Iterative State-Space Reduction for Flexible Computation

Weixiong Zhang
Department of Computer Science
Washington University
St. Louis, MO 63130
E-mail: zhang@cs.wustl.edu.

October 2, 2000

Abstract

Flexible computation is a general framework for decision making under limited computational resources. It enables an agent to allocate limited computational resources to maximize its overall performance or utility. In this paper, we present a strategy for flexible computation, which we call iterative state-space reduction. The main ideas are to reduce a problem space that is difficult to search to one that is relatively easy to explore, to use the optimal solution from the reduced space as an approximate solution to the original problem, and to iteratively apply multiple reductions to progressively find better solutions. The basic operation for state-space reduction is heuristic node pruning, which excludes the nodes that do not seem to lead to high-quality solutions from further examination. Iterative state-space reduction can be combined with a state-space search, such as best-first search or depth-first branch-and-bound (DFBnB). The resulting algorithm can run as an anytime algorithm, which provides a tradeoff between solution quality and computation. Based on an analytical model, we analyze the probability that one iteration of the iterative process finds a solution. Furthermore, by combining it within DFBnB we apply iterative state-space reduction to three combinatorial problems, the maximum Boolean satisfiability, the symmetric TSP and asymmetric TSP. Our experimental results show that iterative state-space reduction is effective and efficient, making DFBnB find better solutions with less computation.
1 Introduction and Overview

In order to behave rationally, an autonomous agent must take into account its limited computational resources in making decisions, and carefully handle time constraints such as unknown deadlines of external events. Flexible computation [20] is a general framework for decision making under limited computational resources. It can assist an agent in allocating its computational resources in such a way that its overall performance or utility is maximized [5, 10, 20, 52]. The performance of the agent depends not only on the quality of its decision, but also on the amount of computational resources that it uses and the penalty introduced by a response delay. This area of research is becoming more and more important and has drawn much attention recently [1, 2, 19].

An important issue of flexible computation is to characterize the relationship between deliberation, which is the process of searching for a high-quality decision, and the payoff of such a decision [5, 10, 20, 52]. The overall performance of a search algorithm can be measured by the quality of the solution found and the amount of computation used to find the solution [20, 52]. Such performance measure can be captured by a performance profile [52] that measures solution quality in terms of the time required. We can measure the quality of a solution by its error relative to the cost of an optimal solution. Denote $prof(A, t)$ as the performance profile of algorithm $A$ at time $t$. We define $prof(A, t)$ as

$$prof(A, t) = 1 - error(A, t),$$  \hspace{1cm} (1)$$

where $error(A, t)$ is the error of solution cost of $A$ at time $t$ relative to the optimal solution cost. During the execution of $A$, $prof(A, t) \leq 1$; and when $A$ finds an optimal solution, $prof(A, t) = 1$.

If an agent has a performance profile of a problem to be solved, it can estimate the amount of computation it needs to find a solution with a satisfactory quality, or vice versa. However, the amount of time allocated for reasoning is generally not known in advance for most applications. Thus, the key to flexible computation is to construct anytime algorithms.

Flexible computation is also closely related to and sometimes depending upon approximation methods. Limited computational resources prohibit finding optimal solutions. In situations where seeking an optimal solution is not feasible, approximation methods enable an agent to find satisfactory solutions with a reasonable amount of computation. Approximation methods can be categorized into two classes. The first class of approximation methods finds solutions of qualities within a predefined acceptance bound [34]. However, finding approximate solutions with a predefined quality for some difficult problems, such as the evaluation of Bayesian belief networks and graph coloring, are NP-hard [8, 34]. In addition, an algorithm in this class cannot usually generate a result before termination. The second class of approximation methods consists of anytime algorithms [10], which find a solution quickly, and then successively improve the quality of the best solution at hand, as long as more computation is available. Therefore, these methods do not have to set their goals in advance. The challenge for anytime algorithms is to find good solutions as soon as possible.

In this paper, we present a general and effective approach for flexible computation, called iterative state-space reduction. This approach can be used to allocate the amount of computation required for a solution of desired quality, and can be used to develop new anytime algorithms. Specifically, a state-space reduction is a process of reducing a state space that is difficult to search, into a state space that may be easier to explore. It leads search efforts to the area of problem space that is likely to provide the best approximate solution with the available
computational resources. After the reduction, the optimal goal in the reduced state space is found and used as an approximate solution to the original problem. By progressively searching more and more complex state spaces, if more computational resource is available, better solutions are incrementally found. The basic operation for state-space reduction is heuristic node pruning, which uses heuristics to exclude some nodes in a state space from further examination.

The paper is organized as follows. In Section 2, we present the iterative state-space reduction strategy and an algorithm description that combines the strategy with depth-first branch-and-bound. In Section 3, we propose a domain-independent heuristic for node pruning. In the next section, we then analyze the probability that state-space reduction will find a solution. We investigate the tradeoff between solution quality and computation as well as anytime performance of iterative state-space reduction in Section 5. We apply iterative state-space reduction to the maximum Boolean satisfiability [14] and the symmetric and asymmetric Traveling Salesman Problems [28] in Section 6. We discuss related work in section 7, and conclude in Section 8.

Previous results of this research appeared in [45, 46].

2 Iterative State-Space Reduction

In this section, we describe the main idea of iterative state-space reduction, discuss the combinations of this approach with state-space search algorithms, and present an anytime algorithm based on iterative reduction and depth-first branch-and-bound.

2.1 The Idea

The main idea of state-space reduction is to transform or reduce a state space that is difficult to search to another state space that is relatively easy to explore. Such a state-space reduction is achieved by pruning the nodes that do not seem to lead to high-quality solutions based on some heuristics. In other words, a state-space reduction through heuristic node pruning reduces a state space to a subset of the original space. Ideally, a reduction is carried out in such a way that high-quality solutions, and hopefully an optimal solution, can be found quickly in the reduced state space.

However, there is no guarantee that a reduced state space will contain a solution of satisfactory quality. If no required solution exists in a reduced space, another reduced, but larger, state space is generated to search for better solutions. A set of reductions can thus be iteratively applied to a difficult state space for finding increasingly better solutions. This progressive process of multiple state-space reductions is called iterative state-space reduction, or iterative reduction for short.

2.2 Iterative State-Space Reduction for Flexible Computation

As discussed in Section 1, the key to flexible computation is to develop efficient anytime algorithms, which finds a solution quickly and continues to search for better solutions when more computation is available. Following the same spirit, iterative state-space reduction is a strategy for anytime problem solving, and it makes a tradeoff between solution quality and computation. It can be integrated with a state-space search algorithm, such as best-first search or depth-first branch-and-bound (DFBnB), to construct an anytime algorithm. By searching in a small, reduced state space, a search algorithm may find a solution quickly and can continue searching for better solutions in larger reduced spaces.
To simplify our discussion in the rest of this paper, a state space generated by state-space reduction is referred to as a *reduced state space*. By the same token, a search in a reduced state space is called a *reduced search*. For instance, a best-first search in a reduced space is referred to as a *reduced best-first search*, and a DFBnB search is simply called a *reduced DFBnB search*.

The actual anytime performance of iterative state-space reduction, when combined with a state-space search algorithm, depends on two factors. The first is whether the state space to be searched contains a large number of leaf nodes that are also goal nodes. An extremely difficult state space is one that contains all *deadend nodes*, which are leaf nodes with no solution, and only one goal node. When applied to such an extreme state space, all the iterations except the last one of an iterative reduction process discard the goal node, making these iterations fail to find a solution. Such an extreme state space is rare, and may be resistant to all types of search strategies. Fortunately, the state spaces of most combinatorial optimization problems, such as those considered in Section 6, have most leaf nodes that provide solutions. When used to solve these problems, iterative state-space reduction is able to find a solution quickly in an early iteration and better solutions in subsequent iterations.

The second factor that affects the anytime performance of iterative state-space reduction is the search algorithm used. Although iterative state-space reduction can be combined in principle with any state-space search algorithm, such as best-first search, depth-first branch-and-bound (DFBnB), or iterative deepening, each different combination has a different anytime performance, depending on how quickly the search algorithm can reach leaf nodes. Given a state space, the best-first search finds the optimal goal node at the end of its execution. Combined with best-first search, iterative reduction can only provide a solution at the end of an iteration. Similar to best-first search, iterative deepening only reaches a goal node at the end of its execution. It also generates more nodes than the best-first search and is sensitive to node costs. When node costs take a large number of different values, iterative deepening typically generates a few new nodes in each iteration in addition to revisiting the nodes examined in the previous iteration, making it an unfavorable method [37]. Therefore, its anytime performance is not better than that of best-first search. DFBnB, on the other hand, can be considered as an anytime algorithm, as it can reach many goal nodes, not necessarily optimal, during its exploration of a state space [44, 47]. Since it always favors a node at a deep depth, DFBnB reaches a leaf node quickly and continues to examine other leaf nodes during its execution. Therefore, the combination of the iterative reduction and DFBnB should have a better anytime performance over best-first search. Furthermore, DFBnB has other favorable features. It uses space that is linear in search depth, making it a feasible algorithm for large problems in practice. It is asymptotically optimal in terms of the number of nodes generated on difficult problems, and it may actually run faster than best-first search due to its low overhead of node generation [50].

### 2.3 An Algorithmic Description

To make our description concrete, we now present, in an algorithmic format, the combination of iterative state-space reduction and DFBnB.

Figure 1 gives the state-space reduction procedure embedded in DFBnB for finding a goal node of the minimum cost, where \( n \) represents the problem to be solved, \( R \) the set of heuristic rules for node pruning, and \( a \) the current upper bound on the cost of an optimal goal. It is invoked as \( \text{Reduction}(\text{problem}, \ R, \ \infty) \).

Note that the reduced DFBnB search in Figure 1 is the same as DFBnB except that it may abandon a node based on heuristic rules \( R \) in line 2. This simple but critical difference makes
Reduction($n$, $R$, $\alpha$)

1) Generate all children of $n$;
2) Discard a child node if it can be pruned by a rule in $R$;
3) Sort the remaining $k$ children in increasing order of cost:
   $n_1, n_2, \ldots, n_k$
4) FOR ($i$ from 1 to $k$)
5)   IF ($\text{cost}(n_i) < \alpha$)
6)     IF ($n_i$ is a goal node) $\alpha \leftarrow \text{cost}(n_i)$;
7)     ELSE $\alpha \leftarrow \text{Reduction}(n_i, R, \alpha)$;
8)   ELSE RETURN $\alpha$
9) RETURN $\alpha$

Figure 1: A DFBnB search in a reduced state space.

IterativeReduction($\text{problem}$, $R$, $\alpha$)

1) DO
2) $\alpha \leftarrow \text{Reduction}(\text{problem}, R, \alpha)$;
3) Weaken heuristic pruning rules in $R$;
4) WHILE (no required goal has been found and
   a rule in $R$ was applied in the last iteration)

Figure 2: An iterative state-space reduction algorithm.

reduced DFBnB terminate sooner than DFBnB, hopefully with a required solution.

The sorting in Line 3 of 1 is called node ordering. It is used to guide the search to the
regions that are more likely to have high-quality solutions. The node ordering can indeed
significantly improve the search efficiency, as observed in most applications including those
considered in Section 6.

The iterative state-space reduction algorithm iteratively runs a series of state-space reduc-
tion procedures using weaker heuristic pruning rules in subsequent iterations. Specifically, the
algorithm combining iterative reduction with a state-space search algorithm runs as follows.
It runs the search algorithm on a reduced state space that is generated by a set of heuristic
pruning rules. If a required solution is found, the new algorithm terminates. Otherwise, the
heuristic pruning rules are weakened by reducing the number of pruning criteria or removing
their individual components. The algorithm then runs another iteration of search in a new
reduced state space. This iterative process continues until a required solution is found, or
no heuristic pruning rule is applied in the last iteration. In the latter case, either no solution
exists, or the best solution found so far is optimal because the last iteration runs in the original
state space and is guaranteed to find an optimal solution. All these cases combined make the
new algorithm complete.

Figure 2 gives an algorithmic description of the combination of iterative state-space reduc-
tion and DFBnB. It is invoked as IterativeReduction($\text{problem}$, $R$, $\infty$), and it calls the
reduced DFBnB search Reduction($\text{problem}$, $R$, $\alpha$) of Figure 1 as a subroutine.

3 Domain-Independent Node-Pruning Heuristic

Whether or not a reduced state space contains a desired solution depends on the heuristic
pruning rules used. Accurate heuristic rules have strong pruning power and can eliminate the
nodes that do not need exploring. On the other hand, very accurate heuristic pruning rules are generally domain dependent and require a deep understanding of a problem domain in order to develop.

In this research, we are interested in domain-independent heuristic node-pruning rules. We propose such a rule in this section. It uses a static, heuristic node evaluation function. Without loss of generality, we assume that the heuristic node evaluation function is monotonically nondecreasing with node depth, i.e., for a node \( n \) and its child node \( n' \), \( \text{cost}(n') \geq \text{cost}(n) \), where \( \text{cost}(n) \) is the cost of \( n \).

Consider again a node \( n \) and its child node \( n' \) in a state space. If \( \text{cost}(n') \) exceeds \( \text{cost}(n) \) by \( \delta \), where \( \delta \) is a predefined, non-negative constant, i.e., \( \text{cost}(n') > \text{cost}(n) + \delta \), then \( n' \) is discarded. Thus, a smaller \( \delta \) represents a stronger pruning rule and a greater \( \delta \) represents a weaker rule. This node-pruning heuristic was derived from the intuition that if a child node has a significantly large cost increase from its parent, it is more likely that the child may lead to a goal node of a large cost. When used in iterative state-space reduction, this pruning rule can be weakened by increasing the value of \( \delta \) after each iteration. By increasing \( \delta \), we reduce the possibility of pruning a child node. The effectiveness of this rule will be examined in Sections 5 and 6.

It is obvious that deadend nodes, which are nodes with no children, will curtail a reduced search’s success of finding a solution. Unfortunately, the state-space reduction strategy creates many unexplored nodes, which are “deadend nodes” as far as the search is concerned. In order to increase the probability that a reduced search reaches a goal node, it is important to reduce the possibility of creating deadends. When a node has more than one child node, the pruning may be too strong if all its children are discarded. Instead of abandoning all the children of a node, keeping at least one child will prevent treating the current node as a deadend. This is a modified or meta rule that controls how the heuristic pruning rules should be used. Intuitively, this modification boosts the probability that a reduced state space contains a goal node. This intuition is supported by an analysis in Section 4.2. In order to find a high-quality goal node, the modified rule chooses to keep the child node with the minimum cost among all the children.

## 4 An Analysis

The use of heuristic pruning rules in state-space reduction is a two-edged sword. On one side, it reduces the amount of search by reducing the size of the state space, solving some problems in a reasonable amount of time. On the other side, it runs the risk of missing a goal, making an iterative state-space reduction algorithm run additional iterations and causing a delay in finding a goal. Therefore, a critical question is how likely a reduced state space generated with a set of given heuristic pruning rules will contain a goal node of the original space. In this section, we carry out an analysis to answer this question by examining the probability that a reduced space has a goal node.

### 4.1 An Analytic Model

We use the following analytic random tree model to facilitate our analysis.

**Definition 4.1** An incremental random tree, or simply random tree, \( T(\beta, d) \) is a tree with random node branching factors and random costs on edges. The branching factors \( \beta \) are nonnegative, discrete, and independent and identically distributed (i.i.d.) random variables with
Figure 3: An incremental random tree.

\[
    p_k = P\{\beta = k\}, \quad k = 0, 1, \ldots, b,
\]

where \(b\) is the maximal branching factor. The costs of edges are non-negative, i.i.d. random variables. The cost of a node is the sum of the costs of the edges on the path from the root to the node. All nodes at depth \(d\) are goal nodes, and an optimal goal is a goal node of minimal cost.

Figure 3 illustrates an incremental random tree. This random tree model is an abstraction of a search space; it has also been used to analyze many search algorithms [23, 26, 31, 32, 48, 49, 50], and develop a new search method called Epsilon search [38, 51]. One advantage of using random trees is that they are easy to reproduce. Thus, different search algorithms can be compared with no interference from a particular application domain.

Since there may exist deadend nodes, a random tree \(T(\beta, d)\) may have a depth of less than \(d\). The i.i.d. assumptions on branching factors and edge costs are introduced to make our analysis feasible. The use of edge costs does not impose real restriction to the model because static node evaluation functions are generally available in practice and an edge cost is the difference between the cost of a node and that of its parent. The most important feature of this abstract model is that it naturally introduces dependencies among node costs. The costs of two nodes are correlated if they share common edges on their paths to the root, with the degree of dependence based on the number of edges they have in common.

### 4.2 Probability of Finding a Solution

We now analyze the probability that a reduced state space, or reduced tree, of a random tree \(T(\beta, d)\) contains a goal node of \(T(\beta, d)\). We consider the heuristic pruning rule and the modified heuristic pruning rule, discussed in Section 3, separately.

#### 4.2.1 Pure Heuristic Pruning Rule

Assume that a node of \(T(\beta, d)\) is pruned by state-space reduction with probability \(q\), independent of any other event. This assumption does not rule out the use of the domain-independent node pruning heuristic of Section 3. This is because this pruning heuristic is based on the edge costs of a random tree, which are independent of each other and other events.
A reduced state-space tree has less nodes than that of the original tree. The number of children of a node in a reduced tree is a random variable \( \beta' \) with distribution of

\[
p'_{k} = P\{\beta' = k\} = \sum_{i=k}^{b} P\{\text{a node } n \text{ has } i \geq k \text{ children, and } k \text{ of them are not pruned}\}
\]

\[
= \sum_{i=k}^{b} P\{i - k \text{ kids are pruned} | n \text{ has } i \geq k \text{ kids}\} P\{n \text{ has } i \geq k \text{ kids}\}
\]

\[
= \sum_{i=k}^{b} \binom{i}{k} q^{i-k} (1 - q)^{k} p_{i}
\]

\[
= \left(\frac{1 - q}{q}\right)^{k} \sum_{i=k}^{b} \binom{i}{k} p_{i} q^{i}, \quad k = 0, 1, \ldots, b.
\]

(3)

where \( b \) is the maximum number of children of a node and \( q \) is the probability that a child node is pruned.

**Theorem 4.1** If a set of node pruning rules has a probability \( q \) to discard a node, the reduced tree of an incremental random tree \( T(\beta, d) \), generated by using the pruning rules, contains a goal node of \( T(\beta, d) \) with probability

\[
P_{\text{goal}} = \begin{cases} 
    0, & \text{when } m' \leq 1 \\
    \frac{1 - s_{1}}{1 - s_{2}}, & \text{when } m' > 1
\end{cases}
\]

(4)

as \( d \to \infty \), where \( m' \) is defined by

\[
m' = \sum_{k=1}^{b} p'_{k} k
\]

(5)

and \( s_{1} \) and \( s_{2} \) are solutions less than 1 to equations

\[
s = \sum_{k=0}^{b} p_{k} s^{k} \quad \text{and} \quad s = \sum_{k=0}^{b} p'_{k} s^{k}, \quad |s| \leq 1,
\]

(6)

respectively, where \( p_{k} \) and \( p'_{k} \) are defined in (2) and (3).

**Proof:** See Appendix A. \( \square \)

Unfortunately, it is difficult to find a closed-form solution to probability \( P_{\text{goal}} \) in general. When \( b \) is greater than five, a closed-form solution does not exist because equations in (6) do not have closed-form solutions. In this case, \( P_{\text{goal}} \) must be calculated numerically.

For special cases when a node has either two children or no children, we have the following results.

**Corollary 4.1** Let \( T(\beta, d) \) be an incremental random tree in which the branching factor of a node may take value zero with probability \( p_{0} > 0 \), or value two with probability \( p_{2} = 1 - p_{0} \). A
Figure 4: Probability of finding a goal node on random trees.

**Proof:** See Appendix A. □

**Corollary 4.2** Let $T(\beta, d)$ be an incremental random tree in which every internal node has two children. A reduced tree of $T(\beta, d)$ has a goal node with probability

$$P_{\text{goal}} = \begin{cases} 
0, & \text{when } p_0 \geq \frac{1}{2} \text{ or } q \geq \frac{1-2p_0}{2(1-q_0)}; \\
\frac{1-4(1-p_0)(1-q)(p_0+q-p_0q)}{4(1-2p_0)(1-q)^2}, & \text{when } p_0 < \frac{1}{2} \text{ and } q \leq \frac{1-2p_0}{2(1-q_0)};
\end{cases}$$

as $d \to \infty$, where $q$ is the probability that heuristic pruning rules prune a node.

**Proof:** See Appendix A. □

Figure 4(a) shows the experimental probability of finding a goal on a random tree $T(\beta, d)$ with $\beta$ equal to 0 with probability $p_0$ or 2 with probability $1-p_0$, and that on $T(\beta, d)$ with $\beta$ equal to 0 or 5, along with the asymptotic probability of reaching a goal based on Theorem 4.1. The horizontal axes are the probability that state-space reduction prunes a node; the vertical axes are the probability of reaching a goal. Each experimental data point of Figure 4 is averaged over 1,000 random trials. As Figure 4(a) shows, the experimental probability approaches its asymptotics as $d$ grows.

### 4.2.2 Modified Pruning Rule

To reiterate, deadend nodes are those of a state space with no children. Intuitively, a reduced state space is more unlikely to have a goal node of the original space when more deadend
nodes exist. This is illustrated in Figure 4 in which the probability of having a goal decreases when the probability of pruning a node increases. To reduce the impact of deadend nodes, we proposed in Section 3 a simple modification method to increase a reduced state space’s probability of having a solution. Following the modification, the best child will be kept if all children of a node are to be pruned based on the heuristic pruning rules. This modification rule tends to keep more nodes in a reduced state space, so that the reduced space will have a high probability of having a goal node of the original space. A formal analysis supports this modification rule, as summarized by the following theorem.

**Theorem 4.2** On an incremental random tree, the modified rule will increase the probability that a reduced random tree contains a goal node.

**Proof:** See Appendix A. □

We experimentally compare the probability that a reduced state space generated with the modified rule has a goal node of the original space to the probability that a reduced state space generated without the modified rule has a goal node.

Figure 5(a) shows the result on random trees with depth 20 and branching factor equal to 0 with probability $p_0$ or equal to 5 with probability $1 - p_0$. The horizontal axis is the probability of pruning a node, and the vertical axis is the probability of having a goal node. Each data point is an average of 1,000 trials. It shows that the modified rule significantly increases the probability of reaching a goal. The remaining question is whether the modified rule may cause a large increase in the total computation. As shown in Figure 5(b) where depth-first branch-and-bound (DFBnB) is used as the basic search algorithm, the computation reductions offered by state-space reduction with and without the modified rule are comparable, indicating that the modified rule does not cause a significant increase in computation. Since the modified rule tends to keep more nodes, more goal nodes are included in the reduced space. The lower goal costs found early in DFBnB search reduce the current upper bound on the optimal goal cost which in turn provide further pruning in DFBnB. In other words, the additional pruning that comes from better goal costs compensates for the additional computation introduced directly by the modified rule.
5 Properties of Iterative State-Space Reduction

Two important features of iterative state-space reduction deserve further investigation: the tradeoff between complexity and solution quality and the anytime performance. A study of these features is the topic of this section.

5.1 Tradeoff Between Complexity and Solution Quality

In general, if the effective branching factor of a search is $\gamma$ and the maximum search depth is $d$, then the search complexity is $O(\gamma^d)$. In essence, state-space reduction reduces search complexity by reducing the effective branching factor of its underlying search method. The amount of reduction is exponential in the search depth $d$. Let $\gamma_1$ be the original effective branching factor, and $\gamma_2$ the effective branching factor of a search in a reduced state space, where $\gamma_2 < \gamma_1$. Then, the amount of reduction is $O(\gamma_1^d) - O(\gamma_2^d) = O(\gamma_2^d)(O(1/\gamma_1^d) - 1) > O(\gamma_2^d)\frac{d}{\gamma_2}$.

Figure 6(a) shows the computation reduction of state-space reduction embedded in depth-first branch-and-bound (DFBnB) over the underlying DFBnB search on incremental random trees. The horizontal axis is the probability that heuristic rules prune a node; the vertical axis is the ratio of the number of nodes expanded by reduced DFBnB to that by DFBnB. The result is averaged over the problem instances on which a reduced DFBnB search finds a goal node among all 1,000 random trees of depth 20 and with branching factor 0 of probability $p_0$ or branching factor 5 of probability $1 - p_0$. It shows that the greater the pruning power is, the more efficient a reduced search becomes when compared to the underlying search method in the case of finding a goal node.

However, the solution quality of a reduced search deteriorates with stronger pruning rules as more nodes are discarded, as shown by Figure 6(b). The horizontal axis is the probability of pruning a node; the vertical axis is the solution quality measured by the error of solution cost relative to the cost of an optimal solution. Figure 6(b) shows that the solution quality decreases when more nodes are pruned. The result is averaged over the same problem instances as those used in Figure 6(a).

The reduced computational complexity of Figure 6(a) and the deteriorating solution quality
of Figure 6(b) indicate that search efficiency comes with a penalty on solution quality even if a reduced search can reach a goal. An interesting and useful implication of this phenomenon is the tradeoff between search complexity and solution quality: Greater computation yields better solutions, while lower computation leads to worse solutions. This tradeoff between computation and solution quality is illustrated in Figure 7, where the horizontal axis is the computation reduction, measured by the ratio of the number of nodes expanded by reduced DFBnB to that by DFBnB, and the vertical axis is the solution quality, measured by the average relative error of solution costs. Similarly, the result of Figure 7 is averaged over the problem instances on which the reduced search finds a goal node among all 1,000 random trees with depth 20 and branching factor 0 or 5.

5.2 Anytime Performance

The iterative state-space reduction algorithm can run as an anytime search algorithm. It can turn a non-anytime algorithm into an anytime algorithm. Specifically, we can combine iterative
reduction with best-first search (BFS) to develop an anytime algorithm, i.e., iterative reduced BFS. Similarly, we can combine iterative reduction with iterative deepening to construct the iterative reduced iterative deepening algorithm. Furthermore, iterative state-space reduction can enhance the anytime performance of an anytime algorithm, such as DFBnB.

We examine the anytime performances of iterative reduced BFS, iterative reduced iterative deepening, and iterative reduced DFBnB on incremental random trees. As discussed previously, the anytime performance of iterative reduced iterative deepening is not better than that of iterative reduced BFS. On problem instances where nodes take a large number of different values, iterative deepening has a very poor performance, and cannot compete with DFBnB. We don’t include the result of iterative deepening in the following discussion.

Figure 8 shows the performance profiles of iterative reduced BFS, iterative reduced DFBnB and DFBnB on incremental random trees. The trees have depth 20 and branching factor equal to 0 with probability 0.2 or equal to 5 with probability 0.8. The horizontal axis represents the computation used in terms of the number of node expansions, and the vertical axis measures performance profile. The results are averaged over 1,000 random trees. In the experiment, iterative state-space reduction can have at most four iterations. The first one has a probability 0.8 of pruning a node, each subsequent iteration reduces the pruning probability of the previous iteration by 0.2, and the last iteration runs BFS or DFBnB.

Figure 8 shows that iterative reduced BFS has a slightly better anytime performance than that of iterative reduced DFBnB when measured by the number of node expansions. However, when measured by CPU time, the anytime performance of iterative reduced BFS is worse than that of iterative reduced DFBnB on these random trees. This is due to different node expansion costs for BFS and DFBnB. The node generation cost for DFBnB is a constant. However, the node generation cost for BFS is a function of the number of open nodes, nodes that have been generated but not expanded. In order to expand a least-cost node in every step, BFS needs to maintain the open nodes in a priority queue such as a heap. This maintenance cost is $O(\log m)$, where $m$ is the number of open nodes. Furthermore, as mentioned earlier, iterative reduced BFS may provide a new solution at the end of an iteration, BFS typically takes longer to reach the first solution than DFBnB. This is shown in Figure 8. Specifically, on the random trees we experimented with, BFS expands more than 500 nodes on average before finding the first solution, while DFBnB only expands about 50 nodes on average for reaching the first goal node.

Figure 8 also shows that the iterative reduced DFBnB finds a better solution significantly sooner than its underlying DFBnB. For instance, at total 1,000, 5,000 and 10,000 node expansions, the average errors of the solution cost from DFBnB, relative to the optimal goal cost, are 36.1% (profile=0.639), 18.4% (profile=0.816) and 10.2% (profile=0.898), respectively, while the average solution errors of iterative reduced DFBnB are 14.8% (profile=0.852), 3.6% (profile=0.964) and 2.1% (profile=0.979), respectively. Furthermore, iterative reduced DFBnB not only outperforms its underlying DFBnB on average, but may also terminate earlier than the latter. The reason is that a better solution cost found in the previous iterations is used as the current upper bound on the optimal goal cost, which may significantly reduce the amount of search in the current iteration. Among the 1,000 instances, iterative reduced DFBnB terminates earlier on 588 cases, a slight bias in favor of iterative reduced DFBnB.

The real challenge is whether iterative state-space reduction can provide better anytime performance to a search algorithm, such as DFBnB, on real problems, since many statistical features, such as the \textit{i.i.d.} assumptions on node branching factors and edge costs, are not valid in practice. This question is favorably answered by the experimental results shown in the next section.
6 Experimental Evaluations

In this section, we apply the iterative state-space reduction approach combined with depth-first branch-and-bound (DFBnB) presented in Section 2.3, the domain-independent pruning heuristic proposed in Section 3, plus the modified rule of Section 3, if necessary, to three NP-complete [14] combinatorial optimization problems, the maximal Boolean satisfiability [14] and the symmetric and asymmetric Traveling Salesman Problems [28]. The purpose of this study is to examine the anytime feature of iterative state-space reduction and to evaluate the effectiveness of the domain-independent pruning rule. In our experiments, we compare the performance profile of iterative state-space reduction combined with DFBnB, called iterative reduced DFBnB for short, against that of DFBnB.

Before we apply the new approach to the problems, we need to address a technical difficulty of using the heuristic pruning rule.

6.1 Learning Edge-Cost Differences and Setting Parameters

Our domain-independent heuristic pruning rule uses a parameter $\delta$. A node will be eliminated if its cost exceeds the cost of its parent by more than $\delta$. In order to set the value of $\delta$ properly, we need information about how much the cost of a node will increase from that of its parent. It would be ideal if we knew the distribution of cost differences between nodes and their parents for a given problem. Such information has to be collected or learned from the problems to be solved. This can be done in two ways. The first is offline sampling. If sample problems from an application domain are available, we can first solve these problems using DFBnB, and use the costs of the nodes encountered during the search as samples to calculate an empirical distribution of node-cost differences. The second method is online sampling, which we use in our experiments. This method can be used when sample problems are not provided. Using this sampling method, the first iteration of an iterative state-space reduction does not apply heuristic pruning rules until a certain number of nodes have been generated in DFBnB. In our experiments, the first iteration of iterative reduction does not use pruning rules until DFBnB has reached the first leaf node. All the nodes generated in the process of reaching the first leaf node are used as initial samples for computing an empirical distribution. The empirical distribution is refined when more samples are available.

Using the empirical distribution of node-cost differences, we can set parameter $\delta$. In our experiments on the following three combinatorial problems, the initial $\delta$ is set to a value $\delta_1$ such that a node-cost difference is less than $\delta_1$ with probability $p$ equal to 0.1. The next iteration increases probability $p$ by 0.1, and so forth, until no heuristic pruning rule has been applied or probability $p$ was equal to one in the latest iteration.

6.2 The Maximum Boolean Satisfiability

We are concerned with Boolean 3-satisfiability (3-Sat), a constraint-satisfaction problem (CSP). A 3-Sat involves a set of Boolean variables and a conjunction of a set of disjunctive clauses, with each clause having 3 literals (variables and their negations). The conjunction defines constraints of acceptable combinations of variables. There are many practical CSPs in which no value assignment that does not violate a constraint can be found; see [13] for discussion and references therein. In this case, one option is to find an assignment such that the total number of satisfied clauses is maximized. In our experiment, we consider the maximum 3-Sat.

The maximum 3-Sat can be optimally solved by the Davis-Putnam algorithm [9], which can be organized as DFBnB. The root of the search tree is the original problem with no variable
One variable is then chosen and set to either true or false, thus decomposing the original problem into two subproblems. Each subproblem is then simplified. For instance, if the selected variable is set to true, a clause can be removed if it contains this variable. A clause can also be discarded if it contains the negation of a variable that is set to false. Furthermore, a variable can be deleted from a clause if the literal is set to false. Since the two values of a variable are mutually exclusive, so are the two subproblems generated. Therefore, the state space of the problem is a binary tree without duplicate nodes. The cost of a node is the total number of clauses violated, which is monotonically nondecreasing with the depth of the node. In our implementation of DF BnB, we use the most occurrence heuristic to choose a variable; in other words, we choose an unspecified variable that occurs most frequently in the set of clauses.

We generated the maximum 3-Sat problem instances by randomly selecting three variables and negating them with probability 0.5 for each clause. Duplicate clauses were removed. Since random 3-Sat problems with a small clause-to-variable ratio are generally satisfiable [7, 33], the problem instances used have a large clause-to-variable ratio.

In our experiments, we studied the effects of the modified rule proposed in Section 3, which retains the best child node if the two children of a node are pruned by the heuristic pruning rule. On random 3-Sat, iterative state-space reduction without the modified rule cannot compete with DF BnB. This is due to two factors. The first is the small branching factor 2 of the state space, so that deadend nodes can be easily generated if the modified rule is not applied. The second is that the initial value of $\delta$ is too small, thus most early iterations of the iterative reduction cannot reach a leaf node in the search space at all. On the other hand, the iterative reduction with the modified rule is significantly superior to DF BnB.

Figure 9 shows the experimental results of iterative state-space reduction using the modified rule on 3-Sat with 30 variables and 450 clauses, averaged over 100 random problem instances. The vertical axis presents the performance profiles of an iterative reduction and DF BnB while the horizontal axis shows the number of nodes generated. The comparison has an almost identical picture when the horizontal axis is plotted in terms of CPU time on a SUN Sparc 2 machine, as the time spent for online sampling is negligible. Figure 9 shows that the iterative reduction significantly improves the anytime performance of DF BnB, finding better solutions sooner. For instance, with a total of 2,000 node generations, the average error of solution found
relative to the optimal is 2.1% (profile=0.979) from the iterative reduction, while the average error is 13% (profile=0.87) from DFBnB.

6.3 The Symmetric TSP

Given \( n \) cities \( \{1, 2, ..., n\} \) and a cost matrix \( (c_{i,j}) \) that defines a cost between each pair of cities, the Traveling Salesman Problem (TSP) is to find a minimum-cost tour that visits each city once and returns to the starting city. When the cost from city \( i \) to city \( j \) is the same as that from city \( j \) to city \( i \), the problem is the symmetric TSP (STSP).

In our implementation of DFBnB, we use the Held-Karp lower bound function [18] to compute node costs, which is a very tight bound on the optimal STSP tour length [22]. This cost function iteratively computes a Lagrangian relaxation on the STSP, with each step constructing a 1-tree. A 1-tree is a minimum spanning tree (MST) [36] on \( n - 1 \) cities plus the two shortest edges from the city not in the MST to two cities in the MST. Note that a complete TSP tour is a 1-tree. If no complete TSP tour has been found after a predefined number of steps of Lagrangian relaxation, which is \( n/2 \) in our experiment, the problem is decomposed into at most three subproblems using Volgenant and Jonker's branching rule [42]. Under this decomposition rule, the state space of the STSP is a tree without duplicate nodes.

We generated STSP problem instances by uniformly choosing a cost between two cities from \( \{0, 1, 2, ..., 2^{32} - 1\} \), the set of 32-bit integers. The experimental results show that the iterative reduction without the modified rule of Section 3 is substantially worse than DFBnB on the maximum 3-sat, due to the same reasons described in Section 6.2.

Figure 10 shows the experimental result of the iterative reduction with the modified rule on 100-city random STSPs, averaged over 100 problem instances. The horizontal axis is the number of 1-trees solved by both algorithms, and the vertical axis presents the performance profiles of an iterative reduction and DFBnB. We do not use the number of generated nodes as a time measure, because generating a node requires to solve a variable number of 1-trees based on Lagrangian relaxation. Figure 10 shows that iterative reduction improves the anytime performance of DFBnB on random STSPs, finding better solutions sooner. For instance, to generate a solution that is 0.06% to the optimal or a performance profile of 0.9994, iterative reduced DFBnB solves only 2,900 1-trees on average, while DFBnB needs to solve 4,850 1-trees.
6.4 The Asymmetric TSP

When the cost from city $i$ to city $j$ is not necessarily equal to that from city $j$ to city $i$, the TSP is the asymmetric TSP (ATSP). The best known cost function for the ATSP is the solution to the assignment problem [36]. The assignment problem is to assign to each city $i$ another city $j$, with $c_{i,j}$ as the cost of this assignment, such that the total cost of all assignments is minimized. The assignment problem is a relaxation of the ATSP since the assignments do not need to form a single tour, but instead can form a collection of disjointed subtours. This provides a lower bound on the cost of the ATSP tour, which can be viewed as an assignment of each city to its successor in the tour. The assignment problem can be solved in $\mathcal{O}(n^3)$ time or can be incrementally solved in $\mathcal{O}(n^2)$ time if the AP solution to the parent problem is given [30]. If the solution to the assignment problem is a single complete tour, then it is also the solution to the ATSP. If, on the other hand, the solution to the assignment problem is not a single complete tour, then it is decomposed and generates subproblems. A subproblem can be produced by including and/or excluding some edges of the solution to the assignment problem in/from the final ATSP tour in order to disallow certain subtours. There are many ways to decompose a problem such that the corresponding state space is a tree with no duplicate nodes. See [3] for a detailed summary of these decomposition methods. In our experiments, we adopted the method proposed by Carpaneto and Toth [6].

It is worth mentioning that the branching factor of an ATSP state space is large and proportional to the number of cities. Therefore, the modified rule of Section 3 does not matter much. In fact, the experimental result of the iterative reduction with the modified rule is slightly worse than that without the modified rule.

We used random ATSP in our experiments. The costs among cities were uniformly chosen from $\{0, 1, 2, \cdots, 2^{32} - 1\}$. Figure 10 shows the experimental results of the iterative reduction without the modified rule on 200-city random ATSP, averaged over 100 instances. The horizontal axis is the number of assignment problems solved, and the vertical axis presents the performance profiles. Figure 11 shows that the iterative reduced DFBnB improves the anytime performance of DFBnB on random ATSPs, finding better solutions sooner. For instance, to find an ATSP tour that is 0.03% to the optimal or a performance profile of 0.9997, iterative reduced DFBnB reduces the total average computation of DFBnB by about half, reducing the number of assignment problems solved from 605 to 299.
6.5 Comments on the Results

The experimental results on the three combinatorial problems all showed that iterative state-
space reduction can improve the anytime performance of DFBNB. It can usually help DFBNB
find a solution of a fixed quality with less than half of the computation required by pure
DFBNB. However, we also need to point out that such improvement on anytime performance
on these combinatorial problems is not as significant as that on a canonical state space such
as a random tree, which was discussed in Section 5.2. This is most likely due to the fact that
many features of the random tree, such as \(i.i.d\). branching factors and \(i.i.d\). edge costs, are
violated in these problems.

7 Related Work and Discussions

The idea of heuristic node pruning can be traced back to beam search [4, 43]. Beam search
executes a best-first search or breadth-first search, but may restrict the search to a set of se-
lected alternatives using a fixed amount of memory and application-specific rules for pruning
nonpromising search alternatives. It was experimentally observed that beam search can sig-
ificantly reduce the computation of its underlying search method [4]. Beam search has been
successfully applied to many problems [11, 12, 29, 35, 40].

Since both state-space reduction and beam search use heuristic node pruning, the former
can be regarded as an extension of the latter. The main contribution of this research to
beam search is twofold. First, we extended the idea of heuristic node pruning to a depth-first
search algorithm and showed the generality of the idea of heuristic node pruning. Second, we
proposed a general approach for constructing complete anytime algorithms using heuristic node
pruning. Therefore, iterative state-space reduction was previously called complete anytime
beam search [45].

In essence, state-space reduction is a nonsystematic search strategy because it explores
only some selected search avenues of a search space and may miss a goal node completely.
State-space reduction is similar to iterative sampling of [27]. The main difference of these
two methods is that state-space reduction uses heuristic rules to decide if a node should be
expanded, while iterative sampling makes that choice randomly.

Iterative state-space reduction is a combination of heuristic node pruning and iterative
weakening [39]. Iterative weakening directly follows iterative deepening [24] and iterative
broadening [15]. These iterative methods all repeatedly apply a search process, but with
stronger or weaker parameters in different passes. It has been shown that a given set of search
policies should be applied in an increasing order of the search complexities that these policies
incur [39].

Iterative state-space reduction bears a close similarity to iterative broadening [15]. Briefly,
iterative broadening first carries out a search with a breadth limit of two. If it fails, the
algorithm repeats the search with breadth limit of three, and so on, until it finds a solution.
Early passes of both algorithms comb through the state space for better solutions, and they
gradually extend the coverage of their exploration by increasing the search breadth. The
difference between these two algorithms is that iterative broadening extends its search breadth
in a predetermined fashion, while iterative state-space reduction broadens its search depending
on how heuristic pruning rules are weakened. If we treat the way that search breadth is
extended in a predefined way as a special heuristic, iterative broadening can then be considered
as a special case of iterative state-space reduction.

Iterative state-space reduction using the domain-independent pruning rule of Section 3 is
symmetric to Epsilon search [38, 51]. In Epsilon search, a node with a cost no greater than its parent’s cost plus \( \epsilon \) is treated as if it has the same cost as its parent, so as to force an early exploration of the node; while in an iterative reduction a node with cost greater than its parent’s cost plus \( \delta \) is considered as if it has an infinitely large cost, so as to postpone the exploration of the node. Iterative state-space reduction and Epsilon search can be, in principle, combined in a single algorithm.

Anytime algorithms are important tools for problem solving in a real-time setting with resource constraints [10, 20, 52]. Many existing incremental refinement and iterative improvement methods have been used as anytime algorithms. One such example is iteratively deepening lookahead search for real-time problem solving [25, 41]. It is used to find the best next execution step under current available computational resources. Problems that such anytime algorithms can solve must allow deliberation after each execution step, i.e., execution steps and deliberation steps interleave with each other. Another anytime algorithm is truncated depth-first branch-and-bound (DFBnB) [21, 44]. Truncated DFBnB executes DFBnB until it exhausts all available computation. The best solution found so far is then an approximate solution. Another approach to constructing anytime algorithm is to use non-admissible heuristic search, as proposed in [16].

8 Conclusions

In this paper, we developed an approach for flexible computation, called iterative state-space reduction. It is a general method based on heuristic node pruning. It can be combined with a state-space search algorithm to develop a complete anytime algorithm, which is guaranteed to provide an optimal solution if sufficient computation is given. We studied in detail the combination of iterative reduction with depth-first branch-and-bound. We proposed a domain-independent heuristic for node pruning, which depends only on a static node evaluation function. Based on an incremental random tree model, we analyzed the properties of state-space reduction, including the probability that a reduced state space contains a goal node of the original space and how to select an important parameter of the node-pruning heuristic. Finally we applied iterative state-space reduction, combined with depth-first branch-and-bound and domain-independent pruning heuristic, to three NP-hard combinatorial optimization problems, namely, the maximum Boolean satisfiability, the symmetric TSPs and the asymmetric TSPs. Our experimental results show that the domain-independent pruning heuristic is effective and the iterative reduction approach is efficient and can significantly improve the efficiency and anytime performance of a state-space search algorithm by finding better solutions sooner.

Acknowledgments

This research was funded in part by National Science Foundation Grant No. IRI-9619554, and in part by Defense Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under cooperative agreement No. F30602-00-2-0508. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright annotation thereon. Thanks to the anonymous reviewers whose insightful comments greatly improved the quality of this paper, and the anonymous reviewers of my previous papers [45, 46], on which this paper is based. Thanks to Lauri Grier and Fanny Mak for proofreading the paper.
A Proofs to the Analytic Results

This appendix contains the proofs to the analytical results given in Section 4. The analysis is based on the observation that the growth of a random tree corresponds to branching processes [17]. To make our discussion self contained, we first briefly describe branching processes.

A.1 A brief description of branching processes

Branching processes [17] describe phenomena in which objects generate additional objects of the same kind, and different objects independently reproduce offspring of the same kind. An initial set of objects, called the 0-th generation, have children that are called the first generation; their children are the second generation, and so on. The growth of an incremental random tree is equivalent to branching processes by the analogy that node expansion processes can be considered as reproduction processes in which the nodes generated will be eventually expanded, producing additional nodes.

Galton-Watson branching processes concern only the sizes of the successive generations. Denote $Z_0, Z_1, Z_2, \ldots$ the numbers of objects in the 0-th, first, second, \ldots generations. $Z_0, Z_1, Z_2, \ldots$ form a Markov chain. In other words, if the size of the $n$-th generation is known, the sizes of later generations do not depend on the sizes of generations preceding the $n$-th. A branching process can be described by a probability distribution

$$p_k = P\{\text{an object has } k \text{ children}\}, \quad k = 0, 1, \cdots$$

with $\sum_k p_k = 1$. Equivalently, a branching process can also be characterized by the generating function

$$f(s) = \sum_k p_k s^k, \quad |s| \leq 1.$$

The iterates of the generating function $f(s)$ is defined by

$$\begin{cases} f_1(s) = f(s); \\ f_{n+1}(s) = f(f_n(s)); \quad n = 2, 3, \cdots \end{cases}$$

It can be verified that each of the iterates is a probability generating function. Furthermore, it can be shown that the generating function of $Z_n$ is the $n$-th iterate $f_n(s)$.

A.2 Proofs to the theorems

Theorem 4.1 If a set of node pruning rules has a probability $q$ to discard a node, the reduced tree of an incremental random tree $T(\beta, d)$, generated by using the pruning rules, contains a goal node of $T(\beta, d)$ with probability

$$P_{\text{goal}} = \begin{cases} 0, & \text{when } m' \leq 1 \\ \frac{1 - x_2}{1 - x_1}, & \text{when } m' > 1 \end{cases}$$

as $d \to \infty$, where $m'$ is the mean branching factor of state-space reduction which is

$$m' = \sum_{k=1}^\beta p_k^b k$$

(9)
and \( s_1 \) and \( s_2 \) are solutions less than 1 to equations

\[
s = \sum_{k=0}^{k} p_k s^k, \quad |s| \leq 1;
\]

and

\[
s = \sum_{k=0}^{k} p_k s^k, \quad |s| \leq 1,
\]

respectively, and \( \{p_0, p_1, \cdots, p_h\} \) and \( \{p'_0, p'_1, \cdots, p'_k\} \) are probability distributions of the branching factors of \( T(\beta, d) \) and that of state-space reduction.

**Proof:** Assume that a node of \( T(\beta, d) \) will be pruned by state-space reduction with probability \( q \), independent of any other event. This assumption will not restrict the domain-independent node pruning heuristics of Section 3. This is because this pruning heuristic is based on the differences of the costs of nodes and that of their parents, i.e., based on the edge costs of \( T(\beta, d) \), which are independent of each other. Call the part of \( T(\beta, d) \) that is explored by state-space reduction a *reduced tree*, and denote a reduced tree embedded in \( T(\beta, d) \) as \( T(\beta', d, q) \), where \( \beta' \) is the new branching factor and \( \beta' \leq \beta \). Let \( P_{\text{goal}} \) be the probability that the reduced tree reaches depth \( d \), thus finding a goal since all leaf nodes at depth \( d \) are goal nodes in the random tree \( T(\beta, d) \) according to Definition 4.1. \( P_{\text{goal}} \) is equal to the probability that the reduced tree \( T(\beta', d, q) \) has depth \( d \) given that \( T(\beta, d) \) has depth \( d \). This means that the necessary condition that a reduced tree can contain a goal of a tree \( T(\beta, d) \) is that the reduced tree has depth \( d \). Therefore, \( P_{\text{goal}} \) can be written as

\[
P_{\text{goal}} = \frac{P\{T(\beta', d, q) \text{ has depth } d \mid T(\beta, d) \text{ has depth } d\}}{P\{T(\beta, d) \text{ has depth } d \text{ and } T(\beta', d, q) \text{ has depth } d\}}
\]

In order to find \( P_{\text{goal}} \) using (12), we examine random trees \( T(\beta, d) \) and \( T(\beta', d, q) \) separately. To compute \( P\{T(\beta, d) \text{ has depth } d\} \), consider tree \( T(\beta, d) \). The key to compute this probability is the observation that the nodes at different depths of \( T(\beta, d) \) comprise a Galton-Watson branching process (see [17] and Section A.1). Let \( Z_i \) be the number of nodes at depth \( i \) of \( T(\beta, d) \), for \( i = 0, 1, \cdots, d \), with \( Z_0 = 1 \). Then \( Z_d \) is the number of leaf nodes of \( T(\beta, d) \). The event that \( T(\beta, d) \) has depth \( d \) can be denoted as \( Z_d \neq 0 \). Therefore,

\[
P\{T(\beta, d) \text{ has depth } d\} = 1 - P\{Z_d = 0\}
\]

Probability \( P\{Z_d = 0\} \) as \( d \to \infty \) is the extinction probability that the branching process terminates. Let \( m \) be the mean number of children of a member in the process, or the mean number of children of a tree node, which is defined as

\[
m = \sum_{k=1}^{h} p_k k,
\]

where \( \{p_0, p_1, \cdots, p_h\} \) is the distribution of \( \beta \) as defined in Definition 4.1. The following result gives the extinction probability of the Galton-Watson branching process.
Lemma 4.1 [17] When \( m \leq 1 \), the extinction probability \( \sum_{d=-\infty}^{\infty} P\{Z_d = 0\} \) is 1. When \( m > 1 \), the extinction probability is the unique nonnegative solution less than 1 of the equation

\[
s = f_1(s),
\]

where \( f_1(s) \) is the probability-generating function of \( Z_1 \), which is defined as

\[
f_1(s) = \sum_{k=0}^{b} p_k s^k, \quad |s| \leq 1.
\]

Lemma 4.1 means that when the average number of children of a node is less than or equal to one, the branching process eventually dies out with probability one. In this case, the problem of finding a goal at depth \( d \) in tree \( T(\beta, d) \) is not well defined. When the average number of children of a node is greater than one, the branching process may become extinct with a probability less than one, and this extinction probability can be computed by solving equation (10). This result, along with equation (13), gives the probability that a random tree \( T(\beta, d) \) grows to depth \( d \).

Now consider the domain-independent pruning heuristic of Section 3 and a reduced tree \( T' \), \( d, q \) of a random tree \( T(\beta, d) \). Since the pruning heuristic depends on the difference of a node cost and that of its parent, whether the node is to be eliminated from further exploration is independent from other events. Therefore, the growth of nodes at different depths of \( T' \) follows a new Galton-Watson branching process. We can use the same technique used for \( P\{T(\beta, d) \} \) to calculate \( P\{T(\beta, d) \} \) and \( P\{T'(\beta, d, q) \} \) all have depth \( d \).

Let \( Z'_i \) be the number of nodes at depth \( i \) of \( T'(\beta', d, q) \), for \( i = 0, 1, \ldots, d \). Recall that the number of children of a tree node \( n \) is a random variable with distribution \( p_k = P\{n \text{ has } k \text{ children}\} \), for \( k = 0, 1, \ldots, b \), and \( n \) may be pruned by state-space reduction with probability \( q \). Then, the probability distribution of \( Z'_i \) is

\[
p'_k = P\{Z'_i = k\} = \sum_{i=k}^{b} P\{\text{a node } n \text{ has } i \geq k \text{ children, and } i - k \text{ of them are pruned}\}
\]

\[
= \sum_{i=k}^{b} P\{i - k \text{ kids are pruned} \mid n \text{ has } i \geq k \text{ kids}\} P\{n \text{ has } i \geq k \text{ kids}\}
\]

\[
= \sum_{i=k}^{b} \binom{i}{k} q^{i-k}(1-q)^{k} p_i
\]

\[
= \left(\frac{1-q}{q}\right)^k \sum_{i=k}^{b} \binom{i}{k} p_i q^i, \quad k = 0, 1, \ldots, b.
\]

The mean number of children of a node in \( T(\beta', d, q) \) is then

\[
m' = \sum_{k=1}^{b} \frac{p'_k}{k}
\]

It is obvious that \( m' \leq m \), i.e., the average number of children of a node in \( T(\beta', d, q) \) is no greater than that in \( T(\beta, d) \), resulting from the additional pruning of state-space reduction.
Furthermore, the probability-generating function of $Z'_1$ is

$$f_2(s) = \sum_{k=0}^{b} p'_k s^k, \quad |s| \leq 1. \quad (19)$$

Using equations (17), (18) and (19) and Lemma 4.1, we can compute the probability $P\{Z'_2 = 0\}$, as $d \to \infty$, by solving the equation

$$s = f_2(s), \quad (20)$$

when $m' > 1$. This result gives the probability $P\{T(\beta, d) \text{ and } T(\beta', d, q) \text{ all have depth } d\}$ by following (13). This and the previous result on probability $P\{T(\beta, d) \text{ has depth } d\}$, along with equation (12), give the probability $P_{\text{goal}}$ that a reduced random tree has a goal node as $d \to \infty$. \hfill \square

**Corollary 4.1** Let $T(\beta, d)$ be an incremental random tree in which the branching factor of a node may take value zero with probability $p_0 > 0$, or value two with probability $p_2 = 1 - p_0$. A reduced tree of $T(\beta, d)$ contains a goal node with probability

$$P_{\text{goal}} = \begin{cases} 
0, & \text{when } p_0 \geq \frac{1}{2} \text{ or } q \geq \frac{1 - 2p_0}{2(1 - p_0)}; \\
\frac{2(1 - p_0)(1 - q) - 1 + \sqrt{1 - 4(1 - p_0)(1 - q)(1 + p_0 - 2p_0)}}{2(1 - 2p_0)(1 - q)^2}, & \text{when } p_0 < \frac{1}{2} \text{ and } q < \frac{1 - 2p_0}{2(1 - p_0)},
\end{cases}$$

as $d \to \infty$, where $q$ is the probability that the heuristic pruning rules prune a node.

**Proof:** First consider $T(\beta, d)$ and its corresponding Galton-Watson branching process. When $m \leq 1$, or $p_0 \geq 1/2$, the branching process will become extinct with probability one according to Lemma 4.1. When $p_0 < 1/2$, the extinction probability of the process is the solution less than one to the equation

$$s = f_1(s), \quad \text{or} \quad (1 - p_0)s^2 - s + p_0 = 0$$

The desired solution is

$$s_1 = \frac{p_0}{1 - p_0} \quad (21)$$

Now consider $T(\beta', d, q)$. Equation (17) becomes

$$\begin{align*}
p'_0 &= p_0 + (1 - p_0)q^2; \\
p'_1 &= 2(1 - p_0)q(1 - q); \\
p'_2 &= (1 - p_0)(1 - q)^2.
\end{align*} \quad (22)$$

The mean value of $\beta'$ is

$$m' = \sum_{k=1}^{b} p'_k k = 2(1 - p_0)(1 - q) \quad (23)$$

When $m' \leq 1$ which is equivalent to $q \geq \frac{1 - 2p_0}{2(1 - p_0)}$, the branching process in $T(\beta', d, q)$ will die out with probability one as $d \to \infty$. Now compute the extinction probability of the branching
process when $q < \frac{1-2p_0}{2(1-p_0)}$. The generating function of the nodes at depth one of $T(\beta', d, q)$ is

$$ f_2(s) = \sum_{k=0}^{2} p'_k s^k $$

$$ = (1 - p_0)(1 - q)s^2 + 2(1 - p_0)q(1 - q)s + p_0 + (1 - p_0)q^2 $$

Thus, equation $s = f_2(s)$ becomes

$$ (1 - p_0)(1 - q)s^2 + 2(1 - p_0)q(1 - q)s + p_0 + (1 - p_0)q^2 = 0 $$

which has a solution $s_2$ less than one

$$ s_2 = \frac{1 - 2(1 - p_0)q(1 - q) - \sqrt{1 - 4(1 - p_0)(1 - q)(q + p_0 - q_0)}}{2(1 - p_0)(1 - q)^2} $$

Finally, using (21), (24) and Theorem 4.1, we have

$$ P_{\text{goal}} = \begin{cases} 0, & \text{when } m' \leq 1 \\ \frac{1 - s_2}{1 - s_1} = \frac{2(1 - p_0)(1 - q) - 1 + \sqrt{1 - 4(1 - p_0)(1 - q)(q + p_0 - q_0)}}{2(1 - 2p_0)(1 - q)^2}, & \text{when } m' > 1 \end{cases} $$

as $d \to \infty$. □

**Corollary 4.2** Let $T(\beta, d)$ be an incremental random tree in which every internal node has two children. A reduced tree of $T(\beta, d)$ has a goal node with probability

$$ P_{\text{goal}} = \begin{cases} 0, & \text{when } q \geq 1/2 \\ 1 - \left(\frac{1}{1-q}\right)^2, & \text{when } q < 1/2 \end{cases} $$

as $d \to \infty$, where $q$ is the probability that heuristic pruning rules prune a node.

**Proof:** This is a special case of Corollary 4.1. When $p_0 = 0$, i.e., every internal node of $T(\beta, d)$ must have two children, (25) can be simplified to (26). □

**Theorem 4.2** On an incremental random tree, the modified rule will increase the probability that a reduced random tree contains a goal node.

**Proof:** Consider a reduced tree $T(\beta', d, q)$ generated without the modification rule and a reduced tree $T(\beta'', d, q)$ produced with the modification rule. The new node branching factor $\beta''$ has a probability distribution

$$ \begin{cases} p'_0'' = p_0; \\ p'_1'' = p'_1 + p'_0 - p_0; \\ p''_k = p'_k; \quad \text{for } k \geq 2. \end{cases} $$

where the $p'_k's$ are defined in (17), which is

$$ p'_k = \left(\frac{1 - q}{q}\right)^k \sum_{i=k}^{b} \binom{i}{k} p_i q^i, \quad k = 0, 1, \ldots, b. $$
We now use Theorem 4.1 to prove that the probability of finding a goal in $T(\beta'', d, q)$, denoted as $P''_{goal}$, is no less than that in $T(\beta', d, q)$, which is $P_{goal}$. The mean value of $\beta''$ is

$$m'' = \sum_{k=1}^{b} p''_k k = p' + \sum_{k=2}^{b} p''_k k = p'_0 - p_0 + \sum_{k=2}^{b} p_k' k = p'_0 - p_0 + m' \geq m', \quad (29)$$

as $p'_0 \geq p_0$. This means that the range in which $P''_{goal} = 0$ is no greater than that in which $P_{goal} = 0$, following Theorem 4.1. We now prove that $P''_{goal} \geq P_{goal}$ in the range of $m'' > 1$. When $m'' > 1$, $P''_{goal} = (1 - s_3)/(1 - s_1)$, where $s_1$ is the solution less than 1 to equation (10), and $s_3$ is the solution less than 1 to the following equation,

$$s = f_3(s), \quad (30)$$

where $f_3(s)$ is the probability-generating function of $Z''_1$, the random number of children in the first generation of tree $T(\beta'', d, q)$, as defined by

$$f_3(s) = \sum_{k=0}^{b} p''_k s^k, \quad |s| \leq 1. \quad (31)$$

Because $P_{goal} = (1 - s_2)/(1 - s_1)$ and $P''_{goal} = (1 - s_3)/(1 - s_1)$, to show that $P''_{goal} \geq P_{goal}$, we only need to prove that $s_3 \leq s_2$, where $s_2$ is the solution less than 1 to (11). Using (27) and (19), we rewrite (31) as follows.

$$f_3(s) = \sum_{k=0}^{b} p''_k s^k = p'_0 + p'_1 s + \sum_{k=2}^{b} p_k'' s^k$$
\[
\begin{align*}
&= p_0 + (p'_1 + p'_0 - p_0)s + \sum_{k=2}^{b} p'_k s^k \\
&= p_0 + (p'_0 - p_0)s - p'_0 + (p'_0 + p'_1 s + \sum_{k=2}^{b} p'_k s^k) \\
&= (p'_0 - p_0)(s - 1) + f_2(s).
\end{align*}
\] (32)

Then, (30) can be written as

\[
[1 - (p'_0 - p_0)]s + (p'_0 - p_0) = f_2(s)
\] (33)

and \(s_3\) is the solution less than 1 to (33).

To compare \(s_2\) with \(s_3\), first consider \(s_2\) which is the solution less than 1 to (11). As illustrated in Figure 12, this solution lies at the intersection of \(y = s\) and \(y = f_2(s)\). Similarly, \(s_3\) is located at the intersection of \(y = [1 - (p'_0 - p_0)]s + (p'_0 - p_0)\) and \(y = f_2(s)\). As shown in Figure 12, \(s_3 \leq s_2\). □

References


27


