Constrained Multiagent Rollout and
Multidimensional Assignment with the Auction Algorithm

by

Dimitri Bertsekas†

Abstract

We consider an extension of the rollout algorithm that applies to constrained deterministic dynamic programming, including challenging combinatorial optimization problems. The algorithm relies on a suboptimal policy, called base heuristic. Under suitable assumptions, we show that if the base heuristic produces a feasible solution, the rollout algorithm has a cost improvement property: it produces a feasible solution, whose cost is no worse than the base heuristic's cost.

We then focus on multiagent problems, where the control at each stage consists of multiple components (one per agent), which are coupled either through the cost function or the constraints or both. We show that the cost improvement property is maintained with an alternative implementation that has greatly reduced computational requirements, and makes possible the use of rollout in problems with many agents. We demonstrate this alternative algorithm by applying it to layered graph problems that involve both a spatial and a temporal structure. We consider in some detail a prominent example of such problems: multidimensional assignment, where we use the auction algorithm for 2-dimensional assignment as a base heuristic. This auction algorithm is particularly well-suited for our context, because through the use of prices, it can advantageously use the solution of an assignment problem as a starting point for solving other related assignment problems, and this can greatly speed up the execution of the rollout algorithm.

---

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

1. INTRODUCTION

We consider a deterministic optimal control problem involving the system

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

where x_k is the state, taking values in some (possibly infinite) set, u_k is the control at time k, taking values in some finite set, and f_k is some function. The initial state is given and is denoted by x_0. A sequence of the form

$$T = (x_0, u_0, x_1, u_1, \ldots, u_{N-1}, x_N), \tag{1.1}$$

where

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1, \tag{1.2}$$

is referred to as a *complete trajectory*. We distinguish a complete trajectory from a *partial trajectory*, which is defined to be a subset of a complete trajectory, consisting of a subsequence of time-contiguous states and controls. Our problem is stated succinctly as

$$\min_{T \in C} G(T), \tag{1.3}$$

where G is a given real-valued cost function and C is a given constraint set of trajectories.†

As an example, we note the common special case of the additive cost

$$G(x_0, u_0, x_1, u_1, \ldots, u_{N-1}, x_N) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k), \tag{1.4}$$

where g_k, $k = 0, 1, \ldots, N$, are given real-valued functions, and the controls satisfy the time-uncoupled (but state-dependent) constraints

$$u_k \in U_k(x_k), \qquad k = 0, 1, \ldots, N-1, \tag{1.5}$$

(so here C is the set of trajectories that are generated by the system equation with controls satisfying the above constraints). This is a standard problem formulation, which is usually taken as the starting point for deterministic dynamic programming (DP for short). Our aim, however, is to address problems involving far more complicated constraints, for which the exact solution of the problem is typically intractable, including

---

† Actually it is not essential that we know the form of the function G. Instead it is sufficient to have access to a human or software expert that enables us to compare any two trajectories T_1 and T_2, without assigning numerical values to them. It is essential, however, that the expert's rankings should have a transitivity property: if T is ranked better than T' and T' is ranked better than T'', then T is ranked better than T''. Of course, the expert should also be able to determine whether a given trajectory T satisfies the constraint $T \in C$.

multiagent problems for which u_k consists of multiple components (one per agent). To this end, we will consider approximate solution methods based on the rollout approach.

The general idea of a rollout algorithm is to start with a suboptimal solution method called *base heuristic*, and to aim at *cost improvement*: a guarantee (under suitable assumptions) that the rollout algorithm produces a feasible solution, whose cost is no worse than the cost corresponding to the base heuristic. Rollout is simple and reliable, and has been used with considerable success for general unconstrained deterministic DP problems of the form (1.3)-(1.5). It has also been used in minimax/game settings, and in stochastic settings, in conjunction with simulation-based evaluation of the expected cost produced by the base heuristic starting from a given state. For applications and related work we refer to Tesauro and Galperin [TeG96], Bertsekas and Tsitsiklis [Ber96], Bertsekas, Tsitsiklis, and Wu [BTW97], Bertsekas [Ber97], Christodouleas [Chr97], Bertsekas and Castañon [BeC99], Duin and Voss [DuV99], Secomandi [Sec00], [Sec01], [Sec03], Ferris and Voelker [FeV02], [FeV04], McGovern, Moss, and Barto [MMB02], Savagaonkar, Givan, and Chong [SGC02], Guerriero and Mancini [GuM03], Tu and Pattipati [TuP03], Wu, Chong, and Givan [WCG03], Chang, Givan, and Chong [CGC04], Meloni, Pacciarelli, and Pranzo [MPP04], Yan, Diaconis, Rusmevichientong, and Van Roy [YDR04], Bertsekas [Ber05a], [Ber05b], Besse and Chaib-draa [BeC08], Sun et al. [SZL08], Bertazzi et al. [BBG13], Sun et al. [SLJ13], Tesauro et al. [TGL13], Antunes and Heemels [AnH14], Beyme and Leung [BeL14], Goodson, Thomas, and Ohlmann [GTO15], Khashooei, Antunes, and Heemels [KAH15], Li and Womer [LiW15], Mastin and Jaillet [MaJ15], Huang, Jia, and Guan [HJG16], Simroth, Holfeld, and Brunsch [SHB15], Lan, Guan, and Wu [LGW16], Ulmer [Ulm17], Bertazzi and Secomandi [BeS18], Guerriero, Di Puglia, and Macrina [GDM19], Sarkale et al. [SNC18], Ulmer at al. [UGM18], Bertsekas [Ber19c], and Chu, Xu, and Li [CXL19]. These works discuss variants and problem-specific adaptations of rollout algorithms for a broad variety of practical problems, and consistently report positive computational experience.

In this paper we will adapt the rollout approach to construct methods that can address suboptimally the constrained DP problem (1.3). Our line of analysis and development are based on the ideas of the paper by Bertsekas, Tsitsiklis, and Wu [BTW97], where rollout was applied to general discrete deterministic optimization problems. Related constrained rollout ideas are also discussed in the author's papers [Ber05a], [Ber05b]. The extension to constrained multiagent rollout (see Section 4) is based on the author's recent paper [Ber19b], which proposed a modification of the standard rollout algorithm to deal efficiently with the special computational demands of the many-agent case.

Generally constrained DP problems can be transformed to unconstrained DP problems. The idea is to redefine the state at stage k to be the partial trajectory

$$y_k = (x_0, u_0, x_1, \ldots, u_{k-1}, x_k),$$

which evolves according to a redefined system equation:

$$y_{k+1} = \big(y_k, u_k, f_k(x_k, u_k)\big). \tag{1.6}$$

The problem then becomes to find a control sequence that minimizes the terminal cost $G(y_N)$ subject to the constraint $y_N \in C$. This is a problem to which the standard form of DP applies.

Unfortunately, with the DP reformulation just described, the exact solution of the problem is typically impractical because the associated computation can be overwhelming. It is much greater than the computation for the corresponding additive cost/time-uncoupled control constraints problem (1.4)-(1.5), where the constraint $T \in C$ is absent. This is true even when C is specified in terms of a finite number of constraint functions that are time-additive, i.e., $T \in C$ if

$$g_N^m(x_N) + \sum_{k=0}^{N-1} g_k^m(x_k, u_k) \le b^m, \qquad m = 1, \ldots, M, \tag{1.7}$$

where g_k^m, $k = 0, 1, \ldots, N$, and b^m, $m = 1, \ldots, M$, are given functions and scalars, respectively. For further appreciation of the issues involved, the reader may consult the author's textbook [Ber17], and the journal literature, which contains several proposals for suboptimal solution of the problem in the case where the constraints are of the form (1.7), using among others, multiobjective optimization ideas; see e.g., Jaffe [Jaf84], Martins [Mar84], Guerriero and Musmanno [GuM01], and Stewart and White [StW91], who also survey earlier work. Generally, experience with constrained DP problems suggests that the use of an approximate solution approach is essentially unavoidable. This is the motivation for the methodology of this paper.

Using a Base Heuristic for Constrained Rollout

We will now describe our rollout algorithm. We assume the availability of a base heuristic, which for any given partial trajectory

$$y_k = (x_0, u_0, x_1, \ldots, u_{k-1}, x_k),$$

can produce a (complementary) partial trajectory†

$$R(y_k) = (x_k, u_k, x_{k+1}, u_{k+1}, \ldots, u_{N-1}, x_N),$$

that starts at x_k and satisfies for every $t = k, \ldots, N-1$ the system equation

$$x_{t+1} = f_t(x_t, u_t).$$

---

† The nature of the base heuristic is essentially arbitrary, and may strongly depend on y_k as well as k. For an extreme but practically interesting possibility, we may have a partition of the set of partial trajectories y_k, and a collection of multiple heuristics that are specially tailored to the sets of the partition. We may then select the appropriate heuristic to use on each set of the partition. Of course the properties of the rollout algorithm and its potential for cost improvement will depend on the design of the base heuristic, and in Section 2, we will discuss properties that are favorable in this regard.

Thus, given y_k and any control u_k, we can use the base heuristic to obtain a complete trajectory as follows:

(a) Generate the next state $x_{k+1} = f_k(x_k, u_k)$.

(b) Extend y_k to obtain the partial trajectory

$$y_{k+1} = \big(y_k, u_k, f_k(x_k, u_k)\big).$$

(c) Run the base heuristic from y_{k+1} to obtain the partial trajectory $R(y_{k+1})$.

(d) Join the two partial trajectories y_{k+1} and $R(y_{k+1})$ to obtain the complete trajectory $\big(y_k, u_k, R(y_{k+1})\big)$, which is denoted by $T_k(y_k, u_k)$:

$$T_k(y_k, u_k) = \big(y_k, u_k, R(y_{k+1})\big). \tag{1.8}$$

A complete trajectory $T_k(y_k, u_k)$ of the form (1.8) is generally feasible for only a subset of controls u_k, which we denote by $U_k(y_k)$:

$$U_k(y_k) = \big\{u_k \mid T_k(y_k, u_k) \in C\big\}. \tag{1.9}$$

Our rollout algorithm starts from a given initial state $\tilde{y}_0 = \tilde{x}_0$, and generates successive partial trajectories $\tilde{y}_1, \ldots, \tilde{y}_N$, of the form

$$\tilde{y}_{k+1} = \big(\tilde{y}_k, \tilde{u}_k, f_k(\tilde{x}_k, \tilde{u}_k)\big), \qquad k = 0, \ldots, N-1, \tag{1.10}$$

where $\tilde{x}_k$ is the last state component of $\tilde{y}_k$, and $\tilde{u}_k$ is a control that minimizes the heuristic cost $H\big(T_k(\tilde{y}_k, u_k)\big)$ over all u_k for which $T_k(\tilde{y}_k, u_k)$ is feasible. Thus at stage k, the algorithm forms the set $U_k(\tilde{y}_k)$ and selects from $U_k(\tilde{y}_k)$ a control $\tilde{u}_k$ that minimizes the cost of the complete trajectory $T_k(\tilde{y}_k, u_k)$:

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big); \tag{1.11}$$

see Fig. 1.1. The objective is to produce a feasible final complete trajectory $\tilde{y}_N$, which has a cost $G(\tilde{y}_N)$ that is no larger than the cost of $R(\tilde{y}_0)$ produced by the base heuristic starting from $\tilde{y}_0$, i.e.,

$$G(\tilde{y}_N) \leq G\big(R(\tilde{y}_0)\big). \tag{1.12}$$

Note that $T_k(\tilde{y}_k, u_k)$ is not guaranteed to be feasible for any given u_k (i.e., it may not belong to C), but for the analysis of Section 2, our assumptions will guarantee that the constraint set $U_k(\tilde{y}_k)$ of the problem (1.11) is nonempty, so that our rollout algorithm is well-defined. Later, in Section 3, we will modify our algorithm so that it is well-defined under the weaker assumption that just *the complete trajectory generated by the base heuristic starting from the given initial state $\tilde{y}_0$ is feasible*, i.e., $R(\tilde{y}_0) \in C$.

It can be seen that our constrained rollout algorithm is not much more complicated or computationally demanding than its unconstrained version where the constraint $T \in C$ is not present (as long as checking

$$T_k(\tilde{y}_k, u_k) = \big(\tilde{y}_k, u_k, R(y_{k+1})\big) \in C$$
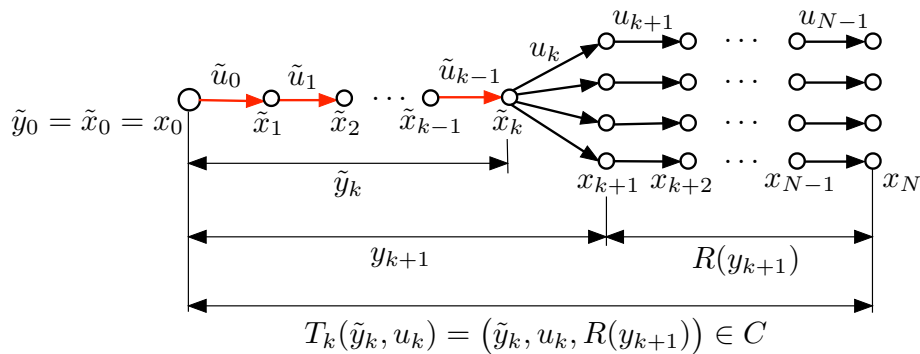
Figure 1.1. Illustration of the rollout algorithm. At stage k, and given the current partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k),$$

which starts at $\tilde{x}_0$ and ends at $\tilde{x}_k$, it considers all possible next states $x_{k+1} = f_k(\tilde{x}_k, u_k)$, and runs the base heuristic starting at $y_{k+1} = (\tilde{y}_k, u_k, x_{k+1})$. It then:

(a) Finds $\tilde{u}_k$, the control that minimizes over all u_k the cost $G\big(T_k(\tilde{y}_k, u_k)\big)$ over all u_k that yield a feasible complete trajectory $T_k(\tilde{y}_k, u_k) \in C$.

(b) Extends $\tilde{y}_k$ by $\big(\tilde{u}_k, f_k(\tilde{x}_k, \tilde{u}_k)\big)$ to form $\tilde{y}_{k+1}$.

feasibility of a complete trajectory T is not computationally demanding). Note, however, that our algorithm makes essential use of the deterministic character of the problem, and does not admit a straightforward extension to stochastic or minimax problems, since checking feasibility of a complete trajectory is typically difficult in stochastic and minimax contexts.

In the next section, we establish the cost improvement properties of our algorithm under the assumptions of sequential consistency and sequential improvement introduced in [BTW97]. In Section 3, we discuss some variants and extensions of the algorithm. In Section 4, we discuss how the algorithm can be applied to multiagent problems with computational complexity that is linear in the number of agents, rather than exponential. In Section 5, we discuss the application of our algorithm to combinatorial problems, including a broad class of layered graph problems, which contains as special cases problems of scheduling, routing, and multidimensional assignment. We will illustrate rollout primarily through the multidimensional assignment problem by using base heuristics that solve repeatedly 2-dimensional assignment problems with the auction algorithm. This algorithm is particularly well-suited to our context because it allows the efficient use of the solution of a given problem as a starting point for the solution of a related assignment problem. Detailed presentations of the auction algorithm are given in several sources, including the author's textbooks [Ber91], [Ber98], and the survey paper [Ber92]. For purposes of easy reference and completeness, we provide an appendix with a description, which is based on the author's brief tutorial paper [Ber01].

2. COST IMPROVEMENT WITH THE ROLLOUT ALGORITHM

Let us summarize the rollout algorithm (1.10)-(1.11). It starts at stage 0 and sequentially proceeds to the last stage. At stage k, it maintains a partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \dots, \tilde{u}_{k-1}, \tilde{x}_k) \tag{2.1}$$

that starts at the given initial state $\tilde{y}_0 = \tilde{x}_0$, and is such that

$$\tilde{x}_{t+1} = f_t(\tilde{x}_t, \tilde{u}_t), \qquad t = 0, 1, \dots, k-1.$$

The algorithm then forms the set of controls

$$U_k(\tilde{y}_k) = \{u_k \mid T_k(\tilde{y}_k, u_k) \in C\}$$

that is consistent with feasibility [cf. Eq. (1.9)], and chooses a control $\tilde{u}_k \in U_k(\tilde{y}_k)$ according to the minimization

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big), \tag{2.2}$$

[cf. Eq. (1.11)], where

$$T_k(\tilde{y}_k, u_k) = \Big(\tilde{y}_k, u_k, R\big(\tilde{y}_k, u_k, f_k(\tilde{x}_k, u_k)\big)\Big);$$

[cf. Eq. (1.8)]. Finally, the algorithm sets

$$\tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k), \qquad \tilde{y}_{k+1} = (\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1}), \tag{2.3}$$

[cf. Eq. (1.10)].

We will introduce conditions guaranteeing that the control set $U_k(\tilde{y}_k)$ in the minimization (2.2) is nonempty, and that the costs of the complete trajectories $T_k(\tilde{y}_k, \tilde{u}_k)$ are improving with each k in the sense that

$$G\big(T_{k+1}(\tilde{y}_{k+1}, \tilde{u}_{k+1})\big) \leq G\big(T_k(\tilde{y}_k, \tilde{u}_k)\big), \qquad k = 0, 1, \dots, N-1,$$

while at the first step of the algorithm we have

$$G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) \leq G\big(R(\tilde{y}_0)\big).$$

It will then follow that the cost improvement condition $G(\tilde{y}_N) \leq G\big(R(\tilde{y}_0)\big)$ [cf. Eq. (1.12)] holds.

Definition 2.1:   We say that the base heuristic is *sequentially consistent* if whenever it generates a partial trajectory

$$(x_k, u_k, x_{k+1}, u_{k+1}, \ldots, u_{N-1}, x_N),$$

starting from a partial trajectory y_k, it also generates the partial trajectory

$$(x_{k+1}, u_{k+1}, x_{k+2}, u_{k+2}, \ldots, u_{N-1}, x_N),$$

starting from the partial trajectory $y_{k+1} = (y_k, u_k, x_{k+1})$.

Sequentially consistent heuristics are often used in practice. For example greedy heuristics tend to be sequentially consistent. Also any policy [a sequence of feedback control functions $\mu_k(y_k)$, $k = 0, 1, \ldots, N - 1$] for the DP problem of minimizing the terminal cost $G(y_N)$ subject to the system equation $y_{k+1} = (y_k, u_k, f_k(x_k, u_k))$ and the feasibility constraint $y_N \in C$ [cf. Eq. (1.6)] can be seen to be sequentially consistent.

For a given partial trajectory y_k, let us denote by $y_k \cup R(y_k)$ the complete trajectory obtained by joining y_k with the partial trajectory generated by the base heuristic starting from y_k. Thus if $y_k = (x_0, u_0, \ldots, u_{k-1}, x_k)$ and $R(y_k) = (x_k, u_{k+1}, \ldots, u_{N-1}, x_N)$, we have

$$y_k \cup R(y_k) = (x_0, u_0, \ldots, u_{k-1}, x_k, u_{k+1}, \ldots, u_{N-1}, x_N).$$

Definition 2.2:   We say that the base heuristic is *sequentially improving* if for every k and partial trajectory y_k for which $y_k \cup R(y_k) \in C$, the set $U_k(y_k)$ is nonempty, and we have

$$G\big(y_k \cup R(y_k)\big) \geq \min_{u_k \in U_k(y_k)} G\big(T_k(y_k, u_k)\big). \tag{2.4}$$

Note that if the base heuristic is sequentially consistent, it is also sequentially improving. The reason is that for a sequentially consistent heuristic, $y_k \cup R(y_k)$ is equal to one of the trajectories contained in the set $\big\{T_k(y_k, u_k) \mid u_k \in U_k(y_k)\big\}$.

Our main result is contained in the following proposition.

Proposition 2.1: Assume that the base heuristic is sequentially improving and generates a feasible complete trajectory starting from the initial state $\tilde{y}_0 = \tilde{x}_0$, i.e., $R(\tilde{y}_0) \in C$. Then for each k, the set $U_k(\tilde{y}_k)$ is nonempty, and we have

$$G\big(R(\tilde{y}_0)\big) \geq G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) \geq G\big(T_1(\tilde{y}_1, \tilde{u}_1)\big) \geq \cdots \geq G\big(T_{N-1}(\tilde{y}_{N-1}, \tilde{u}_{N-1})\big) = G(\tilde{y}_N),$$

where

$$T_k(\tilde{y}_k, \tilde{u}_k) = \big(\tilde{y}_k, \tilde{u}_k, R(\tilde{y}_{k+1})\big);$$

cf. Eq. (1.8). In particular, the final trajectory $\tilde{y}_N$ generated by the constrained rollout algorithm is feasible and has no larger cost than the trajectory $R(\tilde{y}_0)$ generated by the base heuristic starting from the initial state.

Proof: Consider $R(\tilde{y}_0)$, the complete trajectory generated by the base heuristic starting from $\tilde{y}_0$. Since $\tilde{y}_0 \cup R(\tilde{y}_0) = R(\tilde{y}_0) \in C$ by assumption, it follows from the sequential improvement definition, that the set $U_0(\tilde{y}_0)$ is nonempty and we have

$$G\big(R(\tilde{y}_0)\big) \geq G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big),$$

[cf. Eq. (2.4)], while $T_0(\tilde{y}_0, \tilde{u}_0) \in C$.

The preceding argument can be repeated for the next stage, by replacing $\tilde{y}_0$ with $\tilde{y}_1$, and $R(\tilde{y}_0)$ with $T_0(\tilde{y}_0, \tilde{u}_0)$. Since $\tilde{y}_1 \cup R(\tilde{y}_1) = T_0(\tilde{y}_0, \tilde{u}_0) \in C$, from the sequential improvement definition, the set $U_1(\tilde{y}_1)$ is nonempty and we have

$$G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) = G\big(\tilde{y}_1 \cup R(\tilde{y}_1)\big) \geq G\big(T_1(\tilde{y}_1, \tilde{u}_1)\big),$$

[cf. Eq. (2.4)], while $T_1(\tilde{y}_1, \tilde{u}_1) \in C$. Similarly, the argument can be successively repeated for every k, to verify that $U_k(\tilde{y}_k)$ is nonempty and that $G\big(T_k(\tilde{y}_k, \tilde{u}_k)\big) \geq G\big(T_{k+1}(\tilde{y}_{k+1}, \tilde{u}_{k+1})\big)$ for all k.   **Q.E.D.**

Proposition 2.1 implies that for a base heuristic that is sequentially improving and produces a feasible initial complete trajectory, starting from the initial state $\tilde{y}_0$, the rollout algorithm generates at each stage k a feasible complete trajectory that is no worse than its predecessor in terms of cost. It follows that the algorithm produces a final complete trajectory $\tilde{y}_N = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{N-1}, \tilde{x}_N)$ that is feasible and has cost that is no larger than the cost of the initial complete trajectory produced by the base heuristic. On the other hand it is easy to construct examples where the sequential improvement condition (2.4) is violated and the cost of the solution produced by rollout is larger than the cost of the solution produced by the base heuristic starting from the initial state (see [Ber19a], Example 2.4.2).

9

3. VARIANTS AND EXTENSIONS OF THE ROLLOUT ALGORITHM

We will now discuss some variations and extensions of the rollout algorithm of Section 2.

Rollout Without the Sequential Improvement Assumption - Fortified Rollout

Let us consider the case where the sequential improvement assumption is not satisfied. Then it may happen that given the current partial trajectory $\tilde{y}_k$, the set of controls $U_k(\tilde{y}_k)$ that corresponds to feasible trajectories $T_k(\tilde{y}_k, u_k)$ [cf. Eq. (1.9)] is empty, in which case the rollout algorithm cannot extend the partial trajectory $\tilde{y}_k$ further. To bypass this difficulty, we propose a modification, called *fortified rollout algorithm*, which is patterned after an algorithm given in [BTW97] for the case of an unconstrained DP problem (see also [Ber17], Section 6.4, and [Ber19a], Section 2.4.1). For validity of this algorithm, *we require that the base heuristic generates a feasible complete trajectory $R(\tilde{y}_0)$ starting from the initial state $\tilde{y}_0$.*

The fortified rollout algorithm, in addition to the current partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k),$$

maintains a complete trajectory $\hat{T}_k$, called *tentative best trajectory*, which is feasible (i.e., $\hat{T}_k \in C$) and agrees with $\tilde{y}_k$ up to state $\tilde{x}_k$, i.e., $\hat{T}_k$ has the form

$$\hat{T}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k, \overline{u}_k, \overline{x}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N), \tag{3.1}$$

for some $\overline{u}_k, \overline{x}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N$ such that

$$\overline{x}_{k+1} = f_k(\tilde{x}_k, \overline{u}_k), \qquad \overline{x}_{t+1} = f_t(\overline{x}_t, \overline{u}_t), \quad t = k+1, \ldots, N-1.$$

Initially, $\hat{T}_0$ is the complete trajectory $R(\tilde{y}_0)$, generated by the base heuristic starting from $\tilde{y}_0$, which is assumed to be feasible. At stage k, the algorithm forms the subset $\hat{U}_k(\tilde{y}_k)$ of controls $u_k \in U_k(\tilde{y}_k)$ such that the corresponding $T_k(\tilde{y}_k, u_k)$ is not only feasible, but also has cost that is no larger than the one of the current tentative best trajectory:

$$\hat{U}_k(\tilde{y}_k) = \left\{ u_k \in U_k(\tilde{y}_k) \mid G\big(T_k(\tilde{y}_k, u_k)\big) \le G(\hat{T}_k) \right\}.$$

There are two cases to consider at state k:

(1) *The set $\hat{U}_k(\tilde{y}_k)$ is nonempty.* Then the algorithm forms the partial trajectory $\tilde{y}_{k+1} = (\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1})$, where

$$\tilde{u}_k \in \arg \min_{u_k \in \hat{U}_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big), \qquad \tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k),$$

and sets $T_k(\tilde{y}_k, \tilde{u}_k)$ as the new tentative best trajectory, i.e.,

$$\hat{T}_{k+1} = T_k(\tilde{y}_k, \tilde{u}_k).$$

(2) *The set $\hat{U}_k(\tilde{y}_k)$ is empty.* Then, the algorithm forms the partial trajectory $\tilde{y}_{k+1} = (\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1})$, where

$$\tilde{u}_k = \overline{u}_k, \qquad \tilde{x}_{k+1} = \overline{x}_{k+1},$$

and $\overline{u}_k, \overline{x}_{k+1}$ are the control and state subsequent to $\tilde{x}_k$ in the current tentative best trajectory $\hat{T}_k$ [cf. Eq. (3.1)], and leaves $\hat{T}_k$ unchanged, i.e.,

$$\hat{T}_{k+1} = \hat{T}_k.$$

It can be seen that the fortified rollout algorithm will follow the initial complete trajectory $\tilde{T}_0$, the one generated by the base heuristic starting from $\tilde{y}_0$, up to a stage $\overline{k}$ where it will discover a new feasible complete trajectory with smaller cost to replace $\tilde{T}_0$ as the tentative best trajectory. Similarly, the new tentative best trajectory $\tilde{T}_{\overline{k}}$ may be subsequently replaced by another feasible trajectory with smaller cost, etc. Note that if the base heuristic is sequentially improving, and the fortified rollout algorithm will generate the same complete trajectory as the (nonfortified) rollout algorithm given earlier, with the tentative best trajectory $\hat{T}_{k+1}$ being equal to the complete trajectory $T_k(\tilde{y}_k, \tilde{u}_k)$ for all k. The reason is that if the base heuristic is sequentially improving the controls $\tilde{u}_k$ generated by the nonfortified algorithm belong to the set $\hat{U}_k(\tilde{y}_k)$ [by Prop. 2.1, case (1) above will hold].

However, it can be verified that even when the base heuristic is not sequentially improving, the fortified rollout algorithm will generate a complete trajectory that is feasible and has cost that is no worse than the cost of the complete trajectory generated by the base heuristic starting from $\tilde{y}_0$. This is because each tentative best trajectory has a cost that is no worse than the one of its predecessor, and the initial tentative best trajectory is just the trajectory generated by the base heuristic starting from the initial condition $\tilde{y}_0$.

Tree-Based Rollout Algorithm

It is possible to improve the performance of the rollout algorithm at the expense of maintaining more than one partial trajectory. In particular, instead of the partial trajectory $\tilde{y}_k$ of Eq. (2.1), we can maintain a *tree* of partial trajectories that is rooted at $\tilde{y}_0$. These trajectories need not be of equal length, i.e., they need not have the same number of stages. At each step of the algorithm, we select a single partial trajectory from this tree, and execute the rollout algorithm's step as if this partial trajectory were the only one. Let this partial trajectory have k stages and denote it by $\tilde{y}_k$. Then we extend $\tilde{y}_k$ similar to the rollout algorithm of Section 2, with possibly multiple feasible trajectories. There is also a fortified version of this algorithm

11

where a tentative best trajectory is maintained, which is the minimum cost complete trajectory generated thus far.

The aim of the tree-based algorithm is to obtain improved performance, essentially because it can return to extend partial trajectories that were generated and temporarily abandoned at previous stages. The net result is a more flexible algorithm that is capable of examining more alternative trajectories. Note also that there is considerable freedom to select the number of partial trajectories maintained in the tree.

We finally mention a drawback of the tree-based algorithm: it is suitable for off-line computation, but it cannot be applied in an on-line context, where the rollout control selection is made after the current state becomes known as the system evolves in real-time. By contrast, the rollout algorithm of Section 2 and its fortified version are well-suited for on-line application.

Rollout Without States

It is important to note that the constrained deterministic optimal control problem of this paper is very general. In particular, it contains as a special case the fully unstructured discrete optimization problem:

$$
\begin{aligned}
& \text{minimize} \ \ G(u) \\
& \text{subject to} \ \ u \in C,
\end{aligned}
\tag{3.2}
$$

where each solution u has N components; i.e., it has the form $u = (u_0, \dots, u_{N-1})$, where N is a positive integer, C is a finite set of feasible solutions, and $G(u)$ is some cost function.† This is simply the special case of the deterministic optimal control problem where *each state x_k can only take a single value.* Then the state space for each k has a single element, and the system equation $x_{k+1} = f_k(x_k, u_k)$ is trivial and superfluous. Then in effect the partial trajectory y_k is the k-tuple $(u_0, \dots, u_{k-1})$ consisting of the first k components of a solution.

We associate such a k-tuple with the kth stage of the finite horizon DP problem shown in Fig. 3.1. In particular, for $k = 0, \dots, N-1$, we view as the states of the kth stage all the possible k-tuples $(u_0, \dots, u_{k-1})$. The initial state is some artificial state. From this state we may move to any state (u_0), with u_0 belonging to the set

$$
U_0 = \big\{ u_0 \mid \text{there exists a solution of the form } (u_0, \overline{u}_1, \dots, \overline{u}_{N-1}) \in U \big\}.
$$

† The reverse is also true, namely that any constrained deterministic optimal control problem of the form (1.1)-(1.3), can be converted to the general discrete optimization form (3.2), simply by expressing the states x_k as functions of the preceding controls $u_0, \dots, u_{k-1}$ through the system equation (1.2), and eliminating them from the cost function expression and the constraints. This abstraction of the problem may be of value in some contexts because of its inherent simplicity.
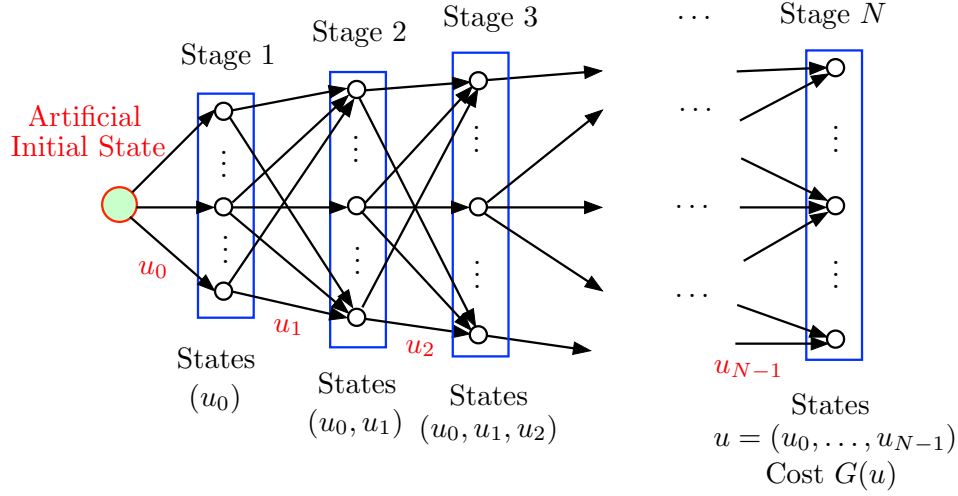
Figure 3.1. Formulation of a discrete optimization problem as a DP problem with N stages. Starting with an artificial initial state, a new control component is chosen at each stage. There is a cost $G(u)$ only at the terminal stage.

Thus U_0 is the set of choices of u_0 that are consistent with feasibility.

More generally, from a state $(u_0, \ldots, u_{k-1})$, we may move to any state of the form $(u_0, \ldots, u_{k-1}, u_k)$, such that u_k belongs to the set

$$U_k(u_0, \ldots, u_{k-1}) = \left\{ u_k \mid \text{ there exists a solution of the form } (u_0, \ldots, u_k, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}) \in U \right\}.$$

These are the set of choices of u_k that are consistent with the preceding choices $u_0, \ldots, u_{k-1}$, and are also consistent with feasibility. The last stage corresponds to the complete solutions $u = (u_0, \ldots, u_{N-1})$, with cost $G(u)$; see Fig. 3.1. All other transitions in this DP problem formulation have cost 0. Of course here the number of states typically grows exponentially with N, but we can still apply the constrained rollout algorithm to the preceding DP formulation, using a suitable base heuristic, which will be applied only N times. As an example, in Section 5 the multidimensional assignment problem will be transformed to the format of problem (3.2), prior to the application of constrained rollout.

The following example describes a treatment by rollout of a classical 0-1 integer programming problem.

Example 3.1 (Facility Location)

We are given a candidate set of N locations, and we want to place in some of these locations a "facility" that will serve the needs of M "clients." Each client $i = 1, \ldots, M$ has a demand d_i for services that may be satisfied at a location $k = 0, \ldots, N-1$ at a cost a_{ik} per unit. If a facility is placed at location k, it has capacity to serve demand up to a known level c_k.

We introduce a 0-1 integer variable u_k to indicate with $u_k = 1$ that a facility is placed at location k at a cost b_k and with $u_k = 0$ that a facility is not placed at location k. Thus if y_{ik} denotes the amount of demand

13
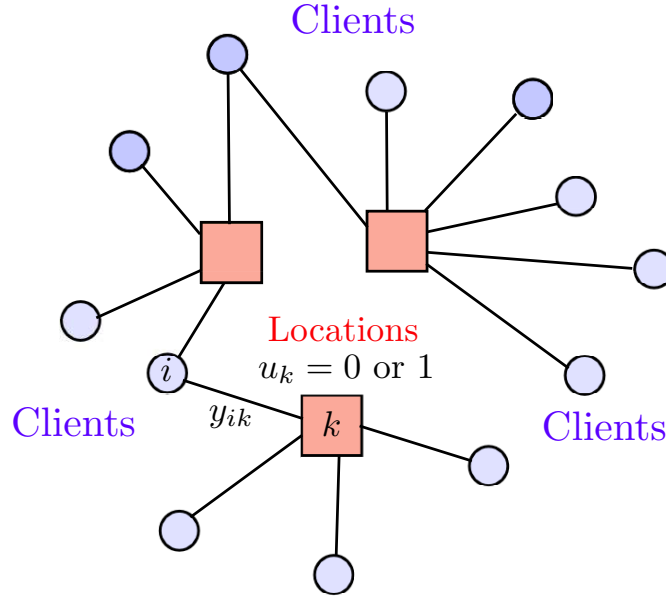
Figure 3.2. Schematic illustration of the facility location problem; cf. Example 3.1. Clients are matched to facilities, and the location of the facilities is subject to optimization.

of client i to be served at facility k, the constraints are

$$\sum_{k=0}^{N-1} y_{ik} = d_i, \qquad i = 1, \ldots, M, \tag{3.3}$$

$$\sum_{i=1}^{M} y_{ik} \leq c_j u_k, \qquad k = 0, \ldots, N-1, \tag{3.4}$$

together with

$$y_{ik} \geq 0, \quad u_k \in \{0, 1\}, \qquad i = 1, \ldots, M, \ k = 0, \ldots, N-1. \tag{3.5}$$

We wish to minimize the cost

$$\sum_{i=1}^{M} \sum_{k=0}^{N-1} a_{ik} y_{ik} + \sum_{k=0}^{N-1} b_k u_k \tag{3.6}$$

subject to the preceding constraints. The essence of the problem is to place enough facilities at favorable locations to satisfy the clients' demand at minimum cost. This can be a very difficult mixed integer programming problem.

On the other hand, when all the variables u_k are fixed at some 0 or 1 values, the problem belongs to the class of *linear transportation* problems (see e.g., [Ber98]), and can be solved by fast polynomial algorithms. Thus the essential difficulty of the problem is how to select the sequence of variables u_k, $j = 0, \ldots, N-1$. This can be viewed as a discrete optimization problem of the type shown in Fig. 3.2. In terms of the notation of this figure, the control components are $u_0, \ldots, u_{N-1}$, where u_k can take the two values 0 or 1.

To address the problem by rollout, we must define a base heuristic at a "state" $(u_0, \ldots, u_{k-1})$, where $u_j = 1$ or $u_j = 0$ specifies that a facility is or is not placed at location j, respectively. A suitable base heuristic

at that state is to place a facility at all of the remaining locations (i.e., $u_j = 1$ for $j = k+1, \ldots, N-1$), and its cost is obtained by solving the corresponding linear transportation problem of minimizing the cost (3.6) subject to the constraints (3.3)-(3.5), with the variables u_j, $j = 0, \ldots, k-1$, fixed at the previously chosen values, and the variables u_j, $j = k, \ldots, N$, fixed at 1.

To illustrate, at the initial state where no placement decision has been made, we set $u_0 = 1$ (a facility is placed at location 0) or $u_0 = 0$ (a facility is not placed at location 0), we solve the two corresponding transportation problems, and we fix u_0, depending on which of the two resulting costs is smallest. Having fixed the status of location 0, we repeat with location 1, set the variable u_1 to 1 and to 0, solve the corresponding two transportation problems, and fix u_1, depending on which of the two resulting costs is smallest, etc.

It is easily seen that if the initial base heuristic choice (placing a facility at every candidate location) yields a feasible solution, i.e.,

$$\sum_{i=1}^{M} d_i \leq \sum_{k=0}^{N-1} c_k,$$

the rollout algorithm will yield a feasible solution with cost that is no larger than the cost corresponding to the initial application of the base heuristic. In fact it can be verified that the base heuristic here is sequentially improving, so it is not necessary to use the fortified version of the algorithm. Regarding computational costs, the number of transportation problems to be solved is at first count $2N$, but it can be reduced to $N+1$ by exploiting the fact that one of the two transportation problems at each stage after the first has been solved at an earlier stage. It is finally worth noting, for readers that are familiar with the integer programming method of branch-and-bound, that the graph of Fig. 3.1 corresponds to the branch-and-bound tree for the problem, so the rollout algorithm amounts to a quick (and imperfect) method to traverse the branch-and-bound tree. This observation may be useful if we wish to use integer programming techniques to add improvements to the rollout algorithm.

We finally note that the rollout algorithm requires the solution of many linear transportation problems, with fairly similar data. It is thus important to use an algorithm that is capable of using effectively the final solution of one transportation problem as a starting point for the solution of the next. The auction algorithm for transportation problems (Bertsekas and Castañon [Ber89]) is particularly well-suited for this purpose.

4. CONSTRAINED MULTIAGENT ROLLOUT

Let us assume a special structure of the control space, where the control u_k consists of m components, $u_k = (u_k^1, \ldots, u_k^m)$, each belonging to a corresponding set U_k^ℓ, $\ell = 1, \ldots, m$. Thus the control space at stage k is the Cartesian product

$$U_k = U_k^1 \times \cdots \times U_k^m. \tag{4.1}$$

We refer to this as the *multiagent case*, motivated by the special case where each component u_k^ℓ, $\ell = 1, \ldots, m$, is chosen by a separate agent ℓ at stage k. Then the rollout minimization (2.2) involves the computation and

comparison of as many as n^m terms $G\big(T_k(\tilde{y}_k, u_k)\big)$, where n is the maximum number of elements of the sets U_k^ℓ [so that n^m is an upper bound to the number of controls in the control space U_k, in view of its Cartesian product structure (4.1)]. Thus the rollout algorithm requires order $O(n^m)$ applications of the base heuristic per stage.

In this section we construct an alternative rollout algorithm that achieves the cost improvement property (1.12) with much smaller computational cost, namely order $O(nm)$ applications of the base heuristic per stage. A key idea here is that the computational requirements of the rollout one-step minimization (2.2) are proportional to the number of controls 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 describe this idea next within the context of this paper; see [Ber19b] for a related discussion, which also considers stochastic multiagent problems and associated rollout algorithms with a cost improvement guarantee over the base heuristic.

Trading off Control Space Complexity with State Space Complexity

We noted that a major issue in rollout is the minimization over u_k

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big),$$

[cf. Eq. (2.2)], which can be very time-consuming when the size of the control space is large. In particular, in the multiagent case when $u_k = (u_k^1, \ldots, u_k^m)$, the time to perform this minimization is typically exponential in m. To deal with this, we 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 rollout.

To this end, we introduce a modified but equivalent problem, involving one-at-a-time agent control selection. In particular, at the generic state x_k, we break down the control u_k into the sequence of the m controls $u_k^1, u_k^2, \ldots, u_k^m$, and between x_k and the next state $x_{k+1} = f_k(x_k, u_k)$, 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 at cost $g_k(x_k, u_k)$ to the next state $x_{k+1} = f_k(x_k, u_k)$ according to the system equation; see Fig. 4.1.

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 structures of the two problems are essentially the same. The motivation for the reformulated problem is that the control space is simplified
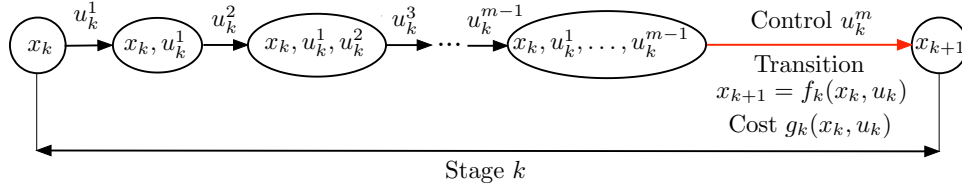
Figure 4.1. Equivalent formulation of the N-stage optimal control problem for the case where the control u_k consists of m components $u_k^1, u_k^2, \ldots, u_k^m$:

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

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 $x_{k+1} = f_k(x_k, u_k)$, and a cost $g_k(x_k, u_k)$ is incurred.

at the expense of introducing $m - 1$ additional layers of states. However, the key point is that *the increase in size of the state space does not adversely affect the operation of rollout.*

Multiagent Rollout Algorithm

Consider now the constrained rollout algorithm of Section 2 applied to the reformulated problem shown in Fig. 4.1, with a base heuristic suitably modified so that it completes a partial trajectory of the form

$$\big(y_k, (x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^\ell)\big), \qquad \ell = 1, \ldots, m.$$

The algorithm involves a minimization over only one control component at the state x_k and at each of 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, one over each of the control components $u_k^1, \ldots, u_k^m$, with the past controls already determined by the rollout algorithm, and the future controls determined by running the base heuristic.* Assuming a maximum of n elements in the control component spaces U_k^ℓ, $\ell = 1, \ldots, m$, 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, for all k and $\ell \leq m$ at the current partial trajectory

$$(\tilde{x}_0, \tilde{u}_0, \ldots, \tilde{x}_k, \tilde{u}_k^1, \ldots, \tilde{u}_k^{\ell-1}),$$

17

and for each of the controls u_k^ℓ, we use the base heuristic to generate a complementary partial trajectory

$$(u_k^{\ell+1}, \ldots, u_k^m, x^{k+1}, u^{k+1}, \ldots, x^{N-1}, u^{N-1}, x_N), \qquad (4.2)$$

up to stage N. We then select the control $\tilde{u}_k^\ell$ for which the resulting complete trajectory is feasible and has minimum cost. There is also a fortified version of this algorithm, which is similar to the one described in Section 3.

Note that the base heuristic used in the reformulated problem must be capable of generating a complementary partial trajectory of the form (4.2), starting from any partial trajectory of states and controls. Note also that instead of selecting the agent controls in a fixed order, it is possible to change the order at each stage k. In fact it is possible to optimize over multiple orders at the same stage.

5. APPLICATION TO MULTIDIMENSIONAL ASSIGNMENT

In this section we demonstrate the application of constrained multiagent rollout within the context of the classical multidimensional assignment problem. This problem is representative of a broad class of layered graph problems, which involve both a temporal and a spacial allocation structure, so that both the dynamic system character and the multiagent character of our algorithms come into play.

Multidimensional assignment problems involve graphs consisting of $N + 1$ subsets of nodes ($N \geq 2$), denoted $\mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_N$, and referred to as *layers*. The arcs of the graphs are directed and are of the form (i, j), where i is a node in a layer $\mathcal{N}_k$, $k = 0, 1, \ldots, N - 1$, and j is a node in the corresponding next layer $\mathcal{N}_{k+1}$. Thus we have a directed graph whose nodes are arranged in $N + 1$ layers and the arcs connect the nodes of each layer to the nodes in their adjacent layers; see Fig. 5.1. Here for simplicity, we assume that each of the layers $\mathcal{N}_k$ contains m nodes, and that there is a unique arc connecting each node in a given layer with each of the nodes of the adjacent layers.

We consider subsets of nodes, referred to as *groupings*, which contain a single node from every layer, and we assume that every grouping is associated with a given cost. A partition of the set of nodes into m disjoint groupings, so that each node belongs to one and only one grouping is called an $(N+1)$-*dimensional assignment*. For each grouping, there is an associated cost, which depends on the N-tuple of arcs that comprise the grouping. The cost of an $(N + 1)$-dimensional assignment is the sum of the costs of its m groupings. The problem is to find an $(N + 1)$-dimensional assignment of minimum cost.

This is a difficult combinatorial problem with many applications. An important special case arises in the context of multi-target tracking and data association; see Blackman [Bla86], Bar-Shalom and Fortman [BaF88], Bar-Shalom [Bar90], Pattipati, Deb, Bar-Shalom, and Washburn [PDB92], Poore [Poo94], Poore and Robertson [PoR97], Popp, Pattipati, and Bar-Shalom [PPB01], and Choi, Brunet, and How [CBH09]. Other
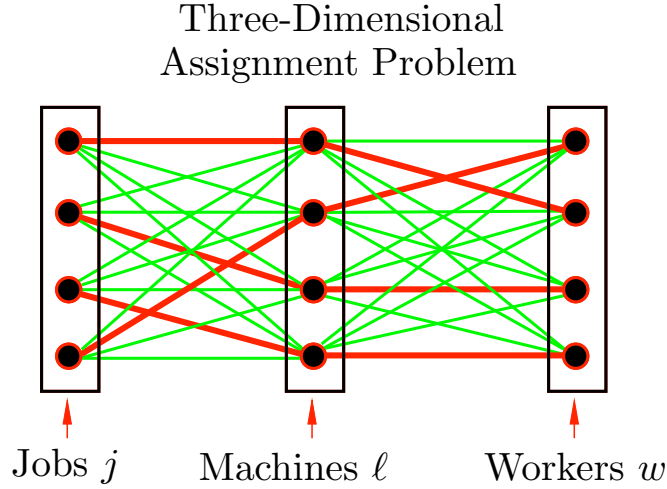
5-Dimensional Assignment Problem
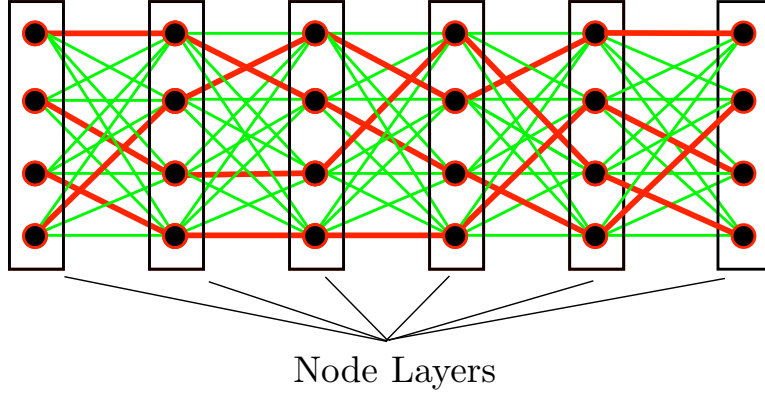
Node Layers

Figure 5.1.   Illustration of the graph of an $(N+1)$-dimensional assignment problem (here $N = 5$). There are $N + 1$ node layers each consisting of m nodes (here $m = 4$). Each grouping consists of $N+1$ nodes, one from each layer, and N corresponding arcs. An $(N+1)$-dimensional assignment consists of m node-disjoint groupings, where each node belongs to one and only one grouping (illustrated in the figure with thick red lines). For each grouping, there is an associated cost, which depends on the $(N+1)$-tuple of arcs that comprise the grouping. The cost of an $(N+1)$-dimensional assignment is the sum of the costs of its m groupings. In the separable case, the cost of a grouping separates into the sum of its N arc costs, and the problem can be solved by solving N decoupled 2-dimensional assignment problems.

challenging combinatorial problems, such as multi-vehicle routing and multi-machine scheduling problems, also share the spatio-temporal type of structure, and are thus well suited for the application of our constrained multiagent rollout approach. For more discussion of related combinatorial applications, we refer to Chapter 10 of the author's network optimization book [Ber98]. Generally, the fine details of such a problem will determine the choice of a suitable base heuristic.

We note that there are several variants of the multidimensional assignment problem illustrated in Fig. 5.1, which are well-suited for the application of constrained rollout. For example, these variants may involve unequal numbers of nodes in each layer, or a sparse structure where some of the possible arcs connecting nodes of adjacent layers are missing. Moreover, there may be cost coupling between collections of groupings that depends on the groupings' compositions. In this paper we will focus on the case where the layers have equal numbers of nodes and where the cost of each grouping depends exclusively on the $N + 1$ nodes that comprise the grouping. This structure favors the use of base heuristics that rely on solution of 2-dimensional assignment problems.

Three-Dimensional Assignment

To simplify the presentation, we will first focus on the 3-dimensional assignment special case ($N = 2$), and for descriptive purposes, we will associate the nodes of the three layers with "jobs," "machines," and "workers,"

Three-Dimensional
Assignment Problem

Figure 5.2. A 3-dimensional assignment problem consisting of assigning each job j to a machine ℓ and to a worker w at cost $a_{j\ell w}$. Each machine assigned to exactly one job and exactly one worker. Here $N = 2$ and $m = 4$.

respectively. Suppose that the performance of a job j requires a single machine ℓ and a single worker w (which cannot be shared by any other job), and that there is a given cost $a_{j\ell w}$ corresponding to the triplet (j, ℓ, w). Given a set of m jobs, a set of m machines, and a set of m workers, we want to find a collection of m job-machine-worker triplets that has minimum total cost. This problem is quite challenging, and is well-suited for demonstration of our constrained rollout approach as it has both a temporal and a spacial character.

To transcribe the problem to the optimal control format of this paper, we use the formulation of Fig. 3.1, and assume that there is only one state at each of three stages [the respective collections of jobs (for the first stage), machines (for the second stage), and workers (for the third stage)], and two decisions to make (the assignment of jobs to machines and the assignment of machines to workers); see Fig. 5.2. Each of the decisions consists of m components, the m outgoing arcs from the m nodes corresponding to the stage. Thus, the application of the multiagent rollout algorithm of Section 4 will involve two stages, a state space consisting of a single element for each state, and a control at each stage that consists of m components (the choice of machine to assign to each job in the first stage, and the choice of worker to assign to each machine in the second stage). These components are computed in sequence according to some predetermined order, which without loss of generality we will assume to be the natural order $1, 2, \ldots, m$.

An important and particularly favorable special case of the problem arises when the costs $a_{j\ell w}$ have the *separable* form

$$a_{j\ell w} = \beta_{j\ell} + \gamma_{\ell w},$$

where $\beta_{j\ell}$ and $\gamma_{\ell w}$ are given scalars. In this case, there is no coupling between jobs and workers, and

the problem can be efficiently (polynomially) solved by solving two decoupled (2-dimensional) assignment problems: one involving the pairing of jobs and machines, with the $\beta_{j\ell}$ as costs, and the other involving the pairing of machines and workers, with the $\gamma_{\ell w}$ as costs. In general, however, the 3-dimensional assignment problem is a difficult integer programming problem, for which there is no known polynomial algorithm.

The Enforced Separation Heuristic and Three-Dimensional Assignment

The separable case motivates a simple heuristic approach for the nonseparable 3-dimensional assignment problem, which is well-suited for the use of constrained rollout. The base heuristic is a two-stage procedure that is based on relaxing each the grouping constraints, by first focusing on assigning machines to workers, and then focusing on assigning jobs to machines, after suitably modifying the costs $a_{j\ell w}$ to make them separable. In particular, we first relax the constraints on the jobs by assuming that each machine-worker pair can be assigned to the most favorable job. Mathematically, this relaxed problem takes the polynomially solvable 2-dimensional assignment form

$$\text{minimize} \quad \sum_{w=1}^{m}\sum_{\ell=1}^{m} c_{\ell w} v_{\ell w}$$
$$\text{subject to} \quad \sum_{\ell=1}^{m} v_{\ell w} = 1, \qquad \forall\, w = 1, \ldots, m,$$
$$\sum_{w=1}^{m} v_{\ell w} = 1, \qquad \forall\, \ell = 1, \ldots, m,$$
$$v_{\ell w} = 0 \text{ or } 1, \qquad \forall\, \ell,\, w = 1, \ldots, m,$$

where

$$c_{\ell w} = \min_{j=1,\ldots,m} a_{j\ell w}, \tag{5.1}$$

$v_{\ell w}$ are the variables of the problem, and $v_{\ell w} = 1$ indicates that machine ℓ is assigned to worker w, so the constraints of the above problem enforce the condition that each machine is assigned to one and only one worker.†

For each w, let w_ℓ be the worker assigned to machine ℓ, according to the solution of this problem. We can now optimally assign jobs j to machine-worker pairs (ℓ, w_ℓ) by using as assignment costs

$$b_{j\ell} = a_{j\ell w_\ell},$$

---

† An alternative is to define each cost $c_{\ell w}$ as a "representative" cost $a_{j_{\ell w} \ell w}$ (for some specially selected job $j_{\ell w}$). Such an alternative may become attractive when extensions of enforced separation are considered for $(N+1)$-dimensional problems with large N. Then the analog of Eq. (5.1) will involve minimization over the (exponential in N) number of all $(N-2)$-tuples of graph arcs that can form a grouping with (ℓ, w), and may become very costly.

and obtain a (suboptimal) 3-dimensional assignment $\{(j_\ell, \ell, w_\ell) \mid \ell = 1, \ldots, m\}$. It can be seen that this approach amounts to *enforced separation*, whereby we replace the costs $a_{j\ell w}$ with the separable approximations $b_{j\ell} + c_{\ell w}$. In fact, it can be shown that if the problem is ϵ-*separable*, in the sense that for some (possibly unknown) $\overline{\beta}_{j\ell}$ and $\overline{\gamma}_{\ell w}$, and some $\epsilon \geq 0$, we have

$$|\overline{\beta}_{j\ell} + \overline{\gamma}_{\ell w} - a_{j\ell w}| \leq \epsilon, \qquad \forall \, j, \, \ell, \, w = 1, \ldots, m,$$

then the assignment $\{(j_\ell, \ell, w_\ell) \mid \ell = 1, \ldots, m\}$ obtained using the preceding enforced separation approach achieves the optimal cost of the problem within $4m\epsilon$ (see the author's network optimization book [Ber98], Exercise 10.31).

The enforced separation approach is simple and can be generalized to problems with more than two stages as we will discuss shortly. Moreover, enforced separation heuristics also apply to several variants of the multidimensional assignment problem. For example, we may have transportation-type constraints, where multiple jobs can be performed on the same machine, and/or multiple machines can be operated by a single worker. In this case, our preceding discussion of the enforced separation heuristic applies similarly, except that we need to solve 2-dimensional transportation problems rather than 2-dimensional assignment problems.

Using Enforced Separation as a Base Heuristic

We will now describe the use of enforced separation as a base heuristic in the context of constrained rollout. The 3-dimensional assignment problem is posed as an optimal control problem involving $m + 1$ sequential choices: the machines assigned to the jobs are first selected one-by-one in some fixed order (m sequential choices), and then the workers assigned to the machines are selected simultaneously. To connect with our earlier optimal control formulation, trajectories here consist of an artificial initial state, the m successive choices of job-machine pairs (these correspond to the controls $u_0^1, \ldots, u_0^m$), and then the m-tuple of machine-worker pairs (these comprise the control u_1). For each of the first m choices a job is selected and the machine to be assigned to this job is fixed by the rollout algorithm, through the use of the base heuristic of enforced separation. At the last stage the m machines are assigned simultaneously to workers using a 2-dimensional assignment algorithm.

To illustrate the fortified rollout algorithm, at the artificial initial state where no job-machine or machine-worker pair has been fixed, the enforced separation heuristic as described above is used to generate a (suboptimal) initial 3-dimensional assignment, which serves as the initial tentative best trajectory, and has cost denoted by $\hat{S}$.

In the first m rollout stages we select in sequence each job $j = 1, \ldots, m$, and we select a machine ℓ to assign to it, by using the enforced separation heuristic. The first rollout stage is as follows:

Stage 1.1: We take job $j = 1$, and fix its assignment to machine 1. We then apply the enforced separation heuristic by solving two 2-dimensional assignment problems. The first of these involves the assignment of machines to workers using as costs

$$c_{\ell w} = \begin{cases} a_{11w} & \text{if } \ell = 1, \\ \min_{j=2,\ldots,m} a_{j\ell w} & \text{if } \ell \neq 1, \end{cases}$$

[cf. Eq. (5.1)]. We thus obtain an assignment of machines to workers of the form (ℓ, w_ℓ), $\ell = 1, \ldots, m$. Having fixed the workers to be assigned to machines, we solve the 2-dimensional assignment problem of assigning the jobs $2, \ldots, m$ to the machines $2, \ldots, m$, where the costs are

$$b_{j\ell} = a_{j\ell w_\ell}, \qquad j, \ell = 2, \ldots, m.$$

By "joining" the solutions of the two 2-dimensional problems just described, we obtain a 3-dimensional assignment that consists of m job-machine-worker groupings whose cost we call S_1.

Stage 1.2: We take job $j = 1$, and fix its assignment to machine 2. We then apply the enforced separation heuristic by solving two 2-dimensional assignment problems. The first involves the assignment of machines to workers using as costs

$$c_{\ell w} = \begin{cases} a_{12w} & \text{if } \ell = 2, \\ \min_{j=2,\ldots,m} a_{j\ell w} & \text{if } \ell \neq 2, \end{cases}$$

[cf. Eq. (5.1)]. We obtain an assignment of machines to workers of the form (ℓ, w_ℓ), $\ell = 1, \ldots, w$. We then solve the 2-dimensional assignment problem of assigning the jobs $2, 3, \ldots, m$ to the machines $1, 3, \ldots, m$, where the costs are

$$b_{j\ell} = a_{j\ell w_\ell}, \qquad j = 2, 3, \ldots, m, \ \ell = 1, 3, \ldots, m.$$

By "joining" the solutions of the two 2-dimensional problems, we obtain a 3-dimensional assignment, whose cost we call S_2.

Stage 1.t: For $t = 3, \ldots, m$, we continue the process described above, where we fix the assignment of job 1 to machine t. We then apply the enforced separation heuristic by solving two 2-dimensional assignment problems, similar to the ones above: first assigning machines to workers using costs

$$c_{\ell w} = \begin{cases} a_{1tw} & \text{if } \ell = t, \\ \min_{j=2,\ldots,m} a_{j\ell w} & \text{if } \ell \neq t, \end{cases}$$

and obtaining an assignment of machines to workers of the form (ℓ, w_ℓ), $\ell = 1, \ldots, w$. We then solve the 2-dimensional assignment problem of assigning the jobs $2, \ldots, m$ to the machines $1, \ldots, t-1, t+1, \ldots, m$, where the costs are

$$b_{j\ell} = a_{j\ell w_\ell}, \qquad j = 2, 3, \ldots, m, \ \ell = 1, \ldots, t-1, t+1, \ldots, m.$$

By "joining" the solutions of the two 2-dimensional problems, we obtain a 3-dimensional assignment, whose cost we call S_t.

Stage 1: We now have m 3-dimensional assignments, with corresponding costs $S_1, \ldots, S_m$, where job 1 is fixed to the machines $1, \ldots, m$, respectively. We choose the machine $\tilde{\ell}$ for which $S_{\tilde{\ell}}$ is minimized over $\ell = 1, \ldots,$, and we permanently assign job 1 to machine $\tilde{\ell}$ if $S_{\tilde{\ell}}$ is less or equal to $\hat{S}$, the cost of the enforced separation heuristic applied to the artificial initial state (the current tentative best trajectory), and we also adopt the corresponding 3-dimensional assignment as the new tentative best trajectory. Otherwise, we set the assignment of job 1 to a machine according to the current tentative best trajectory, which we leave unchanged.

The preceding procedure, the first step of constrained rollout, required the solution of $2m$ 2-dimensional assignment problems, and yielded a permanent assignment of job 1 to a machine. The procedure is then repeated for job 2, taking into account that the assignment of job 1 to a machine has been fixed. This requires similarly the solution of $2(m-1)$ 2-dimensional assignment problems, and yields a permanent assignment of job 2 to a machine, and an update of the current tentative trajectory. Repeating the procedure with jobs $3, \ldots, m$ in sequence, we obtain a permanent assignment of all the jobs to machines, and the corresponding 3-dimensional assignment (which has minimum cost over all the 3-dimensional assignments generated, in view of the use of fortified rollout). The total number of 2-dimensional assignment problems thus solved is

$$2m + 2(m - 1) + 2(m - 2) + \cdots + 2 = m^2.$$

Finally, given the permanent assignment of all the jobs to machines, say (j_ℓ, ℓ), $\ell = 1, \ldots, m$, we obtain the (permanent) assignment of workers to job-machine pairs, by using as costs the scalars

$$c_{\ell w} = a_{j_\ell \ell w}, \qquad \ell, w = 1, \ldots, m.$$

At this point we have obtained by rollout the final suboptimal 3-dimensional assignment, which by construction has the cost improvement property: it has no larger cost than the one obtained by the enforced separation base heuristic starting from the artificial initial condition.

Thus the total number of 2-dimensional assignment problems to be solved by the rollout algorithm is $m^2 + 1$. Each of these problems can be solved very fast using any one of a number of methods. However, because these problems and their solutions are similar, it is important to use a method that can exploit this similarity. A particularly favorable method in this regard is the author's auction algorithm [Ber79] (see the book [Ber98] for a detailed development; we provide an introductory review of auction algorithms in the appendix to this paper). The auction algorithm uses a price variable for each node, such as a worker or a machine, and then adjusts the prices through an auction-like process to achieve a form of economic equilibrium. One can then use the final prices obtained for one 2-dimensional assignment problem as an

efficient starting point for the solution of a related 2-dimensional assignment problem. For this reason, the auction algorithm and its variations have been widely adopted for use in solving 2-dimensional assignment problems in the context of multidimensional assignment algorithms used for multitarget tracking applications, among others (see [PDB92], [Poo94], [PoR97], [PPB01], [CBH09]).

Enforced Separation and Constrained Rollout for Multidimensional Assignment

Let us now consider briefly the extension of the constrained rollout algorithm just described to $(N + 1)$-dimensional assignment problems with $N > 2$. Here we will use an extension of the 3-dimensional enforced separation heuristic. We start again from the last stage, solve the last 2-dimensional assignment problem of the last stage by modifying the arc costs according to the analog of the minimization formula

$$c_{\ell w} = \min_{j=1,\ldots,m} a_{j\ell w},$$

[cf. Eq. (5.1)]. The only difference is that instead of minimizing $a_{j\ell w}$ over jobs j as above, we minimize over all $(N - 1)$-tuples of nodes of the groupings whose final two nodes are (ℓ, w). Once the assignments of the last stage are fixed, a similar procedure can be used to fix the assignments of the next-to-last stage, and so on. The total number of 2-dimensional assignment problems to be solved is $(m + 1)(N - 2)$. Thus the base heuristic's computation time is polynomial in both m and N.

The enforced separation heuristic just described for the artificial initial condition, can be used in suitably modified form for rollout, with the assignments of some stages fixed permanently by rollout, and additional assignments fixed one by one, by applying the enforced separation heuristic, with suitably modified costs that take into account the already fixed assignments.

6. CONCLUDING REMARKS

We have proposed a rollout algorithm for constrained deterministic DP, which is well suited for the suboptimal solution of challenging discrete optimization problems. Under suitable assumptions, we have shown a cost improvement property: the rollout algorithm produces a feasible solution, whose cost is no worse than the cost of the solution produced by the base heuristic.

We have also proposed an efficient variant of the algorithm for multiagent problems, where the control at each stage consists of multiple components. We have shown that the cost improvement property is preserved, and we have applied the algorithm to layered graph problems that involve both a spatial and a temporal structure. In particular, we have focused on multidimensional assignment, using the auction algorithm for 2-dimensional assignment as a base heuristic.

7. REFERENCES

[AnH14] Antunes, D., and Heemels, W.P.M.H., 2014. "Rollout Event-Triggered Control: Beyond Periodic Control Performance," IEEE Transactions on Automatic Control, Vol. 59, pp. 3296-3311.

[BBG13] Bertazzi, L., Bosco, A., Guerriero, F., and Lagana, D., 2013. "A Stochastic Inventory Routing Problem with Stock-Out," Transportation Research, Part C, Vol. 27, pp. 89-107.

[BCE95] Bertsekas, D. P., Castañon, D. A., Eckstein, J., and Zenios, S., 1995. "Parallel Computing in Network Optimization," Handbooks in OR and MS, Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L. (eds.), Vol. 7, North-Holland, Amsterdam, pp. 331-399.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BaF88] Bar-Shalom, Y., and Fortman, T. E., 1988. Tracking and Data Association, Academic Press, N. Y.

[Bar90] Bar-Shalom, Y., 1990. Multitarget-Multisensor Tracking: Advanced Applications, Artech House, Norwood, MA.

[BeC89] Bertsekas, D. P., and Castañon, D. A., 1989. "The Auction Algorithm for Transportation Problems," Annals of Operations Research, Vol. 20, pp. 67-96.

[BeC91] Bertsekas, D. P., and Castañon, 1991. "Parallel Synchronous and Asynchronous Implementations of the Auction Algorithm," Parallel Computing, Vol. 17, pp. 707-732.

[BeC99] Bertsekas, D. P., and Castañon, D. A., 1999. "Rollout Algorithms for Stochastic Scheduling Problems," Heuristics, Vol. 5, pp. 89-108.

[BeC08] Besse, C., and Chaib-draa, B., 2008. "Parallel Rollout for Online Solution of DEC-POMDPs," Proc. of 21st International FLAIRS Conference, pp. 619-624.

[BeG97] Beraldi, P., and Guerriero, F., 1997. "A Parallel Asynchronous Implementation of the Epsilon-Relaxation Method for the Linear Minimum Cost Flow Problem," Parallel Computing, Vol. 23, pp. 1021-1044.

[BeL14] Beyme, S., and Leung, C., 2014. "Rollout Algorithm for Target Search in a Wireless Sensor Network," 80th Vehicular Technology Conference (VTC2014), IEEE, pp. 1-5.

[BeS18] Bertazzi, L., and Secomandi, N., 2018. "Faster Rollout Search for the Vehicle Routing Problem with Stochastic Demands and Restocking," European J. of Operational Research, Vol. 270, pp. 487-497.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., 1989. Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J. (republished in 1997 by Athena Scientific, Belmont, MA).

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

[Ber79] Bertsekas, D. P., 1979. "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Report, MIT, May 1979.

[Ber86] Bertsekas, D. P., 1986. "Distributed Relaxation Methods for Linear Network Flow Problems," Proceedings of 25th IEEE Conference on Decision and Control, Athens, Greece, pp. 2101-2106.

[Ber88] Bertsekas, D. P., 1988. "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Annals of Operations Research, Vol. 14, pp. 105-123.

[Ber91] Bertsekas, D. P., 1991. Linear Network Optimization: Algorithms and Codes, MIT Press, Cambridge, MA.

[Ber92] Bertsekas, D. P., 1992. "Auction Algorithms for Network Flow Problems: A Tutorial Introduction," Computational Optimization and Applications, Vol. 1, pp. 7-66.

[Ber93] Bertsekas, D. P., 1993. "Mathematical Equivalence of the Auction Algorithm for Assignment and the ϵ-Relaxation (Preflow-Push) Method for Min Cost Flow," in Large Scale Optimization: State of the Art, Hager, W. W., Hearn, D. W., and Pardalos, P. M. (eds.), Kluwer, Boston, pp. 27-46.

[Ber97] Bertsekas, D. P., 1997. "Differential Training of Rollout Policies," Proc. of the 35th Allerton Conference on Communication, Control, and Computing, Allerton Park, Ill.

[Ber98] Bertsekas, D. P., 1998. Network Optimization: Continuous and Discrete Models, Athena Scientific, Belmont, MA (available in .pdf form through the author's web site).

[Ber01] Bertsekas, D. P., 2001. "Auction Algorithms," Encyclopedia of Optimization, Kluwer.

[Ber05a] Bertsekas, D. P., 2005. "Rollout Algorithms for Constrained Dynamic Programming," LIDS Report 2646, MIT.

[Ber05b] Bertsekas, D. P., 2005. "Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC," European J. of Control, Vol. 11, pp. 310-334.

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

[Ber19a] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[Ber19b] Bertsekas, D. P., 2019. "Multiagent Rollout Algorithms," arXiv preprint arXiv:1910.00120.

[Ber19c] Bertsekas, D. P., 2019. "Robust Shortest Path Planning and Semicontractive Dynamic Programming," Naval Research Logistics, Vol. 66, pp. 15-37.

[Bla86] Blackman, S. S., 1986. Multi-Target Tracking with Radar Applications, Artech House, Dehdam, MA.

[CBH09] Choi, H. L., Brunet, L., and How, J. P., 2009. "Consensus-Based Decentralized Auctions for Robust Task Allocation," IEEE Transactions on Robotics, Vol. 25, pp. 912-926.

[CGC04] Chang, H. S., Givan, R. L., and Chong, E. K. P., 2004. "Parallel Rollout for Online Solution of Partially Observable Markov Decision Processes," Discrete Event Dynamic Systems, Vol. 14, pp. 309-341.

[CXL19] Chu, Z., Xu, Z., and Li, H., 2019. "New Heuristics for the RCPSP with Multiple Overlapping Modes," Computers and Industrial Engineering, Vol. 131, pp. 146-156.

[Cas93] Castañon, D. A., 1993. "Reverse Auction Algorithms for Assignment Problems," in Algorithms for Network Flows and Matching, Johnson, D. S., and McGeoch, C. C. (eds.), American Math. Soc., Providence, RI, pp. 407-429.

[Chr97] Christodouleas, J. D., 1997. "Solution Methods for Multiprocessor Network Scheduling Problems with Application to Railroad Operations," Ph.D. Thesis, Operations Research Center, Massachusetts Institute of Technology.

[DuV99] Duin, C., and Voss, S., 1999. "The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs," Networks: An International Journal, Vol. 34, pp. 181-191.

[FeV02] Ferris, M. C., and Voelker, M. M., 2002. "Neuro-Dynamic Programming for Radiation Treatment Planning," Numerical Analysis Group Research Report NA-02/06, Oxford University Computing Laboratory, Oxford University.

[FeV04] Ferris, M. C., and Voelker, M. M., 2004. "Fractionation in Radiation Treatment Planning," Mathematical Programming B, Vol. 102, pp. 387-413.

[GDM19] Guerriero, F., Di Puglia Pugliese, L., and Macrina, G., 2019. "A Rollout Algorithm for the Resource Constrained Elementary Shortest Path Problem," Optimization Methods and Software, Vol. 34, pp. 1056-1074.

[GTO15] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W., 2015. "Restocking-Based Rollout Policies for the Vehicle Routing Problem with Stochastic Demand and Duration Limits," Transportation Science, Vol. 50, pp. 591-607.

[GuM01] Guerriero, F., and Musmanno, R., 2001. "Label Correcting Methods to Solve Multicriteria Shortest Path Problems," J. Optimization Theory Appl., Vol. 111, pp. 589-613.

[GuM03] Guerriero, F., and Mancini, M., 2003. "A Cooperative Parallel Rollout Algorithm for the Sequential Ordering Problem," Parallel Computing, Vol. 29, pp. 663-677.

[HJG16] Huang, Q., Jia, Q. S., and Guan, X., 2016. "Robust Scheduling of EV Charging Load with Uncertain Wind Power Integration," IEEE Trans. on Smart Grid, Vol. 9, pp. 1043-1054.

[Jaf84] Jaffe, J. M., 1984. "Algorithms for Finding Paths with Multiple Constraints," Networks, Vol. 14, pp. 95-116.

[KAH15] Khashooei, B. A., Antunes, D. J. and Heemels, W.P.M.H., 2015. "Rollout Strategies for Output-Based Event-Triggered Control," in Proc. 2015 European Control Conference, pp. 2168-2173.

[LGW16] Lan, Y., Guan, X., and Wu, J., 2016. "Rollout Strategies for Real-Time Multi-Energy Scheduling in Microgrid with Storage System," IET Generation, Transmission and Distribution, Vol. 10, pp. 688-696.

[LiW15] Li, H., and Womer, N. K., 2015. "Solving Stochastic Resource-Constrained Project Scheduling Problems by Closed-Loop Approximate Dynamic Programming.," European J. of Operational Research, Vol. 246, pp. 20-33.

[MBT05] Mitchell, I. M., Bayen, A. M., and Tomlin, C. J., 2005. "A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games," IEEE Trans. on Automatic Control, Vol. 50, pp. 947-957.

[MMB02] McGovern, A., Moss, E., and Barto, A., 2002. "Building a Basic Building Block Scheduler Using Reinforcement Learning and Rollouts," Machine Learning, Vol. 49, pp. 141-160.

[MaJ15] Mastin, A., and Jaillet, P., 2015. "Average-Case Performance of Rollout Algorithms for Knapsack Problems," J. of Optimization Theory and Applications, Vol. 165, pp. 964-984.

[Mar84] Martins, E. Q. V., 1984. "On a Multicriteria Shortest Path Problem," European J. of Operational Research, Vol. 16, pp. 236-245.

[MPP04] Meloni, C., Pacciarelli, D., and Pranzo, M., 2004. "A Rollout Metaheuristic for Job Shop Scheduling Problems," Annals of Operations Research, Vol. 131, pp. 215-235.

[PDB92] Pattipati, K. R., Deb, S., Bar-Shalom, Y., and Washburn, R. B., 1992. "A New Relaxation Algorithm and Passive Sensor Data Association," IEEE Trans. Automatic Control, Vol. 37, pp. 198-213.

[PPB01] Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y., 2001. "m-Best SD Assignment Algorithm with Application to Multitarget Tracking," IEEE Transactions on Aerospace and Electronic Systems, Vol. 37, pp. 22-39.

[PoR97] Poore, A. B., and Robertson, A. J. A., 1997. New Lagrangian Relaxation Based Algorithm for a Class of Multidimensional Assignment Problems," Computational Optimization and Applications, Vol. 8, pp. 129-150.

[Poo94] Poore, A. B., 1994. "Multidimensional Assignment Formulation of Data Association Problems Arising from Multitarget Tracking and Multisensor Data Fusion," Computational Optimization and Applications, Vol. 3, pp. 27-57.

[SHB15] Simroth, A., Holfeld, D., and Brunsch, R., 2015. "Job Shop Production Planning under Uncertainty:

A Monte Carlo Rollout Approach," Proc. of the International Scientific and Practical Conference, Vol. 3, pp. 175-179.

[SGC02] Savagaonkar, U., Givan, R., and Chong, E. K. P., 2002. "Sampling Techniques for Zero-Sum, Discounted Markov Games," in Proc. 40th Allerton Conference on Communication, Control and Computing, Monticello, Ill.

[SLJ13] Sun, B., Luh, P. B., Jia, Q. S., Jiang, Z., Wang, F., and Song, C., 2013. "Building Energy Management: Integrated Control of Active and Passive Heating, Cooling, Lighting, Shading, and Ventilation Systems," IEEE Trans. on Automation Science and Engineering, Vol. 10, pp. 588-602.

[SNC18] Sarkale, Y., Nozhati, S., Chong, E. K., Ellingwood, B. R., and Mahmoud, H., 2018. "Solving Markov Decision Processes for Network-Level Post-Hazard Recovery via Simulation Optimization and Rollout," in 2018 IEEE 14th International Conference on Automation Science and Engineering, pp. 906-912.

[SZL08] Sun, T., Zhao, Q., Lun, P., and Tomastik, R., 2008. "Optimization of Joint Replacement Policies for Multipart Systems by a Rollout Framework," IEEE Trans. on Automation Science and Engineering, Vol. 5, pp. 609-619.

[Sec00] Secomandi, N., 2000. "Comparing Neuro-Dynamic Programming Algorithms for the Vehicle Routing Problem with Stochastic Demands," Computers and Operations Research, Vol. 27, pp. 1201-1225.

[Sec01] Secomandi, N., 2001. "A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands," Operations Research, Vol. 49, pp. 796-802.

[Sec03] Secomandi, N., 2003. "Analysis of a Rollout Approach to Sequencing Problems with Stochastic Routing Applications," J. of Heuristics, Vol. 9, pp. 321-352.

[StW91] Stewart, B. S., and White, C. C., 1991. "Multiobjective A^*," J. ACM, Vol. 38, pp. 775-814.

[TGL13] Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., and Prager, J. M., 2013. "Analysis of Watson's Strategies for Playing Jeopardy!," J. of Artificial Intelligence Research, Vol. 47, pp. 205-251.

[TPL04] Tu, F., Pham, D., Luo, J., Pattipati, K. R., and Willett, P., 2004. "Decision Feedback with Rollout for Multiuser Detection in Synchronous CDMA," IEE Proceedings-Communications, Vol. 151, pp. 383-386.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," presented at the 1996 Neural Information Processing Systems Conference, Denver, CO; also in M. Mozer et al. (eds.), Advances in Neural Information Processing Systems 9, MIT Press (1997).

[TsB00] Tseng, P., and Bertsekas, D. P., 2000. "An ϵ-Relaxation Method for Separable Convex Cost Generalized Network Flow Problems," Math. Programming, Vol. 88, pp. 85-104.

[TuP03] Tu, F., and Pattipati, K. R., 2003. "Rollout Strategies for Sequential Fault Diagnosis," IEEE Trans. on Systems, Man and Cybernetics, Part A, pp. 86-99.

[UGM18] Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., and Hennig, M., 2018. "Offline-Online Approximate Dynamic Programming for Dynamic Vehicle Routing with Stochastic Requests," Transportation Science, Vol. 53, pp. 185-202.

[Ulm17] Ulmer, M. W., 2017. Approximate Dynamic Programming for Dynamic Vehicle Routing, Springer, Berlin.

[WCG03] Wu, G., Chong, E. K. P., and Givan, R. L., 2003. "Congestion Control Using Policy Rollout," Proc. 2nd IEEE CDC, Hawaii, pp. 4825-4830.

[YDR04] Yan, X., Diaconis, P., Rusmevichientong, P., and Van Roy, B., 2004. "Solitaire: Man Versus Machine," Advances in Neural Information Processing Systems, Vol. 17, pp. 1553-1560.

APPENDIX: An Overview of the Auction Algorithm for 2-Dimensional Assignment

In this appendix, we describe briefly the auction algorithm, an intuitive method for solving the classical 2-dimensional assignment problem, which outperforms substantially its main competitors, both in theory and in practice, and it can naturally exploit the availability of parallel computation. It is particularly well-suited for contexts of frequent reoptimization, where many problems are solved with slightly different data, as in the rollout algorithm of the present paper when applied to multidimensional assignment. The reason is that the final solution obtained for a given assignment problem by the auction algorithm (a set of prices, as we will describe shortly) is a good starting point for applying the algorithm to a similar problem.

In this appendix, we will sketch the basic principles of the auction algorithm, we will explain its computational properties, and we will discuss some of its extensions to more general network flow problems. These extensions are well-suited for use in rollout algorithms for complex combinatorial optimization problems beyond the multidimensional assignment context. For detailed presentations, we refer to the author's textbooks [Ber91], [Ber98], and the survey paper [Ber92]. For an extensive computational study, we refer to Castañon [Cas93]. Several coded implementations of the auction algorithm are freely available from the internet, including some from the author's web site. The algorithm was first proposed in a 1979 report by the author [Ber79]. The present appendix is based on the author's brief tutorial article [Ber01].

In the classical assignment problem there are n persons and n objects that we have to match on a one-to-one basis. There is a benefit a_{ij} for matching person i with object j and we want to assign persons to objects so as to maximize the total benefit.† Mathematically, we want to find a one-to-one assignment [a set of person-object pairs $(1, j_1), \ldots, (n, j_n)$, such that the objects $j_1, \ldots, j_n$ are all distinct] that maximizes the total benefit $\sum_{i=1}^{n} a_{ij_i}$.

The assignment problem is important in many practical contexts. The most obvious ones are resource allocation problems, such as assigning personnel to jobs, machines to tasks, and the like. There are also many situations where the assignment problem appears as a subproblem in various methods for solving more complex problems, as in the context of the present paper.

The assignment problem is also of great theoretical importance because, despite its simplicity, it embodies a fundamental network optimization structure. All linear single commodity network flow problems, can be reduced to the assignment problem by means of a simple reformulation. Thus, any method for solving the assignment problem can be generalized to solve the linear network flow problem, and in fact this approach is particularly helpful in understanding the extension of auction algorithms to network flow problems that are more general than assignment.

---

† The auction algorithm is more intuitively explained in a context where we want to maximize benefit rather than minimizing cost. We will thus adopt the maximization context, but we can simply convert to minimization by changing the sign of a_{ij}.

The principles on which the auction algorithm is based differ markedly from all other linear network flow methods. In particular, classical methods for assignment are based on iterative improvement of some cost function; for example a primal cost (as in primal simplex methods), or a dual cost (as in Hungarian-like methods, dual simplex methods, and relaxation methods). The auction algorithm departs significantly from the cost improvement idea; at any one iteration, it may deteriorate both the primal and the dual cost, although in the end it finds an optimal assignment. It is based on a notion of approximate optimality, called *ε-complementary slackness*, and while it implicitly tries to solve a dual problem, it actually attains a dual solution that is not quite optimal.

The Auction Process

To develop an intuitive understanding of the auction algorithm, it is helpful to introduce an economic equilibrium problem that turns out to be equivalent to the assignment problem. Let us consider the possibility of matching the n objects with the n persons through a market mechanism, viewing each person as an economic agent acting in his own best interest. Suppose that object j has a price p_j and that the person who receives the object must pay the price p_j. Then, the (net) value of object j for person i is $a_{ij} - p_j$ and each person i would logically want to be assigned to an object j_i with maximal value, that is, with

$$a_{ij_i} - p_{j_i} = \max_{j=1,\ldots,n} \{a_{ij} - p_j\}. \tag{1}$$

We will say that a person i is *happy* if this condition holds and we will say that an assignment and a set of prices are at *equilibrium* when all persons are happy.

Equilibrium assignments and prices are naturally of great interest to economists, but there is also a fundamental relation with the assignment problem; it turns out that an equilibrium assignment offers maximum total benefit (and thus solves the assignment problem), while the corresponding set of prices solves an associated dual optimization problem. This is a consequence of the celebrated duality theorem of linear programming.

Let us consider now a natural process for finding an equilibrium assignment. We will call this process the *naive auction algorithm*, because it has a serious flaw, as will be seen shortly. Nonetheless, this flaw will help motivate a more sophisticated and correct algorithm.

The naive auction algorithm proceeds in "rounds" (or "iterations") starting with *any* assignment and *any* set of prices. There is an assignment and a set of prices at the beginning of each round, and if all persons are happy with these, the process terminates. Otherwise some person who is not happy is selected. This person, call him i, finds an object j_i which offers maximal value, that is,

$$j_i \in \arg \max_{j=1,\ldots,n} \{a_{ij} - p_j\}, \tag{2}$$

and then:

(1) Exchanges objects with the person assigned to j_i at the beginning of the round.

(2) Sets the price of the best object j_i to the level at which he is indifferent between j_i and the second best object, that is, he sets p_{j_i} to

$$p_{j_i} + \gamma_i, \tag{3}$$

where

$$\gamma_i = v_i - w_i, \tag{4}$$

v_i is the best object value,

$$v_i = \max_j \{a_{ij} - p_j\}, \tag{5}$$

and w_i is the second best object value

$$w_i = \max_{j \neq j_i} \{a_{ij} - p_j\}, \tag{6}$$

that is, the best value over objects other than j_i. (Note that γ_i is the largest increment by which the best object price p_{j_i} can be increased, with j_i still being the best object for person i.)

This process is repeated in a sequence of rounds until all persons are happy.

We may view this process as an auction, where at each round the bidder i raises the price of his or her preferred object by the *bidding increment* γ_i. Note that γ_i cannot be negative since $v_i \geq w_i$ [compare Eqs. (5) and (6)], so the object prices tend to increase. Just as in a real auction, bidding increments and price increases spur competition by making the bidder's own preferred object less attractive to other potential bidders.

Unfortunately, this auction does not always work. The difficulty is that the bidding increment γ_i is zero when more than one object offers maximum value for the bidder i [cf. Eqs. (4), (6)]. As a result, a situation may be created where several persons contest a smaller number of equally desirable objects without raising their prices, thereby creating a never ending cycle.

To break such cycles, we introduce a perturbation mechanism, motivated by real auctions where each bid for an object must raise its price by a minimum positive increment, and bidders must on occasion take risks to win their preferred objects. In particular, let us fix a positive scalar ϵ and say that a person i is *almost happy* with an assignment and a set of prices if the value of its assigned object j_i is within ϵ of being maximal, that is,

$$a_{ij_i} - p_{j_i} \geq \max_{j=1,\dots,n} \{a_{ij} - p_j\} - \epsilon. \tag{7}$$

We will say that an assignment and a set of prices are *almost at equilibrium* when all persons are almost happy. The condition (7), introduced first in 1979 in conjunction with the auction algorithm, is known as

ϵ-*complementary slackness* and plays a central role in several optimization contexts. For $\epsilon = 0$ it reduces to ordinary complementary slackness [compare Eq. (1)].

We now reformulate the previous auction process so that the bidding increment is always at least equal to ϵ. The resulting method, the *auction algorithm*, is the same as the naive auction algorithm, except that the bidding increment γ_i is

$$\gamma_i = v_i - w_i + \epsilon, \tag{8}$$

[rather than $\gamma_i = v_i - w_i$ as in Eq. (4)]. With this choice, the bidder of a round is almost happy at the end of the round (rather than happy). The particular increment $\gamma_i = v_i - w_i + \epsilon$ used in the auction algorithm is the maximum amount with this property. Smaller increments γ_i would also work as long as $\gamma_i \geq \epsilon$, but using the largest possible increment accelerates the algorithm. This is consistent with experience from real auctions, which tend to terminate faster when the bidding is aggressive.

We can now show that this reformulated auction process terminates in a finite number of rounds, necessarily with an assignment and a set of prices that are almost at equilibrium. To see this, note that once an object receives a bid for the first time, then the person assigned to the object at every subsequent round is almost happy; the reason is that a person is almost happy just after acquiring an object through a bid, and continues to be almost happy as long as he holds the object (since the other object prices cannot decrease in the course of the algorithm). Therefore, the persons that are not almost happy must be assigned to objects that have never received a bid. In particular, *once each object receives at least one bid, the algorithm must terminate*. Next note that if an object receives a bid in m rounds, its price must exceed its initial price by at least $m\epsilon$. Thus, for sufficiently large m, the object will become "expensive" enough to be judged "inferior" to some object that has not received a bid so far. It follows that only for a limited number of rounds can an object receive a bid while some other object still has not yet received any bid. Therefore, there are two possibilities: either (a) the auction terminates in a finite number of rounds, with all persons almost happy, before every object receives a bid or (b) the auction continues until, after a finite number of rounds, all objects receive at least one bid, at which time the auction terminates. (This argument assumes that any person can bid for any object, but it can be generalized for the case where the set of feasible person-object pairs is limited, as long as at least one feasible assignment exists.)

Optimality Properties at Termination

When the auction algorithm terminates, we have an assignment that is almost at equilibrium, but does this assignment maximize the total benefit? The answer here depends strongly on the size of ϵ. In a real auction, a prudent bidder would not place an excessively high bid for fear that he might win the object at an unnecessarily high price. Consistent with this intuition, we can show that if ϵ is small, then the final assignment will be "almost optimal." In particular, we can show that *the total benefit of the final assignment*

35

is within $n\epsilon$ of being optimal. To see this, note that an assignment and a set of prices that are almost at equilibrium may be viewed as being at equilibrium for a *slightly different* problem where all benefits a_{ij} are the same as before, except for the n benefits of the assigned pairs which are modified by an amount no more than ϵ.

Suppose now that the benefits a_{ij} are all integer, which is the typical practical case (if a_{ij} are rational numbers, they can be scaled up to integer by multiplication with a suitable common number). Then, the total benefit of any assignment is integer, so if $n\epsilon < 1$, a complete assignment that is within $n\epsilon$ of being optimal must be optimal. It follows, that *if*

$$\epsilon < \frac{1}{n},$$

and the benefits a_{ij} are all integer, then the assignment obtained upon termination of the auction algorithm is optimal. Let us also note that the final set of prices is within $n\epsilon$ of being an optimal solution of the dual problem

$$\min_{\substack{p_j \\ j=1,\ldots,n}} \left\{ \sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \max_j \{a_{ij} - p_j\} \right\}. \tag{9}$$

This leads to the interpretation of the auction algorithm as a dual algorithm (in fact an approximate coordinate ascent algorithm; see the cited literature).

ϵ-Scaling and Reoptimization

The auction algorithm exhibits interesting computational behavior, and it is essential to understand this behavior in order to use and implement the algorithm efficiently. First note that the amount of work to solve the problem can depend strongly on the value of ϵ and on the maximum absolute object value

$$C = \max_{i,j} |a_{ij}|.$$

Basically, for many types of problems, the number of bidding rounds up to termination tends to be proportional to C/ϵ. Note also that there is a dependence on the initial prices; if these prices are "near optimal," we expect that the number of rounds to solve the problem will be relatively small.

The preceding observations suggest the idea of ϵ-*scaling*, proposed in the original paper [Ber79]], which consists of applying the algorithm several times, starting with a large value of ϵ and successively reducing ϵ up to an ultimate value that is less than some critical value (for example, $1/n$, when the benefits a_{ij} are integer). Each application of the algorithm provides good initial prices for the next application. This is a common idea in nonlinear programming, encountered for example, in barrier and penalty function methods. An alternative form of scaling, called *cost scaling*, is based on successively representing the benefits a_{ij} with an increasing number of bits, while keeping ϵ at a constant value.

In practice, it is a good idea to at least consider scaling. For sparse assignment problems, that is, problems where the set of feasible assignment pairs is severely restricted, scaling seems almost universally helpful. In theory, scaling leads to auction algorithms with a particularly favorable polynomial complexity (without scaling, the algorithm is pseudopolynomial; see the cited literature).

Finally, let us note that the auction algorithm can be initialized with arbitrary prices, as well arbitrary ϵ. Within this context, it is important to choose initial prices that are close to the final prices obtained upon termination. This can be very important in a reoptimization setting, and in fact this is born out from computational complexity analysis (see [Ber88] or [Ber98]). Indeed, reoptimization with a favorable set of starting prices underlies the idea of ϵ-scaling or cost scaling. The rollout framework of this paper involves extensive reoptimization, with solutions of slightly differing assignment problems. This suggests an overwhelming advantage of the auction algorithm as a base heuristic within the context of this paper.

Parallel and Asynchronous Implementation

Both the bidding and the assignment phases of the auction algorithm are highly parallelizable. In particular, the bidding and the assignment can be carried out for all persons and objects simultaneously. Such an implementation can be termed *synchronous*. There are also *totally asynchronous* implementations of the auction algorithm, which are interesting because they are quite flexible and also tend to result in faster solution in some types of parallel machines. To understand these implementations, it is useful to think of a person as an autonomous decision maker who at unpredictable times obtains information about the prices of the objects. Each person who is not almost happy makes a bid at arbitrary times on the basis of its current object price information (that may be outdated because of communication delays).

Bertsekas and Castañon [BeC91] give a careful formulation of the totally asynchronous model, and a proof of its validity. They include also extensive computational results on a shared memory machine, confirming the advantage of asynchronous over synchronous implementations. In the context of rollout, there is potential for much more parallelization in the context of rollout, since as many as m independent 2-dimensional assignment problems can be solved in parallel at each stage of the rollout computations.

Variations and Extensions

The auction algorithm can be extended to solve a number of variations of the assignment problem, such as the asymmetric assignment problem where the number of objects is larger than the number of persons and there is a requirement that all persons be assigned to some object. Naturally, the notion of an assignment must now be modified appropriately. To solve this problem, the auction algorithm need only be modified in

37

the choice of initial conditions. It is sufficient to require that all initial prices be zero. A similar algorithm can be used for the case where there is no requirement that all persons be assigned.

There have been extensions of the auction algorithm for other types of linear network optimization problems. These extensions find potential application in the context of rollout algorithms for challenging combinatorial problems, which involve graphs and network flows. The general approach for constructing auction algorithms for such problems is to convert them to assignment problems, and then to suitably apply the auction algorithm and streamline the computations. In particular, the classical shortest path problem can be solved correctly by the naive auction algorithm described earlier, once the method is streamlined. Similarly, auction algorithms can be constructed for the max-flow problems, and are very efficient. These algorithms bear a close relation to preflow-push algorithms for the max-flow problem.

Linear transportation problems may also be addressed with the auction algorithm (Bertsekas and Castañon [Ber89]). The basic idea is to convert the transportation problem into an assignment problem by creating multiple copies of persons (or objects) for each source (or sink respectively), and then to modify the auction algorithm to take advantage of the presence of the multiple copies, while appropriately streamlining the computations. This auction algorithm may be used in the context of the facility location problem of Example 3.2.

There are also extensions of the auction algorithm for linear minimum cost flow (transshipment) problems, such as the so called ϵ-relaxation method, and the auction/sequential shortest path algorithm algorithm (see the network optimization textbook [Ber98] for a detailed description and further references). The ϵ-relaxation method was first published by the author in [Ber86], although it was known much earlier (since the development of the mathematically equivalent auction algorithm). It is equivalent to the so called pre-flow push algorithms, as discussed in the author's paper [Ber93]. These methods have interesting theoretical properties and like the auction algorithm, are well suited for parallelization (see the papers [Ber86], [BeG97], the survey by Bertsekas, Castañon, Eckstein, and Zenios [BCE95], and the textbook by Bertsekas and Tsitsiklis [BeT89]).

Let us finally note that there have been proposals of auction algorithms for convex separable network optimization problems with and without gains (but with a single commodity and without side constraints); see Tseng and Bertsekas [TsB00].