September 2019

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 also explore related reinforcement learning and approximate policy iteration algorithms, and we discuss how this cost improvement property is affected when we attempt to improve further the method's computational efficiency through 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.

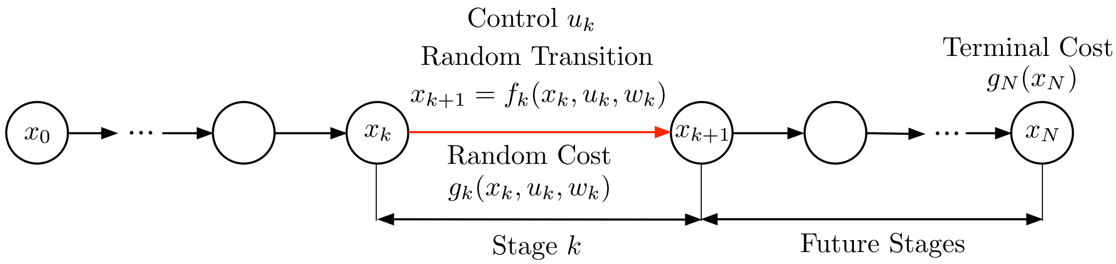

Figure 1.1 Illustration of the N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated according to a system equation

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

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 $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, 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. Standard Rollout

The aim of rollout is policy improvement. In particular, given a policy $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, called *base policy*, with cost-to-go from state x_k at stage k denoted by $J_{k,\pi}(x_k)$, $k = 0, \dots, N$, 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 [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

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

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 ([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 $u_k^1, \dots, u_k^m$,

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

with the component u_k^ℓ, $\ell = 1, \dots, m$, 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 $u_k = (u_k^1, \dots, u_k^m)$, 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.

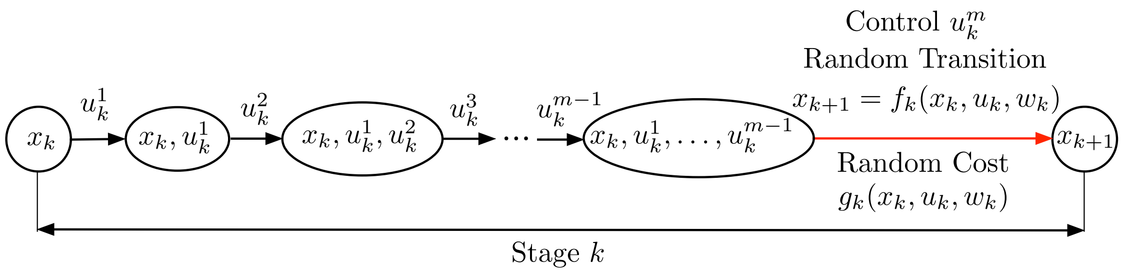

Figure 1.2 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u_k consists of m components $u_k^1, u_k^2, \ldots, u_k^m$:

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

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. Moreover, in a different context it 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 involves a minimization over only one control component at the states x_k and at the intermediate states

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

In particular, *for each stage k, the algorithm requires a sequence of m minimizations, once over each 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.* 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.

To elaborate, at $(x_k,u_k^1,\ldots,u_k^{\ell-1})$ with $\ell\le m$, 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.

---

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

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

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 obtained from the reformulated problem is different from the rollout policy obtained from the original problem. However, the former rollout algorithm is far more efficient than the latter in terms of required computation. 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.

The cost improvement property just described can also be shown analytically by induction, by modifying standard rollout cost improvement proofs (see e.g., [Ber19], Section 2.4.2). For simplicity, we give the proof for the case of just two agents, i.e., $m = 2$.

We will show the inequality

$$J_{k,\tilde{\pi}}(x_k) \le J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k, \tag{2.1}$$

by induction. Clearly it 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,

$$\begin{aligned}
J_{k,\tilde{\pi}}(x_k) &= E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&\le E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^2 \in U_k^2(x_k)} E\Big\{g_k(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k) + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k\big)\Big)\Big\} \\
&\le E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^1 \in U_k^1(x_k)} E\Big\{g_k(x_k, u_k^1, \mu_k^2(x_k), w_k) + J_{k+1,\pi}\Big(f_k\big(x_k, u_k^1, \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&\le E\Big\{g_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= J_{k,\pi}(x_k),
\end{aligned} \tag{2.2}$$

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 (2.1) is thus complete for the case $m = 2$. The proof for an arbitrary number of agents m is entirely similar.

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. (2.2) 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 (2.1) goes through in both cases. Note also that if the base policy π were an optimal policy, Eq. (2.2) would hold as an equality throughout for both rollout algorithms, while the rollout policy $\tilde{\pi}$ would also be optimal.

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 (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 preceding example is reminiscent 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 quire different, however, since we are considering updates of agent controls, and not cost updates at different states.

The simplicity of the preceding example raises serious questions as to whether the cost improvement property (2.1) 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 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 easily modified and generalized to apply to infinite horizon problems. In this context, we may also consider policy iteration 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.

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

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

The component u^ℓ, $\ell = 1, \dots, m$, 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.

† 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.

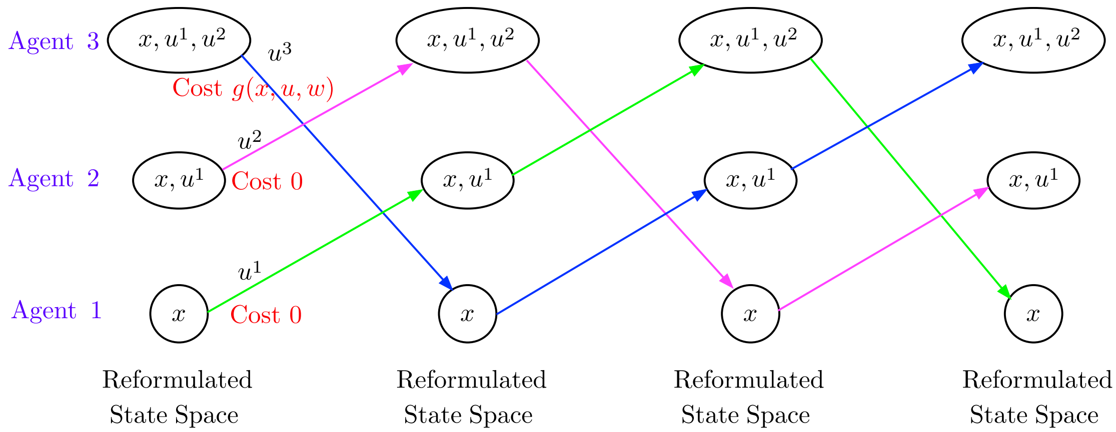

Figure 4.1 Illustration of how to transform an m-agent infinite horizon problem into a stationary infinite horizon problem with fewer control choices available at each state (in this figure $m = 3$). At the typical stage and state x, the first agent chooses u^1 at no cost leading to state (x, u^1). Then 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.

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

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

where x ranges over the original state space, and each u^ℓ, $\ell = 1, \dots, m$, 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 $(x, u^1, \dots, u^{m-1})$, agent m applies $u^m \in U^m(x)$, completing the choice of control $u = (u^1, \dots, u^m)$, and effecting the transition to state $f(x, u, w)$ at a cost $g(x, u, w)$, appropriately discounted.

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

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

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 book [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.

We may also consider approximate policy iteration 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 RL book [Ber19] for more detailed discussions. The common salient characteristic of these policy iteration algorithms is a one-agent-at-a-time policy improvement step, with a far 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 policy iteration, 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 policy iteration 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 highly relevant for distributed implementation of the multiagent policy iteration ideas of the present paper.

5. 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 failure of the cost improvement property, which has been demonstrated by our counterexample. 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, and error bounds for standard rollout apply in suitably reformulated form.

We have also 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. However, the algorithm itself and its cost improvement property are valid even in the case where the control constraint set is infinite. For example, the algorithm can form the basis for a multiagent version of policy iteration for infinite horizon linear-quadratic problems, where the successively generated rollout policies are linear functions of the state (see e.g. [Ber12], [Ber17]).

6. 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.

[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.