International Journal of Artificial Intelligence Tools, Vol. 3, No. 4 (1994) 493–523
© World Scientific Publishing Company

COMPARISON AND EVALUATION OF A CLASS OF IDA* ALGORITHMS[†]

BENJAMIN W. WAH and YI SHANG
Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 West Main Street, Urbana, IL 61801
{wah, shang}@manip.crhc.uiuc.edu

Received 5 December 1994
Revised 11 April 1995

ABSTRACT

In this paper, we study the performance of various IDA*-style searches and investigate methods to improve their performance by predicting in each stage the threshold to be used for pruning. Without loss of generality, we consider minimization problems in this paper. We first present three models to approximate the distribution of the number of search nodes by lower bounds: exponential, geometric, and linear, and illustrate these distributions based on some well-known combinatorial search problems. Based on these distributions, we show the performance of an ideal IDA* algorithm and identify reasons why existing IDA*-style algorithms perform well. In practice, we will be able to know from experience the type of distribution for a given problem instance, but will not be able to know the parameters of this distribution until the instance is solved. Hence, we develop RIDA*, a method that estimates dynamically the parameters of the distribution, and predicts the best threshold to be used in each stage. Finally, we compare the performance of several IDA*-style algorithms — Korf's IDA* and RBFS, RIDA*, IDA*_CR and DFS* — on several application problems, and identify conditions under which each of these algorithms will perform well.

Keywords: iterative search, IDA* algorithm, lower-bound distribution, regression.

1. Introduction

Search is an important tool in problem solving and is applied in many areas in artificial intelligence [1]. In general, blind-search algorithms are inefficient because they treat a search space syntactically without domain-specific information. To improve the performance of a search algorithm, heuristic information about the nature and the structure of the problem domain must be used to limit and guide the search.

Search algorithms for solving an application problem can broadly be classified into two classes according to whether they find optimal or suboptimal (feasible)

[†]Research was supported partially by National Science Foundation Grant MIP 92-18715 and by National Aeronautics and Space Administration Contract NAG 1-613.

solutions. Examples of search methods that find optimal solutions include best-first search, A*, IDA* and depth-first branch-and-bound search. Examples of search methods for finding suboptimal solutions include hill-climbing methods, beam search and anytime algorithms. In this paper, we focus on algorithms for finding optimal solutions.

Many application problems in finding optimal solutions in combinatorially large solution space are NP-hard and are solved by search algorithms that may require exponential time and space. The search algorithm that expands the minimum number of nodes before finding the optimal solution is the *best-first search* (*BFS*); a familiar example is the A* algorithm [1,2]. The drawback of BFS is that it requires an exponential amount of space; hence, it often exceeds the maximum memory capacity even for a relatively small problem. A *guided depth-first search* (*GDFS* or depth-first branch-and-bound search), on the other hand, requires memory space that is linear in the size of the problem. However, with a lack of a good pruning function that prunes futile searches beyond a certain level of a search tree, GDFS is prone to search far deeper than where the optimal solution lies.

The idea of *iterative deepening* was originally used successfully in a program called CHESS 4.5 [3]. In his seminal paper, Korf presented iterative deepening A* or IDA* [4], a search method that operates in a memory space linear in the size of the problem and that can approach asymptotically the behavior of A* (a best-first search with an admissible heuristic function). Because of its efficiency with respect to memory space, IDA* became the first heuristic search algorithm to find optimal solutions for the 15-puzzle problems within reasonable time and space constraints.

Like A*, IDA* requires an admissible lower-bound function. It is a variant of *depth-first iterative deepening* (*DFID*): a series of distinct depth-first searches to progressively greater depths to mimic a breadth-first search. As was originally described, IDA* initially sets its *threshold* to the (lower-bound) value of the root node s, searches depth-first from s, and backtracks when it reaches a node whose value exceeds the threshold. Such a depth-first search is called a *stage* or an *iteration*. If a solution is found in a stage, then this solution is optimal; if not, IDA* sets the threshold to the smallest value borne by any of the leaves of the stage. Then it starts the next stage — a new depth-first search that searches from the root and discards the results of the previous stage.

Korf originally demonstrated the performance of IDA* using the 15-puzzle problem [4]. Since the lower-bound value of a child node in this problem either increases by 0 or 2, threshold values in successive stages always increase by multiples of 2. By noting that there is an exponential growth in the number of search nodes from one stage to another, it is guaranteed that the last depth-first search, which finds the optimal solution, has an overhead that overwhelms the total overhead of all depth-first searches in previous stages. Korf further noted that the original IDA* does not perform well on the traveling-salesman problem (TSP), and suggested improvement by using thresholds sufficiently exceeding the value of the minimum leaf of the previous stage [5].

In general, IDA*-style algorithms can be extended to situations where the heuristic function is not admissible or non-monotonic; examples include path planning problems, chess and Prolog. In these cases, the solutions found are not guaranteed to be optimal, and the discussion is beyond the scope of this paper. In the following subsection, we discuss the various mechanisms for controlling thresholds in applying IDA*-style searches to solve minimization problems.

1.1. *Mechanisms for controlling thresholds in IDA*-style algorithms*

A very important component of IDA*-style algorithms is the thresholding mechanism; the design of which depends on the search objective, information available, and admissibility of the heuristic functions. There are two threshold-control strategies used in IDA*-style algorithms (assuming minimization searches).

(a) *Using additional memory space to store search-related information.* One class of IDA*-style algorithms use extra memory space to store active nodes and useful information in order to avoid re-expanding some search nodes. These include MREC [6], MA* [7], SMA* [2] and ITS [8]. For instance, MREC [6] uses extra memory to save active nodes near the root and to avoid expanding these nodes in future stages. The operations of MREC is illustrated in Figure 1. Initially, m active nodes are generated and are stored in the memory. Each stage starts with these m nodes, and the next threshold is the minimum cost of all nodes whose costs exceeded the current threshold. Note that IDA* is a special case of MREC with $m = 1$. The problem with these algorithms is that for large problems the cost of managing the stored information is usually higher than the benefit of expanding fewer nodes.

(b) *Dynamic control of thresholds.* This class of algorithms control the increase in thresholds in consecutive stages of an IDA* search. Examples include the original IDA*, DFS*, IDA*_CR, RBFS,

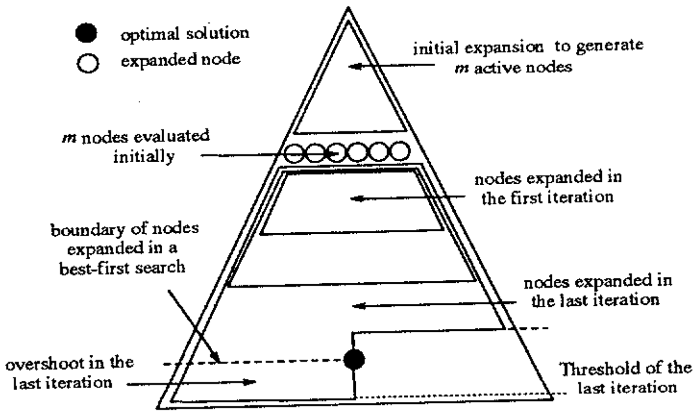

● optimal solution
○ expanded node
initial expansion to generate m active nodes
m nodes evaluated initially
nodes expanded in the first iteration
boundary of nodes expanded in a best-first search
nodes expanded in the last iteration
overshoot in the last iteration
Threshold of the last iteration

Fig. 1. An illustration of MREC. (Nodes in the tree are by their lower-bound values; thresholds indicate lower-bound values of nodes *expanded.*)

The original IDA* and MREC use as the next threshold the minimum lower-bound value of all active nodes exceeding the current threshold. This strategy works well when lower-bound values are discrete and the cumulative distribution of the number of search nodes by lower-bound values is geometric. This is the case in the 15-puzzle problem. The strategy does not work well when lower-bound values are continuous or when the distribution is not geometric.

In DFS* [9], the algorithm initially behaves like IDA* but increases the threshold more liberally by doubling it from one stage to the next. When a feasible solution is found, the algorithm switches to GDFS immediately and proceeds until the optimal solution is found. Thus, in the last stage in which the optimal solution is found, there may be some overshoot if the threshold chosen is larger than the optimal-solution value. The strategy works well when the cumulative distribution is linear and allows the number of nodes searched in successive stages to grow in a geometric fashion. This is the case in the maze problem.

IDA*_CR [10] is a variant of IDA* that collects some statistics in the search process to help determine the threshold in the next stage. It finds an approximate distribution of lower-bound values by dividing the possible lower-bound values into a set of discrete buckets, each recording the number of search nodes pruned in this stage whose lower bounds fall in the range of the bucket. It chooses the next threshold so that the total number of nodes in the buckets within this threshold is larger than a predefined number b^i, where b is the branching degree and i is the stage number. The strategy is hard to apply in practice because the proper choice of the ranges of buckets may be problem-instance dependent.

Recently, Korf proposed Recursive Best-First Search (RBFS) [11] and showed significant improvement over IDA* in certain application problems. RBFS is a recursive best-first search that runs in linear space and always expands new nodes in a best-first order regardless of whether the cost function is monotonic or non-monotonic. It is similar to IDA*, but uses a local cost threshold for each recursive call instead of the global threshold in IDA*. When cost values from the root to a leaf node are non-monotonic, RBFS shows significant improvement over IDA*. However, the improvement is not significant when cost values along a path are monotonic. Further, as RBFS extends thresholds in a similar way as IDA*, it has the same problem as IDA* in determining proper thresholds when cost values of nodes are mostly unique.

Table 1 summarizes the information used in these algorithms and the corresponding strategies. It also lists RIDA*, an algorithm discussed in Section 3.

1.2. *Focus of this paper*

We observe that previous IDA*-style algorithms use different information and heuristics to predict the best threshold to be used in a stage. Since the information and the heuristics used is generally problem dependent, it is possible to use other run-time information to improve performance. We propose in this paper

Table 1. Information and thresholding strategies used in various IDA*-style algorithms.

Algorithm	Information used in each stage in setting thresholds	Strategy
IDA*, MA*, MREC & RBFS*	Minimum lower bound of active nodes exceeding the current threshold	Set the next threshold to this value.
DFS*	Current threshold value	If a feasible solution has been found, then switch to GDFS and continue until the optimal solution is found; otherwise, double the threshold.
IDA*_CR	Lower-bound values of active nodes that are pruned as classified into discrete ranges of buckets	Choose the next threshold so that the total number of nodes in the buckets within this threshold is larger than b^i, where b is a user-defined factor and i is the stage number.
RIDA*	Cumulative distribution of lower-bound values of nodes expanded	Choose the next threshold so that the estimated number of nodes expanded in the next stage grows by a constant ratio.

to use the cumulative distribution of the number of search nodes by lower-bound values to help determine the proper thresholds in each stage. This prediction is possible for the following reasons.

(a) Problem instances of the same application problem generally have the same cumulative distribution, but the parameters of the distribution are problem-instance dependent. Run-time prediction, therefore, entails estimating the parameters of these distributions.

(b) Some problems have cumulative distributions that are hybrid of multiple distributions. For example, an integer programming problem has a cumulative distribution that is bell-shaped: the growth is exponential before any feasible solution is found, and is very small and quickly drops to zero after a feasible solution is found. As another example, the maze problem has a cumulative distribution that is made up two piecewise linear distributions (see Section 2). In this case, it is hard to predict the switch-over point before the search begins, but it may be easier to detect the switch-over using run-time statistics.

In this paper, we compare different prediction mechanisms for determining the threshold to be used in the next stage of an IDA*-style search, and investigate when and why certain algorithms perform well. Our results are summarized as follows.

- Different problem instances of an application problem generally have the same cumulative distribution of the number of search nodes by lower-bound values but with unknown parameters of the distribution. These distributions can be modeled a priori (Section 2) and their parameters, estimated at run time (Section 4).
- Based on these models, we study the behavior of an ideal IDA* algorithm and explains why existing IDA*-style algorithms perform well on some application problems (Section 2). We further show that there is a large disparity in the performance of an ideal IDA* algorithm when the rate of growth of nodes

expanded in successive stages can be chosen from a range of values (Appendix A).

- We present RIDA*, a new algorithm that uses regression to predict the threshold to be used in the next stage of an IDA* algorithm so that the number of nodes searched in successive stages grows in a geometric fashion (Section 3).
- Based on application problems evaluated (Section 4), we show conditions under which a specific IDA*-style algorithm should be used (Section 5).

In this context, we define an *ideal IDA* algorithm* as one that sets the threshold in each stage so that the number of nodes searched in the next stage always grows by a *constant* ratio. The performance of an IDA*-style algorithm evaluated in this paper is compared with that of A* for solving the same problem instance. Let n_{A*} (*resp.*, n_{IDA*}) be the number of nodes expanded by A* (*resp.*, IDA*-style algorithms) for solving a given problem instance. The objective of designing a good thresholding strategy of IDA* is to

$$\text{minimize}\frac{n_{IDA*}}{n_{A*}} \tag{1.1}$$

Note that any IDA*-style search can be evaluated using Eq. (1.1) and that the objective value of a problem instance is unknown until the instance is solved.

2. Performance Modeling of an Ideal IDA* Algorithm

In this section, we analyze the performance of an ideal IDA* algorithm for searches represented in state-space trees. We first present the performance of the ideal IDA* algorithm. We then derive its performance when the distribution of search nodes by lower bounds is exponential, geometric, or linear. Finally, we identify conditions for selecting appropriate thresholds.

2.1. *Performance of ideal IDA* searches*

We present in this section the performance of an ideal IDA* algorithm based on a general statistical distribution of lower-bound values in a search tree. Let $h(i)$ be the lower-bound value of node i, where h is an admissible function ($h(i) \leq h(j)$ if j is a successor of i). Let $f(x)$ be the distribution of the number of nodes whose lower bounds are less than or equal to x. Further, let θ_1, $\theta_2, \ldots, \theta_{n+1}$ be a sequence of thresholds in successive stages of an ideal IDA* algorithm, where θ_1 is the threshold used in the first stage, and θ_{n+1} is the last threshold. (The search always starts with the root, which is considered as stage 0.) Note that θ_1 is chosen heuristically when the search starts, and that $f(\theta_1)$ is the number of nodes expanded in the first stage.

When the optimal solution v_{opt} is found in the last stage, $\theta_n < v_{opt} \leq \theta_{n+1}$. If the number of nodes expanded by A* is $n_{A*} = f(v_{opt})$ and r is the growth ratio maintained by the ideal IDA* algorithm in successive stages, then

$$r = \frac{f(\theta_i)}{f(\theta_{i-1})} \Longrightarrow f(\theta_n) = r^{n-1} f(\theta_1), \quad \text{for} i = 2, \ldots, n\,. \tag{2.1}$$

Since $f(\theta_n) = r^{n-1} f(\theta_1) < f(v_{opt}) \leq f(\theta_{n+1}) = r^n f(\theta_1)$, this implies that

$$n = \left\lceil \log_r \frac{f(v_{opt})}{f(\theta_1)} \right\rceil \tag{2.2}$$

The number of nodes searched by the ideal IDA* algorithm is, therefore,

$$\begin{aligned} n_{IDA*} &= \sum_{i=1}^{n} f(\theta_i) + f'(\theta_{n+1}) = f(\theta_1) \sum_{i=0}^{n-1} r^i + f'(\theta_{n+1}) \\ &= f(\theta_1) \left(\frac{r^n - 1}{r - 1} \right) + f'(\theta_{n+1})\,, \end{aligned} \tag{2.3}$$

where $f'(\theta_{n+1})$ is the number of nodes searched in the last stage, and $f(\theta_n) < f'(\theta_{n+1}) \leq f(\theta_{n+1})$. $f'(\theta_{n+1})$ generally depends on how many search nodes get pruned when the incumbent value is found in the last stage. If GDFS were used in the last stage instead of a regular DFS, then the number of nodes expanded is usually much less.

Given Eq. (2.3), the objective of designing an ideal IDA* algorithm is to choose r such that

$$\min_{r \in (1,\infty)} \frac{n_{IDA*}}{n_{A*}} = \min_{r \in (1,\infty)} \frac{f(\theta_1)(\frac{r^n - 1}{r-1}) + f'(\theta_{n+1})}{f(v_{opt})} \tag{2.4}$$

From Eq. (2.2), when $f(v_{opt}) \leq f(\theta_1)$, there will only be one stage in the IDA* algorithm ($n = 1$). On the other hand, when $r \to 1$, we have $n \to \infty$ and the algorithm reduces to setting $f(\theta_i) = f(\theta_{i-1}) + 1$. It is obvious that a suitably chosen value of r will minimize Eq. (2.4).

In the extreme case in which the optimal solution has a value that is slightly larger than the threshold used in the n'th stage ($f(v_{opt}) = f(\theta_n) + 1$) and all nodes defined by θ_{n+1} are expanded in the $(n+1)$'st stage ($f'(\theta_{n+1}) = f(\theta_{n+1})$), then n_{IDA*}/n_{A*} in Eq. (2.4) is approximately $r^2/(r-1)$, and the optimal r that minimizes it is 2. (A similar result has been reported by Korf before.) Note that this is the value of r that minimizes the overhead of the ideal IDA* algorithm in the worst case. In the average case, the optimal r depends on the performance of GDFS and is difficult to characterize analytically.

For a specific value of r, we need to choose the θ's properly, which are directly related to distribution f. In the next three subsections, we model f using an exponential function, a geometric function, and a linear function.

2.2. *Exponential model*

In this model, the distribution profile of a search problem is approximated by the following continuous exponential function.

$$f(x) = cu^{ax} \tag{2.5}$$

where, c, a and u are positive real constants, and u is the *branching factor.*

Since r can be any real number, Eq. (2.1) becomes

$$r = \frac{f(\theta_i)}{f(\theta_{i-1})} = \frac{cu^{a\theta_i}}{cu^{a\theta_{i-1}}} = u^{a(\theta_i - \theta_{i-1})} \tag{2.6}$$

To achieve the given r, θ_i is set as

$$\theta_i = \theta_{i-1} + \frac{\log r}{a \log u}. \tag{2.7}$$

This means that by increasing θ_i by a constant $\frac{\log r}{a \log u}$, IDA* keeps $f(\theta_i)$ increasing in a geometric fashion by ratio r.

Traveling salesman problems (TSP), production planning problems (PP), and integer programming problems (IP) are examples whose distribution of the number of nodes by lower bounds follows an exponential model. (A detailed description of these problems is shown in Section 4.) As an illustration, we show in Figure 2 the cumulative distribution for a 20-city Euclidean TSP. The end of the line in the top right-hand corner represents the point when the optimal solution is found. To verify that the distribution is exponential, we regressed a linear function on the log plot and found a coefficient of determination R^2 $(0 \leq R^2 \leq 1)$ very close to 1,

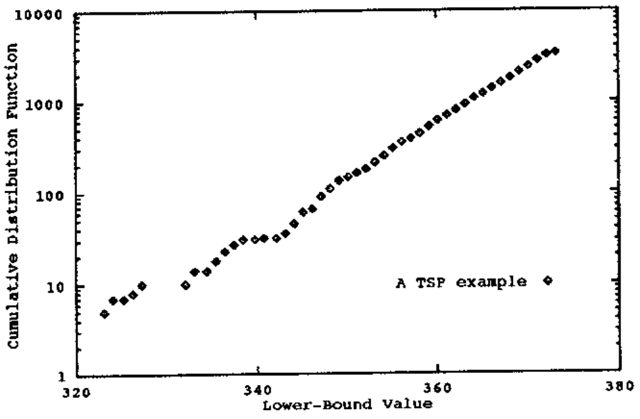


Fig. 2. Cumulative distribution of nodes in the search tree by lower-bound values for a 20-city Euclidean TSP problem instance.

which indicates a good fit. In Table 2, we show the mean and standard deviation of R^2 for 50 random instances of each of the 20-city Euclidean TSP, 18-period PP, and 30-variable IP problems. Note that R^2 for these problems have means very close to 1 and very small standard deviations.

Table 2. Average coefficient of determination R^2 of 50 random instances of the Euclidean TSP, PP, IP and MAZE problems. (Distribution f for the first three problems is exponential, whereas f for the maze problem is piecewise linear.)

Problem	Mean of R^2	Std. Dev. of R^2
20-city TSP	0.977	0.021
18-period PP	0.977	0.027
30-variable IP	0.972	0.032
40-by-40 MAZE	0.947	0.032

2.3. *Geometric model*

In this case, the lower bounds are drawn from a discrete distribution, and the distribution function and r are defined as follows.

$$f(x) = cu^{\lfloor ax \rfloor} \tag{2.8}$$

$$r = \frac{f(\theta_i)}{f(\theta_{i-1})} = \frac{cu^{\lfloor a\theta_i \rfloor}}{cu^{\lfloor a\theta_{i-1} \rfloor}} = u^{\lfloor a\theta_i \rfloor - \lfloor a\theta_{i-1} \rfloor} \tag{2.9}$$

Here, u is not the branching factor of the search tree but rather the average ratio of the number of nodes with lower bounds at one discrete value to the number of nodes at the next discrete value. We call u in the discrete case the *heuristic branching factor*. Note that r is restricted to $1, u, u^2, \ldots, u^s, \ldots$, where s is the number of discrete lower-bound values between the thresholds used in two successive stages of an IDA*-style algorithm. The objective of IDA* as defined in Eq. (1.1) becomes

$$\min_{r \in (u^s | s=0,1,2,\ldots)} \frac{n_{IDA*}}{n_{A*}} \tag{2.10}$$

Eq. (2.10) indicates that we need to find a value of s that minimizes this objective for given u, a, and c. In the following, we assume that the optimal solution is on the L'th discrete level (that is, $f(v_{opt}) = f(\theta_1)u^{L-1} + 1$), and compute the optimal s in the worst and average cases.

(a) *Best s in the worst case.* In the worst case, the optimal solution is not found in the n'th stage because it is pruned, and IDA* expands all nodes within threshold θ_{n+1} in the $n+1$'st stage before finding the optimal solution. Here, θ_{n+1} defines the $(L+s-1)$'st discrete level. Hence,

$$f'(\theta_{n+1}) = f(\theta_{n+1}) = f(\theta_1)u^{L+s-1} \tag{2.11}$$

The approximate total number of nodes expanded (as defined in Eq. (2.3)) is

$$n_{IDA*} = f(\theta_1)u^{L+s-1}(1 + u^{-s} + u^{-2s} + \cdots) = f(\theta_1)\frac{u^{L+2s-1}}{u^s - 1} \tag{2.12}$$

Hence, $s = k$ is better than $s = k + 1$ when

$$f(\theta_1)\frac{u^{L+2k-1}}{u^k - 1} \le f(\theta_1)\frac{u^{L+2k+1}}{u^{k+1} - 1} \Longrightarrow (u^{k+1} - u - 1)(u - 1) \ge 0\,. \tag{2.13}$$

Since $u > 1$, $s = 1$ is better than $s = 2$ when $u \ge 1.618$; $s = 2$ is better than $s = 3$ when $u \ge 1.325$; $s = 3$ is better than $s = 4$ when $u \ge 1.221$; and so on. When u approaches 1, s approaches infinity. Figure 3 shows the optimal s for different u.

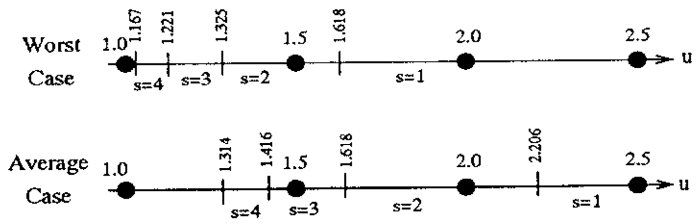


Fig. 3. Optimal s to minimize n_{IDA*}/n_{A*}.

(b) *Best s in the average case.* The exact analysis in the average case is intractable because it depends on the pruning mechanism in GDFS and where the incumbent lies. In the following, we present a simplified analysis.

In the last stage in which the optimal solution is found, θ_{n+1} can be one of the following discrete levels: L, $L + 1, \ldots, L + s - 1$. For simplicity, assume that θ_{n+1} can have any of these levels with equal probability $1/s$. (This probability is assumed to be independent of the number of nodes in each level in the search tree. A similar analysis can be carried out if we assume θ_{n+1} to have a probability that depends on the number of nodes in a level.)

Next, we find n_{IDA*} for each of these possible threshold values. Assuming that the optimal solution is on level L and that successive thresholds are separated by s levels, we identify two distinct situations where θ_{n+1} can lie: (i) $\theta_{n+1} = L + s - 1$, and (ii) $L \le \theta_{n+1} \le L + s - 2$.

In the first case, $\theta_{n+1} = L + s - 1$, which implies that $\theta_n = L - 1$. Since the optimal solution is on level L, the search in the last stage can be terminated immediately once the optimal solution is found. (Note that $\theta_n = L - 1$ implies that all nodes with lower-bound values below level L have been searched in stage n.) Assume on the average that half of the nodes in stage $n + 1$ are expanded.

$$f'(\theta_{n+1}) = \frac{1}{2}f(\theta_1)u^{L+s-1}\,. \tag{2.14}$$

The total number of nodes expanded is, therefore,

$$\begin{aligned} n_{IDA*}|_{\theta_{n+1}=L+s-1} &= f(\theta_1)\frac{u^{L+s-1}}{2} + f(\theta_1)u^{L-1}(1+u^{-s}+u^{-2s}+\cdots) \\ &= f(\theta_1)u^{L+s-1}\left(\frac{1}{2}+\frac{1}{u^s-1}\right) \end{aligned} \tag{2.15}$$

In the second case, $L \le \theta_{n+1} \le L+s-2$, which implies that $\theta_n < L-1$. When a feasible solution is found in Level L in stage $n+1$, the search cannot be terminated because nodes in levels between θ_n and L have not been searched completely. To prove that this solution is indeed optimal, it is necessary to continue the search. Assume on the average that the search expands half of the nodes between level L and threshold θ_{n+1} before finding the feasible solution, that no other feasible solution is found in stage $n+1$ once the first feasible solution has been found, and that it expands all nodes with lower bounds less than level L afterwards, The average number of nodes expanded in the last stage is

$$f'(\theta_{n+1}) = \frac{f(\theta_{n+1})-f(\theta_n)}{2} + f(\theta_n) = \frac{f(\theta_{n+1})+f(\theta_1)u^{L-1}}{2} \tag{2.16}$$

$$f'(\theta_{n+1})|_{\theta_{n+1}=L+i} = \frac{f(\theta_1)(u^{L+i}+u^{L-1})}{2} \tag{2.17}$$

where $i = 0,\ldots,s-2$. Therefore, the total number of nodes expanded by IDA* is

$$\begin{aligned} n_{IDA*}|_{\theta_{n+1}=L+i} &= f(\theta_1)\left[\frac{u^{L+i}+u^{L-1}}{2} + u^{L+i-s}(1+u^{-s}+u^{-2s}+\cdots)\right] \\ &= f(\theta_1)\left[u^{L+i}\left(\frac{1}{2}+\frac{1}{u^s-1}\right)+\frac{u^{L-1}}{2}\right]. \end{aligned} \tag{2.18}$$

Using Eq's (2.15) and (2.18), the average overhead of IDA*, assuming that i can have one of the values of $0,\ldots,s-1$ with equal probability, is

$$\begin{aligned} E[n_{IDA*}] &= \frac{1}{s}\sum_{i=0}^{s-1} n_{IDA*}|_{\theta_{n+1}=L+i} \\ &= \frac{f(\theta_1)}{s}\left\{\sum_{i=0}^{s-2}\left[u^{L+i}\left(\frac{1}{2}+\frac{1}{u^s-1}\right)+\frac{u^{L-1}}{2}\right]+u^{L+s-1}\left(\frac{1}{2}+\frac{1}{u^s-1}\right)\right\} \\ &= f(\theta_1)\left[\frac{u^L(u^s-1)}{s(u-1)}\left(\frac{1}{2}+\frac{1}{u^s-1}\right)+\frac{(s-1)u^{L-1}}{2s}\right] \end{aligned} \tag{2.19}$$

To know when $s=k$ is better than $s=k+1$, we solve

$$\begin{aligned} & f(\theta_1)\left[\frac{u^L(u^k-1)}{k(u-1)}\left(\frac{1}{2}+\frac{1}{u^k-1}\right)+\frac{(k-1)u^{L-1}}{2k}\right] \\ & \le f(\theta_1)\left[\frac{u^L(u^{k+1}-1)}{(k+1)(u-1)}\left(\frac{1}{2}+\frac{1}{u^{k+1}-1}\right)+\frac{[(k+1)-1]u^{L-1}}{2(k+1)}\right] \\ & \Longrightarrow \left[ku^{k+2}-(k+1)u^{k+1}-1\right] \ge 0, \quad \text{where } u>1, k>0, f(\theta_1)>0\,. \end{aligned} \tag{2.20}$$

Note that u^L is eliminated in the analysis, and the condition derived in Eq. (2.20) is independent of the optimal-solution value L. Eq. (2.20) shows that $s = 1$ is better than $s = 2$ when $u \geq 2.206$; $s = 2$ is better than $s = 3$ when $u \geq 1.618$; $s = 3$ is better than $s = 4$ when $u \geq 1.416$; and so on. Figure 3 shows the optimal s values for the average case.

Vertex-cover (VC) and 15-puzzle problems are example applications whose distribution of number of nodes by lower bounds can be modeled by a geometric distribution. Table 3 shows the statistics of an instance of a 30-vertex VC problem with an optimal solution of 19 and an average u of 7.7. We have also collected the average value of u for 50 random instances of the 30-vertex VC problem, and have found the average u to be 9.23 and standard deviation to be 5.22. For the first 50 15-puzzle problems studied by Korf [4], the average u is 10.22, and standard deviation is 8.24. For these problems, $s = 1$ is the optimal choice. (Our analysis is approximate since in any search problem, u is not constant.) This choice coincides with Korf's original IDA* algorithm in solving the 15-puzzle problem.

Table 3. Distribution of nodes in the search tree by lower-bound values of a 30-vertex VC problem.

Lower-Bound Value Value	Number of nodes Within Lower Bound	Heuristic Branching Factor
13	1	—
14	12	12.00
15	102	8.50
16	369	3.62
17	2128	5.77
18	18994	8.93
19	141104	7.43

2.4. *Linear model*

The distribution of the number of nodes by lower bounds is modeled by the following linear function:

$$f(x) = ax + b\,, \tag{2.21}$$

where a and b are constants and $a > 0$. Eq. (2.1) becomes

$$r = \frac{f(\theta_i)}{f(\theta_{i-1})} = \frac{a\theta_i + b}{a\theta_{i-1} + b} \Longrightarrow \theta_i = r\theta_{i-1} + \frac{b(r-1)}{a}\,. \tag{2.22}$$

In this case, θ_i needs to be increased by at least a ratio of r to allow $f(\theta_i)$ to be increased by a ratio of r. Since $r = 2$ is the optimal value in the worst case, doubling θ in each stage is nearly optimal. This strategy was used in DFS* which doubles the threshold in consecutive stages. As a result, DFS* performs very well for problems in this class.

The maze problem is an example whose lower-bound values follow a continuous linear distribution. Figure 4 illustrates this fact for a 40-by-40 maze problem instance generated using the X-maze program similar to that used by Vempaty, Kumar, and Korf [9]. The distribution shown can be modeled by two piecewise linear curves. This is verified by using the median of the smallest and the largest lower bounds as a point to divide the curve into two parts, and by fitting each by a linear line. We show in Table 2 the average coefficient of determination R^2 of regression and the corresponding standard deviation of the two linear fits for 50 random instances of the maze problem. As the average R^2 is very close to 1, the fit is close.

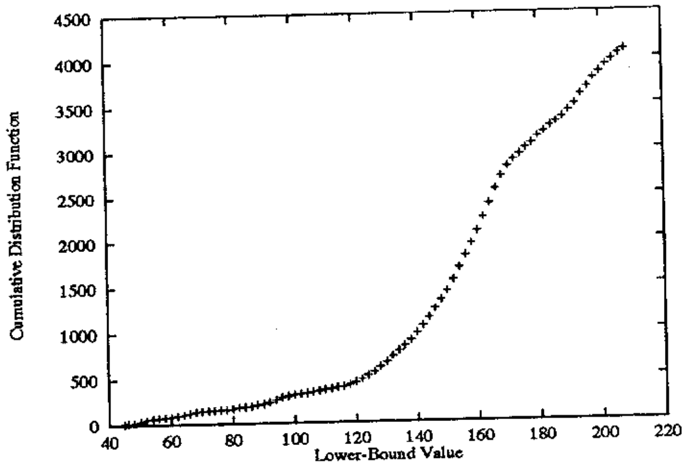


Fig. 4. Cumulative distribution of nodes in the search tree by lower-bound values of a 40-by-40 maze problem instance.)

We have shown in this section suitable choices of thresholds that can minimize the overhead of an ideal IDA* algorithm with respect to an A* algorithm. We have also explained conditions under which Korf's original IDA* algorithm and Vempaty, Kumar, and Korf's DFS* search will work well.

In general, it is hard to select thresholds so that a real IDA* algorithm behaves like an ideal IDA* algorithm. A user may wish to set a desired r in a problem, but the actual r may vary from stage to stage. This variation may lead to an unpredictable overhead of the IDA* algorithm. Appendix A illustrates this phenomenon by showing large variations in overheads for an ideal IDA* algorithm when r can be chosen from a range between r_{min} and r_{max}.

3. Dynamic Determination of Thresholds in RIDA*

We have shown in the last section that there exists an optimal r that can minimize the number of nodes expanded for a problem instance. In this section, we propose

RIDA*, a method that uses run-time distribution on lower bounds and applies regression to determine the best threshold to use in each stage. Our goal is to allow the overheads in successive stages to grow in a geometric fashion with a constant ratio.

RIDA* assumes that the distribution function of lower bounds (exponential, geometric, or linear) for an application problem is known. The parameters of the distribution function, however, are estimated by regressing on the (partial) distribution of lower bounds obtained at run time using a polynomial fit (first-order or second-order) or an exponential fit. The threshold to be used in the next stage is then estimated using the regressed function to achieve the desired growth ratio r.

In many search problems (such as Euclidean TSPs), feasible solutions can be found easily. The best of these solutions is kept in an incumbent, which is used in RIDA* to further reduce the search overhead. Recall that the overhead in the last stage of an IDA* algorithm dominates the overhead of all previous stages. Hence, if we have an incumbent value that is slightly larger than the predicted threshold for the next stage, then we can reduce the overall overhead by using the incumbent value as the next threshold.

The procedure used in RIDA* is as follows. At the end of a stage when the threshold for the next stage is set, the number of nodes $f(v_{inc})$ with incumbent value v_{inc} is calculated based on the estimated distribution function f. Assume that the predicted threshold for the next stage is θ_{i+1} where $f(\theta_{i+1}) = rf(\theta_i)$. We set θ_{i+1} to v_{inc} if $f(v_{inc}) \leq 2rf(\theta_i)$. In this case, the search becomes a guided depth-first branch-and-bound search. The reason for this is that if θ_{i+1} were used as the threshold, then the search may have one more stage, and the cumulative overhead would be larger than $2rf(\theta_i)$. In case that v_{inc} is very close to the current threshold θ_i, this method cuts the overhead by nearly half. This procedure used here is similar to that used in the last stage of DFS* [9].

In summary, RIDA* uses the following rules to find the threshold in each stage.

- Extend the threshold so that the number of nodes expanded in the next stage increases in a geometric fashion with constant ratio r.
- If the threshold does not include at least one unexpanded node, then the threshold is extended to a lower-bound value that includes at least one such node.
- If $f(v_{inc})$, the predicted number of nodes to be expanded when the threshold is set at the current incumbent value v_{inc}, is less than twice the number of nodes based on the predicted threshold in the next stage ($2f(\theta_{i+1}) = 2rf(\theta_i)$), then θ_{i+1} for the next stage is set to v_{inc}.

Note that RIDA* is different from IDA*_CR (Table 1) when setting the threshold for the next stage. RIDA* uses information on lower bounds of nodes expanded rather than lower bounds of active nodes that are pruned due to thresholding. We have found that a node pruned in the current stage does not imply that only one node will be expanded in the next stage, as multiple nodes may be expanded before hitting the next threshold. As a result, the information used by IDA*_CR in setting the threshold for the next stage may under-estimate the actual overhead experienced.

4. Experimental Results

4.1. *Generation of test problems*

The performance of various IDA*-style algorithms are studied by simulations using vertex-cover (VC), Euclidean Traveling-salesman (TSP), production-planning (PP), and integer-programming (IP) problems [12]. We first describe these test problems and their implementations before showing our experimental results.

VC entails finding a minimal set of vertices such that all edges have at least one of their vertices covered by this set. In our experiments on VCs, we generated the edges of graphs randomly with an edge-connectivity probability of 0.1. The lower bound of a graph was computed by finding the minimum number of vertices so that the total number of uncovered edges emanating from these vertices equaled the number of uncovered edges. The upper bound was evaluated by a steepest-descent hill-climbing algorithm.

TSP entails finding the shortest cyclic tour that traverses each city exactly once. In our experiments, we studied Euclidean TSPs in which cities were points on a Cartesian plane, and the distance between two cities was the Euclidean distance. We assume that cities are fully connected and the distance between each pair of cities are symmetric. In generating sample problems, we randomly assigned cities in a 100-by-100 Cartesian plane. We represented the search using a multi-way tree in which the descendant nodes of a parent node represented the set of unvisited cities from this node. (In a binary-tree representation, there are two descendant nodes that represent whether an unvisited city is traversed or not in the next city to be visited. This representation results in worse performance than that of a multi-way state-space representation.) A node in the search tree, therefore, represents a partial tour from the root to this node. We used the minimum spanning-tree algorithm to compute the lower bound of a node, and a nearest-neighbor hill-climbing algorithm to compute the upper bound.

PP entails finding a minimal-cost production schedule. In PPs, all capacities, requirements, production costs, inventory costs, and start-up costs were random integers in the following ranges: demand: [0,3]; capacity: [1,4]; set-up cost: [50,100]; incremental cost: [10,20]; and inventory cost: [5,10]. The number of periods specifies the problem size and is input by the user. We evaluated the lower bound of a problem by finding the minimal-cost schedule while ignoring start-up costs, and the upper bound by a hill-climbing heuristic.

IP entails finding an integral assignment of variables in order to maximize a linear objective function while satisfying a set of linear constraints. In our test problems, we assume that *upsilon*, the number of variables, is the same as the number of constraints. The coefficients in the objective function were random integers in the range [-20, 0]. The constraints were in the form $\mathbf{A}\vec{y} \leq \vec{b}$, where $\mathbf{A}$ was an v-by-v matrix whose elements were random integers in the range [0,20], and $\vec{b}$ was a vector whose elements were random integers in the range $[40v, 120v]$. We computed the lower bound by solving a relaxed linear programming problem using the Simplex method. No upper-bound function is available for solving IPs.

4.2. *Comparison with RBFS*

Recursive Best-First Search (RBFS) [11] has shown significant improvement over IDA* on certain problems. Korf reported that RBFS expanded about the same number of nodes as IDA* for the fifteen-puzzle problem, which shares similar characteristics as VCs. We, therefore, expect RBFS to perform similar to IDA* when applied to solve VCs.

Korf also reported that RBFS performed much better than IDA* for solving TSPs. With 10, 11 and 12 cities, RBFS generated an average of 16%, 16% and 18% of the nodes generated by IDA*, respectively. However, RBFS still generates significantly more nodes than GDFS. In Table 4, we show our experimental results of using A*, GDFS, RBFS and IDA* to solve 10-city, 11-city, and 12-city Euclidean TSPs. Each entry in the table represents the average of overhead normalized with respect to that of A* for 50 runs of randomly generated TSP instances. Our results show that for the 10-city TSPs, GDFS expands 9% more nodes than A* on the average. Although RBFS expands 17% of the nodes expanded by IDA*, it still expands 12.59 times more nodes than A* does. For the 11-city and 12-city TSPs, the improvement of GDFS over RBFS is even greater.

Table 4. Summary of normalized overheads of GDFS, RBFS and IDA* over A* for solving Euclidean TSPs.

Problem	GDFS		RBFS		IDA*	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
10-city TSPs	1.09	0.22	12.59	17.57	72.96	104.46
11-city TSPs	1.08	0.12	15.20	16.90	99.01	120.04
12-city TSPs	1.17	0.25	25.34	28.56	168.86	204.44

We conclude that RBFS should be used when IDA* performs well. This is true in the 15-puzzle problems and the vertex-cover problems studied in this paper.

4.3. *Comparison of RIDA*, GDFS, DFS*, IDA* and IDA*_CR*

In this section, we compare the performance of GDFS, DFS*, IDA*, RIDA* and IDA*_CR using 50 randomly generated instances of 30-vertex VCs, 20-city TSPs, 18-period PPs and 30-variable IPs. In our experiments on RIDA*, we used A* to generate 100 initial nodes and applied regression to pick the first threshold. The desired growth ratio r was set to 3 for both RIDA* and IDA*_CR.

Table 5. Summary of normalized overheads of various IDA*-style algorithms over A* for 50 instances of 30-vertex VCs, 20-city Euclidean TSPs, 18-period PPs and 30-variable IPs.

Problem	GDFS (DFS*)		IDA*		RIDA*		IDA*_CR	
	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.
30-vertex VCs	13.68	30.46	1.41	0.49	1.43	0.49	2.37	3.92
20-city TSPs	1.44	0.60	large	—	1.39	0.21	1.60	0.59
18-period PPs	1.28	0.24	large	—	1.41	0.26	1.82	0.50
30-variable IPs	2.90	2.66	large	—	2.04	0.54	4.47	1.19

Table 5 shows the average normalized overheads of the various algorithms with respect A*. In our experimental results, we have found that the performance of GDFS and that of DFS* is indistinguishable; hence, we present their results in one set. In the experimental results on IDA*, we indicate "large" for test problems in which IDA* takes much longer than the other algorithms and did not finish within a reasonable amount of time. Although RBFS will perform better than IDA* on these problems, we expect it to be worse than GDFS.

Our results show that all the IDA*-style algorithms and GDFS expand more nodes than A*. However, they all use a linear amount of memory whereas A* uses an exponential amount of memory. For this reason, we can only test problems of moderate sizes in our experiments, as we need to compare our results with respect to that of A*. We have also found that the normalized overheads can vary a lot across different problems instances. For example, GDFS has about the same average performance as RIDA* in solving TSPs, but has larger variations than RIDA*.

We cannot use statistical hypothesis testing to determine which of the algorithms has better average performance because such tests require populations with normal distributions, and the performance values are not normally distributed. To show the variance in performance, we present in Table 6 the average ranks of the various algorithms. The rank of an algorithm can be 1, 2, 3, or 4 in solving a problem instance, where 1 is the best rank.

For VCs, the lower bounds are discrete and can be modeled by a geometric distribution (Section 2). For these problems, IDA* has the smallest average overhead

Table 6. Average rank and the number of times of being the first rank of various IDA*-style algorithms for 50 instances of 30-vertex VCs, 20-city Euclidean TSPs, 18-period PPs, and 30-variable IPs.

Problem	GDFS (DFS*)		IDA*		RIDA*		IDA*_CR	
	Avg.	# of 1st	Avg.	# of 1st	Avg.	# of 1st	Avg.	# of 1st
VCs	3.52	7	1.60	25	2.68	2	2.20	16
TSPs	1.64	27	4.0	0	2.06	16	2.30	7
PPs	1.34	35	4.0	0	1.92	13	2.74	2
IPs	1.68	25	4.0	0	1.54	24	2.78	1

(as the heuristic branching degree of the problem is large). RIDA* has slightly more overhead than IDA*, which is attributed mainly to the initial overhead of expanding 100 nodes in order to do regression. IDA*_CR performs worse than IDA* and RIDA* on the average, and also has large variance in its performance. As shown in Table 6, IDA* has the best average rank, and IDA*_CR has better average rank than RIDA*. In 29 out of 50 problem instances, IDA*_CR has better performance than RIDA*. However, there are some instances in which IDA*_CR has poor performance. On the average, GDFS expands many more nodes than the other algorithms. The large standard derivation also indicates that in some instances, GDFS has very poor performance. Overall, IDA* expands the least number of nodes and has the highest chance to be the best.

For TSPs, PPs and IPs, their lower bounds are continuous and can be modeled by exponential distributions. Consequently, IDA* does not perform well for these problems. In these problems, the thresholds computed by IDA* are incremented too slowly, which cause many repeated expansion of the same node.

For TSPs, the number of nodes expanded by A* ranges from 178 to 132,107. For these problems, RIDA* gives the best average normalized overhead, and both GDFS and IDA*_CR have larger variances. GDFS has the best average ranks: in 27 out of 50 instances, GDFS has the best performance, and in most of these instances, the number of nodes expanded by GDFS is just slightly larger than that by A*. However, in a few instances, GDFS performs poorly that causes GDFS to have a larger average overhead and a larger variance.

For PPs and IPs, we have observed similar results. For PPs, GDFS is the best, RIDA* is second, and IDA*_CR is third. For IPs, RIDA* is the best, GDFS is second, and IDA*_CR is third. To visualize the performance difference, we depict in Figure 5 the performance distribution of GDFS, RIDA* and IDA*_CR for 50 random instances of TSPs. The graphs shows that GDFS and IDA*_CR performs poorly on some instances which results in large performance distributions.

In summary, our experiments shows that RIDA* has the smallest standard deviation in performance, and its average performance is very close to that of BFS. GDFS usually has large standard deviation in performance as compared to RIDA* and IDA*_CR. The performance of GDFS depends largely on the quality of the guidance and pruning functions. Hence, it performs well when the guidance function is accurate, and the application problem has many feasible solutions that can be used to prune unpromising nodes. Such is the case in PP; in this case, GDFS has performance very close to that of BFS.

We do not show separately the performance of DFS* in these tables. For all the problems studied, DFS*'s strategy of doubling the threshold each time makes it quickly become GDFS after a few stages. The performance of DFS* is slightly worse than that of GDFS for the problems studied.

We have also compared the empirical performance of RIDA* with respect to the theoretical bounds presented in Appendix A. Figures 6 shows the experimental results of RIDA* for 50 random instances of 20-city Euclidean TSPs and the lower

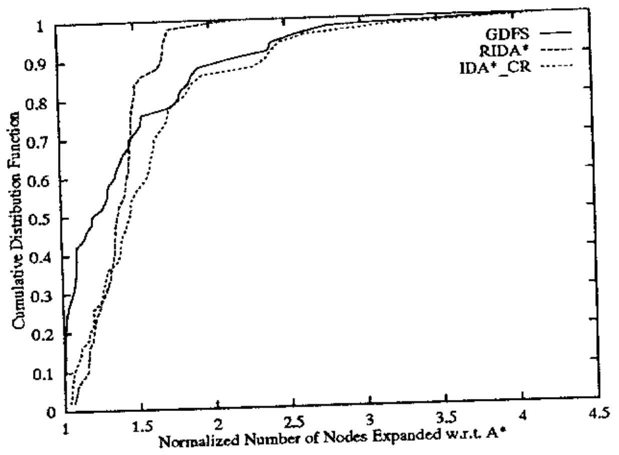


Fig. 5. Cumulative performance distribution of GDFS, RIDA* and IDA*_CR for 50 randomly generated 20-city Euclidean TSP instances.

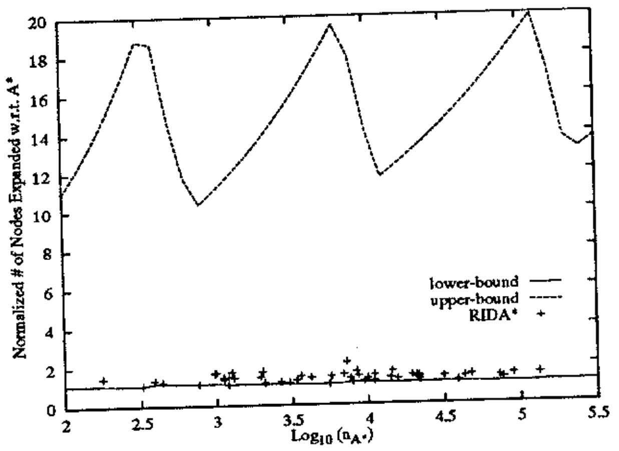


Fig. 6. Performance of RIDA* for 50 randomly generated 20-city Euclidean TSP instances and the corresponding performance bounds presented in Appendix A.)

and upper bounds of performance. Note that the bounds track the performance of RIDA* quite well, and that RIDA* has performance very close to the best performance. The bounds and the performance of RIDA* for other problems (VC, PP, and IP) are similar and are not repeated here.

5. Conclusions

In this paper, we have studied various methods for selecting thresholds in an IDA*-style algorithm. Our goal is to select thresholds so that the number of nodes searched in successive stages grows in a geometric fashion. We have modeled the distribution of lower-bound values by exponential, geometric, and linear distributions, and have derived the optimal thresholds in each case. We have also shown conditions under which Korf's original IDA* algorithm, Vempaty, Kumar, and Korf's DFS* algorithm, and Sarkar, *et al.*'s IDA*_CR will perform the best. Finally, we have presented RIDA*, a method that uses run-time distribution on lower bounds and applies regression to determine the best threshold to use in each stage.

We have evaluated these IDA*-style algorithms using random instances of the vertex-cover, traveling-salesman, production-planning, and integer-programming problems. We have found IDA* to be optimal when lower bounds are discrete and the heuristic branching factor is large (as in vertex-cover and 15-puzzle problems). DFS* performs well when the distribution of lower bounds is linear (as in maze problems). Further, DFS* is similar to GDFS and, therefore, performs well when GDFS performs well (as in production planning problems). We have found that RIDA* performs well when the distribution of lower-bound values can be modeled a priori (as in traveling-salesman problems using a multi-way state-space-tree representation and in integer programming problems). Our experiments also show that RIDA* has the most consistent performance (smallest standard deviation) in terms of deviations from a best-first search. On the other hand, IDA*_CR uses a discrete approximation of the distribution of lower bounds; its prediction is usually imprecise and may lead to unpredictable performance.

In summary, for a given problem instance, the best choice of the IDA*-style algorithm to use is as follows.

- Use Korf's original IDA* if the distribution of lower bounds is discrete and geometric with a large heuristic branching factor.
- Use DFS* if the distribution of lower bounds is linear or when GDFS performs well.
- Use RIDA* if the distribution of lower bounds is exponential.
- Otherwise, use IDA*_CR.

Appendix A. Theoretical Bounds on Performance of the Ideal IDA*

In this section, we derive the upper and lower bounds on performance of the ideal IDA* algorithm. The performance bounds are derived with respect to the exponential, linear and geometric models we have presented in Section 2.

A.1. Performance bounds with respect to the exponential and linear models

In these two models, the growth ratio r defined in Eq. (2.1) can be any real number. In this subsection, we present the upper and lower bounds of n_{IDA*}/n_{A*} defined in Eq. (1.1) for the ideal IDA* algorithm presented in Section 2. Our work is motivated by the observation that the number of nodes expanded by an IDA*-style algorithm may vary significantly when a different ratio of growth r is used.

The following assumptions are made in deriving the bounds.

- There is a constant ratio of growth (r) in number of nodes expanded from one stage of the ideal IDA* algorithm to anther.
- The value r can be any real number between r_{min} and r_{max}.
- Each stage starts with only the root node ($m = 1$ in Figure 1).

The next theorem shows the lower-bound normalized overhead of the ideal IDA* algorithm with respect to A*. This lower bound is derived for any choice of r between r_{min} and r_{max}.

Theorem A.1. For any real number r, $r_{min} \leq r \leq r_{max}$, the lower-bound normalized overhead of the ideal IDA* over A* is

$$\left(\frac{n_{IDA*}}{n_{A*}}\right)_{LB} = \begin{cases} 1 & n_{A*} \leq r_{max} \\ \frac{r_{min}^{k+1}-1}{(r_{min}-1)n_{A*}} + 1 & r_{max}^{k} < n_{A*} \leq r_{min}^{k+1} \text{ and } k \geq 1 \\ \frac{1}{r_{opt}-1}\left(r_{opt} - \frac{1}{n_{A*}}\right) & \text{otherwise} \end{cases} \quad (A.1)$$

where r_{opt} and k are

$$r_{opt} = exp\left[\frac{\log_e n_{A*}}{\lceil \log_{r_{max}} n_{A*} \rceil}\right] \quad (A.2)$$

$$k = \lceil \log_{r_{min}} n_{A*} \rceil - 1 \quad (A.3)$$

Proof. The proof is shown with respect to the three cases in the theorem. We will find the particular r between r_{min} and r_{max} that achieves the lower bound.

Case (1). $n_{A*} \leq r_{max}$. This is straightforward because $r = n_{A*}$ achieves the best performance, and $n_{IDA*}/n_{A*} = 1$.

Case (2). $r_{max}^{k} < n_{A*} \leq r_{min}^{k+1}$ for some integer k, $k \geq 1$. Under this condition, IDA* always takes $k + 1$ stages for any r, $r_{min} \leq r \leq r_{max}$. In the best case, the ideal IDA* expands n_{A*} nodes in the final stage no matter what value of r is used. Using r_{min} in the intermediate stages will lead to the fewest total number of nodes expanded; hence, $r = r_{min}$ is the best.

$$n_{IDA*}|_{r=r_{min}} = \sum_{j=0}^{k} r_{min}^{j} + n_{A*} \Longrightarrow \frac{n_{IDA*}}{n_{A*}} = \frac{r_{min}^{k+1} - 1}{(r_{min} - 1)n_{A*}} + 1 \quad (A.4)$$

where

$$k = \lceil \log_{r_{min}} n_{A*} \rceil - 1 \quad (A.5)$$

For example, if $r_{min} = 3$, $r_{max} = 3.5$, $n_{A*} = 624$, then $k = 5$, and the second condition in Eq. (A.1) is satisfied. n_{IDA*}/n_{A*} is evaluated to be 1.583.

Case (3). In all other cases, there is a set of values $\{r_0, r_1, \ldots, r_n\}$, $r_{max} \geq r_0 > r_1 > \cdots > r_n \geq r_{min}$, such that

$$r_i^{k+i} = n_{A*}, \quad 0 \leq i \leq n, \quad i \text{ and } k \text{ are integers, and} \tag{A.6}$$

$$k = \lceil \log_{r_{max}} n_{A*} \rceil \,. \tag{A.7}$$

In the best case, IDA* using r_i terminates in $k+i$ stages, and the optimal solution just lies *on the threshold* of the last stage.

As an example, when $r_{min} = 3$, $r_{max} = 10$ and $n_{A*} = 624$, we get $r_0 = 8.545$, $r_1 = 4.998$, and $r_2 = r_n = 3.623$ for $n = 2$ and $k = 3$. Using r_1 will result in $(k+1) = 4$ stages.

Let $T(r)$ be the overhead of IDA* using r. The proof for this case is shown in three parts.

1. $T(r_0) < T(r_i)$ for $1 \leq i \leq n$. The difference between using r_i over another r is in the number of nodes expanded in the intermediate stages, which is

$$\sum_{j=0}^{k+i-1} r_i^j, \quad 0 \leq i \leq n \tag{A.8}$$

Since $r_0^k = r_i^{k+i}$ and $r_0^{k-j} < r_i^{k+i-j}$, where $j = 1, 2, \ldots, k$ and $1 \leq i \leq n$, we get $T(r_0) < T(r_i)$.
Continuing with the same example, using $r_0 = 8.545$ results in the sum in Eq. (A.8) to be 82.57. Using $r_1 = 4.998$ results in the sum to be 154.93.
2. $T(\hat{r}_i) > T(r_i)$ for $r_i > \hat{r}_i > r_{i+1}$ and $0 \leq i < n$. IDA* using $\hat{r}_i$ has $k+i+1$ stages while IDA* using r_i has $k+i$ stages. In the best case, IDA* using $\hat{r}_i$ expands n_{A*} nodes in the last stage. Thus the difference between using r_i and $\hat{r}_i$ is in the number of nodes evaluated in the intermediate stages. We know that

$$T(r_i) = \sum_{j=0}^{k+i-1} r_i^j + n_{A*}\,, \tag{A.9}$$

$$T(\hat{r}_i) = \sum_{j=0}^{k+i} \hat{r}_i^{\,j} + n_{A*}\,. \tag{A.10}$$

If we can show that

$$\hat{r}_i^{\,j+1} > r_{i+1}^{j+1} > r_i^j, \quad 0 \leq j \leq k+i-1 \tag{A.11}$$

then $T(\hat{r}_i) > T(r_i)$. The first inequality in Eq. (A.11) is true because of our assumption that $\hat{r}_i > r_{i+1}$. The second inequality can be proved as follows.

We know from Eq. (A.6) that

$$r_{i+1}^{k+i+1} = r_i^{k+i} = n_{A*} \tag{A.12}$$

Dividing all terms by r_{i+1}^{k+i-j}, we get

$$\frac{r_{i+1}^{k+i+1}}{r_{i+1}^{k+i-j}} = r_{i+1}^{j+1} = \frac{r_i^{k+i}}{r_{i+1}^{k+i-j}} > \frac{r_i^{k+i}}{r_i^{k+i-j}} = r_i^j \tag{A.13}$$

Hence, Eq. (A.11) is proved.
Continuing with the previous example, if we choose $\hat{r}_0 = 6$, then $T(r_0) = T(8.545) = 706.57$ and $T(\hat{r}_0) = 883$, assuming that $n_{A*} = 624$ nodes are evaluated in the last stage when $r = \hat{r}$.

3. $T(\hat{r}) > T(r_0)$ for $r_0 < \hat{r} \leq r_{max}$ and both cases of using $\hat{r}$ and r_0 incur k stages. In the best case, both cases expand n_{A*} nodes in the final stage. However, using $\hat{r_0}$ has larger overhead in the intermediate stages. Since $\hat{r} \geq r_0$ and

$$T(r_0) = \sum_{j=0}^{k-1} r_0^j + n_{A*}, \tag{A.14}$$

$$T(\hat{r}) = \sum_{j=0}^{k-1} \hat{r}^j + n_{A*}, \tag{A.15}$$

we get $T(\hat{r}) \geq T(r_0)$.
The above three parts show that using $r_{opt} = r_0$ results in the lower-bound overhead. r_{opt} in Eq. (A.2) is found by substituting the value of k in Eq. (A.7) into Eq. (A.6). The number of nodes expanded using r_{opt} is

$$n_{IDA*}|_{r=r_{opt}=r_0} = \sum_{j=0}^{k} r_{opt}^j = \frac{r_{opt}^{k+1} - 1}{r_{opt} - 1} = \frac{r_{opt} n_{A*} - 1}{r_{opt} - 1} \tag{A.16}$$

This proves Case (3) of Eq. (A.1). $\square$

To illustrate Theorem A.1, the lower-bound normalized overheads for two combinations of r_{min} and r_{max} are shown in Figure 7. We have the following observations from these results: (a) the lower-bound n_{IDA*}/n_{A*} approaches one as r_{max} increases; (b) r_{min} has little effect on the lower-bound n_{IDA*}/n_{A*}; and (c) the frequency of variations is higher for smaller r_{max}. Note that the curves are not smooth because r_{opt} is different for different n_{A*}.

Next, we derive the upper bound on the normalized overhead of the ideal IDA* algorithm.

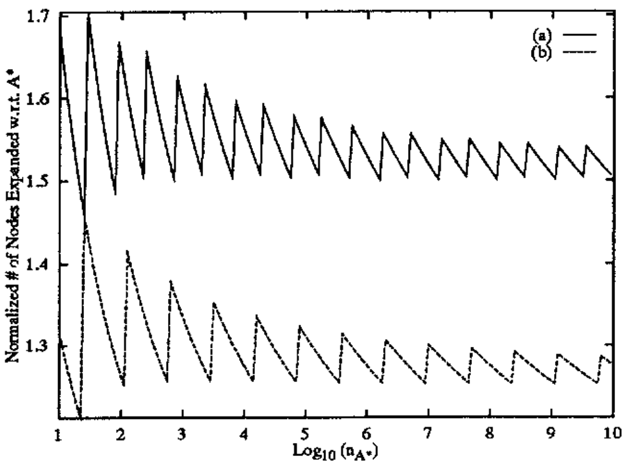


Fig. 7. Lower bounds of n_{IDA*}/n_{A*} for (a) $r_{min} = 2$ and $r_{max} = 3$ (top line), and (b) $r_{min} = 2$ and $r_{max} = 5$ (bottom line).

Theorem A.2. For any real number r, $1 < r_{min} \le r \le r_{max}$, the upper-bound normalized overhead of IDA* with respect to A* is

$$\left(\frac{n_{IDA*}}{n_{A*}}\right)_{UB} = \begin{cases} \frac{r_{max}}{n_{A*}} & n_{A*} \le r_{min} \\ \frac{r_{max}^{k+2}-1}{(r_{max}-1)n_{A*}} & r_{max}^k < n_{A*} \le r_{min}^{k+1},\ k \ge 1 \\ \max\left\{\frac{T(r_0)}{n_{A*}}, \frac{T(r_{max})}{n_{A*}}, \frac{r_n^2(n_{A*}-1)-1}{(r_n-1)n_{A*}}\right\} & \text{otherwise} \end{cases} \tag{A.17}$$

where

$$k = \lceil \log_{r_{min}} n_{A*} \rceil - 1;\ \ k_0 = \lceil \log_{r_{max}} (n_{A*} - 1) \rceil;\ \ r_0 = (n_{A*} - 1)^{\frac{1}{k_0}}; \tag{A.18}$$

$$k_n = \lfloor \log_{r_{min}} (n_{A*} - 1) \rfloor;\ \ r_n = (n_{A*} - 1)^{\frac{1}{k_n}};\ \text{ and }\ T(r) = \frac{r^2(n_{A*} - 1) - 1}{r - 1}. \tag{A.19}$$

Proof. The proof is shown for these three cases, respectively.

Case (1). When $n_{A*} \le r_{min}$, the worst case happens when using r_{max}, and the optimal solution is not found until all other nodes within the threshold are expanded. Thus, $n_{IDA*} = r_{max}$.

Case (2). $r_{max}^k < n_{A*} \le r_{min}^{k+1}$ for some integer k, $k \ge 1$. For any real number r, $r_{min} \le r \le r_{max}$, IDA* has $k + 1$ stages. In the worst situation, IDA* using r_{max} expands more nodes in all the k intermediate and the final stages, and the number of nodes expanded is

$$n_{IDA*} = \sum_{j=0}^{k+1} r_{max}^j = \frac{r_{max}^{k+2} - 1}{r_{max} - 1} \tag{A.20}$$

where $k = \lceil \log_{r_{min}} n_{A*} \rceil - 1$.

Case (3). In all other cases, there is a set of values $\{r_0, r_1, \ldots, r_n\}$, $r_{max} \geq r_0 > r_1 > \cdots > r_n \geq r_{min} = r_{n+1}$, such that

$$n_{A*} - 1 = r_i^{k_0+i}, \; 0 \leq i \leq n \tag{A.21}$$

where k_0 and i are integers, $k_{i+1} = k_i + 1$ and

$$k_0 = \lceil \log_{r_{max}}(n_{A*} - 1) \rceil; \;\; k_{n+1} = \lceil \log_{r_{min}}(n_{A*} - 1) \rceil \tag{A.22}$$

This means that IDA* using r_i terminates in $k_0 + i + 1$ stages.

For example, when $r_{min} = 3$, $r_{max} = 10$, $n_{A*} = 624$, we have $k_0 = \lceil \log_{10} 623 \rceil = 3$, and $k_{n+1} = \lceil \log_3 623 \rceil = 6$.

Let $T(r)$ be the worst possible overhead of the ideal IDA* using r. The proof is shown in three parts.

1. $T(r_i) \leq max(T(r_0), T(r_n))$ for $0 \leq i \leq n$. For each r_i, the worst case happens when the optimal solution is found in the $(k_0 + i + 1)$'st stage and is the last node visited after all nodes within this threshold have been expanded. The number of nodes expanded is, therefore,

$$T(r_i) = \sum_{j=0}^{k_0+i+1} r_i^j = \frac{r_i^{k_0+i+2} - 1}{r_i - 1} = \frac{r_i^2(n_{A*} - 1) - 1}{r_i - 1} \tag{A.23}$$

The minimum value of $T(r_i)$ when r_i is in $(1, \infty)$ is reached when

$$r_i = 1 + \left(1 - \frac{1}{n_{A*} - 1}\right)^{\frac{1}{2}} \tag{A.24}$$

Therefore, for r in the interval $\left[1 + \sqrt{1 - 1/(n_{A*} - 1)}, \infty\right)$, $T(r_i)$ is monotonically increasing with respect to r_i. On the other hand, for r_i in the interval $\left(1, 1 + \sqrt{1 - 1/(n_{A*} - 1}\right]$, $T(r_i)$ is monotonically decreasing with respect to r_i. Either $T(r_0)$ or $T(r_n)$ could be the largest.

Continuing with the example in Case (3) of this proof, we have $T(r_0) = 6,026$, $T(r_3) = 3,117$, and the minimum value of $T(r_i)$ is 2,491 when $r_i = 1.9992$.

2. $T(\hat{r_i}) < T(r_i)$ for $r_{i+1} < \hat{r_i} < r_i$ and $0 \leq i < n$. In both cases of using $\hat{r_i}$ and r_i, IDA* has a maximum of $k_0 + i + 1$ stages.

$$T(r_i) = \sum_{j=0}^{k+i+1} r_i^j \text{ and } T(\hat{r_i}) = \sum_{j=0}^{k+i+1} \hat{r_i}^j. \tag{A.25}$$

Since $r_i^k > \hat{r}_i^k$, $1 \leq k \leq (k_0 + i + 1)$, IDA* using $\hat{r}_i$ has less overhead in all intermediate and the last stages.

3. $T(\hat{r}) \leq T(r_{max})$ for $r_0 < \hat{r} \leq r_{max}$, and IDA* using $\hat{r}$ has k_0 stages. In this case,

$$T(\hat{r}) = \sum_{i=0}^{k_0} \hat{r}^i = \frac{\hat{r}^{k_0+1} - 1}{\hat{r} - 1} \tag{A.26}$$

When $\hat{r}$ increases, $T(\hat{r})$ increases. $T(\hat{r})$ reaches the maximum when $\hat{r} = r_{max}$.

Continuing with the example in this proof, $T(r_{max}) = 1,111$, $T(\hat{r}) = T(9) = 820$.

Since any one of the three values, $T(r_{max})$, $T(r_0)$ or $T(r_n)$ could be the largest, the upper bound on the overhead of IDA* in the worst case is

$$n_{IDA*} = \max(T(r_{max}), T(r_0), T(r_n)) \qquad \square$$

To illustrate Theorem A.2, the upper-bound normalized overhead of IDA* for two combinations of r_{min} and r_{max} are shown in Figure 8. We have the following observations from these results: (a) the upper bound increases either as r_{max} increases or as r_{min} goes to 1; (b) the frequency of variations is higher for smaller r_{max} and when r_{min} is not close to 1.

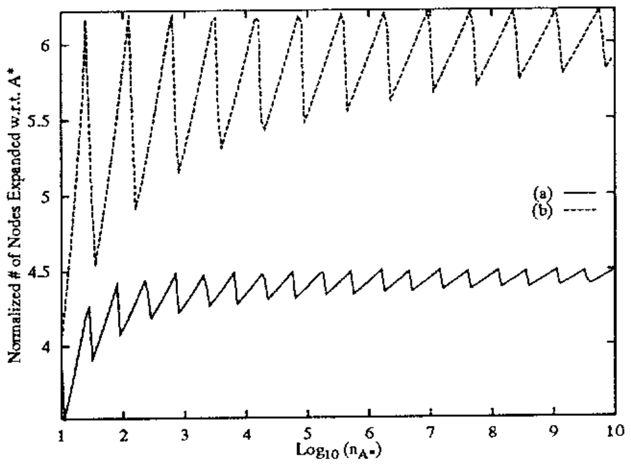


Fig. 8. Upper bounds of n_{IDA*}/n_{A*} for (a) $r_{min} = 2$ and $r_{max} = 3$ (bottom line), and (b) $r_{min} = 2$ and $r_{max} = 5$ (top line).

From Theorems A.1 and A.2, we have found the following results to be consistent with our observations. For a fixed r_{min}, a large r_{max} results in a large disparity between the lower and upper bounds of n_{IDA*}/n_{A*}. When r_{min} goes to 1, the upper bound of n_{IDA*}/n_{A*} goes to infinity whereas the lower bound is not much affected. Hence, small r_{min}'s close to 1 should be avoided.

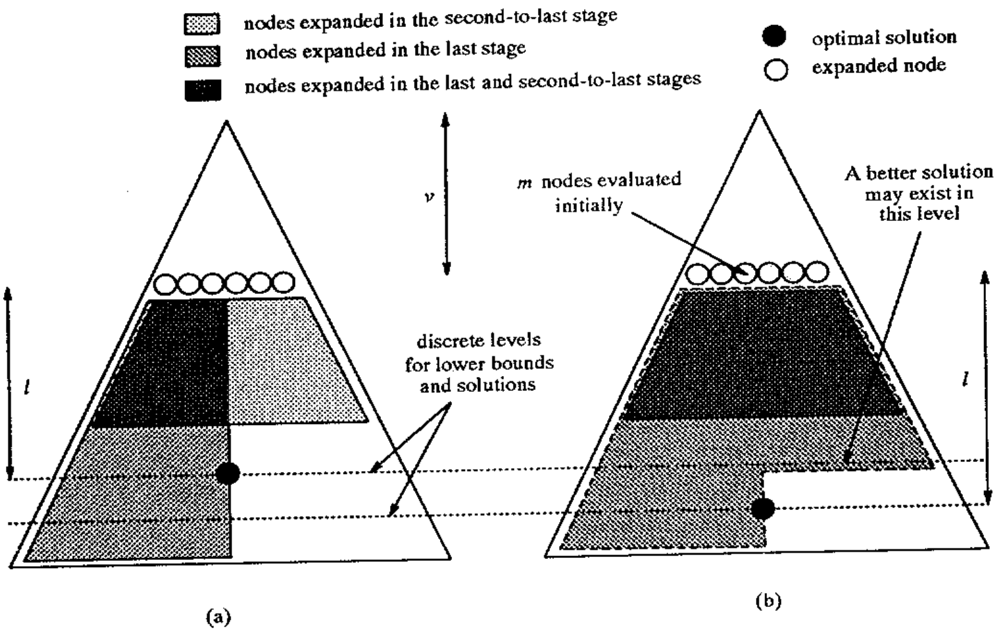
A.2. Performance bounds with respect to the geometric model

In this model, the growth ratio r specified in Eq. (2.9) is restricted to a set of values $\{1, u, u^2, \ldots, u^s, \ldots\}$. In this subsection, we present the upper and lower bounds of overheads of the ideal IDA* algorithm.

When lower-bound values of search nodes are discrete, A* does not always expands the same number of nodes before finding the optimal solution. This happens because the overhead depends on the order in which nodes in the last level are expanded. Therefore, the bounds we derive here are not normalized with respect to n_{A*} as we did in the last subsection. Instead, the performance of IDA* is compared with that of BFS. BFS in this context is a search that expands no other nodes in the same level as the optimal solution, and expands all nodes with lower bounds less than the optimal solution. We denote n_{BFS} as the number of nodes expanded by BFS.

For simplicity, we call the nodes with the same lower-bound values as nodes in the same level. We are interested in finding the number of nodes expanded rather than those generated because nodes in a level of the search tree are either all expanded or none expanded in any stage, whereas they may be partially generated in any given stage.

In the geometric model, two distinct situations can happen. (1) When a solution is found in one level below the threshold used in the second-to-last stage (Figure 9a), IDA* terminates immediately, and the solution found must be optimal because all nodes with lower-bounds less than it have been expanded in the second-to-last stage. In this case, very few nodes may need to be expanded in the last stage. (2) When a solution is found in a level other than the level immediately below the threshold used in the second-to-last stage, IDA* cannot terminate immediately because there may be a better solution in any intermediate levels in between. This phenomenon is illustrated in Figure 9b.



nodes expanded in the second-to-last stage
nodes expanded in the last stage
nodes expanded in the last and second-to-last stages
optimal solution
expanded node
v
l
m nodes evaluated initially
A better solution may exist in this level
discrete levels for lower bounds and solutions
l
(a)
(b)

Fig. 9. Two different situations in discrete IDA* search: (a) $L \bmod s = 1$ or $s = 1$, and (b) $L \bmod s \neq 1$.

The following assumptions are made in deriving the lower and upper bounds of performance of the ideal IDA*

1. $u > 1$ is the constant ratio of the number of nodes with lower bounds at one discrete value to the number of nodes at the next discrete lower-bound value.
2. The threshold of one stage always increases by s levels from the previous stage. (Note that s is in levels and not in lower-bound values.)
3. The value s can be any integer between s_{min} and s_{max} inclusively.
4. L is the level where the optimal solution lies. The root is in level 0.
5. Each stage starts with only the root node ($m = 1$).

Given that s is between s_{min} and s_{max}, the next theorem shows the lower-bound overhead.

Theorem A.3. For any integer s, $s_{min} \le s \le s_{max}$, The lower-bound overhead of IDA* is

$$n_{IDA*} = \min_{s \in [s_{min}, s_{max}]} T(s) \tag{A.27}$$

where $T(s)$, the minimum number of nodes expanded by IDA* using s, is

$$T(s) = \frac{1}{u-1}\left(u \frac{u^{s\lceil \frac{L}{s} \rceil} - 1}{u^s - 1} - \left\lceil \frac{L}{s} \right\rceil \right) + a \tag{A.28}$$

where a is

$$a = \begin{cases} 1 & L \bmod s = 1 \text{ or } s = 1 \\ \frac{u^L - 1}{u - 1} & \text{otherwise} \end{cases} \tag{A.29}$$

Proof. In the best case, no nodes are expanded beyond level L in the final stage, and the optimal solution is the first node expanded in level L. Thus, all other nodes in level L are pruned.

Let $T(s)$ be the overhead of the IDA* search. There are two cases.

1. $L \bmod s = 1$ or $s = 1$. In the best case, IDA* expands only the root in the last stage.
2. In other cases, IDA* expands no nodes in level L in the last stage but all nodes above level L. The total number of nodes expanded is $\sum_{i=0}^{L-1} u^i = \frac{u^L - 1}{u - 1}$.

These two cases only differ in the final stage. The number of nodes that IDA* expands in all intermediate stages is

$$N = \sum_{i=0}^{\alpha} \sum_{j=0}^{i \times s} u^j = \frac{1}{u-1}\left(u \frac{u^{s\lceil \frac{L}{s} \rceil} - 1}{u^s - 1} - \left\lceil \frac{L}{s} \right\rceil \right) \tag{A.30}$$

where $\alpha = \lceil \frac{L}{s} \rceil - 1$. This proves Eq. (A.28). □

To illustrate Theorem A.3, we show in Figures 10 the lower-bound normalized overhead of IDA* over BFS for various combinations of u, s_{min}, and s_{max}. From these results, we have the following observations. (a) The lower-bound normalized overhead goes to 1 as s_{max} increases. (b) A larger u gives a smaller normalized overhead. (c) The frequency of variations is higher for smaller s_{max}'s. Note that the curves are not smooth because s_{opt} is different for different n_{BFS}'s. These observations are consistent with those of the exponential and linear models.

Next, we derive the worst-case normalized overhead of IDA*.

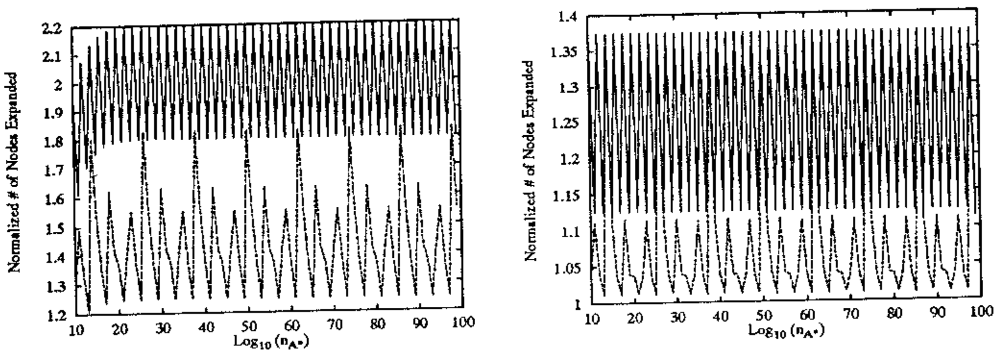


Fig. 10. Lower-bound normalized overhead of IDA*. Left: $u = 1.5$, $s_{min} = 1$, $s_{max} = 2$ (top line), and $u = 1.5$, $s_{min} = 1$, $s_{max} = 4$ (bottom line). Right: $u = 3$, $s_{min} = 1$, $s_{max} = 2$ (top line), and $u = 3$, $s_{min} = 1$, $s_{max} = 4$ (bottom line).

Theorem A.4.. For $s_{min} \le s \le s_{max}$, the upper-bound overhead of IDA* is

$$n_{IDA*} = \max_{s \in [s_{min}, s_{max}]} T(s) \tag{A.31}$$

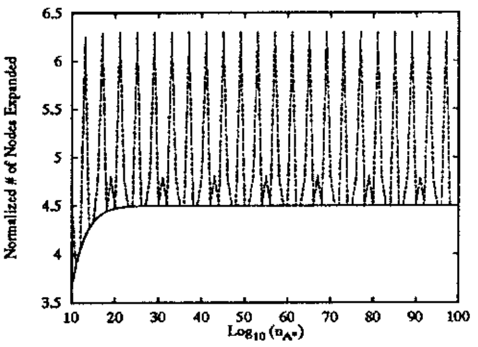
where $T(s)$ is the maximum number of nodes expanded by IDA* using s:

$$T(s) = \frac{1}{u-1}\left(u\frac{u^{s\lceil \frac{L}{s} \rceil + s} - 1}{u^s - 1} - \left\lceil \frac{L}{s} \right\rceil - 1 \right) \tag{A.32}$$

Proof. In the worst case, IDA* expands all nodes with lower bounds up to and including the threshold in the final stage. The total number of nodes expanded is

$$n_{IDA*} = \sum_{i=0}^{\lceil \frac{L}{s} \rceil} \sum_{j=0}^{i \times s} u^j = \frac{1}{u-1}\left(u\frac{u^{s\lceil \frac{L}{s} \rceil + s} - 1}{u^s - 1} - \left\lceil \frac{L}{s} \right\rceil - 1 \right) \quad \square . \tag{A.33}$$

To illustrate Theorem A.4, we plot in Figures 11 the upper-bound normalized overheads of IDA* over BFS for various combinations of u, r_{min}, and r_{max}. From these results, we have the following observations. (a) The upper-bound normalized overhead increases as s_{max} increases. (b) The upper bounds vary in a large range for large u's.



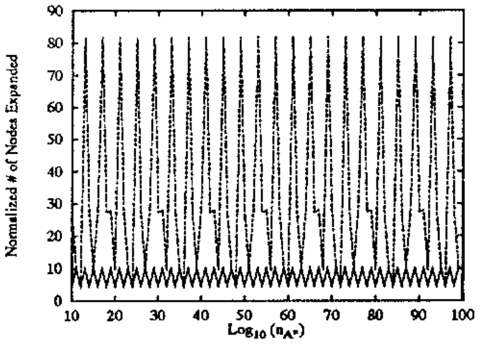


Fig. 11. Upper-bound normalized overhead of IDA*. Left: $u = 1.5$, $s_{min} = 1$, $s_{max} = 2$ (bottom line), and $u = 1.5$, $s_{min} = 1$, $s_{max} = 4$ (top line). Right: $u = 3$, $s_{min} = 1$, $s_{max} = 2$ (bottom line), and $u = 3$, $s_{min} = 1$, $s_{max} = 4$ (top line).

From Theorems A.3 and A.4, we have the following conclusions. (a) For a fixed s_{min}, a large s_{max} results in a better lower-bound n_{IDA*}/n_{BFS} and a worse upper-bound n_{IDA*}/n_{BFS}. (b) For a fixed s_{min} and s_{max}, a large u results in a better lower-bound n_{IDA*}/n_{BFS} and a much worse upper-bound n_{IDA*}/n_{BFS} than those of a small u. Therefore, when u is not close to one, we should avoid using large s_{max}'s. Our conclusions drawn here for the geometric models are consistent with those drawn for the exponential and linear models studied in the last subsection.

Acknowledgments

The authors would like to thank Dr. Lon-Chan Chu for providing his software package WISE [13] that implements the search procedures and application problems studied in this paper.

References

[1] A. Barr and E. A. Feigenbaum, eds., *Handbook of Artificial Intelligence, Volume I.* Los Altos, CA: William Kaufmann (1981).

[2] S. Russell, *Efficient memory-bounded search methods*, Proc. ECAI-92, (Vienna, Austria) (1992).

[3] D. Slate and L. Atkin, *Chess 4.5 — the northwestern university chess program*, Chess Skill in Man and Machine, P. W. Frey, ed., New York: Springer-Verlag (1977).

[4] R. E. Korf, *Depth-first iterative deepening: An optimal admissible tree search*, Artificial Intelligence **27** (1985) 97–109.

[5] R. E. Korf, *Optimal path finding algorithms*, Search in Artificial Intelligence, L. Kanal and V. Kumar, eds., New York: Springer-Verlag (1988) 223–267.

[6] A. K. Sen and A. Bagchi, *Fast recursive formulations for best-fist search that allow controlled use of memory*, Proc. Int'l Joint Conf. on Artificial Intelligence, (Detroit, MI), IJCAI, Inc (1989) 297–302.

[7] P. P. Chakrabarti, S. Ghose, A. Acharya, and S. C. D. Sarkar, *Heuristic search in restricted memory*, Artificial Intelligence **41** (1989) 197–221.

[8] S. Ghosh, A. Mahanti, and D. S. Nau, *Its: An efficient limited-memory heuristic tree search algorithm*, Proc. National Conf. on Artificial Intelligence, AAAI (1994) 1353–1358.

[9] V. N. Rao, V. Kumar, and R. E. Korf, *Depth-first vs best-first search*, Proc. National Conf. on Artificial Intelligence, (Anaheim, CA), AAAI (1991) 434–440.

[10] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, and S. C. D. Sarkar, *Reducing reexpansions in iterative-deepening search by controlling cutoff bounds*, Artificial Intelligence **50** Elsevier Science Publishers (1991) 207–221.

[11] R. E. Korf, *Linear-space best-first search*, Artificial Intelligence **62** (1993) 41–78.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman (1979).

[13] B. W. Wah and L.-C. Chu, *Combinatorial search algorithms with meta-control: Modeling and implementations*, Int'l J. of Artificial Intelligence Tools, vol. 1, World Scientific Publishers (Sept. 1992) 369–397.