A COMPARATIVE STUDY OF IDA*-STYLE SEARCHES *

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

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 use for pruning. We first present three models to approximate the distribution of number of search nodes by lower bounds: exponential, geometric, and linear, and illustrate these distributions based on some well-known combinatorial search problems. We then 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 previous experience the distribution for a given problem instance but will not be able to determine the parameters of the distribution. Hence, we develop RIDA*, a method that estimates dynamically the parameters of the distribution, and predicts the best threshold to use. Finally, we compare the performance of several IDA*-style algorithms — Korf's IDA*, RIDA*, IDA*_CR and DFS* — on several application problems, and identify conditions under which each of these algorithms will perform well.*

1 Introduction

Many search problems for finding optimal solutions in combinatorially large solution spaces 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*). However, it requires an exponential amount of space. A *guided depth-first search* (*GDFS* or depth-first branch-and-bound search), on the other hand, requires memory space linear in the size of the problem. However, with a lack of a good pruning function, GDFS is prone to search far deeper than where the optimal solution lies.

In his seminal paper, Korf presented iterative deepening A* or IDA* [1], 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). Like A*, it 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.

Korf originally demonstrated the performance of IDA* using the 15-puzzle problem [1]. 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 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, and suggested improvement by using thresholds sufficiently exceeding the value of the minimum leaf of the previous stage [2].

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

Proceedings of 6th IEEE International Conference on Tools with Artificial Intelligence, November, 1994

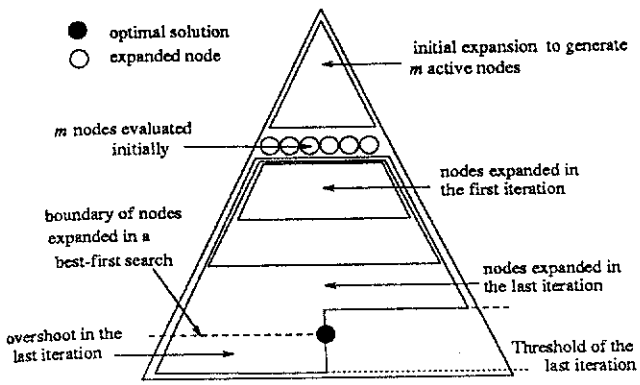

Figure 1: An illustration of MREC. (Nodes in the tree are by their lower-bound values; thresholds indicate lower-bound values of nodes *expanded*.)

A number of search algorithms were developed subsequently to improve the performance of the original IDA*. These algorithms were developed for application problems with characteristics that are different from the 15-puzzle problem. 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. Examples of the various IDA*-style algorithms proposed include MREC [3], DFS* [4], and IDA*_CR [5].

MREC [3] is essentially IDA* that uses extra memory to save active nodes near the root and to avoid expanding these nodes in every stage. The operations of MREC is illustrated in Figure 1. IDA* is a special case of MREC with $m = 1$. DFS* [4] is a hybrid of IDA* and GDFS. IDA*_CR [5] is a variant of IDA* that collects some statistics in the search process to help determine the threshold in the next stage. Table 1 summarizes the information used in these algorithms and the corresponding strategies. It also lists RIDA*, an algorithm discussed in Section 3.

Previous IDA*-style algorithms use different information and heuristics to predict the best threshold to use in a stage. Since the information and the heuristics used is generally problem dependent, it is possible to use other run-time information for this prediction. We propose in this paper to use the cumulative distribution of number of search nodes by lower-bound values.

Our results presented in this paper are summarized as follows.

- Different problem instances of an application problem generally have the same cumulative distribution of number of search nodes by lower-bound values and used during a search. These distributions can be modeled a priori (Section 2).
- 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 present RIDA*, a new algorithm that uses regression to predict thresholds so that the number of nodes searched in successive stages grows in a geometric fashion (Section 3).
- Based on various application problems evaluated (Section 4), we show conditions under which a specific IDA*-style algorithm should be used (Section 5).

The performance of an IDA*-style algorithm evaluated in this paper is compared with that of A*. Let n_{A*} (*resp.*, n_{IDA*}) be the number of nodes expanded by A* (*resp.*, IDA*-style algorithms). The objective of designing a good IDA* strategy is to

$$\text{minimize } \frac{n_{IDA*}}{n_{A*}} \tag{1}$$

2 Modeling of an Ideal IDA*

In this section, we first present the performance of an ideal IDA* algorithm. We then study the selection of thresholds when the distribution of search nodes by lower bounds is exponential, geometric, or linear.

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 $h(n)$ be the lower-bound value of node n, where h is an admissible function ($h(n) \leq h(m)$ if m is a successor of n). Let $f(x)$ be the distribution of the number of nodes whose lower bounds are less than or equal to x. Further, let $\theta_1, \theta_2, \cdots, \theta_{n+1}$ be a sequence of thresholds in successive stages of an ideal IDA* algorithm, where θ_1 is the initial threshold and θ_{n+1} the last.

Let r be the growth ratio maintained by the ideal IDA* algorithm in successive stages, and

$$f(\theta_n) = r^{n-1} f(\theta_1)\,. \tag{2}$$

When the optimal solution v_{opt} is found in the last stage, $\theta_n < v_{opt} \leq \theta_{n+1}$. The number of nodes expanded by A* is $n_{A*} = f(v_{opt})$. The number of nodes

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

Algorithm	*Information used in each stage in setting thresholds*	*Strategy*
IDA*	Minimum lower-bound value of active nodes exceeding the current threshold	Set the next threshold to this value
MREC	Same as IDA*	Same as IDA*
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 a predefined number b_i, where b is a user-defined factor and i is 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 factor

searched by the ideal IDA* algorithm is,

$$n_{IDA*} = \sum_{i=1}^{n} f(\theta_i) + f'(\theta_{n+1}) \tag{3}$$

where $f'(\theta_{n+1})$ is the number of nodes searched in the last stage, and $f(\theta_n) < f(v_{opt}) \leq f'(\theta_{n+1}) \leq f(\theta_{n+1})$. If GDFS is used in each stage of the IDA* algorithm, then $f'(\theta_{n+1})$ depends on how many nodes within θ_{n+1} get pruned in the last stage.

Given Eq. (3), the objective of designing an ideal IDA* algorithm is to choose r such that

$$\min_{r\in(1,\infty)} \frac{n_{IDA*}}{n_{A*}} = \min_{r\in(1,\infty)} \frac{f(\theta_1)(\frac{r^n-1}{r-1}) + f'(\theta_{n+1})}{f(v_{opt})} \tag{4}$$

When $r > f(v_{opt})/f(\theta_1)$, there will only be one stage in the IDA* algorithm ($n = 1$) and will be the same as GDFS. On the other hand, when $r \to 1$, we have $n \to \infty$. It is obvious that a suitably chosen value of r will minimize Eq. (4).

In the extreme case in which the optimal solution 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 searched in the $n+1$'st stage ($f'(\theta_{n+1}) = f(\theta_{n+1})$), then n_{IDA*}/n_{A*} is approximately $r^2/(r-1)$, and the optimal r is 2. (A similar result has been reported by Korf before.) 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.

$$f(x) = cu^{ax} \tag{5}$$

where, c, a and u are positive real constants, and u is the *branching factor*. Eq. (2) becomes

$$r = f(\theta_i)/f(\theta_{i-1}) = u^{a(\theta_i - \theta_{i-1})}.$$

To achieve the given r, θ_i is set as

$$\theta_i = \theta_{i-1} + \frac{\log r}{a \log u} \tag{6}$$

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 number of nodes by lower bounds follows an exponential model. As an illustration, we show in Figure 2 the cumulative distribution for a 16-city fully-connected symmetric TSP. To verify that the distribution is exponential, we regress a linear function on the log plot, and show a coefficient of determination R^2 ($0 \leq R^2 \leq 1$) very close to 1, which indicates a good fit. In Table 2, we show the mean and standard

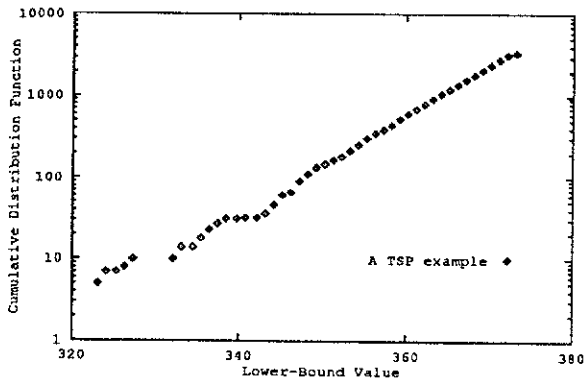

Figure 2: Cumulative distribution of nodes in the search tree by lower-bound values for a 16-city symmetric TSP problem instance.

Table 2: Average Coefficient of determination R^2 of 50 random instances of TSP, PP, IP and MAZE problems. (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
TSP	0.977	0.021
PP	0.977	0.027
IP	0.972	0.032
MAZE	0.947	0.032

deviation of R^2 for 50 random instances of each of the 16-city fully connected symmetric TSP, 18-plant PP, and 30-variable IP problems. Note that the means of R^2 for these problems are all very close to 1 and the standard deviations are very small.

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.

$$f(x) = cu^{\lfloor ax \rfloor} \tag{7}$$

and

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

Here, u is the average 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. We call r in the discrete case the *heuristic branching factor*. Note that r is restricted to 1, u, u^2, ... , u^s, ...

The objective of IDA* as defined in Eq. (4) becomes

$$\min_{r\in(u^s|s=0,1,2,\cdots)} \frac{n_{IDA*}}{n_{A*}} \tag{8}$$

In the following, we assume that the optimal-solution value is L (that is, $f(v_{opt}) = f(\theta_1)u^L$), and compute the optimal s in the worst and average cases.

(1) *Optimal 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. Hence,

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

The approximate total number of nodes expanded is

$$\begin{aligned} n_{IDA*} &= f(\theta_1)u^{L+s-1}(1 + u^{-s} + u^{-2s} + \cdots) \\ &= f(\theta_1)\frac{u^{L+2s-1}}{u^s - 1} \end{aligned}$$

Hence, $s = k$ is better than $s = k+1$ when $(u^{k+1} - u - 1)(u - 1) \geq 0$. Since $u > 1$, by this inequality, $s = 1$ is better than $s = 2$ when $u \geq 1.618$; and so on. Figure 3 shows the optimal s as a function of u.

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

In the last stage, θ_{n+1} can have one of the following values: L, $L+1, \cdots, L+s-1$. Assume that θ_{n+1} can have any of these values 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. Two distinct situations are identified:

(a) $\theta_{n+1} = L + s - 1$. Since the optimal solution is L, the search in the last stage can be terminated immediately once the optimal solution is found. Assume on the average that half of the nodes in stage $n+1$ are expanded, then $f'(\theta_{n+1}) = \frac{1}{2}f(\theta_1)u^{L+s-1}$. The total number of nodes expanded is, therefore,

$$n_{IDA*s-1} = f(\theta_1)u^{L+s-1}\left(\frac{1}{2} + \frac{1}{u^s - 1}\right) \tag{9}$$

(b) $L < \theta_{n+1} \leq L + s - 2$. The search in stage $n+1$ cannot be terminated once a feasible solution is found because this solution cannot be determined to be optimal unless all nodes with lower bounds less

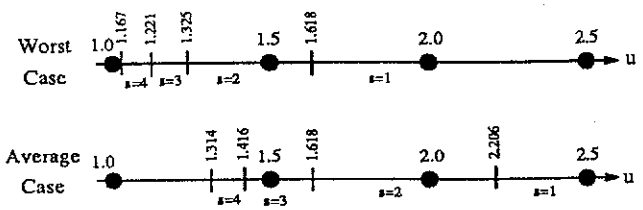

Figure 3: Optimal s to minimize n_{IDA*}/n_{A*}.

than this value have been expanded. Assume on the average that half of the nodes between L and θ_{n+1} are expanded. The average number of nodes expanded in the last stage is

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

where $i = 0, 1, \cdots, s-2$. The total number of nodes expanded by IDA* is

$$n^i_{IDA*} = f(\theta_1)\left[u^{L+i}\left(\frac{1}{2} + \frac{1}{u^s - 1}\right) + \frac{u^{L-1}}{2}\right] \quad (10)$$

Using Eq's (9) and (10), the average overhead of IDA*, assuming that i can have one of the values of $0, 1, \cdots, s-1$ with equal probability, is

$$E[n_{IDA*}] = \frac{1}{s}\sum_{i=0}^{s-1} n^i_{IDA*}$$

$$= f(\theta_1)\left[\frac{u^L(u^s - 1)}{s(u-1)}\left(\frac{1}{2} + \frac{1}{u^s - 1}\right) + \frac{s-1}{2s}u^{L-1}\right].$$

By this equation, $s = k$ is better than $s = k+1$ when

$$\frac{ku^{k+2} - (k+1)u^{k+1} - 1}{ku(k+1)(u-1)} \geq 0$$

where $u > 1$ and $k > 0$. Thus, $s = 1$ is better than $s = 2$ when $u \geq 2.206$; and so on. Figure 3 shows the optimal s values for the average case. Further, when u is less than 2.206, the optimal s is larger in the average case than that in the worst case.

Vertex-cover (VC) and 15-puzzle problems are examples whose distribution of number of nodes by lower bounds can be modeled by a geometric distribution. We collected the average value of u for 50 random instances of the 30-vertex VC problem, and 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, [1] 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.

2.4 Linear Model

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

$$f(x) = ax + b \quad (11)$$

where a and b are constants and $a > 0$. Eq. (2) becomes

$$r = \frac{f(\theta_i)}{f(\theta_{i-1})} = \frac{a\theta_i + b}{a\theta_{i-1} + b} \Rightarrow \theta_i = r\theta_{i-1} + \frac{b(r-1)}{a}.$$

Since $r = 2$ is the optimal value in the worst case, doubling θ in each stage is near optimal. DFS* implements this strategy and 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. We show in Table 2 the average coefficient of determination R^2 of linear regression and the corresponding standard deviation for 50 random instances of the 40-by-40 maze problem. As the average R^2 is very close to 1, the fit is close.

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 this function 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 regression function to achieve the desired growth ratio r.

In many search problems (such as symmetric 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, in a stage where we have an incumbent value

that is slightly larger than the threshold of this stage, then instead of completing this stage and start the final stage, we can reduce the overhead by combining the last two stages into one.

To summarize, RIDA* uses the following rules to compute the threshold in each stage.

1. Extend the threshold so that the number of nodes expanded in the next stage increases in a geometrical fashion with constant r.

2. 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.

3. 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}.

4 Experimental Results

The performance of various IDA*-style algorithms are studied by simulations using 30-vertex VC, 16-city fully connected symmetric TSP, 18-plant PP, and 30-variable IP problems.

We evaluated all the search algorithms using 50 randomly generated instances of each application problem. RIDA* always expanded 100 initial nodes using A* in order for us to apply regression for picking the first threshold. The desired growth ratio r was set to 3 for both RIDA* and IDA*_CR.

Table 3 shows the normalized overheads of the various algorithms over A*, and Table 4 shows the average ranks of the various algorithms. Being the best for a problem instance gives the algorithm rank 1; whereas being the worst gives it rank 4.

As is discussed in Section 2, the lower bounds of VC are discrete and can be modeled by a geometric distribution. For this problem, IDA* has the smallest average overhead (as the heuristic branching degree of the problem is large). RIDA* has slightly more overhead than IDA*. IDA*_CR performs worse than IDA* and RIDA* on the average, and also has large variance in its behavior.

For TSP, PP, and IP, their lower bounds are continuous and can be modeled by exponential distributions. Tables 3 and 4 show the experimental results of these problems. For all these problems, IDA* does not perform well. For TSP, RIDA* gives the best average normalized overhead; both GDFS and IDA*_CR have large variances, and GDFS has the best average ranks. Since the number of nodes expanded by A* for these problems ranges from 95 to 233,410, there are a few instances in which GDFS and IDA*_CR get very poor normalized overheads. For PP, GDFS is the best, and RIDA* is second. For IP, RIDA* is the best, and GDFS is second.

Our experiments show 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.

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.

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 with a constant ratio. 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 traveling-salesman, integer-programming, vertex-cover, and production-planning 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

Table 3: Summary of normalized overheads of various IDA*-style algorithms over BFS for 50 instances of VC, TSP, PP and IP.

Prob.	GDFS (DFS*)		IDA*		RIDA*		IDA*_CR	
	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.	Avg.	S.D.
VC	13.68	30.46	1.41	0.49	1.43	0.49	2.37	3.92
TSP	2.80	5.90	9.49	2.57	1.76	0.29	2.33	4.32
PP	1.28	0.24	94.31	36.80	1.41	0.26	1.82	0.50
IP	2.90	2.66	large	large	2.04	0.54	4.47	1.19

Table 4: Average rank and the times of being the first rank of various IDA*-style algorithms for 50 instances of VC, TSP, PP and IP.

Prob.	GDFS (DFS*)		IDA*		RIDA*		IDA*_CR	
	Avg.	# of 1st	Avg.	# of 1st	Avg.	# of 1st	Avg.	# of 1st
VC	3.52	7	1.60	25	2.68	2	2.20	16
TSP	1.82	22	3.94	0	2.18	15	2.06	13
PP	1.34	35	4.0	0	1.92	13	2.74	2
IP	1.68	25	4.0	0	1.54	24	2.78	1

RIDA* performs well when the distribution of lower-bound values can be modeled a priori (as in traveling-salesman and 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.

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

Acknowledgements

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

References

[1] R. E. Korf, "Depth-first iterative deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, pp. 97–109, 1985.

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

[3] 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), pp. 297–302, IJCAI, Inc, 1989.

[4] N. R. Vempaty, V. Kumar, and R. E. Korf, "Depth-first vs best-first search," *Proc. National Conf. on Artificial Intelligence*, (Anaheim, CA), AAAI, 1991.

[5] 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*, vol. 50, pp. 207–221, Elsevier Science Publishers, 1991.