corresponding constraint sets U_ℓ, $\ell = 1, \ldots, m$. Thus the overall constraint set is $u \in U$, where U is the Cartesian product[1]

$$U = U_1 \times \cdots \times U_m.$$

We associate each control component u_ℓ with the ℓth of m agents.

The term "multiagent" is used widely in the literature, with several different meanings. Here, we use this term as a conceptual metaphor in the context of problems with the multi-component structure (1); it is often insightful to associate control components with agent actions. A common example of a multiagent problem is multi-robot (or multi-person) service systems, often involving a network, such as delivery, maintenance and repair, search and rescue, firefighting, taxicab or utility vehicle assignment, and other related contexts. Here the decisions are implemented collectively by the robots (or persons, respectively), with the aid of information exchange or collection from sensors or from a computational "cloud." The information may or may not be common to all the robots or persons involved. Another example involves a network of facilities, where service is offered to clients that move within the network. Here the agents may correspond to the service facilities or to the clients or to both, with information sharing that may involve errors and/or delays.

Note, however, that the methodology of this paper applies generally to any problem where the control u consists of m components, $u = (u_1, \ldots, u_m)$ [cf. Eq.(1)], independently of the details of the associated practical context. In particular, the practical situation addressed may not involve recognizable "agents" in the common sense of the word, such as multiple robots, automobiles, service facilities, or clients. For example, it may simply involve control with several components, such as a single robot with multiple moving arms, a chemical plant with multiple interacting but independently controlled processes, or a power system with multiple production centers.

As is generally true in DP problems, in addition to control, there is an underlying state, denoted by x, which summarizes all the information that is useful at a given time for the purposes of future optimization. It is assumed that x is perfectly known by all the agents at each stage.[2] In a PI infinite horizon context, given the current policy μ [a function that maps the current state x to an m-component control $\mu(x) = \big(\mu_1(x), \ldots, \mu_m(x)\big)$, 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 the general form

$$\min_{u \in U} H(x, u, J_\mu), \tag{2}$$

where J_μ is the cost function of policy μ (a function of x), and H is a problem-dependent Bellman operator. This minimization may be done off-line (before control has started) or on-line (after control has started), and 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

$$J_{\tilde{\mu}}(x) \leq J_\mu(x), \qquad \text{for all states } x, \tag{3}$$

i.e., the rollout policy yields reduced cost compared with the base policy, for all states x. Assuming that each set U_ℓ is finite (as we do in this paper), there are two difficulties with the lookahead minimization (2), which manifest themselves both in off-line and in on-line settings:

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

(b) To implement the minimization (2), the agents need to coordinate their choices of controls, thus precluding their parallel computation.

In this paper, we develop rollout and PI algorithms, which, as a first objective, aim to alleviate the preceding two difficulties. A key idea is to introduce a form of sequential agent-by-agent one-step lookahead minimization, which we call *multiagent rollout*. It mitigates dramatically the computational bottleneck 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 (3): 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 allow parallel 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 faster distributed and asynchronous on-line control selection by the agents, with a potential sacrifice of performance, which

[1]We will also allow later dependence of the sets U_ℓ on a system state. More complex constraint coupling of the control components can be allowed at the expense of additional algorithmic complications; see [3], [5], [6].

[2]Partial observation Markov decision problems (POMDP) can be converted to problems involving perfect state information by using a belief state; see e.g., the textbook [1]. 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.

depends on the quality of the policy space approximation. Note that *autonomous multiagent rollout makes sense only in the context of distributed computation*. If all computations are performed serially on a single processor, there is no reason to resort to signaling policies and autonomous rollout schemes.

Let us 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 [7], [8], and the book [9]. Also asynchronous PI algorithms have been discussed in a series of papers of the author and Yu [10]−[12], as well as the books [3], [13], [14] . Moreover, distributed DP based on partitioning in conjunction with neural network training on each subset of the partition has been considered in the context of a challenging partial state information problem by Bhattacharya *et al.* [15]. The algorithmic ideas of these works do not directly apply to the multiagent context of this paper. Still one may envision applications where parallelization with state space partitioning is combined with the multiagent parallelization ideas of the present paper. In particular, one may consider PI schemes that involve multiple agents/processors, each using a state space partitioning scheme with a cost function and an agent policy defined over each subset of the partition. The agents may then communicate asynchronously their policies and cost functions to other agents, as described in the paper [10] and book [3] (Section 5.6), and iterate according to the agent-by-agent policy evaluation and policy improvement schemes discussed in this paper. This, however, is beyond our scope and is left as an interesting subject for further research.

2) Classical and Nonclassical Information Patterns

It is worth emphasizing that our multiagent problem formulation requires that all the agents fully share information, including the values of the controls that they have applied in the past, and have perfect memory of all past information. This gives rise to a problem with a so called "classical information pattern," a terminology introduced in the papers by Witsenhausen [16], [17]. A fact of fundamental importance is that problems possessing this structure can be addressed with the DP formalism and approximation in value space methods of RL. Problems where this structure is absent, referred to as problems with "nonclassical information pattern," cannot be addressed formally by DP (except through impractical reformulations), and are generally far more complicated, as illustrated for linear systems and quadratic cost by the famous counterexample of [16].

Once a classical information pattern is adopted, we may assume that all agents have access to a system state[3] and make use of a simple conceptual model: there is 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 as functions of the system state; see Fig. 1. Alternatively, the agent computations can be done at the cloud, and the results may be passed on to the agents in place of the exact state. This scheme is also well suited as a starting point for approximations where the state information made available to the agents is replaced by precomputed "signaling" policies that guess/estimate missing information. The estimates are then treated by the agents as if they were exact. Of course such an approach is not universally effective, but may work well for favorable problem structures.[4] Its analysis is beyond the scope of the present paper, and is left as a subject for further research.

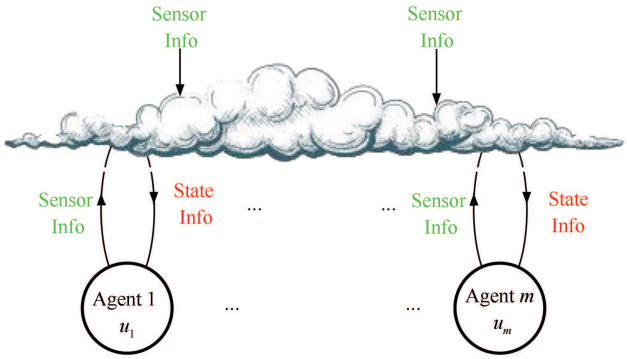


Fig. 1. Illustration of a conceptual structure for our multiagent system. The "cloud" collects information from the environment and from the agents on-line, and broadcasts the state (and possibly other information) to the agents at each stage, who then perform local computations to apply their controls as functions of the state information obtained from the cloud. Of course some of these local computations may be done at the cloud, and the results may be passed on to the agents in place of the exact state. In the case of a problem with partial state observation, the cloud computes the current belief state (rather than the state).

We note that our multiagent rollout schemes relate to 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 [18], Radner [19], Witsenhausen [17], [20], Marschak and Radner [21], Sandell *et al.* [22], Yoshikawa [23], Ho [24]. For more recent works, see Bauso and Pesenti [25], [26], Nayyar, Mahajan, and Teneketzis [27], Nayyar and Teneketzis [28], Li *et al.* [29], Qu and Li [30], Gupta [31], the books by Bullo, Cortes, and Martinez [32], Mesbahi and Egerstedt [33], Mahmoud [34], and Zoppoli, Sanguineti, Gnecco, and Parisini [35], and the references quoted there.

The connection of our work with team theory manifests itself in our infinite horizon DP methodology, which includes value iteration and PI methods that converge to a person-by-person optimal policy. Note that in contrast with the present paper, a large portion of the work on team theory and de-

[3]The system state at a given time is either the common information of all the agents, or a sufficient statistic/summary of this information, which is enough for the computation of a policy that performs arbitrarily close to optimal. For example in the case of a system with partial state observations, we could use as system state a belief state; see e.g., [1].

[4]For example consider a problem where the agent locations within some two-dimensional space become available to the other agents with some delay. It may then make sense for the agents to apply some algorithm to estimate the location of the other agents based on the available information, and use the estimates in a multiagent rollout scheme as if they were exact.

centralized control allows a nonclassical information pattern, whereby the agents do not share the same state information and/or forget information previously received, although they do share the same cost function. In the case of a multiagent system with partially observed state, this type of model is also known as a decentralized POMDP (or Dec-POMDP), a subject that has attracted a lot of attention in the last 20 years; see e.g., the monograph by Oliehoek and Amato [36], and the references quoted there. We may also note the extensive literature on game-theoretic types of problems, including Nash games, where the agents have different cost functions; see e.g., the surveys by Hernandez-Leal *et al.* [37], and Zhang, Yang, and Basar [38]. Such problems are completely outside our scope and require a substantial departure from the methods of this paper. Zero-sum sequential games may be more amenable to treatment with the methodology of this paper, because they can be addressed within a DP framework (see e.g., Shapley [39], Littman [40]), but this remains a subject for further research.

In addition to the aforementioned works on team theory and decentralized control, there has been considerable related work on multiagent sequential decision making from a machine learning perspective, often with the use of variants of policy gradient, Q-learning, and random search methods. Works of this type also have a long history, and they have been surveyed over time by Sycara [41], Stone and Veloso [42], Panait and Luke [43], Busoniu, Babuska, and De Schutter [44], [45], Matignon, Laurent, and Le Fort-Piat [46], Hernandez-Leal, Kartal, and Taylor [47], OroojlooyJadid and Hajinezhad [48], Zhang, Yang, and Basar [38], and Nguyen, Nguyen, and Nahavandi [49], who list many other references. For some representative recent research papers, see Tesauro [50], Oliehoek, Kooij, and Vlassis [51], Pennesi and Paschalidis [52], Paschalidis and Lin [53], Kar, Moura, and Poor [54], Foerster *et al.* [55], Omidshafiei *et al.* [56], Gupta, Egorov, and Kochenderfer [57], Lowe *et al.* [58], Zhou *et al.* [59], Zhang *et al.* [60], Zhang and Zavlanos [61], and de Witt *et al.* [62].

These works collectively describe several formidable difficulties in the implementation of reliable multiagent versions of policy gradient and Q-learning methods, although they have not emphasized the critical distinction between classical and nonclassical information patterns. It is also worth noting that policy gradient methods, Q-learning, and random search are primarily off-line algorithms, as they are typically too slow and noise-afflicted to be applied with on-line data collection. As a result, they produce policies that are tied to the model used for their training. Thus, contrary to rollout, they are not robust with respect to changes in the problem data, and they are not well suited for on-line replanning. On the other hand, it is possible to train a policy with a policy gradient or random search method by using a nominal model, and use it as a base policy for on-line rollout in a scheme that employs on-line replanning.

3) Related Works

The multiagent systems field has a long history, and the range of related works noted above is very broad. However, while the bottleneck due to exponential growth of computation with the number of agents has been recognized [47], [48], it has not been effectively addressed. It appears that the central idea of the present paper, agent-by-agent sequential optimization while maintaining the cost improvement property, has been considered only recently. In particular, the approach to maintaining cost improvement through agent-by-agent rollout was first introduced in the author's papers [5], [6], [63], and research monograph [3].

A major computational study where several of the algorithmic ideas of this paper have been tested and validated is the paper by Bhattacharya *et al.* [64]. This paper considers a large-scale multi-robot 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 and other precomputed signaling policies.

The author's paper [6] and monograph [3] 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. The character of this deterministic constrained rollout methodology differs markedly from the one of the methods of this paper. Still the rollout ideas of the paper [6] are supplementary to the ones of the present paper, and point the way to potential extensions of constrained rollout to stochastic problems. We note also that the monograph [3] describes multiagent rollout methods for minimax/robust control, and other problems with an abstract DP structure.

4) Organization of the Paper

The present paper is organized as follows. We first introduce finite horizon stochastic optimal control problems in Section II, 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 III, we consider the implementation of autonomous multiagent rollout, including schemes that allow the distributed and asynchronous computation of the agents' control components.

We then turn to infinite horizon discounted problems. In particular, in Section IV, we extend the multiagent rollout algorithm, we discuss the cost improvement property, and we provide error bounds for versions of the algorithm involving rollout truncation and simulation. We also discuss two types of multiagent PI algorithms, in Sections IV-A and IV-E, respectively. The first of these, in its exact form, converges to an agent-by-agent optimal policy, thus establishing a connection with the theory of teams. The second PI algorithm, in its exact form, converges to an optimal policy, but must be executed over a more complex state space. Approximate forms of these algorithms, as well as forms of Q-learning, are also discussed, based on the use of policy and value neural networks. These algorithms, in both their exact and their approximate form are strictly off-line methods, but they can be used to provide a base policy for use in an on-line multiagent rollout scheme. Finally, in Section V we discuss autonomous multiagent rollout schemes for infinite horizon discounted

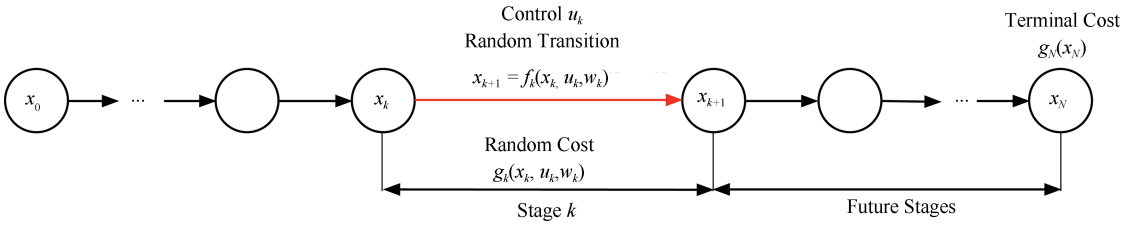


Fig. 2. Illustration of the N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated according to a system equation

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

where w_k is the random disturbance, and a random stage cost $g_k(x_k, u_k, w_k)$ is incurred.

problems, which allow for distributed on-line implementation.

II. Multiagent Problem Formulation - Finite Horizon Problems

We consider a standard form of an N-stage DP problem (see [1], [2]), which involves the discrete-time dynamic system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1, \quad (4)$$

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, with given probability distribution $P_k(\cdot \mid x_k, u_k)$ that may depend explicitly on x_k and u_k, but not on values of prior disturbances $w_{k-1}, \ldots, w_0$. 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.

We consider policies of the form

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

where μ_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 π starting from x_0 is

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

where the expected value operation $E\{\cdot\}$ is with respect to the joint distribution of 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 Π is the set of all policies. An optimal policy π^* is one that attains the minimal cost for every x_0; i.e.,

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0), \qquad \text{for all } x_0.$$

Since the optimal cost function J^* and optimal policy π^* 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.

A. 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 that uses a neural network for policy evaluation or a policy gradient method of the actor/critic type (see e.g., the reinforcement learning book [2]).

The standard rollout algorithm has a long history (see the textbooks [1]−[3], [65], which collectively list a large number of research contributions). The name "rollout" was coined by Tesauro, who among others, has used a "truncated" version of the rollout algorithm for a highly successful application in computer backgammon [66]. The algorithm is widely viewed among the simplest and most reliable RL methods. It provides on-line control of the system as follows:

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

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big) \Big\}. \quad (5)$$

Throughout this paper we will focus on rollout algorithms that involve one-step lookahead minimization as in Eq.(5). 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 (5), which uses $J_{k+1,\pi}$ in place of the optimal cost function J^*, defines a policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, referred to as the *rollout policy*, where for all x_k and k, $\tilde{\mu}_k(x_k)$ is equal to the control $\tilde{u}_k$ obtained from Eq.(5). The rollout policy possesses a fundamental *cost improvement property*: it improves over the

base policy in the sense that

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \forall\ x_k, k, \tag{6}$$

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., [1], Section 6.4, or [2], Section 2.4.2). Extensive experimentation has shown that in practice the rollout policy typically performs significantly better than the base policy, even when the latter policy is quite poor.

In addition to the cost improvement property, the rollout algorithm (5) has a second nice property: it is an on-line algorithm, and hence inherently possesses a *robustness property*: it can adapt to variations of the problem data through on-line replanning. Thus if there are changes in the problem data (such as for example the probability distribution of w_k, or the stage cost function g_k), the performance of the base policy can be seriously affected, but the performance of the rollout policy is much less affected because the computation in Eq.(5) will take into account the changed problem data.

Despite the advantageous properties just noted, the rollout algorithm suffers from a serious disadvantage when the constraint set $U_k(x_k)$ has a large number of elements, namely that the minimization in Eq.(5) involves a large number of alternatives. In particular, let us consider the expected value in Eq.(5), which is the Q-factor of the pair (x_k, u_k) corresponding to the base policy:

$$Q_{k,\pi}(x_k, u_k) = E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big) \Big\}.$$

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 and comparison of Q-factors, grow rapidly with the number of agents, and can become very large. We next introduce a modified rollout algorithm for the multiagent case, which requires much less on-line computation but still maintains the cost improvement property (6).

B. 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 $u_k^1, \ldots, u_k^m$,

$$u_k = (u_k^1, \ldots, u_k^m),$$

with the component u_k^ℓ, $\ell = 1, \ldots, m$, chosen by agent ℓ at stage k, from within a given set $U_k^\ell(x_k)$. Thus the control constraint set is the Cartesian product[5]

$$U_k(x_k) = U_k^1(x_k) \times \cdots \times U_k^m(x_k). \tag{7}$$

Then the minimization (5) involves as many as q^m Q-factors, where q is the maximum number of elements of the sets $U_k^\ell(x_k)$ [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 (7)]. Thus the computation required by the standard rollout algorithm is of order $O(q^m)$ per stage.

We propose an alternative rollout algorithm that achieves the cost improvement property (6) at much smaller computational cost, namely of order $O(qm)$ per stage. A key idea is that the computational requirements of the rollout one-step minimization (5) 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 problem reformulation, first proposed in the neuro-dynamic programming book [65], 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 will next apply this idea within our multiagent rollout context. We note, however, that the idea can be useful in other multiagent algorithmic contexts, including approximate PI, as we will discuss in Section IV-E.

C. 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.(5), 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" $(x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1})$, 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. 3.

[5]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 II-C), 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 our methods.

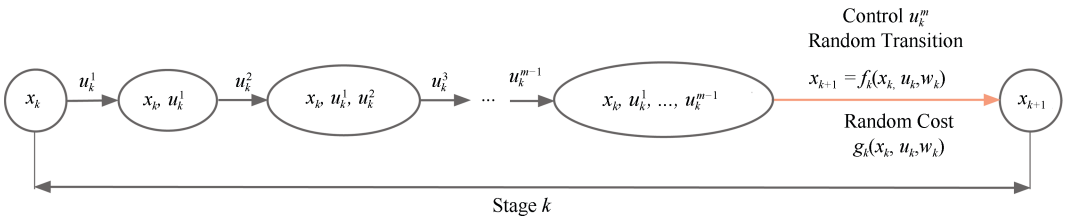


Fig. 3. Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u_k consists of m components $u_k^1, u_k^2, \ldots, u_k^m$:

$$u_k = (u_k^1, \ldots, u_k^m) \in U_k^1(x_k) \times \cdots \times U_k^m(x_k).$$

The figure depicts the kth stage transitions. Starting from state x_k, we generate the intermediate states $(x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1})$, using the respective controls $u_k^1, \ldots, u_k^{m-1}$. The final control u_k^m leads from $(x_k, u_k^1, \ldots, u_k^{m-1})$ to $x_{k+1} = f_k(x_k, u_k^1, \ldots, u_k^m, w_k)$, and a random stage cost $g_k(x_k, u_k^1, \ldots, u_k^m, w_k)$ is incurred.

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

$$\pi = \big\{(\mu_k^1, \ldots, \mu_k^m) \mid k = 0, \ldots, N-1\big\}$$

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 $J_k^1(x_k, u_k^1)$, $J_k^2(x_k, u_k^1, u_k^2), \ldots, J_k^{m-1}(x_k, u_k^1, \ldots, u_k^{m-1})$, 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 (5) 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 function approximation, i.e., with the introduction of cost-to-go approximations

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

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); see Section IV-E.

D. Multiagent Rollout and Cost Improvement

Consider now the standard rollout algorithm applied to the reformulated problem shown in Fig. 3, with a given base policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, which is also a policy of the original problem [so that $\mu_k = (\mu_k^1, \ldots, \mu_k^m)$, with each μ_k^ℓ, $\ell = 1, \ldots, m$, 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

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

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

$$\ldots \qquad \ldots \qquad \ldots$$

$$\tilde{\mu}_k^m(x_k) \in \arg\min_{u_k^m \in U_k^m(x_k)} E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^{m-1}(x_k), u_k^m, w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^{m-1}(x_k), u_k^m, w_k\big)\Big)\Big\}. \quad (8)$$

Thus, when applied on-line, at x_k, *the algorithm generates the control* $\tilde{\mu}_k(x_k) = \big(\tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^m(x_k)\big)$ *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.(8). Assuming a maximum of q elements in the constraint sets $U_k^\ell(x_k)$, 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^ℓ, 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^ℓ 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 ℓ 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 III and V, 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; cf. Fig. 1. 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 our scope.

Note that the rollout policy (8), obtained from the reformulated problem is different from the rollout policy obtained from the original problem [cf. Eq.(5)]. 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 1: Let π be a base policy and let $\tilde{\pi}$ be a corresponding rollout policy generated by the multiagent rollout algorithm (8). We have

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k. \tag{9}$$

Proof: We will show Eq.(9) by induction, and for simplicity, we will give the proof for the case of just two agents, i.e., $m = 2$. Clearly Eq.(9) holds for $k = N$, since $J_{N,\tilde{\pi}} = J_{N,\pi} = g_N$. Assuming that it holds for index $k+1$, i.e., $J_{k+1,\tilde{\pi}} \leq J_{k+1,\pi}$, we have for all x_k,

$$\begin{aligned}
J_{k,\tilde{\pi}}(x_k) &= E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) \\
&\quad + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) \\
&\quad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^2 \in U_k^2(x_k)} E\Big\{g_k(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k) \\
&\quad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big) \\
&\quad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^1 \in U_k^1(x_k)} E\Big\{g_k\big(x_k, u_k^1, \mu_k^2(x_k), w_k\big) \\
&\quad + J_{k+1,\pi}\Big(f_k\big(x_k, u_k^1, \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big) \\
&\quad + J_{k+1,\pi}\Big(f_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= J_{k,\pi}(x_k),
\end{aligned}$$

where in the preceding relation:

(a) The first equality is the DP/Bellman equation for the rollout policy $\tilde{\pi}$.

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

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

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

(e) The last equality is the DP/Bellman equation for the base policy π.

The induction proof of the cost improvement property (9) is thus complete for the case $m = 2$. The proof for an arbitrary number of agents m is entirely similar. ■

Note that there 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.[6] 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 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 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 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 ℓ 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^ℓ is the choice of the ℓth 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. 4, the size of the control space is reduced to ≤ 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

[6]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 the references on team theory cited in Section I.

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 that moving straight towards the fly along a shortest path.

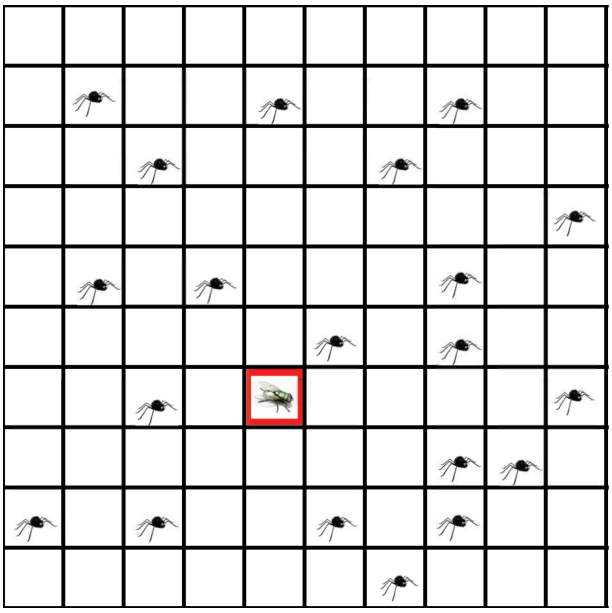


Fig. 4. Illustration of the 2-dimensional spiders-and-fly problem. The state is the set of locations of the spiders and the fly. At each time period, each spider moves to a neighboring location or stays where it is. The spiders make moves with perfect knowledge of the locations of each other and of the fly. The fly moves randomly, regardless of the position of the spiders.

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 (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. 5. 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 are at different locations as in Fig. 5. 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. 5. 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 "farsighted" 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 confirmed by computational experiments with larger two-dimensional problems of the type described in Example 1. It has also been supported by the computational study [64], which deals with a multi-robot repair problem.

E. 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

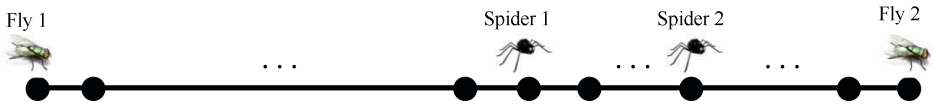


Fig. 5. Illustration of the two-spiders and two-flies problem. The spiders move along integer points of a line. The two flies stay still at some integer locations. The optimal policy is to move the two spiders towards different flies, the ones that are initially closest to them. The base policy directs each spider to move one unit towards the nearest fly position.

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.

correspond to each of the agents $\ell = 1, \ldots, m$ being first in the multiagent rollout order. If ℓ_1 is the agent that produces the minimal Q-factor, we fix ℓ_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 ℓ_2 be the agent that produces the minimal Q-factor, fix ℓ_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

$$m + (m-1) + \cdots + 1 = \frac{m(m+1)}{2}$$

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.

F. 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 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. We will discuss multiagent truncated rollout later in Section IV-F, in the context of infinite horizon problems, where we will give a related error bound.

III. Asynchronous and Autonomous Rollout

In this section we consider multiagent rollout algorithms that are distributed and asynchronous in the sense that the agents may compute their rollout controls in parallel rather than in sequence, aiming at computational speedup. An example of such an algorithm is obtained when at a given stage, agent ℓ 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 other 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 for some initial states.

Example 3 (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 (9) 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 on the desirability of various controls of other agents, then a more flexible inter-agent communication pattern may be sufficient for cost improvement.[7] An important 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.

1) 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.[8] 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 compute their own control components in parallel and asynchronously. We call this algorithm *autonomous multiagent rollout*. While this type of algorithm involves a form of redundant computation, it allows for additional speedup through parallelization.

Similar to Section II, the algorithm at the kth stage uses a base policy $\mu_k = \{\mu_k^1, \ldots, \mu_k^{m-1}\}$, but it also uses *a second policy* $\widehat{\mu}_k = \{\widehat{\mu}_k^1, \ldots, \widehat{\mu}_k^{m-1}\}$, called the *signaling policy*, which is computed off-line, is known to all the agents for on-line use, and is designed to play an agent coordination role. Intuitively, $\widehat{\mu}_k^\ell(x_k)$ *provides an intelligent "guess" about what agent* ℓ *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 uses the base and signaling policies to generate a rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ as follows. At stage k and state x_k, $\tilde{\mu}_k(x_k) = \big(\tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^m(x_k)\big)$, is obtained according to

$$\begin{aligned}
\tilde{\mu}_k^1(x_k) \in \arg \min_{u_k^1 \in U_k^1(x_k)} E\Big\{ & g_k\big(x_k, u_k^1, \mu_k^2(x_k), \\
& \ldots, \mu_k^m(x_k), w_k\big) \\
& + J_{k+1,\pi}\Big(f_k\big(x_k, u_k^1, \mu_k^2(x_k), \\
& \ldots, \mu_k^m(x_k), w_k\big)\Big)\Big\}, \\
\tilde{\mu}_k^2(x_k) \in \arg \min_{u_k^2 \in U_k^2(x_k)} E\Big\{ & g_k\big(x_k, \widehat{\mu}_k^1(x_k), u_k^2, \\
& \ldots, \mu_k^m(x_k), w_k\big) \\
& + J_{k+1,\pi}\Big(f_k\big(x_k, \widehat{\mu}_k^1(x_k), u_k^2, \\
& \ldots, \mu_k^m(x_k), w_k\big)\Big)\Big\}, \\
& \ldots \qquad \ldots \qquad \ldots \\
\tilde{\mu}_k^m(x_k) \in \arg \min_{u_k^m \in U_k^m(x_k)} E\Big\{ & g_k\big(x_k, \widehat{\mu}_k^1(x_k), \\
& \ldots, \widehat{\mu}_k^{m-1}(x_k), u_k^m, w_k\big) \\
& + J_{k+1,\pi}\Big(f_k\big(x_k, \widehat{\mu}_k^1(x_k), \\
& \ldots, \widehat{\mu}_k^{m-1}(x_k), u_k^m, w_k\big)\Big)\Big\}. \qquad (10)
\end{aligned}$$

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

The simplest choice is to *use as signaling policy* $\widehat{\mu}$ *the base policy* μ. However, this choice does not guarantee policy improvement as evidenced by Example 3 (see also Example 7 in Section V). In fact performance deterioration with this choice is not uncommon, and can be observed in more complicated examples, including the following.

Example 4 (Spiders and Flies - Use of the Base Policy for Signaling)

Consider the problem of Example 2, which involves two spiders and two flies on a line, and the base policy μ that moves a spider towards the closest surviving fly (and in case where a spider starts at the midpoint between the two flies, moves the spider to the right). Assume that we use as signaling policy $\widehat{\mu}$ the base policy μ. It can then be verified that if the spiders start from different positions, the rollout policy will be optimal (will move the spiders in opposite directions). If, however, the spiders start from the same position, a completely symmetric situation is created, whereby the rollout controls move both flies in the direction of the fly *furthest away* from the spiders' position (or to the left in the case where the spiders start at the midpoint between the two flies). Thus, the flies end up oscillating around the middle of the interval between the flies and never catch the flies.

The preceding example is representative of a broad class of counterexamples that involve multiple identical agents. If the agents start at the same initial state, with a base policy that has

[7] In particular, one may divide the agents in "coupled" groups, and require coordination of control selection only within each group, while the computation of different groups may proceed in parallel. Note that the "coupled" group formations may change over time, depending on the current state. 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.

[8] The general idea of coordination by sharing information about the agents' policies arises also in other multiagent algorithmic contexts, including some that involve forms of policy gradient methods and Q-learning; see the surveys of the relevant research cited earlier. The survey by Matignon, Laurent, and Le Fort-Piat [46] focuses on coordination problems from an RL point of view.

identical components, and use the base policy for signaling, the agents will select identical controls under the corresponding multiagent rollout policy, ending up with a potentially serious cost deterioration. This example also highlights the role of the sequential choice of the control components $u_k^1, \ldots, u_k^m$, based on the reformulated problem of Fig. 3: it tends to break symmetries and "group think" that guides the agents towards choosing the same controls under identical conditions.

An alternative idea is to choose the signaling policy $\widehat{\mu}_k$ to approximate the multiagent rollout policy of Section II-D [cf. Eq.(8)], which is known to embody coordination between the agents. In particular, we may obtain the policy $\widehat{\mu}_k = (\widehat{\mu}_k^1, \ldots, \widehat{\mu}_k^m)$ by off-line training a neural network (or m networks, one per agent) with training samples generated through the rollout policy of Eq.(8); i.e., *use as signaling policy $\widehat{\mu}_k$ a neural network representation of the rollout policy $\tilde{\mu}_k$ of Eq.(8)*. Note that if the neural network representation were perfect, the policy defined by Eq.(10) would be the same as the rollout policy of Eq.(8). Thus we intuitively expect that if the neural network provides a good approximation of the rollout policy (8), the policy defined by Eq.(10) would have better performance than the base policy. This expectation was confirmed in the context of a large-scale multi-robot repair application in the paper [64]. The advantage of autonomous multiagent rollout with neural network approximations is that it allows approximate policy improvement (to the extent that the functions $\widehat{\mu}_k^i$ are good approximations to $\tilde{\mu}_k^i$), 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 V, in the context of infinite horizon problems.

IV. 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 may also consider multiagent versions of PI algorithms, which generate a sequence of policies $\{\mu^k\}$. They can be viewed as repeated applications of multiagent rollout, with each policy μ^k in the sequence being the multiagent rollout policy that is obtained when the preceding policy μ^{k-1} is viewed as the base policy. For challenging problems, PI must be implemented off-line and with approximations, possibly involving neural networks. However, the final policy obtained off-line by PI (or its neural network representation) can be used as the base policy for an on-line multiagent rollout scheme.

We will focus on discounted problems with finite number of states and controls, so that the problem has a contractive structure (i.e., 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 [13], Chapters 1 and 2, [14], Chapter 2, or [2], Chapter 4. However, a qualitatively similar methodology can be applied to undiscounted problems involving a termination state (e.g., stochastic shortest path problems, see [65], Chapter 2, [13], Chapter 3, and [14], 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 I-B, where $m > 1$. We assume n states $x = 1, \ldots, n$ and a control u that consists of m components u_ℓ, $\ell = 1, \ldots, m$,

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

(for the MDP notation adopted for this section, we switch for convenience to subscript indexing for agents and control components, and reserve superscript indexing for policy iterates). At state x and stage k, a control u is applied, and the system moves to a next state y with given transition probability $p_{xy}(y)$ and cost $g(x, u, y)$. When at stage k, the transition cost is discounted by α^k, where $\alpha \in (0, 1)$ is the discount factor. Each control component u_ℓ 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 μ that applies control $\mu(x) \in U(x)$ at state x is denoted by $J_\mu(x)$, and the optimal cost [the minimum over μ of $J_\mu(x)$] is denoted $J^*(x)$.

An equivalent version of the problem, involving a reformulated/expanded state space is depicted in Fig. 6 for the case $m = 3$. The state space of the reformulated problem consists of

$$x, (x, u_1), \ldots, (x, u_1, \ldots, u_{m-1}), \tag{11}$$

where x ranges over the original state space (i.e., $x \in \{1, \ldots, n\}$), and each u_ℓ, $\ell = 1, \ldots, m$, 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 $(x, u_1, \ldots, u_{m-1})$, 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.

This reformulation involves the type of tradeoff between control space complexity and state space complexity that was proposed in the book [65], Section 6.1.4, and was discussed in Section II-C. 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. Note that 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 [2]. Moreover, the reformulated problem may prove useful in other contexts where the size of the control space is a concern, such as for example Q-learning.

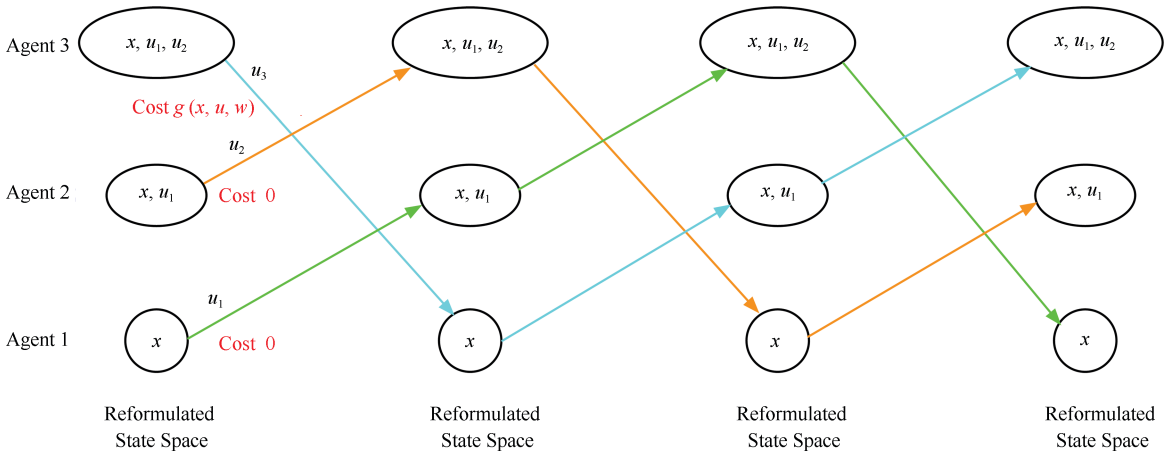


Fig. 6. Illustration of how to transform an m-agent infinite horizon problem into a stationary infinite horizon problem with fewer control choices available at each state (in this figure $m = 3$). At the typical stage only one agent selects a control. For example, at state x, the first agent chooses 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 [3], and the papers [10]−[12], [15].

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.

A. Multiagent Rollout Policy Iteration

The policies generated by the standard PI algorithm for the reformulated problem of Fig. 6 are defined over the larger space and have the form

$$\mu_1(x), \mu_2(x, u_1), \ldots, \mu_m(x, u_1, \ldots, u_{m-1}). \tag{12}$$

We may consider a standard PI algorithm that generates a sequence of policies of the preceding form (see Section IV-E), and which based on standard discounted MDP results, converges to an optimal policy for the reformulated problem, which in turn yields an optimal policy for the original problem. However, policies of the form (12) can also be represented in the simpler form

$$\mu_1(x), \mu_2(x), \ldots, \mu_m(x)$$

i.e., as policies for the original infinite horizon problem. This motivates us to consider an alternative multiagent PI algorithm that uses one-agent-at-a-time policy improvement and operates over the latter class of policies. We will see that this algorithm converges to an agent-by-agent optimal policy (which need not be an optimal policy for the original problem). By contrast, the alternative multiagent PI algorithm of Section IV-E also uses one-agent-at-a-time policy improvement, but operates over the class of policies (12), and converges to an optimal policy for the original problem (rather than just an agent-by-agent optimal policy).

Consistent with the multiagent rollout algorithm of Section IV-D, 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 μ^k, the next policy is obtained as

$$\mu^{k+1} \in \widetilde{\mathcal{M}}_{\mu^k}(J_{\mu^k}), \tag{13}$$

where for given $\mu = (\mu_1, \ldots, \mu_m)$ and J, we denote by $\widetilde{\mathcal{M}}_\mu(J)$ the set of policies

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

satisfying for all states $x = 1, \ldots, n$,

$$\begin{aligned}
\tilde{\mu}_1(x) \in \arg \min_{u_1 \in U_1(x)} & \sum_{y=1}^{n} p_{xy}\big(u_1, \mu_2(x), \ldots, \mu_m(x)\big) \\
& \cdot \Big(g\big(x, u_1, \mu_2(x), \ldots, \mu_m(x), y\big) + \alpha J(y)\Big), \\
\tilde{\mu}_2(x) \in \arg \min_{u_2 \in U_2(x)} & \sum_{y=1}^{n} p_{xy}\big(\tilde{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x)\big) \\
& \cdot \Big(g\big(x, \tilde{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x), y\big) + \alpha J(y)\Big), \\
& \cdots \qquad \cdots \qquad \cdots \\
\tilde{\mu}_m(x) \in \arg \min_{u_m \in U_m(x)} & \sum_{y=1}^{n} p_{xy}\big(\tilde{\mu}_1(x), \tilde{\mu}_2(x), \\
& \qquad \ldots, \tilde{\mu}_{m-1}(x), u_m\big) \\
& \cdot \Big(g\big(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, y\big) \\
& + \alpha J(y)\Big).
\end{aligned} \tag{14}$$

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

Each of the m minimizations (14) 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 II-E.

Similar to the finite horizon case of Section II, the salient feature of the one-agent-at-a-time policy improvement operation (14) 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_ℓ, the preceding components $u_1, \ldots, u_{\ell-1}$ have been computed earlier by the minimization that yielded the policy components $\tilde{\mu}_1, \ldots, \tilde{\mu}_{\ell-1}$, 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 (14) 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.*

B. Multipass Multiagent Policy Improvement

In trying to understand why multiagent rollout of the form (13) succeeds in improving the performance of the base policy, it is useful to think of the multiagent policy improvement operation as an approximation of the standard policy improvement operation. We basically approximate the joint minimization over all the control components $u_1, \ldots, u_m$ with a single "coordinate descent-type" iteration, i.e., a round of single control component minimizations, each taking into account the results of the earlier minimizations.

This coordinate descent view suggests that one may obtain further policy improvements with *multiple rounds of coordinate descent minimizations*. By this we mean that for a given and fixed state x, after computing the multiagent rollout controls $\tilde{\mu}_1(x), \ldots, \tilde{\mu}_m(x)$ using Eq.(14), we use them to replace the base controls $\mu_1(x), \ldots, \mu_m(x)$, and repeat once more the multiagent policy improvement operation [while keeping the function J in Eq.(14) equal to the base policy cost function J_μ].

Mathematically, this amounts to using the control components at x of a policy within the set

$$\widetilde{\mathcal{M}}^2_\mu(J_\mu), \tag{15}$$

defined as the set of all policies in the set $\widetilde{\mathcal{M}}_{\mu'}(J_\mu)$, where μ' is any policy in the set $\widetilde{\mathcal{M}}_\mu(J_\mu)$ defined by Eq. (14) [so $\mu'(x), \ldots, \mu'_m(x)$ are the rollout control components, which are obtained with a single round of coordinate descent minimizations (14)]. The set (15) corresponds to two rounds of coordinate descent minimizations rather than one [note that for the calculations of values of J_μ is Eq.(15), we use the known base policy μ, so the values of $\mu'(x), \ldots, \mu'_m(x)$ are needed only at the given state x].

Similarly, we may consider $k > 2$ rounds of coordinate descent iterations. This amounts to using the control components at x of a policy within the set

$$\widetilde{\mathcal{M}}^k_\mu(J_\mu),$$

defined for all k as the set of all policies in the set $\widetilde{\mathcal{M}}_{\mu'}(J_\mu)$, where μ' is any policy in the set $\widetilde{\mathcal{M}}^{k-1}_\mu(J_\mu)$ [here we define $\widetilde{\mathcal{M}}^1_\mu(J_\mu)$ to be the set $\widetilde{\mathcal{M}}_\mu(J_\mu)$ given by Eq.(14)]. After a finite number of rounds of coordinate descent iterations the values of

$$\begin{aligned}
&\min_{u_1 \in U_1(x)} \sum_{y=1}^n p_{xy}\big(u_1, \tilde{\mu}_2(x), \ldots, \tilde{\mu}_m(x)\big) \\
&\qquad \cdot \Big(g\big(x, u_1, \tilde{\mu}_2(x), \ldots, \tilde{\mu}_m(x), y\big) + \alpha J_\mu(y)\Big), \\
&\qquad \ldots \qquad\qquad \ldots \qquad\qquad \ldots \\
&\min_{u_m \in U_m(x)} \sum_{y=1}^n p_{xy}\big(\tilde{\mu}_1(x), \tilde{\mu}_2(x), \ldots, \tilde{\mu}_{m-1}(x), u_m\big) \\
&\qquad \cdot \Big(g\big(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, y\big) \\
&\qquad + \alpha J_\mu(y)\Big)
\end{aligned}$$

will converge (since the control space is finite). However, the limit of these values need not be the result of the joint control component minimization[9]

$$\begin{aligned}
&\min_{(u_1, \ldots, u_m) \in U_1(x) \cdots U_m(x)} \sum_{y=1}^n p_{xy}(u_1, \ldots, u_m) \\
&\qquad \cdot \Big(g\big(x, u_1, \ldots, u_m, y\big) + \alpha J_\mu(y)\Big).
\end{aligned}$$

It will be instead a value with an agent-by-agent optimality property, to be defined in the next section. This is consistent will the convergence results that we will subsequently obtain (cf. Prop. 2). Still, however, the policy $\tilde{\mu}$ obtained through the preceding multipass multiagent rollout policy has the fundamental policy improvement property $J_{\tilde{\mu}}(x) \leq J_\mu(x)$ for all x. This can be seen by a slight extension of the proof of the subsequent Prop. 2.

[9]Generally, the convergence of the coordinate descent method to the minimum of a multivariable optimization cannot be guaranteed except under special conditions, which are not necessarily satisfied within our context.

C. 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 $\mu \in \widetilde{\mathcal{M}}_\mu(J_\mu)$, or equivalently [cf. Eq.(14)], if for all states $x = 1, \ldots, n$, and agents $\ell = 1, \ldots, m$, we have

$$\begin{aligned}\sum_{y=1}^{n} & p_{xy}\big(\mu_1(x), \ldots, \mu_m(x)\big) \\ & \cdot \Big(g\big(x, \mu_1(x), \ldots, \mu_m(x), y\big) + \alpha J_\mu(y)\Big) \\ = & \min_{u_\ell \in U_\ell(x)} \sum_{y=1}^{n} p_{xy}\big(\mu_1(x), \ldots, \mu_{\ell-1}(x), u_\ell, \\ & \qquad \mu_{\ell+1}(x), \ldots, \mu_m(x)\big) \\ & \cdot \Big(g\big(x, \mu_1(x), \ldots, \mu_{\ell-1}(x), u_\ell, \mu_{\ell+1}(x), \\ & \qquad \ldots, \mu_m(x), y\big) + \alpha J_\mu(y)\Big).\end{aligned}$$

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 ith policy components are fixed at μ_i, while the ℓth 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 μ is agent-by-agent optimal if each component μ_ℓ is optimal for the ℓth single agent problem; in other words by using μ_ℓ, each agent ℓ acts optimally, assuming all other agents $i \neq \ell$ do not deviate from the policy components μ_i. Note that agent-by-agent optimality is related to the notion of a Nash equilibrium where we view the agents as the players of a multi-person game with the same objective function for all the players.

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

Example 5 (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 round of coordinate descent operations (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 [67], Section 3.7). Generally, agent-by-agent optimality may be viewed as an acceptable form of optimality for many types of problems, but there are exceptions.

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

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

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

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

Proposition 2: Let $\{\mu^k\}$ be a sequence generated by the one-agent-at-a-time PI algorithm (13) assuming that ties in the policy improvement operation of Eq.(14) are broken as follows: If for any $\ell = 1, \ldots, m$ and x, the control component $\mu_\ell(x)$ attains the minimum in Eq.(14), 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

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

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

Proof: We recall that for given μ and J, we denote by $\widetilde{\mathcal{M}}_\mu(J)$ the set of policies $\tilde{\mu}$ satisfying Eq.(14). The critical

step of the proof is the following monotone decrease inequality:

$$T_{\tilde{\mu}}J \le T_\mu J \le J, \qquad \text{for all } J \text{ with } T_\mu J \le J \text{ and } \tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J), \tag{16}$$

which yields as a special case $T_{\tilde{\mu}}J_\mu \le J_\mu$, since $T_\mu J_\mu = J_\mu$. This parallels a key inequality for standard PI, namely that $T_{\tilde{\mu}}J_\mu \le J_\mu$, for all $\tilde{\mu}$ such that $T_{\tilde{\mu}}J_\mu = TJ_\mu$, which lies at the heart of its convergence proof. Once Eq.(16) is shown, the monotonicity of the operator $T_{\tilde{\mu}}$ implies the cost improvement property $J_{\tilde{\mu}} \le J_\mu$, 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 (16) for the case $m = 2$. The proof for an arbitrary number of components $m > 2$ is entirely similar. Indeed, if $T_\mu J \le J$ and $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J)$, we have for all states x,

$$\begin{aligned}
(T_{\tilde{\mu}}J)(x) =& \sum_{y=1}^n p_{xy}\big(\tilde{\mu}_1(x), \tilde{\mu}_2(x)\big) \\
&\cdot \Big(g\big(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), y\big) + \alpha J(y)\Big) \\
=& \min_{u_2 \in U_2(x)} \sum_{y=1}^n p_{xy}\big(\tilde{\mu}_1(x), u_2\big) \\
&\cdot \Big(g\big(x, \tilde{\mu}_1(x), u_2, y\big) + \alpha J(y)\Big) \\
\le& \sum_{y=1}^n p_{xy}\big(\tilde{\mu}_1(x), \mu_2(x)\big) \\
&\cdot \Big(g\big(x, \tilde{\mu}_1(x), \mu_2(x), y\big) + \alpha J(y)\Big) \\
=& \min_{u_1 \in U_1(x)} \sum_{y=1}^n p_{xy}\big(u_1, \mu_2(x)\big) \\
&\cdot \Big(g\big(x, u_1, \mu_2(x), y\big) + \alpha J(y)\Big) \\
\le& \sum_{y=1}^n p_{xy}\big(\mu_1(x), \mu_2(x)\big) \\
&\cdot \Big(g\big(x, \mu_1(x), \mu_2(x), y\big) + \alpha J(y)\Big) \\
=& (T_\mu J)(x) \\
\le& J(x),
\end{aligned}$$

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 $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J)$.

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

(4) The last inequality is the assumption $T_\mu J \le J$.

By letting $J = J_{\mu^k}$ in the monotone decrease inequality (16), we have $T_{\mu^{k+1}}J_{\mu^k} \le J_{\mu^k}$. In view of the monotonicity of $T_{\mu^{k+1}}$, we also have $T^{\ell+1}_{\mu^{k+1}}J_{\mu^k} \le T^{\ell}_{\mu^{k+1}}J_{\mu^k}$ for all $\ell \ge 1$, so that

$$J_{\mu^{k+1}} = \lim_{\ell \to \infty} T^{\ell}_{\mu^{k+1}}J_{\mu^k} \le T_{\mu^{k+1}}J_{\mu^k} \le J_{\mu^k}. \tag{17}$$

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.(17). This implies, using also the preceding proof, that

$$\begin{aligned}
&\sum_{y=1}^n p_{xy}\big(\mu_1^{k+1}(x), \mu_2^{k+1}(x)\big) \\
&\qquad \cdot \Big(g\big(x, \mu_1^{k+1}(x), \mu_2^{k+1}(x), y\big) + \alpha J_{\mu^k}(y)\Big) \\
&= \min_{u_2 \in U_2(x)} \sum_{y=1}^n p_{xy}\big(\mu_1^{k+1}(x), u_2\big) \\
&\qquad \cdot \Big(g\big(x, \mu_1^{k+1}(x), u_2, y\big) + \alpha J_{\mu^k}(y)\Big) \\
&= \sum_{y=1}^n p_{xy}\big(\mu_1^{k+1}(x), \mu_2^{k}(x)\big) \\
&\qquad \cdot \Big(g\big(x, \mu_1^{k+1}(x), \mu_2^{k}(x), y\big) + \alpha J_{\mu^k}(y)\Big),
\end{aligned} \tag{18}$$

and

$$\begin{aligned}
&\sum_{y=1}^n p_{xy}\big(\mu_1^{k+1}(x), \mu_2^{k}(x)\big) \\
&\qquad \cdot \Big(g\big(x, \mu_1^{k+1}(x), \mu_2^{k}(x), y\big) + \alpha J_{\mu^k}(y)\Big) \\
&= \min_{u_1 \in U_1(x)} \sum_{y=1}^n p_{xy}\big(u_1, \mu_2^{k}(x)\big) \\
&\qquad \cdot \Big(g\big(x, u_1, \mu_2^{k}(x), y\big) + \alpha J_{\mu^k}(y)\Big) \\
&= \sum_{y=1}^n p_{xy}\big(\mu_1^{k}(x), \mu_2^{k}(x)\big) \\
&\qquad \cdot \Big(g\big(x, \mu_1^{k}(x), \mu_2^{k}(x), y\big) + \alpha J_{\mu^k}(y)\Big).
\end{aligned}$$

In view of our tie breaking rule, this equation implies that $\mu_1^{k+1} = \mu_1^k$, and then Eq.(18) 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 μ^{k+1} and μ^k are agent-by-agent optimal. ■

D. Variants - Value and Policy 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 [5] and in the monograph [3] (Sections 5.4−5.6). It is outside the scope of this paper.

As Example 5 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 6 (Dependence of the Final Policy on the Agent Iteration Order)

Consider the problem of Example 5. In this problem there are two agent-by-agent optimal policies: the optimal policy μ^* where $\mu_1^*(x) = 1$ and $\mu_2^*(x) = 1$, and the suboptimal policy $\bar{\mu}$ where $\bar{\mu}_1(x) = 0$ and $\bar{\mu}_2(x) = 0$. Let the starting policy be μ^0 where $\mu_1^0(x) = 1$ and $\mu_2^0(x) = 0$. Then if agent 1 iterates first, the algorithm will terminate with the suboptimal policy, $\mu^1 = \bar{\mu}$, while if agent 2 iterates first, the algorithm will terminate with the optimal policy, $\mu^1 = \mu^*$.

As noted in Section II-E, 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 ℓ_1 is the agent that produces the minimal Q-factor, we fix ℓ_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.

1) 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 (14) 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 $\big(x^s, \tilde{\mu}(x^s)\big)$, $s = 1, \ldots, q$, which can be used to train a (policy) neural network to generate an approximation $\hat{\mu}$ to the policy $\tilde{\mu}$.[10] 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. 7). 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 potentially large state space is overcome by training value and policy networks.

The RL books [2] and [3] 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 [15], which applies partitioning to the solution of a challenging class of partial state information problems.

Note also that the policy evaluation $\tilde{J}_\mu$ of the base policy μ in the context of approximate PI may be done in several different ways. These include methods that compute iteratively the projection of J_μ onto a subspace spanned by basis functions or features, such as temporal difference methods, including TD(λ) and LSPE(λ), or methods based on matrix inversion such as LSTD(λ). We refer to RL textbooks, such as [65], [79], and the approximate DP book [13] for detailed accounts of these methods. We next discuss an alternative that is based on aggregation.

2) 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 [13] and [2], and the references quoted there. In particular, let us consider the aggregation with representative features framework of [2], Section 6.2 (see also [13], 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 [80]. Briefly, in this framework we introduce an expanded DP problem involving a finite number of additional states $i = 1, \ldots, s$, 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, $i = 1, \ldots, s$, 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 ϕ_{yj} mapping system states y to subsets of aggregate states X_j.

A base policy μ defines a set of aggregate state costs $r_\mu(j)$, $j = 1, \ldots, s$, which can be computed by simulation involving an "aggregate" Markov chain (see [2], [13]). The aggregate costs $r_\mu(j)$ define an approximation $\hat{J}_\mu$ of the cost function J_μ of the base policy, through the equation

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

Then an (approximate) multiagent rollout policy $\tilde{\mu}$ can be defined by one-step lookahead using $\hat{J}_\mu$ in place of J_μ, i.e., $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(\hat{J}_\mu)$, where the set $\widetilde{\mathcal{M}}_\mu(J)$ is defined for any μ and J by Eq.(14). In other words, the multiagent rollout algorithm with aggregation is defined by $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(\hat{J}_\mu)$ instead of its counterpart without aggregation, which is defined by $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J_\mu)$.

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 [2] and [13]). In particular, a multiagent PI algorithm based on aggregation admits a convergence result like the one of Prop. 2, except that this result asserts convergence to an agent-by-agent optimal policy for the associated aggregate problem. By contrast, approximate multiagent PI with value and policy networks (cf. Fig. 7) generically oscillates, as shown in sources such as [2], [13], [65], [81].

E. Policy Iteration and Q-Learning for the Reformulated Problem

Let us return to the equivalent reformulated problem introduced at the beginning of Section IV and illustrated in Fig. 6.

[10] There are quite a few methods for training an approximation architecture to represent a given policy by using training data that is generated by using this policy. In principle, these methods can be based on classification methodology, whereby a policy is represented as a classifier that associates states to controls; see [68]−[70]. There are also several related methods, known by names such as imitation learning, apprenticeship learning, or learning from demonstrations; see [71]−[78].

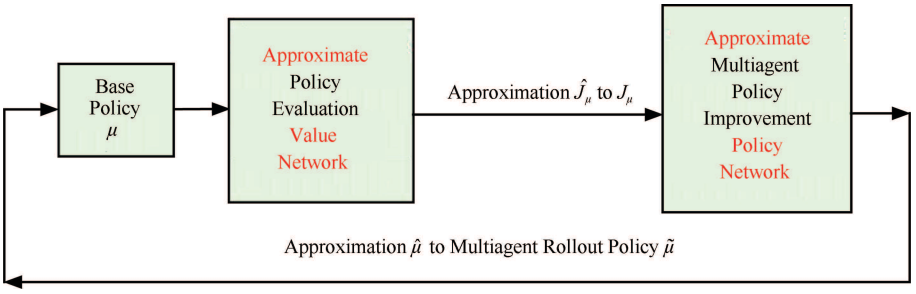


Fig. 7. Approximate multiagent PI with value and policy networks. The value network provides a trained approximation to the current base policy μ. The policy network provides a trained approximation $\hat{\mu}$ to the corresponding multiagent rollout policy $\tilde{\mu}$. The policy network may consist of m separately trained policy networks, one for each of the agent policies $\hat{\mu}_1, \ldots, \hat{\mu}_m$.

Instead of applying approximate multiagent PI to generate a sequence of multiagent policies

$$\mu^k(x) = \big(\mu_1^k(x), \mu_2^k(x), \ldots, \mu_m^k(x)\big) \tag{19}$$

as described in Section IV-A [cf. Eqs.(13) and (14)], we can use an ordinary type of PI method for the reformulated problem. The policies generated by this type of PI will exhibit not only a dependence on the state x [like the policies (19)], but also a dependence on the agents' controls, i.e., the generated policies will have the form

$$\big(\mu_1^k(x), \mu_2^k(x, u_1), \ldots, \mu_m^k(x, u_1, \ldots, u_{m-1})\big); \tag{20}$$

cf. the state space of Eq.(11) of the reformulated problem. Thus the policies are defined over a space that grows exponentially with the number of agents. This is a different PI method than the one of Section IV-A, and will generate a different sequence of policies, even when the initial policy is the same.

The exact form of this PI algorithm starts iteration k with a policy of the form (20), computes its corresponding evaluation (i.e., the cost function of the policy, defined over the state space of the reformulated problem)

$$J_k^0(x), J_k^1(x, u_1), \ldots, J_k^{m-1}(x, u_1, \ldots, u_{m-1}), \tag{21}$$

and generates the new policy

$$\big(\mu_1^{k+1}(x), \mu_2^{k+1}(x, u_1), \ldots, \mu_m^{k+1}(x, u_1, \ldots, u_{m-1})\big)$$

through the following policy improvement operation:

$$\begin{aligned}
&\mu_1^{k+1}(x) \in \arg \min_{u_1 \in U_1(x)} J_k^1(x, u_1),\\
&\mu_2^{k+1}(x, u_1) \in \arg \min_{u_2 \in U_2(x)} J_k^2(x, u_1, u_2),\\
&\qquad \ldots \qquad\qquad \ldots \qquad\qquad \ldots\\
&\mu_{m-1}^{k+1}(x, u_1, \ldots, u_{m-2}) \in\\
&\qquad \arg \min_{u_{m-1} \in U_{m-1}(x)} J_k^{m-1}(x, u_1, \ldots, u_{m-2}, u_{m-1}),\\
&\mu_m^{k+1}(x, u_1, \ldots, u_{m-1}) \in\\
&\qquad \arg \min_{u_m \in U_m(x)} \sum_{y=1}^{n} p_{xy}(u_1, \ldots, u_m)\\
&\qquad \cdot \big(g(x, u_1, \ldots, u_m, y) + \alpha J_k^0(y)\big).
\end{aligned} \tag{22}$$

According to the standard theory of discounted MDP, the preceding exact form of PI will terminate in a finite number of iterations with an optimal policy

$$\big(\hat{\mu}_1(x), \hat{\mu}_2(x, u_1), \ldots, \hat{\mu}_m(x, u_1, \ldots, u_{m-1})\big)$$

for the reformulated problem, which in turn can yield an optimal policy $\mu^* = (\mu_1^*, \ldots, \mu_m^*)$ for the original problem through the successive substitutions

$$\begin{aligned}
&\mu_1^*(x) = \hat{\mu}_1(x),\\
&\mu_2^*(x) = \hat{\mu}_2\big(x, \mu_1^*(x)\big),\\
&\qquad \ldots\\
&\mu_m^*(x) = \hat{\mu}_m\big(x, \mu_1^*(x), \ldots, \mu_{m-1}^*(x)\big),\\
&\text{for all} \quad x = 1, \ldots, n.
\end{aligned}$$

For example, the reader can verify that the algorithm will find the optimal policy of the one-state/two controls problem of Example 5 in two iterations, when started with the strictly suboptimal agent-by-agent optimal policy $\mu_1(x) = 0$, $\mu_2(x, u_1) \equiv 0$ of that problem.

Note that *the policy improvement operation (22) requires optimization over single control components rather over the entire vector $u = (u_1, \ldots, u_m)$, but it is executed over a larger and more complex state space*, whose size grows exponentially with the number of agents m. The difficulty with the large state space can be mitigated through approximate implementation with policy networks, but for this it is necessary to construct m policy networks at each iteration, with the ℓth agent network having as input $(x, u_1, \ldots, u_{\ell-1})$; cf. Eq.(20). Similarly, in the case of approximate implementation with value networks, it is necessary to construct m value networks at each iteration, with the ℓth agent network having as input $(x, u_1, \ldots, u_{\ell-1})$; cf. Eq.(21). Thus generating policies of the form (20) requires more complex value and policy network approximations. For a moderate number of agents, however, such approximations may be implementable without overwhelming difficulty, while maintaining the advantage of computationally tractable one-agent-at-a-time policy improvement operations of the form (22).

We may also note that the policy improvement operations (22) can be executed in parallel for all states of the reformulated problem. Moreover, the corresponding PI method has a potentially significant advantage: it aims to approximate an

optimal policy rather than one that is merely agent-by-agent optimal.

1) Q-Learning for the Reformulated Problem

The preceding discussion assumes that the base policy for the multiagent rollout algorithm is a policy generated through an off-line exact or approximate PI algorithm. We may also use the reformulated problem to generate a base policy through an off-line exact or approximate value iteration (VI) or Q-learning algorithm. In particular, the exact form of the VI algorithm can be written in terms of multiple Q-factors as follows:

$$\begin{aligned} J^{k+1}(x) &= \min_{u_1 \in U_1(x)} Q_1^k(x,u_1), \qquad x = 1,\ldots,n, \\ Q_1^{k+1}(x,u_1) &= \min_{u_2 \in U_2(x)} Q_2^k(x,u_1,u_2), \\ &\qquad x = 1,\ldots,n,\ u_1 \in U_1(x), \\ &\quad \ldots \qquad \ldots \qquad \ldots \end{aligned} \tag{23}$$

$$\begin{aligned} &Q_{m-1}^{k+1}(x,u_1,\ldots,u_{m-1}) \\ &\quad = \min_{u_m \in U_m(x)} Q_m^k(x,u_1,\ldots,u_{m-1},u_m), \\ &\qquad x = 1,\ldots,n,\ u_\ell \in U_\ell(x),\ \ell = 1,\ldots,m-1, \\ &Q_m^{k+1}(x,u_1,\ldots,u_m) = \sum_{y=1}^{n} p_{xy}(u_1,\ldots,u_m) \\ &\qquad \cdot \big(g(x,u_1,\ldots,u_m,y) + \alpha J^k(y)\big), \\ &\qquad x = 1,\ldots,n,\ (u_1,\ldots,u_m) \in U(x). \end{aligned}$$

It gives both the value iterate sequence $\{J^k\}$ and the Q-factor iterate sequences $\{Q_\ell^k\}$, $\ell = 1,\ldots,m$, at the states of the reformulated problem [cf. Eq.(11)]. The convergence of the preceding algorithm, as well as its asynchronous stochastic approximation/Q-learning variants, is covered by the classical theory of infinite horizon DP and the theory of the Q-learning method applied to the reformulated problem (see the analysis of Tsitsiklis [82], and subsequent mathematical works on the convergence of Q-learning and variations). In particular, the sequence $\{J^k\}$ converges to J^* (the optimal cost function), while each sequence $\{Q_\ell^k(x,u_1,\ldots,u_\ell)\}$ converges to $Q_\ell^*(x,u_1,\ldots,u_\ell)$, the optimal cost that can obtained if we start at x, the agents $1,\ldots\ell$ choose next the controls $u_1,\ldots,u_\ell$, respectively, and all the subsequent agent controls are chosen optimally.

Note that all of the iterations (23) involve minimization over a single agent control component, but are executed over a state space that grows exponentially with the number of agents. On the other hand one may use approximate versions of the VI and Q-learning iterations (23) (such as SARSA [78], and DQN [83]) to mitigate the complexity of the large state space through the use of neural networks or other approximation architectures. Once an approximate policy is obtained through a neural network-based variant of the preceding algorithm, it can be used as a base policy for on-line multiagent rollout that involves single agent component minimizations.

F. 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 II-E. Here, we use multiagent one-step lookahead, we then apply rollout with base policy μ for a limited number of steps, and finally we approximate the cost of the remaining steps using some terminal cost function approximation J. In truncated rollout schemes, J may be heuristically chosen, may be based on problem approximation, or may be based on a more systematic simulation methodology. For example, the values $J_\mu(x)$ can be computed by simulation for all x in a subset of representative states, and J can 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 may be used on-line in place of J_μ as a terminal cost function approximation.

We have the following performance bounds the proofs of which are given in [3] (Prop. 5.2.7).

Proposition 2: (Performance Bounds for Multiagent Truncated Rollout)
Let μ be a base 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 μ for a given number of steps, and followed by a terminal cost function approximation J. Let $\tilde{\mu}$ be the generated rollout policy.
(a) We have

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

where

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

(b) We have

$$J_{\tilde{\mu}}(x) \le J_\mu(x) + \frac{2}{1-\alpha} \max_{y=1,\ldots,n} \big|J(y) - J_\mu(y)\big|, \qquad x = 1,\ldots,n.$$

These error bounds provide some guidance for the implementation of truncated rollout, as discussed in Section 5.2.6 of the book [3]. An important point is that the error bounds do not depend on the number of agents m, so the preceding proposition guarantees the same level of improvement of the rollout policy over the base policy for one-agent-at-a-time and all-agents-at-once rollout. In fact there is no known error bound that is better for standard rollout than for multiagent rollout. This provides substantial analytical support for the multiagent rollout approach, and is consistent with the results of computational experimentation available so far.

V. Autonomous Multiagent Rollout For Infinite Horizon Problems - Signaling Policies

The autonomous multiagent rollout scheme of Section III can be extended to infinite horizon problems. The idea is again to use in addition to the base policy $\mu = (\mu_1,\ldots,\mu_m)$, a *signaling policy* $\widehat{\mu} = (\widehat{\mu}_1,\ldots,\widehat{\mu}_m)$, which is computed off-line and embodies agent coordination.

In particular, given a base policy μ and a signaling policy $\widehat{\mu}$, the autonomous multiagent rollout algorithm gener-

ates a policy $\tilde{\mu}$ as follows. At state x, it obtains $\tilde{\mu}(x) = \big(\tilde{\mu}_1(x), \ldots, \tilde{\mu}_m(x)\big)$, according to

$$\begin{aligned}
\tilde{\mu}_1(x) \in \arg \min_{u_1 \in U_1(x)} & E\Big\{g\big(x, u_1, \mu_2(x), \ldots, \mu_m(x), w\big) \\
& + \alpha J_\mu\Big(f\big(x, u_1, \mu_2(x), \ldots, \mu_m(x), w\big)\Big)\Big\}, \\
\tilde{\mu}_2(x) \in \arg \min_{u_2 \in U_2(x)} & E\Big\{g\big(x, \widehat{\mu}_1(x), u_2, \ldots, \mu_m(x), w\big) \\
& + \alpha J_\mu\Big(f\big(x, \widehat{\mu}_1(x), u_2, \ldots, \mu_m(x), w\big)\Big)\Big\}, \\
& \cdots \qquad\qquad \cdots \qquad\qquad \cdots \\
\tilde{\mu}_m(x) \in \arg \min_{u_m \in U_m(x)} & E\Big\{g\big(x, \widehat{\mu}_1(x), \ldots, \widehat{\mu}_{m-1}(x), u_m, w\big) \\
& + \alpha J_\mu\Big(f\big(x, \widehat{\mu}_1(x), \ldots, \widehat{\mu}_{m-1}(x), u_m, w\big)\Big)\Big\}.
\end{aligned} \tag{24}$$

Note that the preceding computation of the controls $\tilde{\mu}_1(x), \ldots, \tilde{\mu}_m(x)$ can be done asynchronously and in parallel, without agent intercommunication of their computed controls, since the signaling policy values $\widehat{\mu}_1(x), \ldots, \widehat{\mu}_{m-1}(x)$ 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., $\widehat{\mu} = \mu$. However, this choice does not guarantee policy improvement and can lead to poor performance, as evidenced by Example 3. Still, using the base policy as signaling policy can be an attractive possibility, which one may wish to try (perhaps in some modified form) on specific problems, in view of its simplicity and its parallelization potential. On the other hand, if the signaling policy is taken to be the (nonautonomous) multiagent rollout policy $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J_\mu)$ [cf. Eq.(14)], i.e., $\widehat{\mu} = \tilde{\mu}$, 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 $\widehat{\mu}$ to approximate the multiagent rollout policy $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J_\mu)$. In particular, we may obtain the policy $\widehat{\mu} = (\widehat{\mu}_1, \ldots, \widehat{\mu}_{m-1})$, by off-line training and approximation in policy space using a neural network, with the training set generated by the multiagent rollout policy $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J_\mu)$; cf. Section IV-C and Fig. 7. Here are two possibilities along these lines:

(a) We may use the approximate multiagent PI algorithm with policy network approximation (cf. Section IV-D), start with some initial policy μ^0, and produce k new policies $\mu^1, \ldots, \mu^k$. Then the rollout scheme would use μ^k as signaling policy, and μ^{k-1} as base policy. The final rollout policy thus obtained can be implemented on-line with the possibility of on-line replanning and the attendant robustness property.

(b) We may generate a base policy μ by a policy gradient or random search method, and approximate the corresponding multiagent rollout policy $\tilde{\mu} \in \widetilde{\mathcal{M}}_\mu(J_\mu)$ by off-line neural network training. Then the rollout scheme would use the neural network policy thus obtained as signaling policy, and μ as base policy. Again, the final rollout policy thus obtained can be implemented on-line with the possibility of on-line replanning and the attendant robustness property.

Note that if the neural network were to provide a perfect approximation of the rollout policy, the policy defined by Eq.(24) 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 (14), the policy defined by Eq.(24) will have better performance than both the base policy and the signaling policy. This was confirmed by the computational results of the paper [64], within the context of a multi-robot 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 $\widehat{\mu}_i$ are good approximations to $\tilde{\mu}_i$), while allowing the speedup afforded by autonomous agent operation, as well as on-line replanning when the problem data varies over time. The following example aims to illustrate these ideas.

Example 7 (Autonomous Spiders and Flies)

Let us return to the two-spiders-and-two-flies problem of Examples 2 and 4, and use it as a test of the sensitivity of autonomous multiagent rollout algorithm with respect to variations in the signaling policy. Formally, we view the problem as an infinite horizon MDP of the stochastic shortest path type. Recall that the base policy moves each spider selfishly towards the closest surviving fly with no coordination with the other spider, while both the standard and the multiagent rollout algorithms are optimal.

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). Then it can be verified that the autonomous multiagent rollout policy with arbitrary signaling policy acts optimally as long as the spiders are initially separated on the line by at least one unit. What is happening here is that the Q-factors that are minimized in Eq.(24) involve a first stage cost (which is fixed at 1 and is independent of the signaling policy), and the cost of the base policy $J_\mu(y)$ starting from the next state y, which is not sufficiently affected by the signaling policy $\widehat{\mu}$ to change the outcome of the Q-factor minimizations (24).

On the other hand, we saw in Example 4 that if we use as signaling policy the base policy, and the two spiders start at the same position, the spiders cannot coordinate their move choices, and they never separate. Thus the algorithm gets locked onto an oscillation where the spiders keep moving together back and forth, and (in contrast with the base policy) never capture the flies!

The preceding example shows how a misguided choice of signaling policy (namely the base policy), may lead to very poor performance starting from some initial states, but also a very good performance starting from other initial states. Since detecting the "bad" initial states may be tricky for a complicated problem, it seems that one should be careful to support with analysis (to the extent possible), as well as substantial experimentation the choice of a signaling policy.

The example also illustrates a situation where 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 (24) whose dominance is not affected much by the choice of the signaling policy. As noted in Section III, 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 using on-line simulation will be proportional to the size of the largest group rather than proportional to the number of agents.[11] Note, however, that the "coupled" groups may depend on the current state, and that deciding which agents to include within each group may not be easy.

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: the agents can compute on-line their respective controls asynchronously and in parallel without explicit inter-agent coordination, while taking advantage of local information for on-line replanning.

VI. 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. 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 standard rollout algorithm (which is in fact intractable for only a modest number of agents).

It is useful to keep in mind that the multiagent rollout policy is essentially the standard (all-agents-at-once) rollout policy applied to the (equivalent) reformulated problem of Fig. 3 (or Fig. 6 in the infinite horizon case). As a result, known insights, results, error bounds, and approximation techniques for standard rollout apply in suitably reformulated form. Moreover, the reformulated problem may form the basis for an approximate PI algorithm with agent-by-agent policy improvement, as we have discussed in Section IV-E.

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 II-E of the RL book [2]), 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.

The two multiagent PI algorithms that we have proposed in Sections IV-A and IV-E differ in their convergence guarantees when implemented exactly. In particular the PI algorithm of Section IV-A, in its exact form, is only guaranteed to terminate with an agent-by-agent optimal policy. Still in many cases (including the problems that we have tested computationally) it may produce comparable performance to the standard PI algorithm, which however involves prohibitively large computation even for a moderate number of agents. The PI algorithm of Section IV-E, in its exact form, is guaranteed to terminate with an optimal policy, but its implementation must be carried out over a more complex space. Its approximate form with policy networks has not been tested on challenging problems, and it is unclear whether and under what circumstances it offers a tangible performance advantage over approximate forms of the PI algorithm of Section IV-A.

Our multiagent PI convergence result of Prop.2 can be extended beyond the finite-state discounted problem 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, Q-learning and other related methods. The analysis of such extensions is reported separately; see [3] and [5].

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 perfect 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. Analysis and computational experimentation with such schemes should be very useful and may lead to improved understanding of their properties.

[11]The concept of weakly coupled subsystems figures prominently in the literature of decentralized control of systems with continuous state and control spaces, where it is usually associated with a (nearly) block diagonal structure of the Hessian matrix of a policy's Q-factors (viewed as functions of the agent control components $u_1, \ldots, u_m$ for a given state). In this context, the blocks of the Hessian matrix correspond to the coupled groups of agents. This analogy, while valid at some conceptual level, does not fully apply to our problem, since we have assumed a discrete control space.

Several unresolved questions remain regarding algorithmic variations and conditions that guarantee that our PI algorithm of Section IV-A obtains an optimal policy rather than one that is agent-by-agent optimal (the paper [5] provides relevant discussions). Moreover, approximate versions of our PI algorithms that use value and policy network approximations are of great practical interest, and are a subject for further investigation (the papers by Bhattacharya *et al*. [15] and [64] discuss in detail various neural network-based implementations, in the context of some challenging POMDP multi-robot repair applications). 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 address special problem types that involve multicomponent control structures.

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 book [3] and the paper [6].

REFERENCES

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. I*. 4th ed. Belmont, USA: Athena Scientific, 2017.

[2] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, USA: Athena Scientific, 2019.

[3] D. P. Bertsekas, *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Belmont, USA: Athena Scientific, 2020.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and Shogi by self-play with a general reinforcement learning algorithm," arXiv preprint arXiv: 1712.01815, 2017.

[5] D. P. Bertsekas, "Multiagent value iteration algorithms in dynamic programming and reinforcement learning," arxiv: 2005.01627, 2020.

[6] D. P. Bertsekas, "Constrained multiagent rollout and multidimensional assignment with the auction algorithm," arXiv:2002.07407, 2020.

[7] D. P. Bertsekas, "Distributed dynamic programming," *IEEE Trans. Autom. Control*, vol. 27, no. 3, pp. 610−616, Jun. 1982.

[8] D. P. Bertsekas, "Asynchronous distributed computation of fixed points," *Math. Programming*, vol. 27, no. 1, pp. 107−120, Sep. 1983.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, USA: Prentice-Hall, 1989.

[10] D. P. Bertsekas and H. Z. Yu, "Asynchronous distributed policy iteration in dynamic programming," in *Proc. 48th Annu. Allerton Conf. Communication, Control, and Computing*, Allerton, USA, 2010, pp. 1368−1374.

[11] D. P. Bertsekas and H. Z. Yu, "Q-learning and enhanced policy iteration in discounted dynamic programming," *Math. Oper. Res.*, vol. 37, pp. 66−94, Feb. 2012.

[12] H. Z. Yu and D. P. Bertsekas, "Q-learning and policy iteration algorithms for stochastic shortest path problems," *Ann. Oper. Res.*, vol. 208, no. 1, pp. 95−132, Sep. 2013.

[13] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. II*. 4th ed. Belmont, USA: Athena Scientific, 2012.

[14] D. P. Bertsekas, *Abstract Dynamic Programming*. Belmont, USA: Athena Scientific, 2018.

[15] S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil, and D. P. Bertsekas, "Reinforcement learning for POMDP: Partitioned rollout and policy iteration with application to autonomous sequential repair problems," *IEEE Rob. Autom. Lett.*, vol. 5, no. 3, pp. 3967−3974, Jul. 2020.

[16] H. S. Witsenhausen, "A counterexample in stochastic optimum control," *SIAM J. Control*, vol. 6, no. 1, pp. 131−147, 1968.

[17] H. S. Witsenhausen, "Separation of estimation and control for discrete time systems," *Proc. IEEE*, vol. 59, no. 11, pp. 1557−1566, Nov. 1971.

[18] J. Marschak, "Elements for a theory of teams," *Manage. Sci.*, vol. 1, no. 2, pp. 127−137, Jan. 1975.

[19] R. Radner, "Team decision problems," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 857−881, Sep. 1962.

[20] H. S. Witsenhausen, "On information structures, feedback and causality," *SIAM J. Control*, vol. 9, no. 2, pp. 149−160, 1971.

[21] J. Marschak and R. Radner, *Economic Theory of Teams*. New Haven, USA: Yale University Press, 1976.

[22] N. Sandell, P. Varaiya, M. Athans, and M. Safonov, "Survey of decentralized control methods for large scale systems," *IEEE Trans. Autom. Control*, vol. 23, no. 2, pp. 108−128, Apr. 1978.

[23] T. Yoshikawa, "Decomposition of dynamic team decision problems," *IEEE Trans. Autom. Control*, vol. 23, no. 4, pp. 627−632, Aug. 1978.

[24] Y. C. Ho, "Team decision theory and information structures," *Proc. IEEE*, vol. 68, no. 6, pp. 644−654, Jun. 1980.

[25] D. Bauso and R. Pesenti, "Generalized person-by-person optimization in team problems with binary decisions," in *Proc. American Control Conf.*, Seattle, USA, 2008, pp. 717−722.

[26] D. Bauso and R. Pesenti, "Team theory and person-by-person optimization with binary decisions," *SIAM J. Control Optim.*, vol. 50, no. 5, pp. 3011−3028, Jan. 2012.

[27] A. Nayyar, A. Mahajan, and D. Teneketzis, "Decentralized stochastic control with partial history sharing: A common information approach," *IEEE Trans. Autom. Control*, vol. 58, no. 7, pp. 1644−1658, Jul. 2013.

[28] A. Nayyar and D. Teneketzis, "Common knowledge and sequential team problems," *IEEE Trans Autom. Control*, vol. 64, no. 12, pp. 5108−5115, Dec. 2019.

[29] Y. Y. Li, Y. J. Tang, R. Y. Zhang, and N. Li, "Distributed reinforcement learning for decentralized linear quadratic control: A derivative-free policy optimization approach," arXiv:1912.09135, 2019.

[30] G. Qu and N. Li, "Exploiting Fast Decaying and Locality in Multi-Agent MDP with Tree Dependence Structure," in *Proc. of CDC*, Nice, France, 2019.

[31] A. Gupta, "Existence of team-optimal solutions in static teams with common information: A topology of information approach," *SIAM J. Control Optim.*, vol. 58, no. 2, pp. 998−1021, Apr. 2020.

[32] F. Bullo, J. Cortes, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. St. Princeton, USA: Princeton University Press, 2009.

[33] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton, USA: Princeton University Press, 2010.

[34] M. S. Mahmoud, *Multiagent Systems: Introduction and Coordination Control*. Boca Raton, USA: CRC Press, 2020.

[35] R. Zoppoli, M. Sanguineti, G. Gnecco, and T. Parisini, *Neural Approximations for Optimal Control and Decision*, Springer, 2020.

[36] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*, Springer International Publishing, 2016.

[37] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," arXiv:1707.09183, 2017.

[38] K. Q. Zhang, Z. R. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," arXiv:1911.10635, 2019.

[39] L. S. Shapley, "Stochastic games," *Proc. Natl. Acad. Sci.*, vol. 39, no. 10, pp. 1095−1100, 1953.

[40] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirsh, Eds. Amsterdam, The Netherlands: Elsevier, 1994, pp. 157−163.

[41] K. P. Sycara, "Multiagent systems," *AI Mag.*, vol. 19, no. 2, pp. 79−92, Jun. 1998.

[42] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Auton. Rob.*, vol. 8, no. 3, pp. 345−383, Jun. 2000.

[43] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Auton. Agen. Multi-Agent Syst.*, vol. 11, no. 3, pp. 387−434, Nov. 2005.

[44] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern., Part C*, vol. 38, no. 2, pp. 156−172, Mar. 2008.

[45] L. Busoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications-1*, D. Srinivasan and L. C. Jain, Eds. Berlin, Germany: Springer, 2010, pp. 183−221.

[46] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems," *Knowl. Eng. Rev.*, vol. 27, no. 1, pp. 1−31, Feb. 2012.

[47] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Auton. Agent. Multi-Agent Syst.*, vol. 33, no. 6, pp. 750−797, Oct. 2019.

[48] A. OroojlooyJadid and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," arXiv:1908.03963, 2019.

[49] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Trans Cybern.*, vol. 50, no. 9, pp. 3826−3839, Sep. 2020.

[50] G. Tesauro, "Extending Q-learning to general adaptive multi-agent systems," in *Proc. 16th Int. Conf. Neural Information Processing Systems*, 2004, pp. 871−878.

[51] F. A. Oliehoek, J. F. P. Kooij, and N. Vlassis, "The cross-entropy method for policy search in decentralized POMDPs," *Informatica*, vol. 32, no. 4, pp. 341−357, 2008.

[52] P. Pennesi and I. C. Paschalidis, "A distributed actor-critic algorithm and applications to mobile sensor network coordination problems," *IEEE Trans. Autom. Control*, vol. 55, no. 2, pp. 492−497, Feb. 2010.

[53] I. C. Paschalidis and Y. W. Lin, "Mobile agent coordination via a distributed actor-critic algorithm," in *Proc. 19th Mediterranean Conf. Control Automation*, Corfu, Greece, 2011, pp. 644−649.

[54] S. Kar, J. M. F. Moura, and H. V. Poor, "*QD*-Learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1848−1862, Apr. 2013.

[55] J. N. Foerster, Y. M. Assael, N. De Freitas, and S. Whiteson, "Learning to Communicate with Deep Multi-Agent Reinforcement Learning," in *Proc. 30th Int. Conf. Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 2137−2145.

[56] S. Omidshafiei, A. A. Agha-Mohammadi, C. Amato, S. Y. Liu, J. P. How, and J. Vian, "Graph-based cross entropy method for solving multi-robot decentralized POMDPs," in *Proc. IEEE Int. Conf. Robotics and Automation*, Stockholm, Sweden, 2016, pp. 5395−5402.

[57] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Autonomous Agents and Multiagent Systems*, Best Papers, Brazil, 2017, pp. 66−83.

[58] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Information Processing Systems*, Long Beach, USA, 2017, pp. 6379−6390.

[59] M. Zhou, Y. Chen, Y. Wen, Y. D. Yang, Y. F. Su, W. N. Zhang, D. Zhang, and J. Wang, "Factorized Q-learning for large-scale multi-agent systems," arXiv:1809.03738, 2018.

[60] K. Q. Zhang, Z. R. Yang, H. Liu, T. Zhang, and T. Başar, "Fully decentralized multi-agent reinforcement learning with networked agents," arXiv:1802.08757, 2018.

[61] Y. Zhang and M. M. Zavlanos, 2019 "Distributed off-policy actor-critic reinforcement learning with policy consensus," in *Proc. IEEE 58th Conf. Decision and Control*, Nice, France, 2018, pp. 4674−4679.

[62] C. S. de Witt, J. N. Foerster, G. Farquhar, P. H. S. Torr, W. Boehmer, and S. Whiteson, "Multi-agent common knowledge reinforcement learning", in *Proc. 31st Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019, pp. 9927−9939.

[63] D. P. Bertsekas, "Multiagent rollout algorithms and reinforcement learning," arXiv: 2002.07407, 2019.

[64] S. Bhattacharya, S. Kailas, S. Badyal, S. Gil, and D. P. Bertsekas, "Multiagent rollout and policy iteration for POMDP with application to multi-robot repair problems," in *Proc. Conf. Robot Learning, 2020*; also arXiv preprint, arXiv:2011.04222.

[65] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, USA: Athena Scientific, 1996.

[66] G. Tesauro, and G. R. Galperin, "On-line policy improvement using Monte-Carlo search," in *Proc. 9th Int. Conf. Neural Information Processing Systems*, Denver, USA, 1996, pp. 1068−1074.

[67] D. P. Bertsekas, *Nonlinear Programming*. 3rd ed. Belmont, USA: Athena Scientific, 2016.

[68] M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," in *Proc. 20th Int. Conf. Machine Learning*, Washington, USA, 2003, pp. 424−431.

[69] C. Dimitrakakis and M. G. Lagoudakis, "Rollout sampling approximate policy iteration," *Mach. Learn.*, vol. 72, no. 3, pp. 157−171, Jul. 2008.

[70] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Analysis of a classification-based policy iteration algorithm," in *Proc. 27th Int. Conf. Machine Learning*, Haifa, Israel, 2010.

[71] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Machine Learning*, Banff, Canada, 2004.

[72] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Rob. Auton. Syst.*, vol. 57, no. 5, pp. 469−483, May 2009.

[73] G. Neu and C. Szepesvari, "Apprenticeship learning using inverse reinforcement learning and gradient methods," arXiv:1206.5264, 2012.

[74] H. Ben Amor, D. Vogt, M. Ewerton, E. Berger, B. Jung, and J. Peters, "Learning responsive robot behavior by imitation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Tokyo, Japan, 2013, pp. 3257−3264.

[75] J. Lee, "A survey of robot learning from demonstrations for human-robot collaboration," arXiv:1710.08789, 2017.

[76] M. K. Hanawal, H. Liu, H. H. Zhu, and I. C. Paschalidis, "Learning policies for Markov decision processes from data," *IEEE Trans. Autom. Control*, vol. 64, no. 6, pp. 2298−2309, Jun. 2019.

[77] D. Gagliardi and G. Russo, "On a probabilistic approach to synthesize control policies from example datasets," arXiv:2005.11191, 2020.

[78] T. T. Xu, H. H. Zhu, and I. C. Paschalidis, "Learning parametric policies and transition probability models of Markov decision processes from data," *Eur. J. Control*, 2020.

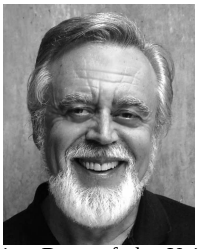
[79] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd Ed. Cambridge, USA: MIT Press, 2018.

[80] D. P. Bertsekas, "Feature-based aggregation and deep reinforcement learning: A survey and some new implementations," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 1, pp. 1−31, Jan. 2019.

[81] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310−335, Jul. 2011; Expanded version appears as Lab. for Info. and Decision System Report LIDS-2833, MIT, 2011.

[82] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Mach. Learn.*, vol. 16, no. 3, pp. 185−202, Sep. 1994.

[83] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529−533, 2015.



Dimitri Bertsekas undergraduate studies were in engineering at the National Technical University of Athens, Greece. He obtained his MS in electrical engineering at the George Washington University, Wash. DC in 1969, and his Ph.D. in system science in 1971 at the Massachusetts Institute of Technology.

Dr. Bertsekas has held faculty positions with the Engineering-Economic Systems Dept., Stanford University (1971-1974) and the Electrical Engineering Dept. of the University of Illinois, Urbana (1974-1979). From 1979 to 2019 he was with the Electrical Engineering and Computer Science Department of the Massachusetts Institute of Technology (M.I.T.), where he served as McAfee Professor of Engineering. In 2019, he was appointed Fulton Professor of Computational Decision Making, and a full time faculty member at the department of Computer, Information, and Decision Systems Engineering at Arizona State University, Tempe, while maintaining a research position at MIT. His research spans several fields, including optimization, control, large-scale computation, and data communication networks, and is closely tied to his teaching and book authoring activities. He has written numerous research papers, and eighteen books and research monographs, several of which are used as textbooks in MIT classes. Most recently Dr Bertsekas has been focusing on reinforcement learning, and authored a textbook in 2019, and a research monograph on its distributed and multiagent implementation aspects in 2020.

Professor Bertsekas was awarded the INFORMS 1997 Prize for Research Excellence in the Interface Between Operations Research and Computer Science for his book "Neuro-Dynamic Programming", the 2000 Greek National Award for Operations Research, the 2001 ACC John R. Ragazzini Education Award, the 2009 INFORMS Expository Writing Award, the 2014 ACC Richard E. Bellman Control Heritage Award for "contributions to the foundations of deterministic and stochastic optimization-based methods in systems and control," the 2014 Khachiyan Prize for Life-Time Accomplishments in Optimization, and the SIAM/MOS 2015 George B. Dantzig Prize. In 2018, he was awarded, jointly with his coauthor John Tsitsiklis, the INFORMS John von Neumann Theory Prize, for the contributions of the research monographs "Parallel and Distributed Computation" and "Neuro-Dynamic Programming". In 2001, he was elected to the United States National Academy of Engineering for "pioneering contributions to fundamental research, practice and education of optimization/control theory, and especially its application to data communication networks."

Dr. Bertsekas' recent books are "Introduction to Probability: 2nd Edition" (2008), "Convex Optimization Theory" (2009), "Dynamic Programming and Optimal Control," Vol. I, (2017), and Vol. II: (2012), "Abstract Dynamic Programming" (2018), "Convex Optimization Algorithms" (2015), "Reinforcement Learning and Optimal Control" (2019), and "Rollout, Policy Iteration, and Distributed Reinforcement Learning"(2020), all published by Athena Scientific.