Rollout Algorithms for Discrete Optimization: A Survey

Dimitri P. Bertsekas

Contents

Abstract

This chapter discusses rollout algorithms, a sequential approach to optimization problems, whereby the optimization variables are optimized one after the other. A rollout algorithm starts from some given heuristic and constructs another heuristic with better performance than the original. The method is particularly simple to implement and is often surprisingly effective. This chapter explains the method and its properties for discrete deterministic optimization problems.

1 Introduction

Rollout is a form of sequential optimization that originated in dynamic programming (DP for short). It may be viewed as a single iteration of the fundamental method of policy iteration. The starting point is a given policy (called the base policy), whose performance is evaluated in some way, possibly by simulation. Based on the evaluation, an improved policy is obtained by one-step lookahead. When the problem is discrete and deterministic, as will be assumed in this chapter, the

D.P. Bertsekas (✉)
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: dimitrib@mit.edu

method is very simple to implement: the base policy is just some heuristic, and the rollout policy consists of repeated application of this heuristic. The rollout policy is guaranteed to improve the performance of the base policy, often very substantially in practice. In this chapter, rather than using the dynamic programming formalism, the method is explained starting from first principles.

Consider the minimization of a function

$$g(x_1, \dots, x_N)$$

of the discrete variables $x_1, \dots, x_N$, assumed to take values in some finite set. This can be done (at least conceptually) by minimizing over x_1 the result of minimization of g over the remaining variables $x_2, \dots, x_N$, with x_1 held fixed, that is, finding

$$x_1^* \in \arg\min_{x_1} J_1(x_1)$$

where the function J_1 is defined by

$$J_1(x_1) = \min_{x_2,\dots,x_N} g(x_1, x_2, \dots, x_N).$$

Leaving aside for the moment the difficulty of calculating $J_1(x_1)$, given the optimal value x_1^*, the optimal value of x_2 can be found by the minimization

$$x_2^* \in \arg\min_{x_2} J_2(x_1^*, x_2),$$

where the function J_2 is defined by

$$J_2(x_1, x_2) = \min_{x_3,\dots,x_N} g(x_1, x_2, x_3, \dots, x_N).$$

Similarly, for every $k = 1, \dots, n$, given the optimal values $x_1^*, \dots, x_{k-1}^*$, the optimal value of x_k can be found by the minimization

$$x_k^* \in \arg\min_{x_k} J_k(x_1^*, \dots, x_{k-1}^*, x_k), \qquad k = 1, \dots, N, \tag{1}$$

where the function J_k is defined by

$$J_k(x_1, \dots, x_k) = \min_{x_{k+1},\dots,x_N} g(x_1, \dots, x_k, x_{k+1}, \dots, x_N). \tag{2}$$

This is a calculation that is typical of DP, where the functions J_k are called the *optimal cost-to-go functions*, and are defined by the recursion

$$J_k(x_1, \dots, x_k) = \min_{x_{k+1}} J_{k+1}(x_1, \dots, x_k, x_{k+1}),$$

starting from the boundary condition

$$J_N(x_1, \ldots, x_N) = g(x_1, \ldots, x_N).$$

[The validity of this recursion can be seen from the definition of J_k.]

Unfortunately, there is a serious difficulty with the preceding approach: calculating numerically and storing the functions J_k in tables is impossible in practice (except for very special problems; if each of the variables $x_1, \ldots, x_k$ may take m distinct values, the storage of J_k requires a table of size m^k). In DP this is known as *the curse of dimensionality*.

In the rollout approach, J_k is *approximated* by a function that is more easily calculated and does not require excessive storage. In particular, for any given $x_1, \ldots, x_k$, an easily implementable heuristic (called *base heuristic*) is used to approximate the minimization (2). If $H_k(x_1, \ldots, x_k)$ denotes the corresponding approximately optimal value, the *rollout algorithm* obtains a suboptimal solution by replacing J_k with H_k in Eq. (1):

$$x_k \in \arg\min_{x_k} H_k(\tilde{x}_1, \ldots, \tilde{x}_{k-1}, x_k), \qquad k = 1, \ldots, N.$$

These minimizations are carried out in sequence, first obtaining $\tilde{x}_1 \in \arg\min_{x_1} H_1(x_1)$, then obtaining $\tilde{x}_2 \in \arg\min_{x_2} H_2(\tilde{x}_1, x_2)$, and proceeding with $\tilde{x}_3, \ldots, \tilde{x}_N$ in that order.

With some analysis and algorithmic refinement, it can be shown that rollout results in no performance deterioration over just applying the base heuristic once to the original problem. Most importantly, experimentation has shown that rollout typically results in significant and often dramatic improvement over the common approach of just applying the base heuristic. This improvement comes at the expense of a substantial but often tractable (polynomial) increase in computational requirements. The following example provides some insight into the nature of cost improvement.

Example 1 (The Breakthrough Problem) Consider a binary tree with N stages as shown in Fig. 1. Stage k of the tree has 2^k nodes. There are two types of tree arcs: *free* and *blocked*. A free (or blocked) arc can (cannot, respectively) be traversed in the direction from the root to the leaves. The objective is to break through the graph with a sequence of free arcs (a free path) starting from the root and ending at one of the leaves.

One may use a DP-like algorithm to discover a free path (if one exists) by starting from the last stage and by proceeding backward to the root node. The kth step of the algorithm determines for each node of stage $N-k$ whether there is a free path from that node to some leaf node, by using the results of the preceding step. The amount of calculation at the kth step is $O(2^{N-k})$. Adding the calculations for the N stages, it can be seen that the total amount of calculation is $O(N2^N)$, so it increases exponentially with the number of stages.

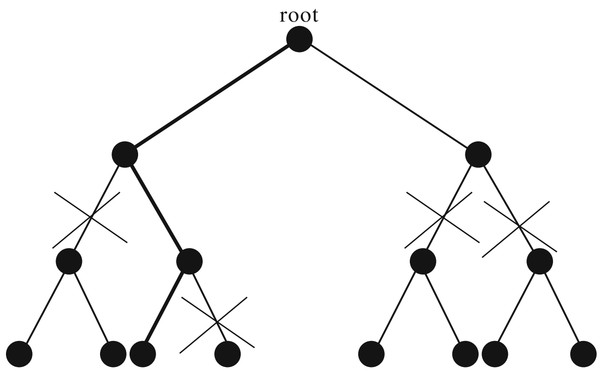

Fig. 1 Binary tree with N stages for the breakthrough problem. Each arc is either free or is blocked (crossed out in the figure). The problem is to find a path from the root to one of the leaves, which is free (such as the one shown with *thick lines*)

As an alternative, one may suboptimally use a *greedy* algorithm, which starts at the root node, selects a free outgoing arc (if one is available), and tries to construct a free path by adding successively nodes to the path. Generally, at the current node, if one of the outgoing arcs is free and the other is blocked, the greedy algorithm selects the free arc. Otherwise, it selects one of the two outgoing arcs according to some fixed rule that depends only on the current node (and not on the status of other arcs). Clearly, the greedy algorithm may fail to find a free path even if such a path exists, as can be seen from Fig. 1 (e.g., if the choice at the root node is the right-hand side arc). On the other hand the amount of computation associated with the greedy algorithm is $O(N)$, which is much faster than the $O(N2^N)$ computation of the optimal algorithm. Thus, one may view the greedy algorithm as a fast heuristic, which is suboptimal in the sense that there are problem instances where it fails while the DP algorithm succeeds.

Consider also the rollout algorithm that uses the greedy algorithm as the base heuristic. This algorithm starts at the root and tries to construct a free path by exploring alternative paths constructed by the greedy algorithm. At the current node, it proceeds according to the following two cases:

(a) If at least one of the two outgoing arcs of the current node is blocked, the rollout algorithm adds to the current path the arc that the greedy algorithm would select at the current node.

(b) If both outgoing arcs of the current node are free, the rollout algorithm considers the two end nodes of these arcs, and from each of them it runs the greedy algorithm. If the greedy algorithm succeeds in finding a free path that starts from at least one of these nodes, the rollout algorithm stops with a free path having been found; otherwise, the rollout algorithm moves to the node that the greedy algorithm would select at the current node.

Thus, when both outgoing arcs are free, the rollout algorithm explores further the suitability of these arcs, as in case (b) above. Because of this additional discriminatory capability, the rollout algorithm always does at least as well as the greedy (it always finds a free path when the greedy algorithm does, and it also finds a free path in some cases where the greedy algorithm does not). This is consistent with our earlier discussion of the generic cost improvement property of the rollout algorithm over the base heuristic. On the other hand, the rollout algorithm applies the greedy heuristic as many as $2N$ times, so that it requires $O(N^2)$ amount of computation – this is intermediate between the $O(N)$ computation of the greedy and the $O(N2^N)$ computation of the DP algorithm.

The greedy and the rollout algorithms may be evaluated by calculating the probabilities that they will find a free path given a randomly chosen breakthrough problem. In particular, the graph of the problem may be generated randomly, by selecting each of its arcs to be free with probability p, independently of the other arcs. If the corresponding probabilities of success for the greedy and the rollout algorithms are calculated, it follows that the rollout algorithm has an $O(N)$ times larger probability of finding a free path than the greedy algorithm, while requiring $O(N)$ times more computation (see [3], Example 6.4.2). This type of tradeoff is qualitatively typical: the rollout algorithm achieves a substantial performance improvement over the base heuristic at the expense of extra computation that is equal to the computation time of the base heuristic times a factor that is a low order polynomial of the problem size.

2 The Basic Rollout Algorithm for Discrete Optimization

The rollout algorithm will now be formalized by introducing a graph search problem that can serve as a general model for discrete optimization. A graph is given that has a finite set of nodes $\mathcal{N}$, a finite set of arcs $\mathcal{A}$, and a special node s, called the *origin*. The arcs are directed in the sense that arc (i, j) is distinct from arc (j, i). A subset $\overline{\mathcal{N}}$ of nodes is also given, called *destinations* and a cost $g(i)$ for each destination i. The destination nodes are terminal in the sense that they have no outgoing arcs. The problem is to find a path that starts at the origin s, ends at one of the destination nodes $i \in \overline{\mathcal{N}}$, and is such that the cost $g(i)$ is minimized.

In our terminology, a path is a sequence of arcs

$$(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m),$$

all of which are oriented in the forward direction. The nodes i_1 and i_m are called the *start node* and the *end node* of the path, respectively. For convenience, and without loss of generality,[1] it will be assumed that given an ordered pair of nodes (i, j), there is at most one arc with start node i and end node j, which (if it exists) will be

[1]In the case where there are multiple arcs connecting a node pair, all these arcs can be merged to a single arc, since the set of destination nodes that can be reached from any non-destination node will not be affected.

denoted by (i, j). In this way, a path consisting of arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m)$ is unambiguously specified as the sequence of nodes $(i_1, i_2, \ldots, i_m)$.

Let us assume the availability of a heuristic path construction algorithm, denoted $\mathcal{H}$, which given a non-destination node $i \notin \overline{\mathcal{N}}$, constructs a path $(i, i_1, \ldots, i_m, \overline{i})$ starting at i and ending at one of the destination nodes $\overline{i}$. Implicit in this assumption is that for every non-destination node, there exists at least one path starting at that node and ending at some destination node. The algorithm $\mathcal{H}$ is referred to as the *base heuristic* and will be used as the basic building block for constructing the rollout algorithm to be introduced shortly.

The end node $\overline{i}$ of the path constructed by the base heuristic $\mathcal{H}$ is completely specified by the start node i. The node $\overline{i}$ is called the *projection of* i *under* $\mathcal{H}$ and is denoted by $p(i)$. The corresponding cost is denoted by $H(i)$,

$$H(i) = g\bigl(p(i)\bigr).$$

The projection of a destination node is the node itself by convention, so that $i = p(i)$ and $H(i) = g(i)$ for all $i \in \overline{\mathcal{N}}$. Note that while the base heuristic $\mathcal{H}$ will generally yield a suboptimal solution, the path that it constructs may involve a fairly sophisticated suboptimization. For example, $\mathcal{H}$ may construct several paths ending at destination nodes according to some heuristics and then select the path that yields minimal cost.

One possibility for suboptimal solution of the problem is to start at the origin s and use the base heuristic $\mathcal{H}$ to obtain the projection $p(s)$. It is instead proposed to use $\mathcal{H}$ to construct a path to a destination node sequentially. At the typical step, a path that starts at s and ends at a node i is available, the base heuristic $\mathcal{H}$ is run starting from each of the downstream neighbors j of i, and the corresponding projections and costs are obtained. The neighbor that gives the best projection is chosen as the next node in the current path. This sequential version of $\mathcal{H}$ is called the *rollout algorithm based on* $\mathcal{H}$ and is denoted by $\mathcal{RH}$.

Formally, let $N(i)$ denote the set of downstream neighbors of a non-destination node i,

$$N(i) = \bigl\{j \mid (i, j) \text{ is an arc}\bigr\}.$$

The rollout algorithm $\mathcal{RH}$ starts with the origin node s. At the typical step, given a node sequence $(s, i_1, \ldots, i_m)$, where i_m is not a destination, $\mathcal{RH}$ adds to the sequence a node i_{m+1} such that

$$i_{m+1} \in \arg \min_{j \in N(i_m)} H(j). \qquad (3)$$

If i_{m+1} is a destination node, $\mathcal{RH}$ terminates. Otherwise, the process is repeated with the sequence $(s, i_1, \ldots, i_m, i_{m+1})$ replacing $(s, i_1, \ldots, i_m)$; see Fig. 2.

Once $\mathcal{RH}$ has terminated with a path $(s, i_1, \ldots, i_m)$, the projection $p(i_k)$ of each of the nodes i_k, $k = 1, \ldots, m$, will have been obtained. The best of these projections yields a cost

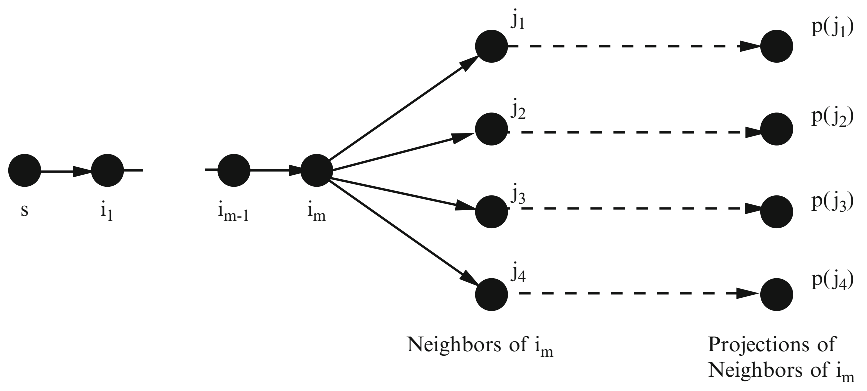

Fig. 2 Illustration of the rollout algorithm. After m steps, the algorithm has constructed the path $(s, i_1, \ldots, i_m)$. To extend this path at the next step, the set $N(i_m)$ of neighbors of the terminal node i_m is generated, and the neighbor that has the best projection is selected from this set, that is, $i_{m+1} \in \arg\min_{j \in N(i_m)} H(j) \in \arg\min_{j \in N(i_m)} g\big(p(j)\big)$

$$\min_{k=1,\ldots,m} H(i_k) = \min_{k=1,\ldots,m} g\big(p(i_k)\big),$$

and the projection that corresponds to the minimum above may be taken as the final (suboptimal) solution produced by the rollout algorithm. The above minimal cost may also be compared with the cost $g\big(p(s)\big)$ of the projection $p(s)$ of the origin, so that $p(s)$ is used as the final solution if it produces a smaller cost. This will ensure that the rollout algorithm will produce a solution that is no worse than the one produced by the base heuristic. Note that while the best neighbor of i_m is i_{m+1} according to Eq. (3), it does not necessarily follow that i_{m+1} has a better projection than i_m under $\mathcal{H}$, that is, that

$$H(i_{m+1}) \leq H(i_m). \tag{4}$$

The reason is that if the path constructed by $\mathcal{H}$ starting from i_m is $(i_m, j_1, \ldots, j_k, \bar{i})$, it is not necessarily true that the path constructed by $\mathcal{H}$ starting from j_1 is $(j_1, \ldots, j_k, \bar{i})$. If this were so, then Eq. (4) would hold, since i_{m+1} would have no worse projection that j_1 according to Eq. (3). This argument will be the basis for further analysis to be given later (see Proposition 1).

Example 2 (Traveling Salesman Problem) Let us consider the traveling salesman problem, whereby a salesman wants to find a minimum mileage/cost tour that visits each of N given cities exactly once and returns to the city he started from. With each city $i = 1, \ldots, N$, a node is associated and an arc (i, j) with traversal cost a_{ij} is introduced for each ordered pair of nodes i and j. Note that the graph is assumed complete; that is, there exists an arc for each ordered pair of nodes. There is no loss of generality in doing so because a very high cost a_{ij} can be assigned to an arc

(i, j) that is precluded from participation in the solution. The problem is to find a cycle that goes through all the nodes exactly once and whose sum of arc costs is minimum.

There are many heuristic approaches for solving the traveling salesman problem. For illustration purposes, consider a simple *nearest neighbor* heuristic. It starts from a path consisting of just a single node i_1, and at each iteration, it enlarges the path with a node that does not close a cycle and minimizes the cost of the enlargement. In particular, after k iterations, a path $\{i_1, \ldots, i_k\}$ consisting of distinct nodes has been constructed, and at the next iteration, an arc (i_k, i_{k+1}) that minimizes $a_{i_k i}$ over all arcs (i_k, i) with $i \neq i_1, \ldots, i_k$, is added. After $N - 1$ iterations, all nodes are included in the path, which is then converted to a tour by adding the final arc (i_N, i_1).

The traveling salesman problem can be formulated as a graph search problem as follows: A starting city, say i_1, is chosen corresponding to the origin of the graph search problem. Each node of the graph search problem corresponds to a path $(i_1, i_2, \ldots, i_k)$, where $i_1, i_2, \ldots, i_k$ are distinct cities. The neighbor nodes of the path $(i_1, i_2, \ldots, i_k)$ are paths of the form $(i_1, i_2, \ldots, i_k, i_{k+1})$ that correspond to adding one more unvisited city $i_{k+1} \neq i_1, i_2, \ldots, i_k$ at the end of the path. The destinations are the cycles of the form $(i_1, i_2, \ldots, i_N)$, and the cost of a destination in the graph search problem is the cost of the corresponding cycle. Thus, a path from the origin to a destination in the graph search problem corresponds to constructing a cycle in $N - 1$ arc addition steps and at the end incurring the cost of the cycle.

Let us now use as base heuristic the nearest neighbor method. The corresponding rollout algorithm operates as follows: After k iterations, a path $\{i_1, \ldots, i_k\}$ consisting of distinct nodes has been constructed. At the next iteration, the nearest neighbor heuristic is run starting from each of the paths of the form $\{i_1, \ldots, i_k, i\}$ where $i \neq i_1, \ldots, i_k$, and a corresponding cycle is obtained. The node i_{k+1} of the path is selected to be the node i that corresponds to the best cycle thus obtained.

2.1 Termination and Sequential Consistency

The rollout algorithm $\mathcal{RH}$ is said to be *terminating* if it is guaranteed to terminate finitely starting from any node. Contrary to the base heuristic $\mathcal{H}$, which by definition, has the property that it yields a path terminating at a destination starting from any node, the rollout algorithm $\mathcal{RH}$ need not have this property in the absence of additional conditions. The termination question can usually be resolved quite easily, and a few different methods by which this can be done will now be discussed.

One important case where $\mathcal{RH}$ is terminating is *when the graph is acyclic*, since then the nodes of the path generated by $\mathcal{RH}$ cannot be repeated within the path, and their number is bounded by the number of nodes in $\mathcal{N}$. As a first step toward developing another case where $\mathcal{RH}$ is terminating, consider the following definition, which will also set the stage for further analysis of the properties of $\mathcal{RH}$.

Definition 1 The base heuristic $\mathcal{H}$ is said to be *sequentially consistent* if for every node i, it has the following property: If $\mathcal{H}$ generates the path $(i, i_1, \ldots, i_m, \bar{i})$ when it starts at i, it generates the path $(i_1, \ldots, i_m, \bar{i})$ when it starts at the node i_1.

Thus, $\mathcal{H}$ is sequentially consistent if all the nodes of a path that it generates have the same projection. There are many examples of sequentially consistent algorithms that are used as heuristics in combinatorial optimization, including the following.

Example 3 (Greedy Algorithms as Base Heuristics) Consider a function F, which for each node i, provides a scalar estimate $F(i)$ of the optimal cost starting from i, that is, the minimal cost $g(\bar{i})$, that can be obtained with a path that starts at i and ends at one of the destination nodes $\bar{i} \in \overline{\mathcal{N}}$. Then, F can be used to define a base heuristic, called the *greedy algorithm with respect to* F, as follows:

The greedy algorithm starts at a node i with the (degenerate) path that consists of just node i. At the typical step, given a path $(i, i_1, \ldots, i_m)$, where i_m is not a destination, the algorithm adds to the path a node i_{m+1} such that

$$i_{m+1} \in \arg \min_{j \in N(i_m)} F(j). \tag{5}$$

If i_{m+1} is a destination, the algorithm terminates with the path $(i, i_1, \ldots, i_m, i_{m+1})$. Otherwise, the process is repeated with the path $(i, i_1, \ldots, i_m, i_{m+1})$ replacing $(i, i_1, \ldots, i_m)$.

An example of a greedy algorithm is the nearest neighbor heuristic for the traveling salesman problem (cf. Example 2). Recall from that example that nodes of the graph search problem correspond to paths (sequences of distinct cities), and a transition to a neighbor node corresponds to adding one more unvisited city to the end of the current path. The function F in the nearest neighbor heuristic specifies the cost of the addition of the new city.

It is also interesting to note that by viewing F as a cost-to-go approximation, the greedy algorithm may be considered to be a special type of one-step lookahead policy. Furthermore, if $F(j)$ is chosen to be the cost obtained by some base heuristic starting from j, then the greedy algorithm becomes the corresponding rollout algorithm. Thus, it may be said that the rollout algorithm is a special case of a greedy algorithm. However, the particular choice of F used in the rollout algorithm is responsible for special properties that are not shared by other types of greedy algorithms.

Let us denote by $\mathcal{H}$ the greedy algorithm described above and assume that it terminates starting from every node (this has to be verified independently). Let us also assume that whenever there is a tie in the minimization of Eq. (5), $\mathcal{H}$ resolves the tie in a manner that is fixed and independent of the starting node i of the path, for example, by resolving the tie in favor of the numerically smallest node j that attains the minimum in Eq. (5). Then, it can be seen that $\mathcal{H}$ is sequentially consistent, since by construction, every node on a path generated by $\mathcal{H}$ has the same projection.

For a sequentially consistent base heuristic $\mathcal{H}$, a restriction will be imposed in the way the rollout algorithm $\mathcal{RH}$ resolves ties in selecting the next node on its path; this restriction will guarantee that $\mathcal{RH}$ is terminating. In particular, suppose that after m steps, $\mathcal{RH}$ has produced the node sequence $(s, i_1, \ldots, i_m)$, and that the path generated by $\mathcal{H}$ starting from i_m is $(i_m, i_{m+1}, i_{m+2}, \ldots, \bar{i})$. Suppose that among the neighbor set $N(i_m)$, the node i_{m+1} attains the minimum in the selection test

$$\min_{j \in N(i_m)} H(j), \tag{6}$$

but there are also some other nodes, in addition to i_{m+1}, that attain this minimum. Then, the tie is broken in favor of i_{m+1}, that is, the next node added to the current sequence $(s, i_1, \ldots, i_m)$ is i_{m+1}. Under this convention for tie-breaking, the following proposition shows that the rollout algorithm $\mathcal{RH}$ terminates at a destination and yields a cost that is no larger than the cost yielded by the base heuristic $\mathcal{H}$.[2]

Proposition 1 *Let the base heuristic $\mathcal{H}$ be sequentially consistent. Then, the rollout algorithm $\mathcal{RH}$ is terminating. Furthermore, if $(i_1, \ldots, i_{\tilde{m}})$ is the path generated by $\mathcal{RH}$ starting from a non-destination node i_1 and ending at a destination node $i_{\tilde{m}}$, the cost of $\mathcal{RH}$ starting from i_1 is less or equal to the cost of $\mathcal{H}$ starting from i_1. In particular,*

$$H(i_1) \geq H(i_2) \geq \cdots \geq H(i_{\tilde{m}-1}) \geq H(i_{\tilde{m}}). \tag{7}$$

Furthermore, for all $m = 1, \ldots, \tilde{m}$,

$$H(i_m) = \min \left\{ H(i_1), \min_{j \in N(i_1)} H(j), \ldots, \min_{j \in N(i_{m-1})} H(j) \right\}. \tag{8}$$

Proof Let i_m and i_{m+1} be two successive nodes generated by $\mathcal{RH}$, and let $(i_m, i'_{m+1}, i'_{m+2}, \ldots, \bar{i}_m)$ be the path generated by $\mathcal{H}$ starting from i_m, where $\bar{i}_m$ is the projection of i_m. Then, since $\mathcal{H}$ is sequentially consistent,

$$H(i_m) = H(i'_{m+1}) = g(\bar{i}_m).$$

Furthermore, since $i'_{m+1} \in N(i_m)$, using the definition of $\mathcal{RH}$ [cf. Eq. (3)],

[2]For an example where this convention for tie-breaking is not observed and as a consequence $\mathcal{RH}$ does not terminate, assume that there is a single destination d and that all other nodes are arranged in a cycle. Each non-destination node i has two outgoing arcs: one arc that belongs to the cycle and another arc which is (i, d). Let $\mathcal{H}$ be the (sequentially consistent) base heuristic that, starting from a node $i \neq d$, generates the path (i, d). When the terminal node of the path is node i, the rollout algorithm $\mathcal{RH}$ compares the two neighbors of i, which are d and the node next to i on the cycle, call it j. Both neighbors have d as their projection, so there is tie in Eq. (6). It can be seen that if $\mathcal{RH}$ breaks ties in favor of the neighbor j that lies on the cycle, then $\mathcal{RH}$ continually repeats the cycle and never terminates.

$$H(i'_{m+1}) \geq \min_{j \in N(i_m)} H(j) = H(i_{m+1}).$$

Combining the last two relations,

$$H(i_m) \geq H(i_{m+1}) = \min_{j \in N(i_m)} H(j). \tag{9}$$

To show that $\mathcal{RH}$ is terminating, note that in view of Eq. (9), either $H(i_m) > H(i_{m+1})$ or else $H(i_m) = H(i_{m+1})$. In the latter case, in view of the convention for breaking ties that occur in Eq. (6) and the sequential consistency of $\mathcal{H}$, the path generated by $\mathcal{H}$ starting from i_{m+1} is the tail portion of the path generated by $\mathcal{H}$ starting from i_m and has one arc less. Thus, the number of nodes generated by $\mathcal{RH}$ between successive times that the inequality $H(i_m) > H(i_{m+1})$ holds is finite. On the other hand, the inequality $H(i_m) > H(i_{m+1})$ can occur only a finite number of times, since the number of destination nodes is finite, and the destination node of the path generated by $\mathcal{H}$ starting from i_m cannot be repeated if the inequality $H(i_m) > H(i_{m+1})$ holds. Therefore, $\mathcal{RH}$ is terminating.

Finally, if $(i_1, \dots, i_{\tilde{m}})$ is the path generated by $\mathcal{RH}$, the relation (9) implies the desired relations (7) and (8). **Q. E. D.**

Proposition 1 shows that in the sequentially consistent case, the rollout algorithm $\mathcal{RH}$ has an important "automatic cost sorting" property, whereby it follows the best path generated by the base heuristic $\mathcal{H}$. In particular, when $\mathcal{RH}$ generates a path $(i_1, \dots, i_{\tilde{m}})$, it does so by using $\mathcal{H}$ to generate a collection of other paths and corresponding projections starting from all the successor nodes of the intermediate nodes $i_1, \dots, i_{\tilde{m}-1}$. However, $(i_1, \dots, i_{\tilde{m}})$ is guaranteed to be the best among this path collection, and $i_{\tilde{m}}$ has minimal cost among all generated projections [cf. Eq. (8)]. Of course, this does not guarantee that the path generated by $\mathcal{RH}$ will be a near-optimal path, because the collection of paths generated by $\mathcal{H}$ may be "poor." Still, the property whereby $\mathcal{RH}$ at all times follows the best path found so far is intuitively reassuring.

The following example illustrates the preceding concepts.

Example 4 (One-Dimensional Walk) Consider a person who walks on a straight line and at each time period takes either a unit step to the left or a unit step to the right. There is a cost function assigning cost $g(i)$ to each integer i. Given an integer starting point on the line, the person wants to minimize the cost of the point where he will end up after a given and fixed number N of steps.

This problem can be formulated as a graph search problem of the type discussed in the preceding section. In particular, without loss of generality, assume that the starting point is the origin, so that the person's position after n steps will be some integer in the interval $[-n, n]$. The nodes of the graph are identified with pairs (k, m), where k is the number of steps taken so far ($k = 1, \dots, N$) and m is the person's position ($m \in [-k, k]$). A node (k, m) with $k < N$ has two outgoing arcs

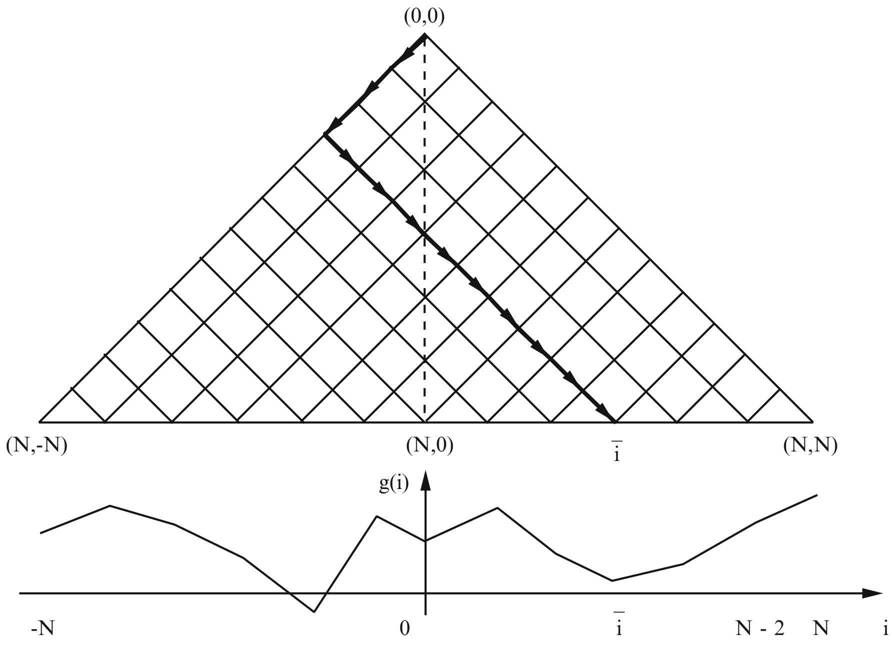

Fig. 3 Illustration of the path generated by the rollout algorithm $\mathcal{RH}$ in Example 4. The algorithm keeps moving to the *left* up to the time where the base heuristic $\mathcal{H}$ generates two destinations $(N, \bar{i})$ and $(N, \bar{i} - 2)$ with $g(\bar{i}) \leq g(\bar{i} - 2)$. Then it continues to move to the *right* ending at the destination $(N, \bar{i})$, which corresponds to the local minimum closest to N

with end nodes $(k+1, m-1)$ (corresponding to a left step) and $(k+1, m+1)$ (corresponding to a right step). The starting state is $(0,0)$ and the terminating states are of the form (N, m), where m is of the form $N - 2l$ and $l \in [0, N]$ is the number of left steps taken.

Let the base heuristic $\mathcal{H}$ be defined as the algorithm, which, starting at a node (k, m), takes $N - k$ successive steps to the right and terminates at the node $(N, m + N - k)$. Note that $\mathcal{H}$ is sequentially consistent. The rollout algorithm $\mathcal{RH}$, at node (k, m), compares the cost of the destination node $(N, m + N - k)$ (corresponding to taking a step to the right and then following $\mathcal{H}$) and the cost of the destination node $(N, m + N - k - 2)$ (corresponding to taking a step to the left and then following $\mathcal{H}$).

Let us say that an integer $i \in [-N + 2, N - 2]$ is a *local minimum* if $g(i-2) \geq g(i)$ and $g(i) \leq g(i+2)$. Let us also say that N (or $-N$) is a local minimum if $g(N-2) \leq g(N)$ [or $g(-N) \leq g(-N+2)$, respectively]. Then, it can be seen that starting from the origin $(0,0)$, $\mathcal{RH}$ obtains the local minimum that is closest to N (see Fig. 3). This is no worse (and typically better) than the integer N obtained by $\mathcal{H}$. This example illustrates how $\mathcal{RH}$ may exhibit "intelligence" that is totally lacking from $\mathcal{H}$ and is in agreement with the result of Proposition 1.

2.1.1 Sequential Improvement

It is possible to show that the rollout algorithm improves on the base heuristic (cf. Proposition 1) under weaker conditions. To this end the following definition is introduced.

Definition 2 The base heuristic $\mathcal{H}$ is said to be *sequentially improving* if for every non-destination node i,

$$H(i) \geq \min_{j \in N(i)} H(j). \tag{10}$$

It can be seen that a sequentially consistent $\mathcal{H}$ is also sequentially improving, since sequential consistency implies that $H(i)$ is equal to one of the values $H(j)$, $j \in N(i)$. The following proposition generalizes Proposition 1.

Proposition 2 *Let the base heuristic $\mathcal{H}$ be sequentially improving and assume that the rollout algorithm $\mathcal{RH}$ is terminating. Let $(i_1, \ldots, i_{\tilde{m}})$ be the path generated by $\mathcal{RH}$ starting from a non-destination node i_1 and ending at a destination node $i_{\tilde{m}}$. Then, the cost of $\mathcal{RH}$ starting from i_1 is less or equal to the cost of $\mathcal{H}$ starting from i_1. In particular, for all $m = 1, \ldots, \tilde{m}$,*

$$H(i_m) = \min \left\{ H(i_1), \min_{j \in N(i_1)} H(j), \ldots, \min_{j \in N(i_{m-1})} H(j) \right\}. \tag{11}$$

Proof For each $m = 1, \ldots, \tilde{m} - 1$,

$$H(i_m) \geq \min_{j \in N(i_m)} H(j),$$

by the sequential improvement assumption, while

$$\min_{j \in N(i_m)} H(j) = H(i_{m+1}),$$

by the definition of the rollout algorithm. These two relations imply Eq. (11). Since the cost of $\mathcal{RH}$ starting from i_1 is $H(i_{\tilde{m}})$, the result follows. **Q. E. D.**

Example 5 (One-Dimensional Walk: Continued) Consider the one-dimensional walk problem of Example 4, and let $\mathcal{H}$ be defined as the algorithm that, starting at a node (k, m), compares the cost $g(m + N - k)$ (corresponding to taking all of the remaining $N - k$ steps to the right) and the cost $g(m - N + k)$ (corresponding to taking all of the remaining $N - k$ steps to the left) and accordingly moves to node

$$(N, m + N - k) \qquad \text{if} \qquad g(m + N - k) \leq g(m - N + k),$$

or to node

$$(N, m - N + k) \qquad \text{if} \qquad g(m - N + k) < g(m + N - k).$$

It can be seen that $\mathcal{H}$ is not sequentially consistent, but is instead sequentially improving. Using Eq. (11), it follows that starting from the origin $(0, 0)$, $\mathcal{RH}$ obtains the global minimum of g in the interval $[-N, N]$, while $\mathcal{H}$ obtains the better of the two points $-N$ and N.

Proposition 2 actually follows from a general equation for the cost of the path generated by the rollout algorithm, which holds for any base heuristic (not necessarily one that is sequentially improving). This is given in the following proposition.

Proposition 3 *Assume that the rollout algorithm $\mathcal{RH}$ is terminating. Let $(i_1, \ldots, i_{\tilde{m}})$ be the path generated by $\mathcal{RH}$ starting from a non-destination node i_1 and ending at a destination node $i_{\tilde{m}}$. Then, the cost of $\mathcal{RH}$ starting from i_1 is equal to*

$$H(i_1) + \delta_{i_1} + \cdots + \delta_{i_{\tilde{m}-1}},$$

where for every non-destination node i,

$$\delta_i = \min_{j \in N(i)} H(j) - H(i).$$

Proof By the definition of the rollout algorithm

$$H(i_m) + \delta_{i_m} = \min_{j \in N(i_m)} H(j) = H(i_{m+1}), \qquad m = 1, \ldots, \tilde{m} - 1.$$

By adding these equations over m,

$$H(i_1) + \delta_{i_1} + \cdots + \delta_{i_{\tilde{m}-1}} = H(i_{\tilde{m}}).$$

Since the cost of $\mathcal{RH}$ starting from i_1 is $H(i_{\tilde{m}})$, the result follows. **Q. E. D.**

If the base heuristic is sequentially improving, there holds $\delta_i \leq 0$ for all non-destination nodes i, so it follows from Proposition 3 that the cost of the rollout algorithm is less or equal to the cost of the base heuristic (cf. Proposition 2).

2.1.2 The Fortified Rollout Algorithm

A variant of the rollout algorithm will now be described, which implicitly uses a sequentially improving base heuristic, so that it has the cost improvement property of Proposition 2. This variant, called the *fortified rollout algorithm*, and denoted by $\overline{\mathcal{RH}}$, starts at the origin s, and after m steps, maintains, in addition to the current sequence of nodes $(s, i_1, \ldots, i_m)$, a path

$$P(i_m) = (i_m, i'_{m+1}, \ldots, i'_k),$$

ending at a destination i'_k. Roughly speaking, the path $P(i_m)$ is the tail portion of the best path found after the first m steps of the algorithm, in the sense that the destination i'_k has minimal cost over all the projections of nodes calculated thus far.

In particular, initially $P(s)$ is the path generated by the base heuristic $\mathcal{H}$ starting from s. At the typical step of the fortified rollout algorithm $\mathcal{R}\overline{\mathcal{H}}$, a node sequence $(s, i_1, \ldots, i_m)$ has been constructed and the path $P(i_m) = (i_m, i'_{m+1}, \ldots, i'_k)$ is available, where i_m is not a destination. Then, if

$$\min_{j \in N(i_m)} H(j) < g(i'_k), \tag{12}$$

$\mathcal{R}\overline{\mathcal{H}}$ adds to the node sequence $(s, i_1, \ldots, i_m)$ the node

$$i_{m+1} \in \arg \min_{j \in N(i_m)} H(j)$$

and sets $P(i_{m+1})$ to the path generated by $\mathcal{H}$, starting from i_{m+1}. On the other hand, if

$$\min_{j \in N(i_m)} H(j) \geq g(i'_k), \tag{13}$$

$\mathcal{R}\overline{\mathcal{H}}$ adds to the node sequence $(s, i_1, \ldots, i_m)$ the node

$$i_{m+1} = i'_{m+1}$$

and sets $P(i_{m+1})$ to the path $(i_{m+1}, i'_{m+2}, \ldots, i'_k)$. If i_{m+1} is a destination, $\mathcal{R}\overline{\mathcal{H}}$ terminates, and otherwise, $\mathcal{R}\overline{\mathcal{H}}$ repeats the process with $(s, i_1, \ldots, i_{m+1})$ replacing $(s, i_1, \ldots, i_m)$ and $P(i_{m+1})$ replacing $P(i_m)$, respectively.

The idea behind the construction of $\mathcal{R}\overline{\mathcal{H}}$ is to follow the path $P(i_m)$ unless a path of lower cost is discovered through Eq. (12). It can be shown that $\mathcal{R}\overline{\mathcal{H}}$ may be viewed as the rollout algorithm $\mathcal{R}\mathcal{H}$ corresponding to a modified version of $\mathcal{H}$, called *fortified* $\mathcal{H}$, and denoted $\overline{\mathcal{H}}$. This algorithm is applied to a slightly modified version of the original problem, which involves an additional downstream neighbor for each node i_m that is generated in the course of the algorithm $\mathcal{R}\overline{\mathcal{H}}$ and for which the condition (13) holds. For every such node i_m, the additional neighbor is a copy of i'_{m+1}, and the path generated by $\overline{\mathcal{H}}$ starting from this copy is $(i'_{m+1}, \ldots, i'_k)$. From every other node, the path generated by $\overline{\mathcal{H}}$ is the same as the path generated by $\mathcal{H}$.

It can be seen that $\overline{\mathcal{H}}$ is sequentially improving, so that $\mathcal{R}\overline{\mathcal{H}}$ is terminating and has the automatic cost sorting property of Proposition 2; that is,

$$H(i_m) = \min \left\{ H(i_1), \min_{j \in N(i_1)} H(j), \ldots, \min_{j \in N(i_{m-1})} H(j) \right\}.$$

The above property can also be easily verified directly, using the definition of $\mathcal{R}\overline{\mathcal{H}}$. Finally, it can be seen that when $\mathcal{H}$ is sequentially consistent, the rollout algorithm $\mathcal{R}\mathcal{H}$ and its fortified version $\mathcal{R}\overline{\mathcal{H}}$ coincide.

2.1.3 Using Multiple Base Heuristics: Parallel Rollout

In many problems, several promising path construction heuristics may be available. It is then possible to use all of these heuristics in parallel within the rollout framework, essentially by combining them into a single "superheuristic." In particular, let us assume that K algorithms $\mathcal{H}_1, \ldots, \mathcal{H}_K$ are available. The kth of these algorithms, given a non-destination node i, produces a path $(i, i_1, \ldots, i_m, \bar{i})$ that ends at a destination node $\bar{i}$, and the corresponding cost is denoted by $H_k(i) = g(\bar{i})$. The K algorithms can be incorporated in a generalized version of the rollout algorithm, which uses the minimal cost

$$H(i) = \min_{k=1,\ldots,K} H_k(i), \tag{14}$$

in place of the cost obtained by any one of the K algorithms $\mathcal{H}_1, \ldots, \mathcal{H}_K$.

In particular, the algorithm starts with the origin node s. At the typical step, given a node sequence $(s, i_1, \ldots, i_m)$, where i_m is not a destination, the algorithm adds to the sequence a node i_{m+1} such that

$$i_{m+1} \in \arg \min_{j \in N(i_m)} H(j).$$

If i_{m+1} is a destination node, the algorithm terminates, and otherwise, the process is repeated with the sequence $(s, i_1, \ldots, i_m, i_{m+1})$ replacing $(s, i_1, \ldots, i_m)$.

An interesting property, which can be readily verified by using the definitions, is that if all the algorithms $\mathcal{H}_1, \ldots, \mathcal{H}_K$ are sequentially improving, the same is true for $\mathcal{H}$. This is evident from the fact that the heuristics run in parallel from a given node may be viewed as a single heuristic, which has the cost improvement property of Definition 2. The fortified version of the rollout algorithm $\overline{\mathcal{RH}}$ easily generalizes for the case of Eq. (14), by defining the path generated starting from a node i as the path generated by the path construction algorithm, which attains the minimum in Eq. (14).

In an alternative version of the rollout algorithm that uses multiple path construction heuristics, the results of the K algorithms $\mathcal{H}_1, \ldots, \mathcal{H}_K$ are weighted with some fixed scalar weights r_k to compute $H(i)$ for use in Eq. (3):

$$H(i) = \sum_{k=1}^{K} r_k H_k(i). \tag{15}$$

The weights r_k may be adjusted by trial and error or more sophisticated techniques which may be found in the literature (see e.g., [6]).

2.1.4 Extension for Intermediate Arc Costs

Let us consider a variant of the graph search problem where in addition to the terminal cost $g(i)$, there is a cost $c(i, j)$ for a path to traverse an arc (i, j). Within this context, the cost of a path $(i_1, i_2, \ldots, i_n)$ that starts at i_1 and ends at a destination node i_n is redefined to be

$$g(i_n) + \sum_{k=1}^{n-1} c(i_k, i_{k+1}). \tag{16}$$

Note that when the cost $g(i)$ is zero for all destination nodes i, this is the problem of finding a shortest path from the origin node s to one of the destination nodes, with $c(i, j)$ viewed as the length of arc (i, j). However, here we are interested in problems where the number of nodes is very large, and the use of the shortest path algorithms is impractical.

One way to transform the problem with arc costs into one involving a terminal cost only is to redefine the graph of the problem so that nodes correspond to sequences of nodes in the original problem graph. Thus, having arrived at node i_k using path $(i_1, \ldots, i_k)$, the choice of i_{k+1} as the next node is viewed as a transition from $(i_1, \ldots, i_k)$ to $(i_1, \ldots, i_k, i_{k+1})$. Both nodes $(i_1, \ldots, i_k)$ and $(i_1, \ldots, i_k, i_{k+1})$ are viewed as nodes of a redefined graph. Furthermore, in this redefined graph, a destination node has the form $(i_1, i_2, \ldots, i_n)$, where i_n is a destination node of the original graph, and has a cost given by Eq. (16).

After the details are worked out, it can be seen that to recover our earlier algorithms and analysis, the cost of the heuristic algorithm $\mathcal{H}$ needs to be modified as follows: If the path $(i_1, \ldots, i_n)$ is generated by $\mathcal{H}$ starting at i_1, then

$$H(i_1) = g(i_n) + \sum_{k=1}^{n-1} c(i_k, i_{k+1}).$$

Furthermore, the rollout algorithm $\mathcal{RH}$ at node i_m selects as next node i_{m+1} the node

$$i_{m+1} \in \arg \min_{j \in N(i_m)} \big[c(i_m, j) + H(j)\big];$$

[cf. Eq. (3)]. The definition of a sequentially consistent algorithm remains unchanged. Furthermore, Proposition 1 remains unchanged except that Eqs. (7) and (8) are modified to read

$$H(i_k) \geq c(i_k, i_{k+1}) + H(i_{k+1}) = \min_{j \in N(i_k)} \big[c(i_k, j) + H(j)\big], \quad k = 1, \ldots, m-1.$$

A sequentially improving algorithm should now be characterized by the property

$$H(i_k) \geq c(i_k, i_{k+1}) + H(i_{k+1}),$$

where i_{k+1} is the next node on the path generated by $\mathcal{H}$ starting from i_k. Furthermore, Proposition 2 remains unchanged, except that Eq. (11) is modified to read

$$H(i_k) \geq \min_{j \in N(i_k)} \big[c(i_k, j) + H(j)\big], \qquad k = 1, \ldots, m-1.$$

Finally, the criterion $\min_{j\in N(i_m)} H(j) < g(i'_k)$ [cf. Eq. (12)] used in the fortified rollout algorithm, given the sequence $(s, i_1, \ldots, i_m)$, where $i_m \notin \overline{\mathcal{N}}$, and the path $P(i_m) = (i_m, i'_{m+1}, \ldots, i'_k)$, should be replaced by

$$\min_{j\in N(i_m)} \big[c(i_m, j) + H(j)\big] < g(i'_k) + c(i_m, i'_{m+1}) + \sum_{l=m+1}^{k-1} c(i'_l, i'_{l+1}).$$

2.1.5 Rollout Algorithms with Multistep Lookahead

It is possible to incorporate *multistep lookahead* into the rollout framework. To describe the case of 2-step lookahead, suppose that after m steps of the rollout algorithm, the current node sequence is $(s, i_1, \ldots, i_m)$. Then, the set of all 2-step-ahead neighbors of i_m is considered, defined as

$$N_2(i_m) = \big\{ j \in \mathcal{N} \mid \; j \in N(i_m) \text{ and } j \in \overline{\mathcal{N}}, \text{ or } j \in N(n) \text{ for some } n \in N(i_m) \big\}.$$

The base heuristic $\mathcal{H}$ is then run starting from each $j \in N_2(i_m)$, and the node $\overline{j} \in N_2(i_m)$ that has projection of minimum cost is found. Let $i_{m+1} \in N(i_m)$ be the node next to i_m on the (one- or two-arc) path from i_m to $\overline{j}$. If i_{m+1} is a destination node, the algorithm terminates. Otherwise, the process is repeated with the sequence $(s, i_1, \ldots, i_m, i_{m+1})$ replacing $(s, i_1, \ldots, i_m)$.

Note that a fortified version of the rollout algorithm described above is possible along the lines described earlier. Also, it is possible to eliminate from the set $N_2(i_m)$ some of the 2-step neighbors of i_m that are judged less promising according to some heuristic criterion, in order to limit the number of applications of the base heuristic. This may be viewed as *selective depth lookahead*. Finally, the extension of the algorithm to look ahead more than two steps is straightforward: The 2-step-ahead neighbor set $N_2(i_m)$ is simply replaced with a suitably defined k-step-ahead neighbor set $N_k(i_m)$.

2.1.6 Interpretation in Terms of DP

Let us now reinterpret the graph-based rollout algorithm within the context of deterministic DP. The base heuristic will be viewed as a suboptimal policy, and the rollout algorithm will be viewed as a policy obtained by a process of policy improvement, provided the base heuristic is sequentially consistent.

To this end, the graph search problem is cast as a sequential decision problem, where each node corresponds to a state of a dynamic system. At each non-destination node/state i, a node j must be selected from the set of neighbors $N(i)$; then, if j is a destination, the process terminates with cost $g(j)$, and otherwise, the process is repeated with j becoming the new state. The DP algorithm calculates for every node i, the minimal cost that can be achieved starting from i, that is, the smallest value of $g(\overline{i})$ that can be obtained using paths that start from i and end at destination nodes $\overline{i}$. This value, denoted $J^*(i)$, is the optimal cost-to-go starting at node i. Once $J^*(i)$ is computed for all nodes i, an optimal path $(i_1, i_2, \ldots, i_m)$

can be constructed starting from any initial node/state i_1 by successively generating nodes using the relation

$$i_{k+1} \in \arg \min_{j \in N(i_k)} J^*(j), \qquad k = 1, \ldots, m-1, \tag{17}$$

up to the point where a destination node i_m is encountered.[3]

A base heuristic $\mathcal{H}$ defines a policy π, that is, an assignment of a successor node to any non-destination node. However, starting from a given node i, the cost of π need not be equal to $H(i)$ because if a path $(i_1, i_2, i_3, \ldots, i_m)$ is generated by $\mathcal{H}$ starting from node i_1, it is not necessarily true that the path $(i_2, i_3, \ldots, i_m)$ is generated by the base heuristic starting from i_2. Thus, the successor node chosen at node i_2 by policy π may be different than the one used in the calculation of $H(i_1)$. On the other hand, if $\mathcal{H}$ is sequentially consistent, the cost of policy π starting from a node i is $H(i)$, since sequential consistency implies that the path that the base heuristic generates starting at the successor node is part of the path it generates at the predecessor node. It turns out that the cost improvement property of the rollout algorithm in the sequentially consistent case is a special case of a cost improvement property for rollout algorithms that holds for more general DP contexts, including stochastic ones.

Generally, in the DP context the rollout algorithm $\mathcal{RH}$ is viewed as a one-step lookahead policy that uses $H(j)$ as a cost-to-go approximation from state j. In some cases, $H(j)$ is the cost of some policy (in the DP sense), such as when $\mathcal{H}$ is sequentially consistent, as explained above. In general, however, this need not be so, in which case $H(j)$ can be viewed as a convenient cost-to-go approximation that is derived from the base heuristic. Still, the rollout algorithm $\mathcal{RH}$ may improve on the cost of the base heuristic (e.g., when $\mathcal{H}$ is sequentially improving, cf. Proposition 2).

3 Applications in Discrete Optimization

Finally, to provide some perspective on the rollout methodology, let us explore the connections with some important discrete optimization problems and methods. In particular, let us consider the generic discrete optimization problem of minimizing a cost function $g(x)$ over a finite set X of feasible solutions. This problem will be reformulated as a graph search problem. It may be noted that several reformulations are possible, and different choices of the underlying graph give rise to different rollout algorithms. Thus, it is important to select a reformulation that matches the type of rollout algorithm one wishes to develop. Some reformulations and corresponding rollout algorithms will be discussed in what follows, and these algorithms will be related to some general computational methods.

[3]It is assumed here that there are no termination/cycling difficulties of the type illustrated in the footnote following Example 3.

Suppose that each solution x has N components; that is, it has the form $x = (x_1, \dots, x_N)$, where N is a positive integer. For example, in a 0-1 integer programming problem, each component x_k may correspond to a single variable that can take the values 0 or 1, or alternatively, it may correspond to a multidimensional vector involving several variables each taking the values 0 or 1. In a network optimization problem, each component x_k may correspond to a vector involving the flows of several arcs of the network. One way to reformulate the problem

$$\text{minimize } g(x)$$
$$\text{subject to } x \in X$$

into the framework of the search problem is to introduce an acyclic graph involving an artificial origin node s and N subsets of nodes $I_1, \dots, I_N$. In particular, for each feasible solution $x \in X$ and each $k = 1, \dots, N$, the node set I_k contains a node $(x(k), k)$, where $x(k)$ consists of the first k components of x [two feasible solutions $x, x' \in X$ whose first k components are identical are mapped onto the same node $(x(k), k)$ of I_k]. Each node $(x(N), N) \in I_N$ is viewed as a destination node of the graph and has cost $g(x)$, where x is the feasible solution mapping onto $x(N)$. The origin node is connected with an arc to each node $(x(1), 1) \in I_1$. Furthermore, for every $k = 1, \dots, N-1$, each node $(x(k), k) \in I_k$ is connected with an arc to each node $(x(k+1), k+1) \in I_{k+1}$ such that the components $x_1, \dots, x_k$ of $x(k)$ and the first k components $x(k+1)$ are identical. A few observations may be made:

(a) Selecting one neighbor out of the set of neighbors of the origin node amounts to selecting the first component x_1 of x, while selecting one neighbor out of the set of neighbors of a node $(x(k), k) \in I_k$ amounts to selecting the $(k+1)$st component x_{k+1} of x.

(b) Choosing a path that starts at s and ends at a destination node $(x(N), N)$ amounts to a sequential choice of the components of x: The first component is chosen when the arc connecting s to a node $(x(1), 1) \in I_1$ is selected, and the kth component is chosen $(k = 2, \dots, N)$ when the arc connecting a node $(x(k-1), k-1) \in I_{k-1}$ to a node $(x(k), k) \in I_k$ is selected. For each $k = 2, \dots, N$, the first $k-1$ components of $x(k-1)$ and $x(k)$ are identical.

(c) Any base heuristic that starts at node s amounts to a sequential choice of the components x_k, $k = 1, \dots, N$, so that the final result $(x_1, \dots, x_N)$ is feasible (belongs to X). Any base heuristic that starts at a non-destination node $(x(k), k) \in I_k$ amounts to a sequential choice of the components $x_{k+1}, \dots, x_N$, so that after they are added to the k components $x_1, \dots, x_k$ specified by $x(k)$, the final result $(x_1, \dots, x_N)$ is feasible.

Given a base heuristic $\mathcal{H}$, as described in (c) above, the kth step of the rollout algorithm $\mathcal{RH}$ minimizes g with respect to the kth component x_k, while keeping the preceding components $x_1, \dots, x_{k-1}$ at the values selected at the preceding steps, and using the base heuristic $\mathcal{H}$ to supply the remaining components $x_{k+1}, \dots, x_N$.

Here is an example where the base heuristic is trivial, and the rollout algorithm leads to a well-known method.

Example 6 (Coordinate Descent) Assume that the set X has the (Cartesian product) form

$$\{(x_1,\dots,x_N) \mid x_k \in X_k,\ k=1,\dots,N\}, \tag{18}$$

where X_k, $k = 1,\dots,N$, are some given finite sets. In principle, there is no loss of generality in this assumption since sets X_k such that the set (18) contains X can always be found, and the cost $g(x)$ can be set to a very high value for every $x \notin X$ that belongs to the set (18).

Let $\overline{x}$ be some given feasible solution. Consider the base heuristic that operates as follows:

(a) Starting from the origin s, it generates the solution $\overline{x}$.

(b) Starting from $(x(k),k) \in I_k$, $k = 1,\dots,N_1$, it generates the solution that has the first k components equal to the corresponding k components of $x(k)$ and the last $N-k$ components equal to the corresponding last $N-k$ components of $\overline{x}$.

Then, it can be seen that the rollout algorithm is equivalent to a coordinate descent method that starts from $\overline{x}$ and yields $(\tilde{x}_1,\dots,\tilde{x}_N)$ according to

$$\tilde{x}_1 \in \arg\min_{x_1\in X_1} g(x_1,\overline{x}_2,\dots,\overline{x}_N),$$

$$\tilde{x}_k \in \arg\min_{x_k\in X_k} g(\tilde{x}_1,\dots,\tilde{x}_{k-1},x_k,\overline{x}_{k+1},\dots,\overline{x}_N), \qquad k=2,\dots,N.$$

Note that a more general version of coordinate descent is obtained by using a base heuristic similarly defined by multiple solutions $\overline{x}^1,\dots,\overline{x}^m$ in place of $\overline{x}$. Then, the preceding equation is replaced by

$$\tilde{x}_1 \in \arg\min_{x_1\in X_1} \min\big\{g(x_1,\overline{x}_2^1,\dots,\overline{x}_N^1),\dots,g(x_1,\overline{x}_2^m,\dots,\overline{x}_N^m)\big\},$$

$$\tilde{x}_k \in \arg\min_{x_k\in X_k} \min\big\{g(\tilde{x}_1,\dots,\tilde{x}_{k-1},x_k,\overline{x}_{k+1}^1,\dots,\overline{x}_N^1),\dots,$$

$$g(\tilde{x}_1,\dots,\tilde{x}_{k-1},x_k,\overline{x}_{k+1}^m,\dots,\overline{x}_N^m)\big\}, \quad k=2,\dots,N.$$

The preceding example provides a baseline. It shows what can be achieved in a coordinate-based formulation of the rollout algorithm, even with a very trivial base heuristic. One can expect much better performance with more sophisticated base heuristics.

There are interesting variations of the coordinate-based reformulation of the generic discrete optimization problem into a graph search problem. In particular, the problem has been reformulated so that the components of $x_1,\dots,x_N$ are selected in a specific order. Alternative orders are possible, and in fact an attempt to optimize the choice of order may be effected through the rollout algorithm. This can be done by introducing the index of the component of x as part of the specification of a node. In particular, the nodes in the "layer" I_k of the graph may have the form $(x(k),k,n)$ where n specifies the next component to be chosen by the rollout algorithm.

Let us also explore the relation between rollout algorithms and local search methods, which are a broad and important class of heuristics for the generic discrete optimization problem of minimizing $g(x)$ over the finite set X. A local search method uses the notion of a *neighborhood* $N(x)$ of a feasible solution $x \in X$, which is a (usually small) subset of X, containing solutions that are "close" to x in some sense.

In particular, given a solution x, the method selects among the solutions in the neighborhood $N(x)$ a successor solution x', according to some rule. The process is then repeated with x' replacing x (or stops when some termination criterion is met). Thus, a local search method is characterized by:

(a) The method for choosing a starting solution
(b) The definition of the neighborhood $N(x)$ of a solution x
(c) The rule for selecting a successor solution from within $N(x)$
(d) The termination criterion

The definition of a neighborhood often involves intricate calculations and suboptimizations that aim to bring to consideration promising neighbors. While the definition of neighborhood is typically problem dependent, some general classes of procedures for generating neighborhoods have been developed. An example of such a class is the well-known *genetic algorithms*.

The criterion for selecting a solution from within a neighborhood is usually the cost of the solution, so that a neighbor of minimum cost is selected. Then, by assuming that each $x \in X$ belongs to its own neighborhood $N(x)$, the local search is *cost improving* and effectively stops at a *local minimum*, that is, a solution that is no worse than all other solutions within its neighborhood. Attention will be restricted to such methods, but there are important alternatives, such as in the methods of *tabu search* and *simulated annealing*, which will not be discussed here.

Consider a cost-improving local search method, as described above, and let $N(x)$ and $\overline{x}$ be the neighborhood definition and the starting point of the method, respectively. Let us assume that there is a given limit M to the number of iterations, so that the method terminates when it reaches this limit (if it encounters a local minimum before M iterations are performed, it may be assumed that it simply repeats the local minimum, until the limit M is reached). We will provide a reformulation of the problem into the framework of the search problem of the preceding section, so that the rollout algorithm becomes identical to the local search method described above.

To this end, an acyclic graph is introduced, which consists of an origin node that corresponds to the starting solution $\overline{x}$, and M subsets of nodes $I_1, I_2, \ldots, I_M$, which may be viewed as replicas of the feasible set X. In particular, for each feasible solution $x \in X$ and each $k = 1, \ldots, M$, the node set I_k contains a node (x, k). Each node $(x, M) \in I_M$ is viewed as a destination node of the graph and has cost $g(x)$. The origin node is connected with an arc to each node $(x, 1)$ such that $x \in N(\overline{x})$, while for every $k = 1, \ldots, M-1$, each node $(x, k) \in I_k$ is connected with an arc to each node $(x', k+1) \in I_{k+1}$ such that $x' \in N(x)$.

Consider now the base heuristic that, starting from a node $(x, k) \in I_k$, generates the destination (x, N) with cost $g(x)$. Then, it can be seen that the rollout algorithm reduces to the local search method. In particular, the rollout algorithm, given x after k steps [i.e., when at node (x, k)], it considers all x' in the neighborhood $N(x)$ and runs the base heuristic starting at x' and yielding the cost $g(x')$. It then selects $x' \in N(x)$ that yields the minimum cost. This is exactly what the local search method also does.

The preceding reformulation suggests that rollout algorithms can provide an additional dimension to local search methods, whereby intermediate infeasible solutions may be generated, and these solutions are evaluated through their projections, which are obtained through a base heuristic. Thus, while local search methods rely on a single construct, namely, neighborhoods, for selecting successive solutions, rollout algorithms bring to bear two independent constructs, neighborhoods *and* base heuristics. It should be mentioned also that rollout algorithms embody some additional important methodological ideas, namely, DP and policy iteration. For this reason, rollout algorithms admit natural extensions to stochastic control problems, for which there is no known analog of a local search method.

4 Further Reading

The main idea of rollout algorithms, obtaining an improved policy starting from some other suboptimal policy using a one-time policy improvement, has appeared in several DP application contexts. In the context of game-playing computer programs, it has been proposed by Abramson [1] and by Tesauro and Galperin [24]. The name "rollout" was coined by Tesauro in specific reference to rolling the dice in the game of backgammon. In Tesauro's proposal, a given backgammon position is evaluated by "rolling out" many games starting from that position, using a simulator, and the results are averaged to provide a "score" for the position. The internet contains a lot of material on computer backgammon and the use of rollout, in some cases in conjunction with multistep lookahead and cost-to-go approximation.

The application of rollout algorithms to discrete optimization problems has its origin in the neuro-dynamic programming work of the author and J. Tsitsiklis [6]. The formalization as a path construction algorithm, including the notions of sequential consistency, sequential improvement, and fortified and parallel rollout, was given in the paper by Bertsekas, Tsitsiklis, and Wu [7]. The subsequent paper by Bertsekas and Castanon [5] considered its application to stochastic DP and stochastic scheduling. The analysis of the breakthrough problem (Example 1) is given in the DP book by Bertsekas [3] and is based on unpublished work by Bertsekas, Castanon, and Tsitsiklis. An analysis of the optimal policy and some suboptimal policies for this problem is given by Pearl [18]. A discussion of rollout algorithms as applied to network optimization problems may be found in the author's network optimization book [2].

For applications of rollout algorithms, see Christodouleas [11], Duin and Voss [12], Secomandi [20–22], Ferris and Voelker [13, 14], McGovern, Moss, and Barto [16], Savagaonkar, Givan, and Chong [19], Bertsimas and Popescu [8], Guerriero and Mancini [15], Tu and Pattipati [25], Wu, Chong, and Givan [26], Chang, Givan, and Chong [10], Meloni, Pacciarelli, and Pranzo [17], Yan, Diaconis, Rusmevichientong, and Van Roy [27], Besse and Chaib-draa [9], and Sun, Zhao, Lun, and Tomastik [23]. These works discuss a broad variety of applications and case studies and generally report positive computational experience. The survey [4] discusses rollout algorithms from a control theory point of view and explores its close connection with model predictive control (MPC).

Recommended Reading

1. B. Abramson, Expected-outcome: a general model of static evaluation. IEEE Trans. Pattern Anal. Mach. Intell. **12**, 182–193 (1990)
2. D.P. Bertsekas, *Network Optimization: Continuous and Discrete Models* (Athena Scientific, Belmont, 1998)
3. D.P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. I (Athena Scientific, Belmont, 2005)
4. D.P. Bertsekas, Dynamic programming and suboptimal control: a survey from ADP to MPC, in *Fundamental Issues in Control*. Eur J. Control, **11**(4–5), 310–334 (2005)
5. D.P. Bertsekas, D.A. Castanon, Rollout algorithms for stochastic scheduling problems. Heuristics, **5**, 89–108 (1999)
6. D.P. Bertsekas, J.N. Tsitsiklis, *Neuro-Dynamic Programming* (Athena Scientific, Belmont, 1996)
7. D.P. Bertsekas, J.N. Tsitsiklis, C. Wu, Rollout algorithms for combinatorial optimization. Heuristics, **3**, 245–262 (1997)
8. D. Bertsimas, I. Popescu, Revenue management in a dynamic network environment. Transp. Sci. **37**, 257–277 (2003)
9. C. Besse, B. Chaib-draa, Parallel rollout for online solution of DEC-POMDPs, in *Proceedings of 21st International FLAIRS Conference*, Coconut Grove, FL, 2008, pp. 619–624
10. H.S. Chang, R.L. Givan, E.K.P. Chong, Parallel rollout for online solution of partially observable Markov decision processes. Discret. Event Dyn. Syst. **14**, 309–341 (2004)
11. J.D. Christodouleas, Solution methods for multiprocessor network scheduling problems with application to railroad operations, Ph.D. thesis, Operations Research Center, Massachusetts Institute of Technology, 1997
12. C. Duin, S. Voss, The pilot method: a strategy for heuristic repetition with application to the Steiner problem in graphs. Networks, **34**, 181–191 (1999)
13. M.C. Ferris, M.M. Voelker, Neuro-dynamic programming for radiation treatment planning. Numerical Analysis Group Research Report NA-02/06, Oxford University Computing Laboratory, Oxford University, 2002
14. M.C. Ferris, M.M. Voelker, Fractionation in radiation treatment planning. Math. Program. B **102**, 387–413 (2004)
15. F. Guerriero, M. Mancini, A cooperative parallel rollout algorithm for the sequential ordering problem. Parallel Comput. **29**, 663–677 (2003)
16. A. McGovern, E. Moss, A. Barto, Building a basic building block scheduler using reinforcement learning and rollouts. Mach. Learn. **49**, 141–160 (2002)
17. C. Meloni, D. Pacciarelli, M. Pranzo, A rollout metaheuristic for job shop scheduling problems. Ann. Oper. Res. **131**, 215–235 (2004)
18. J. Pearl, *Heuristics* (Addison-Wesley, Reading, 1984)
19. U. Savagaonkar, R. Givan, E.K.P. Chong, Sampling techniques for zero-sum, discounted Markov games, in *Proceedings of 40th Allerton Conference on Communication, Control and Computing*, Monticello, IL, 2002
20. N. Secomandi, Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. Comput. Oper. Res. **27**, 1201–1225 (2000)

21. N. Secomandi, A rollout policy for the vehicle routing problem with stochastic demands. Oper. Res. **49**, 796–802 (2001)
22. N. Secomandi, Analysis of a rollout approach to sequencing problems with stochastic routing applications. J. Heuristics, **9**, 321–352 (2003)
23. T. Sun, Q. Zhao, P. Lun, R. Tomastik, Optimization of joint replacement policies for multipart systems by a rollout framework. IEEE Trans. Autom. Sci. Eng. **5**, 609–619 (2008)
24. G. Tesauro, G.R. Galperin, On-line policy improvement using Monte Carlo search. Presented at the 1996 neural information processing systems conference, Denver, CO, 1996; also in *Advances in Neural Information Processing Systems 9*, ed. by M. Mozer et al. (MIT, 1997)
25. F. Tu, K.R. Pattipati, Rollout strategies for sequential fault diagnosis. IEEE Trans. Syst. Man Cybern. Part A **33**, 86–99 (2003)
26. G. Wu, E.K.P. Chong, R.L. Givan, Congestion control using policy rollout, in *Proceedings of 2nd IEEE CDC*, Maui, HI, 2003, pp. 4825–4830
27. X. Yan, P. Diaconis, P. Rusmevichientong, B. Van Roy, Solitaire: man versus machine. Adv. Neural Inf. Process. Syst. **17**, 1553–1560 (2005).