September 2019 (Revised March 2020)

Multiagent Rollout Algorithms and Reinforcement Learning

Dimitri Bertsekas†

Abstract

We consider finite and infinite horizon dynamic programming problems, where the control at each stage consists of several distinct decisions, each one made by one of several agents. We introduce an approach, whereby at every stage, each agent's decision is made by executing a local rollout algorithm that uses a base policy, together with some coordinating information from the other agents. The amount of local computation required at every stage by each agent is independent of the number of agents, while the amount of total computation (over all agents) grows linearly with the number of agents. By contrast, with the standard rollout algorithm, the amount of total computation grows exponentially with the number of agents. Despite the drastic reduction in required computation, we show that our algorithm has the fundamental cost improvement property of rollout: an improved performance relative to the base policy. We explore related reinforcement learning and approximate policy iteration algorithms, and we discuss how the cost improvement property steers the algorithm towards convergence to an agent-by-agent optimal solution. We also discuss possibilities to improve further the method's computational efficiency through limited agent coordination and parallelization of the agents' computations.

1. MULTIAGENT PROBLEM FORMULATION - FINITE HORIZON PROBLEMS

We consider a standard form of an N-stage dynamic programming (DP) problem (see [Ber17], [Ber19]), which involves the discrete-time dynamic system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1, \tag{1.1}$$

where x_k is an element of some (possibly infinite) state space, the control u_k is an element of some finite control space, and w_k is a random disturbance, which is characterized by a probability distribution $P_k(\cdot \mid x_k, u_k)$ 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. 1.1.

We consider policies of the form

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

† McAfee Professor of Engineering, MIT, Cambridge, MA, and Fulton Professor of Computational Decision Making, ASU, Tempe, AZ.

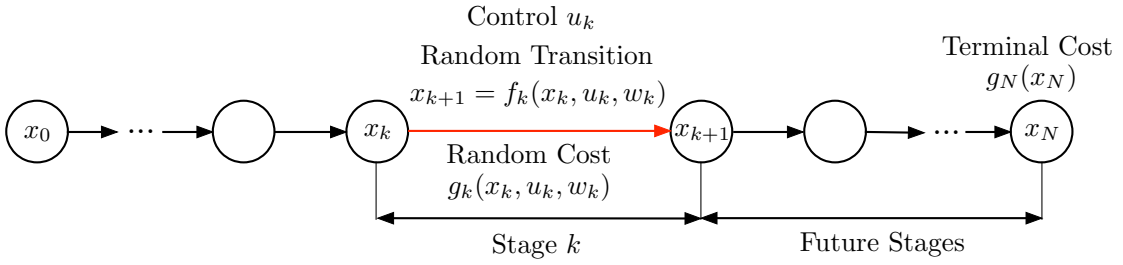

Figure 1.1 Illustration of the N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated according to a system equation

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

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

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 at 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 over all the random variables w_k and x_k. The optimal cost is the function J^* of the initial state x_0, defined by

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

where Π is the set of all policies, while an optimal policy π^* is one that attains the minimal cost for every initial state; i.e.,

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0).$$

Since J^* and π^* are typically hard to obtain by exact DP, we consider reinforcement learning (RL) algorithms for suboptimal solution, and focus on rollout, which we describe next.

1.1. The Standard Rollout Algorithm

The aim of rollout is policy improvement. In particular, 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 wish to use rollout to obtain an improved policy, i.e., one that achieves cost that is at most $J_{k,\pi}(x_k)$ starting from each x_k. The standard rollout algorithm effects on-line control of the system as follows (see the textbooks [BeT96], [Ber17], [Ber19]):

Standard One-Step Lookahead Rollout Algorithm:

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

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big)\Big\}. \tag{1.2}$$

The one-step minimization (1.2), which uses $J_{k+1,\pi}$ in place of the optimal cost-to-go function, defines a policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, where for all x_k and k, $\tilde{\mu}_k(x_k)$ is equal to the control $\tilde{u}_k$ obtained from Eq. (1.2). This policy is referred to as the *rollout policy*. The fundamental cost improvement result here is that the rollout policy improves over the base policy in the sense that

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \forall\ x_k, k, \tag{1.3}$$

where $J_{k,\tilde{\pi}}(x_k)$, $k = 0, \ldots, N$, is the cost-to-go of the rollout policy starting from state x_k ([Ber17], Section 6.4, or [Ber19], Section 2.4.2).

The expected value in Eq. (1.2) is the Q-factor of the pair (x_k, u_k) corresponding to the base policy:

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

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). Unfortunately, in the multiagent context to be discussed shortly, the number of controls in $U_k(x_k)$, and the attendant computation of Q-factors, grow rapidly with the number of agents, and can become very large. The purpose of this paper is to introduce a modified rollout algorithm for the multiagent case, which requires much less computation while maintaining the cost improvement property (1.3).

1.2. The Multiagent Case

Let us assume a special structure of the control space, corresponding to a multiagent version of the problem.†

† While we focus on multiagent problems, our methodology applies to any problem where the control u_k consists of m components, $u_k = (u_k^1, \ldots, u_k^m)$.

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†

$$U_k(x_k) = U_k^1(x_k) \times \cdots \times U_k^m(x_k). \tag{1.4}$$

Then the minimization (1.2) involves as many as n^m Q-factors, where n is the maximum number of elements of the sets $U_k^i(x_k)$ [so that n^m is an upper bound to the number of controls in $U_k(x_k)$, in view of its Cartesian product structure (1.4)]. Thus the computation required by the rollout algorithm is of order $O(n^m)$ per stage.

In this paper we propose an alternative rollout algorithm that achieves the cost improvement property (1.3) at much smaller computational cost, namely of order $O(nm)$ per stage. A key idea here is that the computational requirements of the rollout one-step minimization (1.2) are proportional to the number of controls in the set $U_k(x_k)$ and are independent of the size of the state space. This motivates a reformulation of the problem, first suggested in the neuro-dynamic programming book [BeT96], Section 6.1.4, whereby control space complexity is traded off with state space complexity by "unfolding" the control u_k into its m components, which are applied one *agent-at-a-time* rather than *all-agents-at-once*. We discuss this idea next within the multiagent context.

1.3. Trading off Control Space Complexity with State Space Complexity

We noted that a major issue in rollout is the minimization over $u_k \in U_k(x_k)$ in Eq. (1.2), which may be very time-consuming when the size of the control constraint set is large. In particular, in the multiagent case where $u_k = (u_k^1, \ldots, u_k^m)$, the time to perform this minimization is typically exponential in m. In this case, we can reformulate the problem by breaking down the collective decision u_k into m individual component decisions, thereby reducing the complexity of the control space while increasing the complexity of the state space. The potential advantage is that the extra state space complexity does not affect the computational requirements of some RL algorithms, including rollout.

To this end, we introduce a modified but equivalent problem, involving *one-agent-at-a-time control selection*. At the generic state x_k, we break down the control u_k into the sequence of the m controls

† The Cartesian product structure of the constraint set is adopted here for simplicity of exposition, particularly when arguing about computational complexity. The idea of trading off control space complexity and state space complexity (cf. Section 1.3), on which this paper rests, does not depend on a Cartesian product constraint structure. Of course when this structure is present, it simplifies the computations of the methods of this paper.

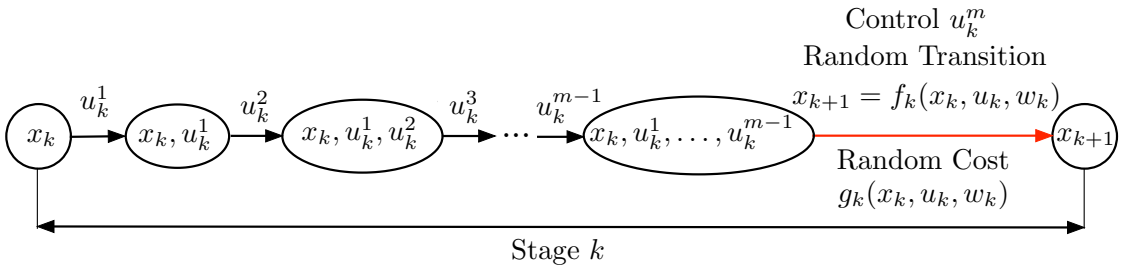

Figure 1.2 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u_k consists of m components $u_k^1, u_k^2, \ldots, u_k^m$:

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

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^m leads from $(x_k, u_k^1, \ldots, u_k^{m-1})$ to $x_{k+1} = f_k(x_k, u_k, w_k)$, and a random stage cost $g_k(x_k, u_k, w_k)$ is incurred.

$u_k^1, u_k^2, \ldots, u_k^m$, 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. 1.2.

It is evident that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, while the cost structure of the two problems is the same. In particular, every policy

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

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.†

† It would superficially appear that the reformulated problem contains more policies than the original problem, because its form is more general: in the reformulated problem the policy at the kth stage applies the control components

$$\mu_k^1(x_k), \mu_k^2(x_k, u_k^1), \ldots, \mu_k^m(x_k, u_k^1, \ldots, u_k^{m-1}),$$

where $u_k^1 = \mu_k^1(x_k)$ and for $\ell = 2, \ldots, m$, u_k^ℓ is defined sequentially as

$$u_k^\ell = \mu_k^\ell(x_k, u_k^1, \ldots, u_k^{\ell-1}).$$

Still, however, this policy is equivalent to the policy of the original problem that applies the control components

$$\mu_k^1(x_k), \hat{\mu}_k^2(x_k), \ldots, \hat{\mu}_k^m(x_k),$$

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 (1.2) is performed for just one state at each stage. Moreover, in a different context, the increase in size of the state space can be dealt with by using function approximation, i.e., with the introduction of cost-to-go approximations

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

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).

2. MULTIAGENT ROLLOUT

Consider now the standard rollout algorithm applied to the reformulated problem shown in Fig. 1.2, with a given base policy $\pi=\{\mu_0,\ldots,\mu_{N-1}\}$, which is also a policy of the original problem [so that $\mu_k=(\mu_k^1,\ldots,\mu_k^m)$, 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 $\tilde\mu_k^\ell$, $\ell=1,\ldots,m$,

$$\tilde\mu_k=(\tilde\mu_k^1,\ldots,\tilde\mu_k^m),$$

and is obtained for all x_k according to

$$\begin{aligned}
\tilde\mu_k^1(x_k)&\in\arg\min_{u_k^1\in U_k^1(x_k)}E\Big\{g_k\big(x_k,u_k^1,\mu_k^2(x_k),\ldots,\mu_k^m(x_k),w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,u_k^1,\mu_k^2(x_k),\ldots,\mu_k^m(x_k),w_k\big)\Big)\Big\},\\
\tilde\mu_k^2(x_k)&\in\arg\min_{u_k^2\in U_k^2(x_k)}E\Big\{g_k\big(x_k,\tilde\mu_k^1(x_k),u_k^2,\ldots,\mu_k^m(x_k),w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,\tilde\mu_k^1(x_k),u_k^2,\ldots,\mu_k^m(x_k),w_k\big)\Big)\Big\},\\
&\qquad\qquad\cdots\qquad\qquad\cdots\qquad\qquad\cdots\\
\tilde\mu_k^m(x_k)&\in\arg\min_{u_k^m\in U_k^m(x_k)}E\Big\{g_k\big(x_k,\tilde\mu_k^1(x_k),\tilde\mu_k^2(x_k),\ldots,u_k^m,w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,\tilde\mu_k^1(x_k),\tilde\mu_k^2(x_k),\ldots,u_k^m,w_k\big)\Big)\Big\}.
\end{aligned}\tag{2.1}$$

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 the agent controls* $u_k^1,\ldots,u_k^m$, *with the past controls*

where for $\ell=2,\ldots,m$, $\hat\mu_k^\ell(x_k)$ is defined sequentially as

$$\hat\mu_k^\ell(x_k)=\mu_k^\ell\big(x_k,\mu_k^1(x_k),\hat\mu_k^2(x_k),\ldots,\hat\mu_k^{\ell-1}(x_k)\big).$$

determined by the rollout policy, and the future controls determined by the base policy; cf. Eq. (2.1). Assuming a maximum of n elements in the constraint sets $U_k^i(x_k)$, the computation required at each stage k is of order $O(n)$ 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 $O(nm)$ 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.

Note that the rollout policy (2.1), obtained from the reformulated problem is different from the rollout policy obtained from the original problem [cf. Eq. (1.2)]. Generally, it is unclear how the two rollout policies perform relative to each other in terms of attained cost. On the other hand, both rollout policies perform no worse than the base policy, since the performance of the base policy is identical for both the reformulated problem and for the original problem. This is shown formally in the following proposition.

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

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k. \tag{2.2}$$

Proof: We will show Eq. (2.2) by induction, and for simplicity, we will give the proof for the case of just two agents, i.e., $m = 2$. Clearly Eq. (2.2) holds for $k = N$, since $J_{N,\tilde{\pi}} = J_{N,\pi} = g_N$. Assuming that it holds

for index $k+1$, we have for all x_k,

$$\begin{aligned}
J_{k,\tilde{\pi}}(x_k) &= E\Big\{g_k\big(x_k,\tilde{\mu}_k^1(x_k),\tilde{\mu}_k^2(x_k),w_k\big)+J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k,\tilde{\mu}_k^1(x_k),\tilde{\mu}_k^2(x_k),w_k\big)\Big)\Big\}\\
&\le E\Big\{g_k\big(x_k,\tilde{\mu}_k^1(x_k),\tilde{\mu}_k^2(x_k),w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,\tilde{\mu}_k^1(x_k),\tilde{\mu}_k^2(x_k),w_k\big)\Big)\Big\}\\
&= \min_{u_k^2\in U_k^2(x_k)} E\Big\{g_k(x_k,\tilde{\mu}_k^1(x_k),u_k^2,w_k)+J_{k+1,\pi}\Big(f_k\big(x_k,\tilde{\mu}_k^1(x_k),u_k^2,w_k\big)\Big)\Big\}\\
&\le E\Big\{g_k\big(x_k,\tilde{\mu}_k^1(x_k),\mu_k^2(x_k),w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,\tilde{\mu}_k^1(x_k),\mu_k^2(x_k),w_k\big)\Big)\Big\} \qquad (2.3)\\
&= \min_{u_k^1\in U_k^1(x_k)} E\Big\{g_k(x_k,u_k^1,\mu_k^2(x_k),w_k)+J_{k+1,\pi}\Big(f_k\big(x_k,u_k^1,\mu_k^2(x_k),w_k\big)\Big)\Big\}\\
&\le E\Big\{g_k\big(x_k,\mu_k^1(x_k),\mu_k^2(x_k),w_k\big)+J_{k+1,\pi}\Big(f_k\big(x_k,\mu_k^1(x_k),\mu_k^2(x_k),w_k\big)\Big)\Big\}\\
&= J_{k,\pi}(x_k),
\end{aligned}$$

where in the preceding relation:

(a) The first equality is the DP 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 rollout algorithm as it pertains to agent 2.

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

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

The induction proof of the cost improvement property (2.2) is thus complete for the case $m = 2$. The proof for an arbitrary number of agents m is entirely similar. **Q.E.D.**

Note the difference in the proof argument between the all-agents-at-once and one-agent-at-a-time rollout algorithms. In the former algorithm, the second equality in Eq. (2.3) would be over both $u_k^1 \in U_k^1(x_k)$ and $u_k^2 \in U_k^2(x_k)$, and the second inequality and third equality would be eliminated. Still the proof of the cost improvement property (2.2) goes through in both cases. Note also that if the base policy were optimal, Eq. (2.3) would hold as an equality throughout for both rollout algorithms, while the rollout policy $\tilde{\pi}$ would also be optimal.

On the other hand, there is an important situation where the all-agents-at-once rollout algorithm can improve the base policy but the one-agent-at-a-time algorithm will not. This is the case where 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.† Such a policy may not be optimal, except under special conditions. Thus if the base policy is agent-by-agent-optimal, multiagent rollout will

† This is a concept that has received much attention in the theory of team optimization, known as *person-by-person optimality*. It has been studied in the context of somewhat different problems, which involve imperfect

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 practical situations, since ordinarily a base policy must be reasonably simple, readily available, and easily simulated.

Let us also note that the qualitative difference between all-agents-at-once versus one-agent-at-a-time rollout is reminiscent of the context of value iteration (VI) algorithms, which involve minimization of a Bellman equation-like expression over the control constraint. In such algorithms one may choose between Gauss-Seidel methods, where the cost of a single state (and the control at that state) is updated at a time, while taking into account the results of earlier state cost computations, and Jacobi methods, where the cost of all states is updated at once. The tradeoff between Gauss-Seidel and Jacobi methods is well-known in the VI context: generally, Gauss-Seidel methods are faster, while Jacobi methods are also valid, as well as better suited for distributed asynchronous implementation; see [BeT89], [Ber12]. Our context in this paper is quite different, however, since we are considering updates of agent controls, and not cost updates at different states.

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

Example 2.1 (Spiders and Fly)

Here there are m spiders and one fly moving on a 2-dimensional grid. During each time period the fly moves to a some other position according to a given state-dependent probability distribution. Each spider learns the current state (the vector of spiders and fly locations) at the beginning of each time period, and either moves to a neighboring location or stays where it is. Thus each spider i has as many as five choices at each time period. The control vector is $u = (u^1, \ldots, u^m)$, where u^i is the choice of the ith 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. 1.2, 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 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. The problem can be made more complicated by introducing terrain obstacles, travel costs, or multiple flies.

Sample computations with this example indicate that the multiagent rollout algorithm of this section

state information that may not be shared by all the agents; see Marschak [Mar55], Radner [Rad62], Witsenhausen [Wit71], Ho [Ho80], and for some recent works, see Nayyar, Mahajan, and Teneketzis [NMT13], Nayyar and Teneketzis [NaT19], Li et al. [LTZ19], and the references quoted there.

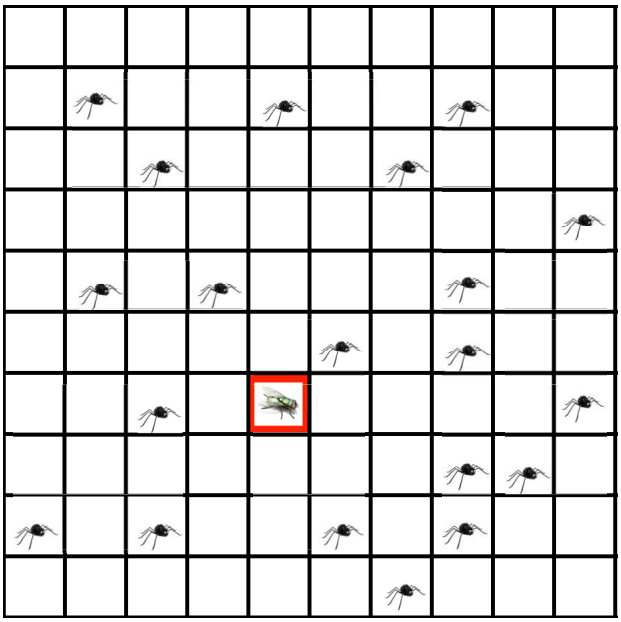

Figure 2.1 Illustration of the 2-dimensional spiders-and-fly problem. The state is the vector of distances between spiders and fly. At each time period, each spider moves to a neighboring location or stays where it is. The spiders make moves with perfect knowledge of the locations or each other and of the fly. The fly moves randomly, regardless of the position of the spiders.

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.

The following example is similar to the preceding one, but involves two spiders and two flies moving along a line, and admits an exact solution. It illustrates analytically how the multiagent rollout policy may exhibit intelligence and agent coordination that is totally lacking from the base policy. The behavior described in the example has been supported by computational experiments with larger two-dimensional problems of the type described in the preceding example.

Example 2.2 (Spiders and Flies)

There are two spiders and two flies moving along integer locations on a straight line. For simplicity we will assume that the flies' positions are fixed at some integer locations, although the problem is qualitatively similar when the flies move randomly. The spiders have the option of moving either left or right by one unit; see Fig. 2.2. The objective is to minimize the time to capture both flies. 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

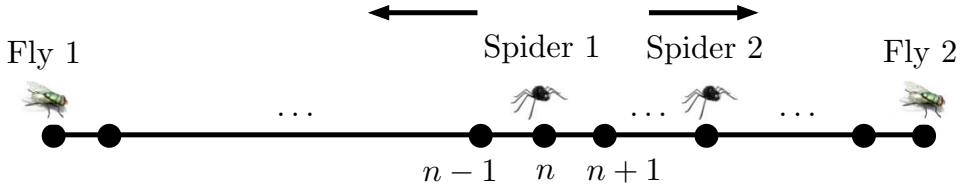

Figure 2.2 Illustration of the two-spiders and two-flies problem. The spiders move along integer points of a line. The two flies stay still at some integer locations. The optimal policy is to move the two spiders towards different flies, the ones that are initially closest to them.

Multiagent rollout with the given base policy starts with spider 1 at location n, and calculates the two Q-factors of 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 spiders 1 and 2 towards flies 1 and 2, respectively, then spider 2 first captures fly 2, and then spider 1 captures fly 1.

to them (with ties broken arbitrarily). The minimal time to capture is the sum of the initial distances of the two spider-fly pairs.

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. 2.2. 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 heuristic to obtain the next move of spider 2, and 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 heuristic will do*; see Fig. 2.2. 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 happen for both standard and agent-by-agent 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 in can benefit from control calculations that reach far into future stages.

3. ROLLOUT VARIANTS FOR FINITE HORIZON PROBLEMS

It is worth noting a few variants of the rollout algorithm for the reformulated problem.

(a) Instead of selecting the agent controls in a fixed order, it is possible to change the order at each stage k (the preceding cost improvement proof goes through again by induction). In fact it is possible to optimize over multiple orders at the same stage, or to base the order selection on various features of the state x.

(b) The algorithm can be applied to a partial state information problem (POMDP), after it has been transformed to a perfect state information problem, using a belief state formulation, where the conditional probability distribution of the state given the available information plays the role of x_k (note here that we have allowed the state space to be infinite, thereby making our methodology applicable to the POMDP/belief state formulation).

(c) We may use rollout variants involving multistep lookahead, truncated rollout, and terminal cost function approximation, in the manner described in the RL book [Ber19]. Of course, in such variants the cost improvement property need not hold strictly, but it holds within error bounds, some of which are given in [Ber19], Section 5.1, for the infinite horizon discounted problem case.

We may also consider multiagent rollout algorithms that are asynchronous in the sense that the agents may compute their rollout controls in parallel or in some irregular order rather than in sequence, and they may also communicate these controls asynchronously with some delays. Algorithms of this type are discussed in generality in the book [BeT89], and also in the papers [BeY10], [BeY12], [YuB13], within the present DP context [see also the books [Ber12] (Section 2.6), and [Ber18] (Section 2.6)]. 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. While such an algorithm is likely to work well for many problems, it may not possess the cost improvement property. In fact we can construct a simple example involving a single state, two agents, and two controls per agent, where the second agent does not take into account the control applied by the first agent, and as a result the rollout policy performs worse than the base policy.

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

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

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

The difficulty in the preceding example stems from inadequate coordination between the two agents. In particular, the second agent uses rollout to compute his/her local control, while thinking that the other agent 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 also raises serious questions as to whether the cost improvement property (2.2) 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. Still, however, such an algorithm is computationally attractive in view of its potential for efficient distributed implementation, and may be worth considering in a practical setting. A noteworthy property of this algorithm is that if the base policy is optimal, the same is true of the rollout policy. This suggests that if the base policy is nearly optimal, the same is true of the rollout policy. One may also speculate that if agents are naturally "weakly coupled" in the sense that their choice of control has little impact in the desirability of various controls of other agents, then a more flexible inter-agent communication pattern may be sufficient for cost improvement.† A computational comparison of various multiagent rollout algorithms with flexible communication patterns may shed some light on this question. The key question is whether and under what circumstances agent coordination is essential, i.e., there is a significant performance

† 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. 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.

loss when the computations of different agents are done to some extent concurrently rather than sequentially with intermediate information exchange.

4. MULTIAGENT PROBLEM FORMULATION - INFINITE HORIZON DISCOUNTED PROBLEMS

The multiagent rollout ideas that we have discussed so far can be modified and generalized to apply to infinite horizon problems. In this context, we may also consider policy iteration (PI for short) methods, which generate a sequence of policies $\{\pi^k\}$, and can be viewed as repeated or perpetual rollout, i.e., π^{k+1} is the rollout policy obtained when π^k is the base policy. We will focus on discounted problems with a bounded cost per stage so that the Bellman operator is a contraction mapping, and the strongest version of the available theory applies (the solution of Bellman's equation is unique, and strong convergence results hold for value and policy iteration algorithms); see [Ber12], Chapters 1 and 2, and [Ber18], Chapter 2. However, a qualitatively similar methodology can be applied to undiscounted problems involving a termination state (e.g., stochastic shortest path problems, see [Ber12], Chapter 3, and [Ber18], Chapters 3 and 4).

In particular, we consider a stationary infinite horizon α-discounted version of the finite horizon m-agent problem of Section 1.2, where $m > 1$, $\alpha \in (0, 1)$ is the discount factor, and the control has the form

$$u = (u^1, \ldots, u^m).$$

The component u^ℓ, $\ell = 1, \ldots, m$, is chosen by agent ℓ, from within a given state-dependent constraint set $U^\ell(x)$. At state x and stage k, a control u is applied, and the system transitions to a next state $f(x, u, w)$ at a cost $\alpha^k g(x, u, w)$, where w a random disturbance with known distribution that depends on (x, u). The cost per stage function g is uniformly bounded over the range of values of (x, u, w), and stationarity of the system equation and the cost per stage is assumed throughout.

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

$$x, (x, u^1), \ldots, (x, u^1, \ldots, u^{m-1}),$$

where x ranges over the original state space, 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 $f(x, u, w)$ at a cost $g(x, u, w)$, appropriately discounted.

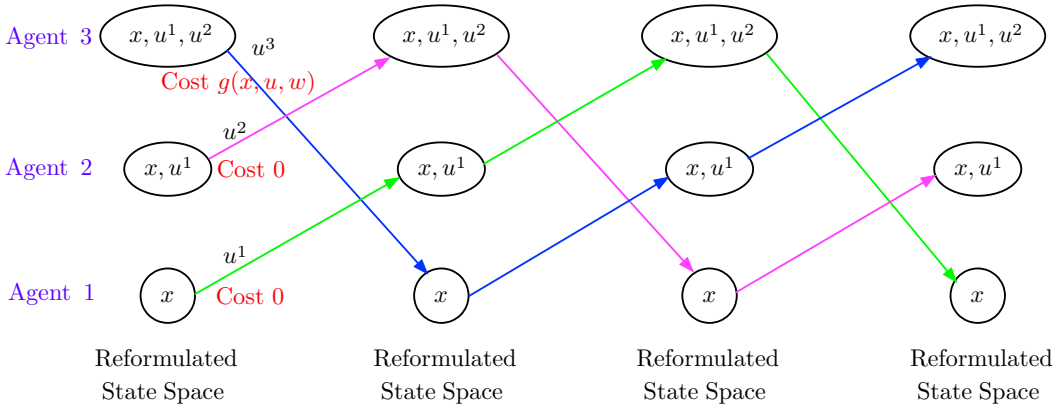

Figure 4.1 Illustration of how to transform an m-agent infinite horizon problem into a stationary infinite horizon problem with fewer control choices available at each state (in this figure $m = 3$). At the typical stage and state x, the first agent chooses u^1 at no cost leading to state (x, u^1). Then the second agent applies u^2 at no cost leading to state (x, u^1, u^2). Finally, the third agent applies u^3 at cost $g(x, u, w)$ leading to state $f(x, u, w)$, 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 this reformulation involves the type of tradeoff between control space complexity and state space complexity that we discussed in Section 1.3. The reformulated problem involves m cost-to-go functions

$$J^0(x), J^1(x, u^1), \ldots, J^{m-1}(x, u^1, \ldots, u^{m-1}), \tag{4.1}$$

with corresponding sets of Bellman equations, but a much smaller control space. Moreover, the existing analysis of rollout algorithms, including implementations, variations, and error bounds, applies to the reformulated problem; see Section 5.1 of the author's RL textbook [Ber19]. Similar to the finite horizon case, the implementation of the rollout algorithm involves one-agent-at-a-time policy improvement, and is much more economical for the reformulated problem, while maintaining the basic cost improvement and error bound properties of rollout, as they apply to the reformulated problem.

We may also consider approximate PI algorithms and variations for the reformulated problem. These variations are simply adaptations of existing algorithms for the single agent case, and include optimistic versions, Q-learning versions, distributed asynchronous versions, and actor-critic versions, which involve approximations in value space of the m cost functions (4.1), as well as approximation in policy space of the m agent policies; see the book [Ber19] for more detailed discussions. The common salient characteristic of these PI algorithms is a one-agent-at-a-time policy improvement step, with a potentially more efficient implementation resulting. The issues relating to parallelization of the policy improvement (or rollout) step that we discussed at the end of Section 3 for finite horizon problems, also apply to infinite horizon problems.

Regarding the policy evaluation step of PI, the natural partition of the state space illustrated in Fig. 4.1 suggests a distributed implementation (which may be independent of any parallelization in the policy

improvement step). In particular, distributed asynchronous PI algorithms based on state space partitions are proposed and analyzed in the work of Bertsekas and Yu [BeY10] [see also [BeY12], [YuB13], and the books [Ber12] (Section 2.6), and [Ber18] (Section 2.6)]. These algorithms are relevant for distributed implementation of the multiagent PI ideas of the present paper.

Note, however, that the policy evaluation step requires the calculation of m cost-to-go functions of the form (4.1), so the policy evaluation must be done over a much larger space than the original state space. Moreover, the policies generated by this algorithm are also defined over a larger space and have the form

$$\mu^0(x), \mu^1(x, u^1), \ldots, \mu^{m-1}(x, u^1, \ldots, u^{m-1}). \tag{4.2}$$

Motivated by this fact, we may consider a multiagent PI algorithm that operates over the simpler class of policies of the form

$$\mu(x) = \big(\mu^0(x), \mu^1(x), \ldots, \mu^{m-1}(x)\big), \tag{4.3}$$

i.e., the policies for the original infinite horizon problem. We describe such an algorithm for finite-state discounted Markovian decision problems in the next section.

5. ONE-AGENT-AT-A-TIME POLICY ITERATION FOR INFINITE HORIZON DISCOUNTED PROBLEMS

In this section, we will focus on the standard discounted Markovian decision problem (MDP for short) involving n states, denoted $1, \ldots, n$, and a finite number of controls u. Given that we restrict ourselves to finite-state problems, we will adopt in this section a more suitable notation, whereby states are denoted by i and j, the transition probability from i to j under control u is denoted $p_{ij}(u)$, and the corresponding transition cost is denoted $g(i, u, j)$.

The control u consists of m components u_ℓ, $\ell = 1, \ldots, m$ (with the MDP notation adopted for this section, it will be convenient for us to switch to subscript indexing for control components). Each control component u_ℓ is separately constrained to lie in a given finite set $U_\ell(i)$ when the system is at state i. Thus the control constraint is $u \in U(i)$, where $U(i)$ is the finite Cartesian product set

$$U(i) = U_1(i) \times \cdots \times U_m(i).$$

When at stage k the transition cost is discounted by α^k, where $\alpha \in (0, 1)$ is the discount factor. The cost function of a stationary policy μ that applies control $\mu(i) \in U(i)$ at state i is denoted by $J_\mu(i)$, and the optimal cost function [the minimum over μ of $J_\mu(i)$] is denoted $J^*(i)$.

To introduce the PI algorithm and state our results, we introduce the Bellman equation operator T, which maps a cost function/n-dimensional vector $J = \big(J(1), \ldots, J(n)\big)$ to the n-dimensional vector

$TJ = \big((TJ)(1), \ldots, (TJ)(n)\big)$ according to

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J(j)\big), \qquad i = 1, \ldots, n, \tag{5.1}$$

and for each policy μ, the corresponding Bellman equation operator T_μ defined by

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J(j)\Big), \qquad i = 1, \ldots, n. \tag{5.2}$$

It is well known that T and T_μ are contraction mappings of modulus α with respect to the sup norm, and their unique fixed points are J^* and J_μ, respectively, i.e., $J^* = TJ^*$ and $J_\mu = T_\mu J_\mu$.

In the preceding expressions, we will often expand u to write it in terms of its components. In particular, we may write $p_{ij}(u_1, \ldots, u_m)$ and $g(i, u_1, \ldots, u_m, j)$ in place of $p_{ij}(u)$ and $g(i,u,j)$, respectively. Similarly, we will denote the components of a policy μ as $\mu_1, \ldots, \mu_m$.

The Standard Policy Iteration Algorithm

For each policy μ, we introduce the subset of policies

$$\mathcal{M}_\mu = \{\tilde{\mu} \mid T_{\tilde{\mu}} J_\mu = TJ_\mu\}. \tag{5.3}$$

Equivalently, we have $\tilde{\mu} \in \mathcal{M}_\mu$ if $\tilde{\mu}$ is obtained from μ using the standard policy improvement operation,

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu(j)\big), \qquad i = 1, \ldots, n, \tag{5.4}$$

or

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} Q_\mu(i,u), \qquad i = 1, \ldots, n, \tag{5.5}$$

where $Q_\mu(i,u)$ is the Q-factor of the state-control pair (i,u) corresponding to μ, given by

$$Q_\mu(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu(j)\big). \tag{5.6}$$

The standard form of PI generates a sequence $\{\mu^k\}$ of policies, starting from a given policy μ^0 (see e.g., [Ber12]). Given the current policy μ^k, it generates a new policy from the set of "improved" policies $\mathcal{M}_{\mu^k}$ of Eq. (5.3):

$$\mu^{k+1} \in \mathcal{M}_{\mu^k}. \tag{5.7}$$

Thus the kth iteration of the standard PI algorithm can be separated into two phases:

(a) *Policy evaluation*, which computes J_{μ^k}.

(b) *Policy improvement*, which computes a new policy $\mu^{k+1} \in \mathcal{M}_{\mu^k}$ by the minimization over $u \in U(i)$ of the Q-factor $Q_{\mu^k}(i,u)$; cf. Eq. (5.4).

The Multiagent Policy Iteration Algorithm

Our proposed one-agent-at-a-time PI algorithm uses a modified form of policy improvement, whereby the control $u = (u_1, \ldots, u_m)$ is optimized one-component-at-a-time, similar to Section 2. In particular, given the current policy μ^k, the next policy is obtained as

$$\mu^{k+1} \in \widehat{\mathcal{M}}_{\mu^k}, \tag{5.8}$$

where for a given μ, we denote by $\widehat{\mathcal{M}}_\mu$ the set of policies $\tilde{\mu} = (\tilde{\mu}_1, \ldots, \tilde{\mu}_m)$ satisfying for all $i = 1, \ldots, n$,

$$\tilde{\mu}_1(i) \in \arg\min_{u_1 \in U_1(i)} \sum_{j=1}^{n} p_{ij}\big(u_1, \mu_2(i), \ldots, \mu_m(i)\big)\Big(g\big(i, u_1, \mu_2(i), \ldots, \mu_m(i), j\big) + \alpha J_\mu(j)\Big),$$

$$\tilde{\mu}_2(i) \in \arg\min_{u_2 \in U_2(i)} \sum_{j=1}^{n} p_{ij}\big(\tilde{\mu}_1(i), u_2, \mu_3(i), \ldots, \mu_m(i)\big)\Big(g\big(i, \tilde{\mu}_1(i), u_2, \mu_3(i), \ldots, \mu_m(i), j\big) + \alpha J_\mu(j)\Big),$$

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

$$\tilde{\mu}_m(i) \in \arg\min_{u_m \in U_m(i)} \sum_{j=1}^{n} p_{ij}\big(\tilde{\mu}_1(i), \tilde{\mu}_2(i), \ldots, \tilde{\mu}_{m-1}(i), u_m\big)\Big(g\big(i, \tilde{\mu}_1(i), \tilde{\mu}_2(i), \ldots, \tilde{\mu}_{m-1}(i), u_m, j\big) + \alpha J_\mu(j)\Big). \tag{5.9}$$

Note that each of the m minimizations (5.9) can be performed for each state i independently, i.e., the computations for state i 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 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 the ordering of the components is not important, and it may change from one policy improvement operation to the next.

Similar to the finite horizon case of Sections 2 and 3, the salient feature of the one-agent-at-a-time policy improvement operation (5.9) 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 $u_1, ..., u_m$. 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 $U_\ell(i)$ is bounded by a number q, the modified policy improvement phase requires at most qmn calculations of a Q-factor of the generic form (5.6). By contrast, since the number of elements in the constraint set $U(i)$ is bounded by q^m, the corresponding number of Q-factor calculations in the standard policy improvement is bounded by $q^m n$. Thus the one-agent-at-a-time policy improvement where the number of Q-factors grows linearly with m, as opposed to the standard policy improvement, which requires a number of Q-factor calculations that grows exponentially with m.

We say that a policy $\mu = \{\mu_1, \ldots, \mu_m\}$ is *agent-by-agent optimal* if it satisfies $\mu \in \widehat{\mathcal{M}}_\mu$, or equivalently, for all $i = 1, \ldots, n$, and $\ell = 1, \ldots, m$, we have

$$\sum_{j=1}^{n} p_{ij}\big(\mu_1(i), \ldots, \mu_m(i)\big)\Big(g\big(i, \mu_1(i), \ldots, \mu_m(i), j\big) + \alpha J_\mu(j)\Big)$$

$$= \min_{u_\ell \in U_\ell(i)} \sum_{j=1}^{n} p_{ij}\big(\mu_1(i), \ldots, \mu_{\ell-1}(i), u_\ell, \mu_{\ell+1}(j), \ldots, \mu_m(j)\big)$$

$$\cdot \Big(g\big(\mu_1(j), \ldots, \mu_{\ell-1}(j), u_\ell, \mu_{\ell+1}(j), \ldots, \mu_m(j)\big) + \alpha J_\mu(j)\Big).$$

Our main result is that the one-agent-at-a-time PI algorithm just described converges to an agent-by-agent optimal policy in a finite number of iterations (in the terminology of team theory such a policy may also be called "person-by-person optimal"). For the proof, we 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.

Proposition 5.1: Let $\{\mu^k\}$ be a sequence generated by the one-agent-at-a-time PI algorithm (5.8) assuming that ties in the policy improvement operation of Eq. (5.9) are broken as follows: If for any $\ell = 1, \ldots, m$ and i, the control component $\mu_\ell(i)$ attains the minimum in Eq. (5.9), we choose $\tilde{\mu}_\ell(i) = \mu_\ell(i)$ [even if there are other control components within $U_\ell(i)$ that attain the minimum in addition to $\mu_\ell(i)$]. Then for all i and k, we have $J_{\mu^{k+1}}(i) \leq J_{\mu^k}(i)$, 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: 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 $J \leq J'$ if $J(i) = J'(i)$ for all i. The critical step of the proof is the following monotone decrease inequality:

$$T_{\tilde{\mu}} J \leq T_\mu J \leq J, \qquad \text{for all } \tilde{\mu} \in \widehat{\mathcal{M}}_\mu \text{ and } J \text{ with } T_\mu J \leq J, \tag{5.10}$$

which yields as a special case $T_{\tilde{\mu}} J_\mu \leq J_\mu$, since $T_\mu J_\mu = J_\mu$. This parallels a key inequality for standard PI, namely that $T_{\tilde{\mu}} J_\mu \leq J_\mu$, for all $\tilde{\mu} \in \mathcal{M}_\mu$, which lies at the heart of its convergence proof. Once Eq. (5.10) is shown, the monotonicity of the operator $T_{\tilde{\mu}}$ implies the cost improvement property $J_{\tilde{\mu}} \leq J_\mu$, and by using the finiteness of the set of policies, the finite convergence of the algorithm will follow.

We will give the proof of the monotone decrease inequality (5.10) for the case $m = 2$. The proof for an arbitrary number of components $m > 2$ is entirely similar. Indeed, if $T_\mu J \leq J$, we have for all states i,

$$\begin{aligned}
(T_{\tilde{\mu}} J)(i) &= \sum_{j=1}^{n} p_{ij}\big(\tilde{\mu}_1(i), \tilde{\mu}_2(i)\big)\Big(g\big(i, \tilde{\mu}_1(i), \tilde{\mu}_2(i), j\big) + \alpha J(j)\Big) \\
&= \min_{u_2 \in U_2(i)} \sum_{j=1}^{n} p_{ij}\big(\tilde{\mu}_1(i), u_2\big)\Big(g\big(i, \tilde{\mu}_1(i), u_2, j\big) + \alpha J(j)\Big) \\
&\leq \sum_{j=1}^{n} p_{ij}\big(\tilde{\mu}_1(i), \mu_2(i)\big)\Big(g\big(i, \tilde{\mu}_1(i), \mu_2(i), j\big) + \alpha J(j)\Big) \\
&= \min_{u_1 \in U_1(i)} \sum_{j=1}^{n} p_{ij}\big(u_1, \mu_2(i)\big)\Big(g\big(i, u_1, \mu_2(i), j\big) + \alpha J(j)\Big) \\
&\leq \sum_{j=1}^{n} p_{ij}\big(\mu_1(i), \mu_2(i)\big)\Big(g\big(i, \mu_1(i), \mu_2(i), j\big) + \alpha J(j)\Big) \\
&= (T_\mu J)(i) \\
&\leq J(i),
\end{aligned} \tag{5.11}$$

where:

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

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

(3) The last equality is the Bellman equation for policy μ.

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

By letting $J = J_{\mu^k}$ in the monotone decrease inequality (5.10), we have $T_{\mu^{k+1}} J_{\mu^k} \leq J_{\mu^k}$. In view of the monotonicity of $T_{\mu^{k+1}}$, we also have $T^{\ell+1}_{\mu^{k+1}} J_{\mu^k} \leq T^{\ell}_{\mu^{k+1}} J_{\mu^k}$ for all $\ell \geq 1$, so that

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

It follows that either $J_{\mu^{k+1}} = J_{\mu^k}$, or else we have strict policy improvement, i.e., $J_{\mu^{k+1}}(i) < J_{\mu^k}(i)$ for at least one state i. 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. (5.11) when $\mu = \mu^k$, $\tilde{\mu} = \mu^{k+1}$, and $J = J_{\mu^k}$. This implies that

$$\begin{aligned}
&\sum_{j=1}^{n} p_{ij}\big(\mu_1^{k+1}(i), \mu_2^{k+1}(i)\big)\Big(g\big(i, \mu_1^{k+1}(i), \mu_2^{k+1}(i), j\big) + \alpha J_{\mu^k}(j)\Big) \\
&\qquad = \min_{u_2 \in U_2(i)} \sum_{j=1}^{n} p_{ij}\big(\mu_1^{k+1}(i), u_2\big)\Big(g\big(i, \mu_1^{k+1}(i), u_2, j\big) + \alpha J_{\mu^k}(j)\Big) \\
&\qquad = \sum_{j=1}^{n} p_{ij}\big(\mu_1^{k+1}(i), \mu_2^{k}(i)\big)\Big(g\big(i, \mu_1^{k+1}(i), \mu_2^{k}(i), j\big) + \alpha J_{\mu^k}(j)\Big),
\end{aligned} \tag{5.12}$$

and

$$\sum_{j=1}^{n} p_{ij}\big(\mu_1^{k+1}(i),\mu_2^k(i)\big)\Big(g\big(i,\mu_1^{k+1}(i),\mu_2^k(i),j\big)+\alpha J_{\mu^k}(j)\Big)$$

$$=\min_{u_1\in U_1(i)}\sum_{j=1}^{n} p_{ij}\big(u_1,\mu_2^k(i)\big)\Big(g\big(i,u_1,\mu_2^k(i),j\big)+\alpha J_{\mu^k}(j)\Big) \tag{5.13}$$

$$=\sum_{j=1}^{n} p_{ij}\big(\mu_1^k(i),\mu_2^k(i)\big)\Big(g\big(i,\mu_1^k(i),\mu_2^k(i),j\big)+\alpha J_{\mu^k}(j)\Big).$$

In view of the normalization rule, Eq. (5.13) implies that $\mu_1^{k+1}=\mu_1^k$, and then Eq. (5.12) implies that $\mu_2^{k+1}=\mu_2^k$. Thus we have $\mu^{k+1}=\mu^k$, and from Eqs. (5.12) and (5.13), are agent-by-agent optimal. **Q.E.D.**

The line of proof based on the monotone decrease inequality (5.10) given above can be used to establish the validity of some variants of one-agent-at-a-time PI. One such variant, which we will not pursue further, enlarges the set $\widehat{\mathcal{M}}_\mu$ to allow approximate minimization over the control components in Eq. (5.9). In particular, we require that in place of Eq. (5.11), each control $\tilde\mu_\ell(i)$, $\ell=1,\ldots,m$, satisfies

$$\sum_{j=1}^{n} p_{ij}\big(\tilde\mu_1(i),\ldots,\tilde\mu_{\ell-1}(i),\tilde\mu_\ell(i),\mu_{\ell+1}(i),\ldots,\mu_m(i)\big)$$

$$\Big(g\big(i,\tilde\mu_1(i),\ldots,\tilde\mu_{\ell-1}(i),\tilde\mu_\ell(i),\mu_{\ell+1}(i),\ldots,\mu_m(i),j\big)+\alpha J_\mu(j)\Big)$$

$$<\sum_{j=1}^{n} p_{ij}\big(\tilde\mu_1(i),\ldots,\tilde\mu_{\ell-1}(i),\mu_\ell(i),\mu_{\ell+1}(i),\ldots,\mu_m(i)\big)$$

$$\Big(g\big(i,\tilde\mu_1(i),\ldots,\tilde\mu_{\ell-1}(i),\mu_\ell(i),\mu_{\ell+1}(i),\ldots,\mu_m(i),j\big)+\alpha J_\mu(j)\Big),$$

whenever there exists $u_\ell\in U_\ell(i)$ that can strictly reduce the corresponding minimized expression in Eq. (5.11). It can be seen that even with this approximate type of minimization over control components, the convergence proof of Prop. 5.1 still goes through.

Another important variant of one-agent-at-a-time PI is an optimistic version, whereby policy evaluation is performed by using a finite number of one-agent-at-a-time variants of value iteration. Finally, there are many possibilities for approximate one-agent-at-a-time PI versions, including the use of value and policy neural networks. In particular, the multiagent policy improvement operation (5.9) may be performed at a sample set of states i^s, $s=1,\ldots,q$, thus yielding a training set of state-rollout control pairs $\big(i^s,\tilde\mu(i^s)\big)$, $s=1,\ldots,q$, which can be used to train a (policy) neural network to generate an approximation $\hat\mu$ to the policy $\tilde\mu$. The policy $\hat\mu$ can be used in turn to train a (value) neural network that approximates its cost function value $J_{\hat\mu}$, and the approximate multiagent policy iteration cycle can be continued. Thus in this scheme, the difficulty with a large control space is overcome by one-agent-at-a-time policy improvement, while the difficulty with a large state space is overcome by training value and policy networks. A further discussion of this type of approximate schemes is beyond the scope of the present paper.

6. CONCLUDING REMARKS

We have shown that in the context of multiagent problems, a one-agent-at-a-time 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 many variations of rollout algorithms for multiagent problems, which deserve attention, despite the potential lack of strict cost improvement in the case of a suboptimal base policy that is agent-by-agent optimal. Computational tests in some practical multiagent settings will be helpful in comparatively evaluating some of these variations.

In this paper we have primarily focused on the cost improvement property, and the practically important fact that it can be achieved at a much reduced computational cost. However, it is important to keep in mind that the one-agent-at-a-time rollout algorithm is simply the standard all-agents-at-once rollout algorithm applied to the (equivalent) reformulated problem of Fig. 1.2 (or Fig. 4.1 in the infinite horizon case). As a result, all known insights, results, error bounds, and approximation techniques for standard rollout apply in suitably reformulated form.

In this paper, we have assumed that the control constraint set is finite in order to argue about the computational efficiency of the one-agent-at-a-time rollout algorithm. The algorithm itself and its cost improvement property are valid even in the case where the control constraint set is infinite, including the model predictive control context (cf. Section 2.5 of the RL book [Ber19]), and linear-quadratic problems. However, it may be hard to argue that one-agent-at-a-time rollout offers an advantage in this case.

We have also discussed a one-agent-at-a-time version of PI for infinite horizon problems, which uses one-component-at-a-time policy improvement. While this algorithm may terminate with a suboptimal policy that is agent-by-agent optimal, it may produce comparable performance to the standard PI algorithm, which however may be computationally intractable even for a moderate number of agents.

We finally mention that the idea of one-agent-at-a-time rollout also applies within the context of challenging deterministic discrete/combinatorial optimization problems, which involve constraints that couple the controls of different stages. We discuss corresponding rollout algorithms separately in the paper [Ber20].

7. REFERENCES

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., 1989. Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, NJ; republished in 1996 by Athena Scientific, Belmont, MA.

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming, Athena Scientific, Belmont, MA.

[BeY10] Bertsekas, D. P., and Yu, H., 2010. "Asynchronous Distributed Policy Iteration in Dynamic Programming," Proc. of Allerton Conf. on Communication, Control and Computing, Allerton Park, Ill, pp.

1368-1374.

[BeY12] Bertsekas, D. P., and Yu, H., 2012. "Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming," Math. of OR, Vol. 37, pp. 66-94.

[Ber12] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th edition, Athena Scientific, Belmont, MA.

[Ber17] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th edition, Athena Scientific, Belmont, MA.

[Ber18] Bertsekas, D. P., 2018. Abstract Dynamic Programming, Athena Scientific, Belmont, MA.

[Ber19] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[Ber20] Bertsekas, D. P., 2020. "Constrained Multiagent Rollout and Multidimensional Assignment with the Auction Algorithm," arXiv preprint, arXiv:2002.07407.

[Ho80] Ho, Y. C., 1980. "Team Decision Theory and Information Structures," Proceedings of the IEEE, Vol. 68, pp. 644-654.

[LTZ19] Li, Y., Tang, Y., Zhang, R., and Li, N., 2019. "Distributed Reinforcement Learning for Decentralized Linear Quadratic Control: A Derivative-Free Policy Optimization Approach," arXiv preprint arXiv:1912.09135.

[Mar55] Marschak, J., 1975. "Elements for a Theory of Teams," Management Science, Vol. 1, pp. 127-137.

[NMT13] Nayyar, A., Mahajan, A. and Teneketzis, D., 2013. "Decentralized Stochastic Control with Partial History Sharing: A Common Information Approach," IEEE Transactions on Automatic Control, Vol. 58, pp. 1644-1658.

[NaT19] Nayyar, A. and Teneketzis, D., 2019. "Common Knowledge and Sequential Team Problems," IEEE Transactions on Automatic Control, Vol. 64, pp. 5108-5115.

[Rad62] Radner, R., 1962. "Team Decision Problems," Ann. Math. Statist., Vol. 33, pp. 857-881.

[Wit71] Witsenhausen, H., 1971. "Separation of Estimation and Control for Discrete Time Systems," Proceedings of the IEEE, Vol. 59, pp. 1557-1566.

[YuB13] Yu, H., and Bertsekas, D. P., 2013. "Q-Learning and Policy Iteration Algorithms for Stochastic Shortest Path Problems," Annals of Operations Research, Vol. 208, pp. 95-132.