A Reach and Bound Algorithm for Acyclic Dynamic Programming Networks *

Matthew D. Bailey
Department of Industrial Engineering
University of Pittsburgh

Robert L. Smith
Department of Industrial and Operations Engineering
University of Michigan

Jeffrey M. Alden
General Motors Research and Development Center

June 18, 2007

Abstract

Node pruning is a commonly used technique for solution acceleration in a dynamic programming network. In pruning, nodes are adaptively removed from the dynamic programming network when they are determined to not lie on an optimal path. We introduce an *ε-pruning* condition that extends pruning to include a possible error in the pruning step. This results in a greater reduction of the computation time; however,

*This work was supported in part by a National Science Foundation GOALI (DMI-9900267) and a grant from General Motors

as a result of the inclusion of this error, the solution can be sub-optimal or possibly infeasible. This condition requires the ability to compare the costs of an optimal path from a node to a terminal node. Therefore, we focus on the class of acyclic dynamic programming networks with monotonically decreasing optimal costs-to-go. We provide an easily implementable algorithm, *Reach and Bound*, which maintains feasibility and bounds the solution's error. We conclude by illustrating the applicability of Reach and Bound on a problem of single location capacity expansion.

Keywords: Dynamic Programming Networks, Shortest Path Problem, Pruning

1 Introduction

We confront the intractability of large-scale dynamic programs (DPs) through approximation methods that build on the success of known optimal solution techniques. We introduce an error-bounded pruning device to eliminate nodes that fail to be within a user-specified value of being potentially optimal. We provide an efficient implementation for the class of deterministic acyclic dynamic programs with nonnegative costs. We develop the notion of ε-pruning and construct an algorithm, Reach and Bound, that systematically incorporates this pruning device, while bounding the overall error. Reach and Bound is most effective in DP networks that are simultaneously created and solved, so that "descendant" nodes and arcs of pruned nodes may never be generated. In Section 4, we introduce a problem of this type in capacity expansion and provide computational results for an application of Reach and Bound.

2 Node Pruning

We begin with an acyclic network $\mathcal{N} = (N, A, C)$, where $N = \{0, 1, 2, \ldots, n\}$ is the set of nodes, $A \subseteq \{(i, j) \mid i, j \in N\}$ is the set of arcs, and C is the set of nonnegative arc costs c_{ij}. We assume that the nodes have a topological ordering, where if $(i, j) \in A$ then $i < j$ with origin $0 \in N$ and destination $n \in N$. We can, and often will, view the problem as searching for a shortest path in the DP network from node 0 to node n, where the length of an arc

is its associated arc cost. When a node i is *pruned* from the network $\mathcal{N}$, $\mathcal{N}$ is replaced by $\mathcal{N} \setminus \{i\}$, where we define $\mathcal{N} \setminus T$ as the resulting network when the node set N is replaced by $N \setminus T$ and the arc set A is replaced by $A \setminus \{(i,j) \in A \mid i \in T \text{ or } j \in T\}$. We define $\mathcal{P}_{ij}(\mathcal{N})$ as the set of paths from node i to node j in network $\mathcal{N}$, where paths are defined by a sequence of nodes in N. The cost or length of a path p is defined as the sum of the arc costs between consecutive nodes on the path and is denoted as $l(p)$. Under this framework, we provide the following definitions.

$h(s,\mathcal{N}) = \min\{l(p) \mid p \in \mathcal{P}_{0,n}(\mathcal{N}), s \in p\}$, i.e., the length of an optimal path from the origin to the destination where node $s \in N$ is an intermediate node on the path.

$f(s,\mathcal{N}) = \min\{l(p) \mid p \in \mathcal{P}_{0,s}(\mathcal{N})\}$, i.e., the length of an optimal path from the origin to node s in network $\mathcal{N}$.

$g(s,\mathcal{N}) = \min\{l(p) \mid p \in \mathcal{P}_{s,n}(\mathcal{N})\}$, i.e., the length of an optimal path from node s to the destination node n in network $\mathcal{N}$. This will also be referred to as the *optimal cost-to-go* from node s.

$f^*(\mathcal{N}) = \min\{l(p) \mid p \in \mathcal{P}_{0,n}(\mathcal{N})\}$, i.e, the cost of an optimal path from the origin to the destination in network $\mathcal{N}$.

Since we are restricted to acyclic networks, it is clear that $h(s,\mathcal{N}) = f(s,\mathcal{N}) + g(s,\mathcal{N})$. It is necessary to define each of these values with respect to the current network $\mathcal{N}$, since the pruning of nodes could potentially change their values.

Under standard pruning, node $s \in N$ prunes node $t \in N$ if it is known that the cost of an optimal path through s is less than or equal to the cost of an optimal path through t. By construction, the cost of the pruned network's optimal solution is identical to the cost of the original network's optimal solution. However, the cost of an optimal path *through* a node is typically not known and instead the following *standard pruning condition* is used:

$$\begin{aligned} f(s,\mathcal{N}) &\leq f(t,\mathcal{N}) \text{ and} \\ g(s,\mathcal{N}) &\leq g(t,\mathcal{N}). \end{aligned} \tag{1}$$

By summing the two inequalities in (1), we conclude that $h(s,\mathcal{N}) \leq h(t,\mathcal{N})$. Therefore, the standard pruning condition implies that the value of an optimal path through node s is at

most the value of an optimal path through t. If there does not exist a zero cost path between s and t, we are assured that t cannot lie on an optimal path through s, since one of the two conditions would be violated. In general, it is only necessary to know that these conditions hold; the explicit costs $f(s,\mathcal{N})$, $f(t,\mathcal{N})$, $g(s,\mathcal{N})$, and $g(t,\mathcal{N})$ are not necessary. As a result, we will assume that knowledge about the relationships between the optimal cost-to-go from the nodes can be inferred through problem structure or known bounds. Typically, in solution algorithms for DP networks, the costs of optimal paths *to* a subset of nodes, i.e., $f(s,\mathcal{N})$, are computed as the algorithm progresses (see for example [4]). We will assume that this information is available for a subset of nodes and will illustrate this within the algorithm Reach and Bound.

According to Denardo and Fox [5], pruning was first used by Gilmore and Gomory [7] for the group knapsack problem. Denardo and Fox [5] investigated pruning as an accelerating device applied to the reaching algorithm. Reaching is a label-setting algorithm that allows the exploitation of network structure. Their technique was primarily used to remove arcs emanating from a node that could not lie on an optimal path. Pearl and Kim [12] developed an error-bounded solution acceleration technique based on the algorithm A^* by allowing an error in the node selection sequence criteria. Although their technique is related to that presented here, it does not explicitly prune the nodes. The nodes that are passed over for selection are still eligible for subsequent selection. Lark et al. [8] also developed a variant of A^* for which the cost-to-go estimate can be used to eliminate sub-optimal nodes. Pruning has also been successfully applied as an accelerating device in several applications. Nemhauser and Ullman [11] applied reaching with pruning to capital budgeting. Morin and Marsten [9] applied the technique to resource allocation. Morin and Marsten [10] also view pruning as an application of the branch-and-bound technique from integer programming. In their applications, they assumed the existence of a bound on the optimal cost and used this to prune nodes for which an optimal path through the node exceeded the bound. This bound is found from a known feasible solution or from problem structure. They illustrated the technique by applying it to the traveling-salesman problem and to the nonlinear knapsack problem. Building on this paper, Easton [6] investigated the relationship between the quality of the bound on the optimal cost and the solution time as applied to the assembly line

balancing problem. Alden and Yano [2] defined standard pruning and successfully applied it to a lot sizing problem to significantly reduce the size of the network.

An extension of the type of pruning performed by Gilmore and Gomory [7] was applied to the single location capacity expansion problem by Smith [14]. In this instance, the author found significant time savings, since the pruning was performed simultaneously as the network was created. The pruning of a node resulted in descendant nodes not being created. Thereby, it was possible that with the removal of one node an entire subnetwork of the DP network was extracted. This motivates the inclusion of a pruning error, which will allow us to prune an even greater number of nodes at the expense of a possible error in the final solution.

2.1 ε-Pruning

ε-Pruning is an extension of standard pruning that allows the possibility that the cost of a path through the pruned node is less than the cost of the path through the pruning node. By expanding the class of nodes that are pruned by a node, the reduction in computation time will be even more significant. However, such pruning may result in a higher cost solution. We will show that under certain conditions the resulting error will be contained within an a priori user-defined error bound.

We adjust the standard pruning condition (1) to include an error when pruning. The *ε-pruning condition* is then

$$\begin{aligned} f(s,\mathcal{N}) &\leq f(t,\mathcal{N}) + \varepsilon \text{ and} \\ g(s,\mathcal{N}) &\leq g(t,\mathcal{N}). \end{aligned} \tag{2}$$

Although an allowable error could also be included in the relationship between the costs-to-go, we restrict the allowable error to the relationship between the costs to the nodes.

By summing the two inequalities in (2), we see that it is possible that the cost of the optimal path through t is actually less than the cost of the optimal path through s, but it is at most ε less. As a result of using condition (2), the complexity of the network can be further simplified as more nodes may be pruned using (2) than by using condition (1) exclusively.

There can often be multiple optimal paths to a node; however, to maintain the error bound we need only maintain one of these paths. As a result, we select a designated optimal path to a node. It is assumed that the designated optimal path to node j consists of the designated optimal paths to each node that lies on the designated optimal path to j. By construction, this will be satisfied in Reach and Bound.

2.1.1 ε-Pruning Complications

As stated previously, using the standard pruning conditions will result in no loss of optimality. However, the same is not true using condition (2). As a result, the error bound may no longer be valid. Therefore, *to maintain the validity of the error bound, we restrict pruning to nodes not on the designated optimal path through the current pruning node.* It is actually only necessary to maintain any path whose cost is at most the cost of an optimal path through s. However, an optimal path through s is typically not known. In the absence of this information, we maintain the designated optimal path through s to guarantee that the final solution cost is at most ε more than the cost of the optimal path.

A complication also arises with ε-pruning in that the error in the resulting solution can accumulate. We will show that the error in the resulting solution is nonetheless at most $\sum_{i=1}^{p} \epsilon_i$, where p is the number of nodes used for pruning and ϵ_i is the allowable error associated with the i^{th} pruning node. In addition, we will show that this bound can be tightened and refined as the algorithm progresses.

3 Reach and Bound

In order to implement the ε-pruning condition for two nodes, it is necessary to compare the costs of an optimal path to the nodes and the optimal costs-to-go from the nodes. We will compute bounds on the optimal costs to nodes and assume the following structure:

(A1) For all nodes i and j if $i < j$ then $g(j, \mathcal{N}) \leq g(i, \mathcal{N})$.

We assume that the DP network has monotonically decreasing optimal costs-to-go. We expect to see the greatest computational gains in problems for which the network is created

as we discover a solution. We therefore define $succ(i)$ as the set of successor nodes of node i: i.e., $succ(i) = \{j \mid (i,j) \in A\}$.

Standard dynamic programming formulations of many problems in various applications arise that have monotonically decreasing optimal costs-to-go. Specific examples include production line design [1] and single location capacity expansion [14]. The applications allow us to verify assumption A1 a priori. This structure can be exploited to create a greatly simplified algorithm for utilizing ε-pruning. The algorithm presented will be an extension and generalization of the reaching algorithm presented in Smith [14], which was motivated by the reaching algorithm presented in Gilmore and Gomory [7] and in Shapiro and Wagner [13]. We will first present the algorithm and its properties for acyclic DP networks that satisfy assumption A1 and then directly apply the technique to the single location capacity expansion problem in the next section. We will show that nodes can be pruned based only on the cost of a *feasible* path and still maintain the error bound.

3.1 Reach and Bound Algorithm

The Reach and Bound algorithm is an extension of a label-setting algorithm. The labels of the nodes, v_j, are upper bounds for the cost of an optimal path to the node in the current network, i.e., an upper bound for $f(j, \mathcal{N})$. Initially, they are all set to zero and the algorithm begins by reaching from or expanding node $s_0 = 0$, the origin, where *expanding* a node means updating the labels, v_j, of all nodes $j \in succ(s_0)$. After expanding node s_i, we select s_{i+1}, the next node that will be expanded. We first determine the set of unexpanded nodes j for which v_j is at most ε more than the label of any other unexpanded node. From this set, we select the largest node with respect to the topological ordering and define this as s_{i+1}. As illustrated in Figure 1, this effectively prunes the nodes numbered between s_i and s_{i+1}. It is assumed that the user has specified the sequence of pruning step errors allowed, $\{\varepsilon_1, \varepsilon_2, \ldots\}$. This sequence is associated with $\{s_1, s_2, \ldots\}$, the sequence of pruning nodes selected in the algorithm. By construction, at iteration i the current label v_j is the minimum cost of traveling through the previous pruning nodes $\{s_0, s_1, \ldots, s_i\}$ to j.

Reach and Bound Algorithm

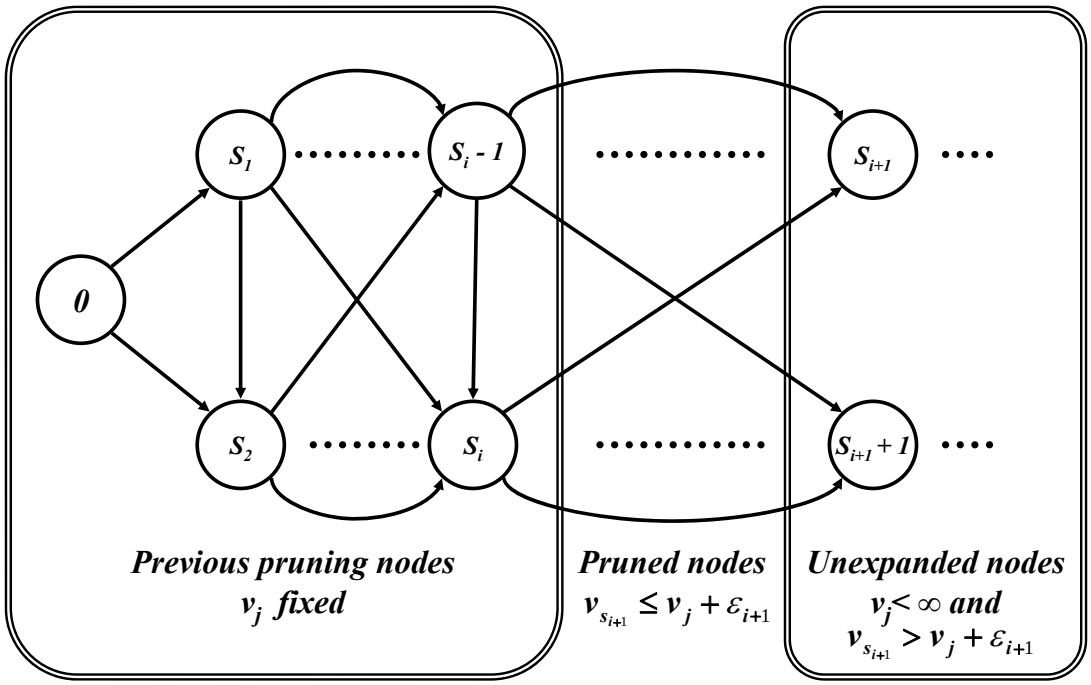

Figure 1: Reach and Bound in networks with monotonically decreasing costs-to-go.

1. Set $i = 0$, $s_0 = 0$, $v_0 = 0$, $N = \{0\}$.

2. For each $j \in succ(s_i)$, if $j \notin N$ set $v_j = v_{s_i} + c_{s_i,j}$ and $N = N \cup \{j\}$, otherwise let

$$v_j \leftarrow \min\{v_j, v_{s_i} + c_{s_i,j}\}.$$

3. Set s_{i+1} to the largest $k > s_i, k \in N$ such that $v_k \leq \min\{v_j \mid j > s_i, j \in N\} + \varepsilon_{i+1}$.

4. Set $i = i + 1$. If $s_i < n$, go to Step 2, otherwise stop.

During iteration i, any node t such that $s_i < t < s_{i+1}$ is pruned from the network, i.e., any node passed over for expansion is removed from the network. These nodes can be explicitly removed from the generated network; however the computational savings is a result of not exploring paths through these nodes, so the added effort to explicitly remove them is unnecessary. Therefore, at the beginning of iteration i, the current subnetwork only contains nodes with topological node numbering less than s_i that were expanded and all nodes with numbering greater than or equal to s_i that are descendants of these nodes.

3.1.1 Satisfying ε-Pruning Conditions

In Reach and Bound, we are pruning nodes using upper bounds on the optimal cost v_j to each of the nodes in the current network. However, as the following results show, the relationships of these upper bounds are sufficient to satisfy the previous ε-pruning conditions. If we define m_{i+1} as the highest numbered node m greater than s_i such that $v_m = \min_{j>s_i} v_j$ in the current subnetwork of iteration i, then this is the node that would be expanded if $\varepsilon_{i+1} = 0$ for iteration i. The following lemma states that the label of node m_{i+1} is less than or equal to the cost of an optimal path in the current subnetwork to any node greater than s_i.

Lemma 1 *At the conclusion of Step (2) in iteration i of* Reach and Bound*, if node s_i was the last node to be expanded within the current subnetwork $\mathcal{N}_i$, then $f(j, \mathcal{N}_i) \geq v_{m_{i+1}} \geq f(m_{i+1}, \mathcal{N}_i)$ for all $j > s_i$.*

Proof. By construction $v_{m_{i+1}}$ is the cost of an optimal path to m_{i+1} restricted to the previously expanded nodes in $\mathcal{N}_i$, therefore $v_{m_{i+1}} \geq f(m_{i+1}, \mathcal{N}_i)$.

We show $f(j, \mathcal{N}_i) \geq v_{m_{i+1}}$ by induction on $j > s_i$. For $j = s_i + 1$, since all of its ancestors in $\mathcal{N}_i$ have been previously expanded

$$\begin{aligned} f(s_i + 1, \mathcal{N}_i) &= v_{s_i+1} \\ &\geq v_{m_{i+1}} \text{ by definition.} \end{aligned}$$

Now assume

$$v_{m_{i+1}} \leq f(j, \mathcal{N}_i), \text{ for } s_i < j \leq k.$$

Now, for node $k+1$ either its optimal ancestor is after s_i, i.e.,

$$f(k+1, \mathcal{N}_i) = f(t, \mathcal{N}_i) + c_{t,k+1} \text{ for some } s_i < t < k+1,$$

or its optimal ancestor is before s_i, i.e.,

$$f(k+1, \mathcal{N}_i) = f(t, \mathcal{N}_i) + c_{t,k+1} \text{ for some } 1 \leq t \leq s_i.$$

If $s_i < t < k+1$ then

$$\begin{aligned} f(k+1, \mathcal{N}_i) &= f(t, \mathcal{N}_i) + c_{t,k+1}, \\ &\geq v_{m_{i+1}} + c_{t,k+1}, \text{ by assumption,} \\ &\geq v_{m_{i+1}}, \text{ since } c_{t,k+1} \geq 0. \end{aligned}$$

If $1 \leq t \leq s_i$ then

$$\begin{aligned} f(k+1, \mathcal{N}_i) &= f(t, \mathcal{N}_i) + c_{t,k+1}, \\ &= v_{k+1} \text{ by construction}, \\ &\geq v_{m_{i+1}}, \text{ by definition.} \end{aligned}$$

□

Using the above result, we show that a pruning node s_{i+1} in Reach and Bound satisfies the ε-pruning condition (2).

Proposition 1 *For a given $\varepsilon_{i+1} \geq 0$ and computed pruning node s_{i+1}, at the conclusion of Step (2) in iteration i of* Reach and Bound*,*

$$\begin{aligned} f(s_{i+1}, \mathcal{N}_i) &\leq f(j, \mathcal{N}_i) + \varepsilon_{i+1} \text{ and} \\ g(s_{i+1}, \mathcal{N}_i) &\leq g(j, \mathcal{N}_i) \end{aligned}$$

for $s_i < j < s_{i+1}$, where node s_i was the last node to be expanded in $\mathcal{N}_i$.

Proof.

$$\begin{aligned} f(s_{i+1}, \mathcal{N}_i) &\leq v_{s_{i+1}}, & (3) \\ &\leq v_{m_{i+1}} + \varepsilon_{i+1}, \text{ by definition}, \\ &\leq f(j, \mathcal{N}_i) + \varepsilon_{i+1}, \text{ by Lemma 1 for } s_i < j < s_{i+1}. & (4) \end{aligned}$$

Since $s_{i+1} > j$, then

$$g(s_{i+1}, \mathcal{N}_i) \leq g(j, \mathcal{N}_i),$$

by assumption A1. □

From this proposition, we know that at iteration i for a fixed $\varepsilon_{i+1} > 0$, the cost of the designated optimal path through node s_{i+1} is at most ε_{i+1} more than the optimal cost through all nodes j, $s_i < j < s_{i+1}$. Therefore, if we remove all such nodes j from $\mathcal{N}_i$ and maintain the designated optimal path through s_{i+1}, then the cost of an optimal path in the resulting subnetwork is at most ε_{i+1} more than the cost of an optimal path in $\mathcal{N}_i$.

Reach and Bound does not actually check whether the designated optimal path through s_{i+1} is maintained; *instead it maintains a path through* s_{i+1} *with the cost of the path to* s_{i+1} *given by the label* $v_{s_{i+1}}$. However, in the following proposition we show that maintaining such a feasible path is sufficient to maintain the fixed error bound for an individual pruning step.

Proposition 2 *For a given* $\varepsilon_{i+1} \geq 0$,

$$v_{s_{i+1}} + g(s_{i+1}, \mathcal{N}_i) \leq h(j, \mathcal{N}_i) + \varepsilon_{i+1},$$

for $s_i < j < s_{i+1}$.

Proof. For $s_i < j < s_{i+1}$,

$$v_{s_{i+1}} \leq f(j, \mathcal{N}_i) + \varepsilon_{i+1},$$

by inequalities (3) and (4). Hence,

$$v_{s_{i+1}} + g(s_{i+1}, \mathcal{N}_i) \leq h(j, \mathcal{N}_i) + \varepsilon_{i+1},$$

by assumption A1. □

Then, from Lemma 1, we know that $f(m_{i+1}, \mathcal{N}_i) \leq f(j, \mathcal{N}_i)$ for $s_i < j < m_{i+1}$. Also, by assumption A1, $h(m_{i+1}, \mathcal{N}_i) \leq h(j, \mathcal{N}_i)$. Therefore, if all nodes j, where $s_i < j < m_{i+1}$, are removed from the network, then an optimal path in the current network is still intact. In other words, the only error incurred in this technique is by pruning node m_{i+1} and the nodes between m_{i+1} and s_{i+1}. In addition, from Proposition 2, if all nodes k, where $m_{i+1} \leq k < s_{i+1}$, are removed then there still exists a path in the pruned network with cost $v_{s_{i+1}} + g(s_{i+1}, \mathcal{N}_i)$. The costs of the paths removed were at worst ε_{i+1} less than $v_{s_{i+1}} + g(s_{i+1}, \mathcal{N}_i)$. Therefore, the cost of an optimal path in the resulting pruned network is within ε_{i+1} of the cost of an optimal path in the network $\mathcal{N}_i$. As a result, the error in the total solution is at most $\sum_{k=1}^{i} \varepsilon_k$ after i iterations of Reach and Bound.

3.1.2 Error Bound Updating

Although the maximum allowed error ε_{i+1} at any pruning step is user-defined, in Reach and Bound we determine a tighter bound on the actual maximum error incurred. The amount

of the error bound "used" can be tracked during the solution so that a more accurate error bound can be reported.

Lemma 2 *After Step (2) of iteration i of* Reach and Bound *for networks with nonincreasing costs-to-go, an upper bound on the actual maximum error incurred for that pruning step is*

$$v_{s_{i+1}} - v_{m_{i+1}}.$$

Proof. For every $s_i < j < s_{i+1}$,

$$\begin{aligned} h(s_{i+1}, \mathcal{N}_i) - h(j, \mathcal{N}_i) &= f(s_{i+1}, \mathcal{N}_i) + g(s_{i+1}, \mathcal{N}_i) - f(j, \mathcal{N}_i) - g(j, \mathcal{N}_i), \\ &\leq f(s_{i+1}, \mathcal{N}_i) + g(s_{i+1}, \mathcal{N}_i) - v_{m_{i+1}} - g(j, \mathcal{N}_i), \text{ by Lemma 1,} \\ &\leq v_{s_{i+1}} + g(s_{i+1}, \mathcal{N}_i) - f(m_{i+1}, \mathcal{N}_i) - g(j, \mathcal{N}_i) \qquad (5) \\ &\leq v_{s_{i+1}} - v_{m_{i+1}}, \text{ by assumption A1.} \end{aligned}$$

□

As we see from inequality (5), a tighter bound can be obtained if the difference $g(s_{i+1}, \mathcal{N}_i) - g(j, \mathcal{N}_i)$ can also be bounded by a greater bound than zero.

If the user specifies an a priori total acceptable error ϵ from which to allocate the per iteration error bounds ε_i, this error could be adaptively partitioned and allocated to each pruning iteration. Initial iterations may be allocated a larger amount of the error bound to, for example, prune a larger number of nodes initially. The ε_i would be set to zero when $\sum_{k=0}^{i} \left(v_{s_{k+1}} - v_{m_{k+1}}\right) = \epsilon$. Such allocation schemes would be highly application-dependent and are beyond the scope of this paper.

4 Application to Single Location Capacity Expansion

To illustrate the potential effectiveness of Reach and Bound, we focus on the problem of single location capacity expansion. This is an example of the problem investigated by Smith [14]. In this work, he considered the problem of sequentially selecting capacity additions from a finite set of possible facility sizes to meet growing demand at a single location over an infinite decision horizon. When the current capacity is exhausted, an additional facility

must be selected to expand capacity. It is shown that there exists an optimal policy such that after a time t^*, the least average-cost facility is repeatedly installed. As a result, once t^* has been determined, the problem reduces to a dynamic program over a horizon of length t^*. The problem is to find, at minimum cost, a sequence of facility installations, in which the additional capacity of the last facility ensures that the total capacity will not be exhausted until after t^*.

We will adopt the same notation and assumptions as in [14]. We assume exponential growth in demand for capacity at time t, i.e., $D(t) = D_0(e^{dt} - 1)$ where $D_0 > 0$ is the initial demand completely met by the installed capacity and $d > 0$ is the demand growth rate. Additional demand can be met by any of a finite set of distinct facilities $1, 2, \ldots, n$ where a facility provides capacity. Each facility l has fixed cost $F_l > 0$ and an integer capacity $X_l > 0$, which is incurred at the time of installation. All costs are discounted continuously using an interest rate r. We alter the problem slightly and seek a sequence of facility installations that satisfies demand for capacity over a finite horizon at a cost within a specified error of the minimum discounted cost. We call this error ε. An installation epoch is defined as a time at which the installed capacity is exhausted and the next facility is installed. We stage on installation epochs and restate the problem as the following dynamic program:

> Find $g(0)$ where $g(t) = \min_{l=1,2,\ldots,n}\{F_l + e^{-r\tau_l(t)}g(t + \tau_l(t))\}$ and $\tau_l(t)$ is the time from installation of facility l at time t until its capacity is exhausted when installed. We define $g(t) = 0$ for $t \geq t^*$.

Since $D(t + \tau_l(t)) = D_0(e^{d(t+\tau_l(t))} - 1) = D(t) + X_l$, then $\tau_l(t) = d^{-1}\ln(1 + \frac{X_l}{D_0 e^{dt}})$. In a typical telephone transmission facility application, there are about 10^{16} potential installation sequences.

4.1 Reach and Bound in Capacity Expansion

The algorithm begins by stepping-off at time zero with each possible facility type. Thus it generates n potential installation epochs/nodes. Figure 2 illustrates how the network is dynamically created from node t and three facility choices with all installation costs converted to their present values at the initial installation epoch. The pruning device of rejecting

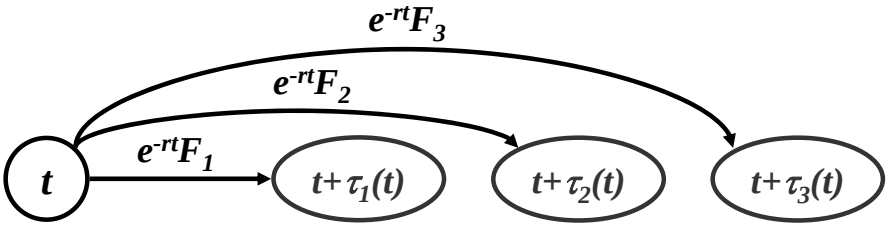

Figure 2: Capacity expansion DP network generation.

potential installation epochs of ε-dominated partial sequences leads to the next installation epoch, node s_{i+1}. This is repeated successively until all candidate sequences span the finite horizon time t^*. When node t is expanded, v_t is the minimum discounted cost of reaching time t in the pruned network.

In the original DP network, for two arbitrary times t and t', if $t > t'$ then the optimal sequence of facility installations from t that spans t^* could also be installed from t'. In addition, since the costs are discounted, the cost of the optimal sequence of facilities that span t^* from t is at least the cost of the optimal sequence of facilities from t'. Therefore the original dynamic programming network satisfies assumption A1. As a result, the number of nodes pruned by the algorithm can be significantly increased by allowing an error in the resulting solution.

Reach and Bound in Capacity Expansion Algorithm

1. Set $i = 1$, $s_1 = 0$, $v_0 = 0$, $N = \{0\}$

2. For $l = 1, 2, \ldots, n$, let $t = s_i + \tau_l(s_i)$. If $t \notin N$ set $v_t = v_{s_i} + F_l e^{-rs_i}$ and $N = N \cup \{t\}$, otherwise, let
$$v_t \leftarrow \min\{v_t, v_{s_i} + F_l e^{-rs_i}\}.$$

3. Set s_{i+1} to the largest time $k > s_i$ such that $v_k \leq \min\{v_j \mid j > s_i, j \in N\} + \varepsilon_{i+1}$.

4. Set $i = i + 1$. If $s_i < t^*$, go to Step 2, otherwise stop.

This technique is a slight variant of the Reach and Bound algorithm described earlier, since the network is dynamically created as the problem is solved. At Step 2, we expand the current

node by investigating the capacity exhaust epochs created by installing the n facilities from that epoch. At Step 3, the next node to be expanded, s_{i+1}, is selected, thus pruning all nodes/epochs t, where $s_i < t < s_{i+1}$. In Reach and Bound, we find a feasible solution whose cost is within $\sum_{j=1}^{k} \varepsilon_j$ of the optimal cost for the partial decision sequence, where k is the total number of iterations.

4.2 Computational Example

We illustrate the above algorithm on the following data with four hundred facility choices over 90 periods. We are given a set of facility sizes X_l in thousands of circuits, where $X_l = 2l$ for $l = 1, \ldots, 400$, with fixed cost $K(X)$ in thousands of dollars given by the following Dixon-Clapp law [15]:

$$K(X) = KX^{1-\alpha},$$

where K is a constant and α is the economy of scale factor. The values used for our example and all other necessary data are given in Table 1.

Initial Demand (thousands of circuits)	Growth rate	Interest rate	D-C constant	Economy of scale factor
10	0.09	0.11	10	0.5

Table 1: Transmission facilities data.

This problem was solved using standard reaching and several variations of the Reach and Bound algorithm coded in C++ on a 3.4 GHz Pentium 4 with 1.00 GB of RAM. Using Reaching, the cost of the optimal solution was found to be \$80,571 after 1.66 seconds. For simplicity, the allowable error per iteration was set at a fixed value. The value of the fixed ε_i allowed per pruning iteration was incremented from fifty dollars to five hundred dollars. For each solution instance, the total error bound was adaptively updated as described in Section 3.1.2. The actual error, updated error bound, and computation time are provided in Table 2. Costs are reported in thousands of dollars; errors are reported in percentage of optimal cost; and computation time is in seconds.

Allowable Error per Iteration	% Total Nodes Expanded	Solution Cost	Actual % Error from Optimal	Reported % Error Bound	Run Time
0.00	100	80.571	0	0.00	3.81
0.05	1.72	80.633	0.077	8.78	0.08
0.10	1.22	80.6206	0.062	11.00	0.06
0.15	0.98	80.6581	0.108	13.50	0.05
0.20	0.86	80.6395	0.085	15.21	0.04
0.25	0.77	80.5959	0.031	16.92	0.03
0.30	0.72	80.6309	0.074	18.23	0.03
0.35	0.69	80.6495	0.097	18.73	0.03
0.40	0.65	80.6006	0.037	19.34	0.03
0.45	0.63	80.5883	0.021	19.75	0.03
0.50	0.60	80.6618	0.113	20.85	0.03

Table 2: Reach and Bound results in capacity expansion.

We see from Table 2 that there is a large deviation between the actual error incurred and the reported error bound. We have found, in practice, that these bounds are typically very loose. Part of this is explained by the deviation in the optimal cost-to-go from a node and the pruned nodes. We use a worst-case bound on this value and assume that this deviation is zero. One might expect that the actual error in the solution would monotonically increase as the error allowed per iteration is increased. However, these are only bounds on the error and as a result of problem structure, we could see a slightly improved solution with a larger allowance for error.

Typically, such an installation problem is solved for each location in a large-scale transmission network and for each evaluated route in a network routing heuristic [16]. As a result, the single location capacity expansion problem is solved repeatedly for hundreds of locations. The example illustrates the potential power of Reach and Bound, since we are able to compute a solution within 1% of optimal in almost an order of magnitude less time than it takes to find the optimal solution. The extension of this solution technique over all locations for

each evaluated transmission routing would result in significant computational savings for a network routing heuristic.

5 Conclusions

ε-Pruning is an error bounded solution acceleration technique based on a known zero-error technique. By permitting an error in the pruning, we can quickly find a feasible solution with a bounded error. We provided conditions for which ε-pruning can be applied to maintain an error bound and provided an efficient, simplified algorithm, Reach and Bound. The algorithm can be used alone to quickly find a quality sub-optimal solution or as an iterative step within a larger heuristic. As shown, Reach and Bound is easily implementable and can significantly decrease the portion of the network explored while controlling the quality of the resulting solution. The effectiveness of the algorithm and the error bound can be improved through an application-specific allocation of the errors per iteration.

References

[1] J. Alden, M. Bailey, and R. Smith, A dynamic programming formulation for the assembly line design problem, Technical report 00-06, University of Michigan, 2000.

[2] J. Alden and C. Yano, A forward dynamic programming approach for general uncapacitated multi-stage lot-sizing problems, Technical report 86-11, University of Michigan, 1986.

[3] M. Bailey, Approximate solution techniques for acyclic, deterministic dynamic programming, Ph.D. Thesis, University of Michigan, 2002.

[4] E. Denardo, Dynamic programming: Theory and applications, Prentice Hall, Englewood Cliffs, N.J., 1980.

[5] E. Denardo and B. Fox, Shortest-route methods: 1. reaching, pruning, and buckets, Oper Res 27 (1979), 161–186.

[6] F. Easton, A dynamic program with fathoming and dynamic upper bounds for the assembly line balancing problem, Comput Oper Res 17 (1990), 163–175.

[7] P. Gilmore and R. Gomory, The theory and computation of knapsack functions, Oper Res 14 (1966), 1045–1074.

[8] J. Lark, C. White, and K. Syverson, A best-first search algorithm guided by a set-valued heuristic, IEEE Trans Syst, Man, Cybernetics 25 (1995), 1097–1101.

[9] T. Morin and R. Marsten, An algorithm for nonlinear knapsack problems, Manage Sci 22 (1976), 1147–1158.

[10] T. Morin and R. Marsten, Branch-and-bound strategies for dynamic programming, Oper Res 24 (1976), 611–627.

[11] G. Nemhauser and Z. Ullman, Discrete dynamic programming and capital allocation, Manage Sci 15 (1969), 494–505.

[12] J. Pearl and J. Kim, Studies in semi-admissible heuristics, IEEE Trans Pattern Anal Machine Intelligence 4 (1982), 392–399.

[13] J. Shapiro and H. Wagner, A finite renewal algorithm for the knapsack and turnpike models, Oper Res 15 (1967), 319–341.

[14] R. Smith, Turnpike results for single location capacity, Manage Sci 25 (1979), 474–484.

[15] N. Valcoff, Optimal size of transmission systems, IFAC Symp Optimal Syst Planning (1968), 26–35.

[16] B. Yaged, Minimum cost routing for dynamic network models, Networks 3 (1973), 193–224.