# PARALLEL ITERATIVE REFINING A* SEARCH

*Guo-jie Li*
National Research Center
for Intelligent Computing Systems
P. O. Box 2704
Beijing, P. R. China

*Benjamin W. Wah*
Coordinated Science Laboratory
University of Illinois
1101 W. Springfield Avenue
Urbana, IL 61801
wah@aquinas.csl.uiuc.edu

## ABSTRACT

In this paper, we study a new search scheme called Iterative Refining A* (IRA*). The scheme solves a class of combinatorial optimization problems whose lower-bound values and heuristic solutions of partial problems are easy to compute. An example of problems in this class is the symmetric traveling-salesman problem for a set of fully-connected cities. The scheme consists of a sequence of guided depth-first searches, each searching for a more accurate solution than the previous one. It is an extension from Korf's IDA* search, except that it prunes by approximation rather than by thresholding. In IRA*, global upper bound (or incumbents), rather than lower-bound information, is passed between the successive depth-first searches. The time required by IRA* for finding optimal solutions to problems in this class is very close to that required by a best-first search, while the memory space required is the same as that of a depth-first search. We derive the performance bounds of parallel IRA* and found that it is more robust and has much better worst-case behavior than a parallel depth-first search.

**INDEX TERMS**: Approximation, combinatorial optimization, iterative-deepening A*, iterative-refining A* (IRA*), parallel processing, performance bounds.

## 1. INTRODUCTION

NP-hard combinatorial optimization problems (COPs) require an exponential amount of computation time and memory space in the worst case in solving them. It has been proved that the A* algorithm with an admissible lower-bound function expands the minimum number of nodes in finding the optimal solution or a solution with a guaranteed deviation from the optimal solution. *Our objective in this paper is to find an efficient and robust parallel search scheme for solving COPs so that the memory space required is bounded, and that the ratio of the total number of nodes expanded in the best sequential (or A*) algorithm to that of the proposed (parallel) search scheme is maximum.*

There are a number of recent results on parallel combinatorial searches [1, 3]. Most of them are focused on load balancing, variable sharing, and reducing communication overheads. A number of studies were devoted to finding the best speedup of a parallel algorithm as compared to a sequential algorithm using the *same* search strategy. For instance, near-linear or superlinear speedups have been observed for a parallel depth-first search as compared to a sequential depth-first search [1]. Similar behavior has been observed for parallel IDA* search as compared to a sequential one [5]. These issues are important in parallel processing; however, they do not indicate the actual speedup achieved in solving the given problem. *The speedup achieved by parallel processing must be compared against the best sequential algorithm for solving the given problem and not the sequential version of the parallel algorithm.* Using the number of nodes expanded as a measure of performance, the best sequential performance is achieved by the best-first strategy. In this paper we characterize the aggregate speedup achieved as a function of the speedup achieved by parallel processing (using the same search strategy) and the slowdown resulted due to the use of a more space-efficient algorithm (such as depth-first search) than the best-first search.

In general, combinatorial optimization problems can be roughly divided into two classes according to whether feasible solutions can be found easily. In the first class, finding feasible solutions may be as hard as finding optimal ones. Examples of problems in this class include integer programming, solving 15-puzzle problems, and finding vertex covers. These problems can be solved efficiently by IDA* [2]. In the second class, it is relatively easy to find feasible solutions and hence upper-bound values to partially evaluated problem instances. Examples of problems include the symmetric traveling-salesman problem on fully-connected cities and many unconstrained scheduling problems. In this paper, we address problems in the second class. Without loss of generality, we discuss our results with respect to minimization problems.

In developing an efficient parallel search strategy, it is necessary to decompose the search into tasks so that the processing resources are scheduled to search independent subtrees. It is important that memory and time limitations be observed, that anomalies in parallelism be addressed, and that the scheme be applicable to a wide variety of problems. The Iterative Refining A* (in short, IRA*) search proposed in this paper are developed

This research was supported in part by National Aeronautics and Space Administration under grant NCC 2-481 and in part by National Science Foundation under grant MIP 88-10584.

This research was undertaken when the G. J. Li was a visiting scholar in the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign.

International Conference on Parallel Processing, St. Charles, IL, August 13-15, 1991.

with these objectives in mind.

In general, parallelism of COPs can be exploited in two levels: intra-node and inter-node levels. Current techniques in parallel processing, such as pipelining and array processing, are suitable for intra-node parallelism. In contrast, inter-node parallelism can be broadly classified into four types.

(a) In *tree-splitting* or *tree-decomposition*, each processor has a local list for storing active nodes, each constituting a disjoint part of the search tree. When a processor finishes evaluating all nodes in the local list, it gets an unsearched node from another processor. The major issues here are in finding efficient load balancing and variable-sharing (or information communication) strategies and in coping with anomalies in parallelism.

(b) In *parallel window searches*, each processor expands the nodes whose lower bounds fall into a given window, namely, a lower-bound interval. This approach is suitable for bottom-up searches, such as game-tree search, but suffers from the high overhead of node distribution found in top-down searches.

(c) In *parallel iterative searches*, the tree searched is expanded in successive iterations and may overlap with each other. In the original IDA* algorithm [2], thresholds are defined so all nodes with values less than a given threshold are searched in an iteration. Trees in successive iterations are searched sequentially. These trees can be searched in parallel. Parallel iterative search has the benefit of obtaining a desirable approximate solution early in the process and in communicating these solutions to all trees searched in parallel.

(d) The last type of inter-node parallelism is based on exploiting multiple heuristics and has not been addressed seriously in the literature. In this form, multiple heuristics are run in parallel. This takes advantage of the large fluctuations in computation times of various heuristics in generating feasible solutions.

The parallel IRA* search scheme studied in this paper utilizes the four types of inter-node parallelism discussed above and addresses three important issues: memory-space limitation, detrimental anomalies in parallel search [4], and achieving a performance as close as possible to the theoretical minimum number of nodes expanded in a best-first search. It performs better than a pure depth-first search or a guided depth-first search, which only address the issue on memory limitation. We derive bounds on performance of parallel IRA* searches. These bounds show that the IRA* algorithms are more robust than pure depth-first searches with respect to parallel processing and approximate computations.

In this paper, we evaluate our results using the traveling-salesman problem. These problems are generated by calling the random number generator *rand()*, which randomly determines the location of cities on a 100-by-100 Cartesian plane. All cities are assumed fully connected, and distances between cities are symmetric; that is, the triangular inequality is satisfied. The lower-bound function used here is computed by the spanning-tree method, and the upper bounds by a hill-climbing heuristic. Since the method for computing the upper bound generates a feasible solution, the global minimum upper bound is the same as the incumbent.

## 2. A BASIC PRINCIPLE OF PARALLEL PROCESSING

In general, two criteria can be used to measure the efficiency of parallel solution of a COP.

Type I Minimize $T(P)$
Subject to $\alpha \geq \alpha_r$ and $M(P) \leq M_{max}$,

Type II: Maximize $\alpha$
Subject to $T(P) \leq T_{max}$ and $M(P) \leq M_{max}$,

where $T(P)$ and $M(P)$ are the computation time and memory space spent by the algorithm using P processors; $\alpha$ and $\alpha_r$ are, respectively, the achieved and desired accuracies of the solution; and $T_{max}$ and $M_{max}$ are the time and space constraints. In this paper, the criterion used is of Type I.

Let $T_{best}(1)$ be the computation time required by the best sequential algorithm for solving a given problem. Since $T_{best}$ is fixed, Type I criterion is equivalent to

$$\text{maximize} \quad \frac{T_{best}(1)}{T(P)} \qquad \text{subject } \alpha \geq \alpha_r \text{ and } M(P) \leq M_{max}$$

Let T(1) be the underlying sequential algorithm corresponding to the parallel algorithm to be measured. Then,

$$\frac{T_{best}(1)}{T(P)} = \frac{T_{best}(1)}{T(1)} \cdot \frac{T(1)}{T(P)} = R \cdot S \tag{2.1}$$

where $R = \frac{T_{best}(1)}{T(1)}$ and $S = \frac{T(1)}{T(P)}$. It is clear that $R$ characterizes the efficiency of the underlying sequential algorithm, and S characterizes the efficiency of the parallel implementation of the underlying sequential algorithm. A desirable objective of a parallel search is that it achieves the maximum of $R \cdot S$ rather than only $R$ or $S$ individually, and that it satisfies the constrains listed in Type I or II criterion.

A parallel best-first search may achieve the maximum $R \cdot S$. However, it violates the memory-space constraints unless the COP solved is very small. For the original IDA* algorithm, the average $S$ achieved for different problem instances may be close to $P$, although anomalous behavior occurs, as reported by Rao, Kumar, and Ramesh [5]. $R$ achieved, however, is small for solving the second class of COPs, such as TSPs. For the conventional depth-first search, $R$ achieved is quite small, and $S$ is unpredictable due to its anomalous behavior: acceleration anomalies may result in superlinear speedups, and detrimental anomalies are frequent. *The basic idea in this paper is, therefore, on obtaining both R and S as large as possible and on making S predictable under the restriction of memory space.*

## 3. EFFECTS OF INITIAL UPPER-BOUNDS

Most search algorithms including A* can be formulated as a version of a branch-and-bound algorithm (B&B). In the B&B computation for solving a minimization problem, the global minimum upper bound, or *incumbent*, is initially set to a given value $u_0$ (or infinity) and monotonically decreases until it achieves $f_o$, the optimal solution, which remains unchanged until the end of computation. In the following sections, the incumbent initially obtained before starting the computation of an iteration is called the *initial upper bound* of the iteration.

In general, a combinatorial search process can be split into two phases:

(a) computation to obtain an optimal solution

Table 1. Comparison of $T_f$ and $T_v$ of guided depth-first searches for 5 symmetric 20-city TSPs.

| | Initial Seeds | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $T_f$ | 54806 | 44143 | 181299 | 146335 | 106305 |
| $T_v$ | 12981 | 13483 | 42658 | 10602 | 29547 |
| $T_b$ | 16235 | 23491 | 77292 | 13867 | 48062 |

(b) computation to verify its optimality.

To analyze the performance independent of the details of the underlying computer architecture, we use the number of nodes expanded as the performance measure throughout this paper. Let $T_f$ and $T_v$ be the numbers of nodes expanded before and after the first optimal solution is found. Let $T_b$ be the number of nodes expanded in an A* search. Obviously, the total number of nodes expanded in any search algorithm, $T$, satisfies

$$T = T_f + T_v \geq T_b$$

In an A* algorithm, the best-first search strategy is adopted, so $T_f = T_b$ and $T_v = 0$. This implies that the initial upper bound has no impact on the A* algorithm. However, in depth-first searches, $T_f$ is usually much larger than $T_v$ and $T_b$. Table 1 shows $T_f$, $T_v$, and $T_b$ for 5 symmetric TSPs of 20-cities. These examples show that in most cases, $T_f$ is the dominant part of search time for depth-first searches. To reduce the computation time of depth-first searches, the most important concern is to make $T_f$ as small as possible.

For general heuristic searches, $T_f$ depends mainly on the accuracy of the guidance function, while $T_v$ depends uniquely on the accuracy of the lower-bound function. The only way to reduce $T_v$ is to design more accurate lower-bound functions, which is difficult since it is problem dependent. In a depth-first search, since no heuristic function is used to guide the selection of nodes, the upper-bound function has a strong influence on $T_f$, especially when approximations are applied.

So far, a lot of work has been carried out on improving the guidance and lower-bound functions, but the impact of the upper-bound function has not been studied seriously. In this paper, we do not attempt to compare different upper-bound functions, but rather study the influence of the initial upper bound on the performance of a search.

We have extensively studied the computational behavior of guided depth-first searches[1] for solving TSPs and knapsack problems. It is found that a decrease of the initial upper bound results in a significant decrease in the computation time in most cases. When an optimal solution is sought, a smaller initial upper bound will always have a positive effect, *i.e.*, reducing $T_f$ for the depth-first search. Due to space limitation, this claim is stated without proof in Theorem 1.

**Theorem 1.** Suppose that $A(u_1)$ and $A(u_2)$ are B&B algorithms using the different initial upper bounds $u_1$ and $u_2$ and that $u_1 > u_2$. Assume that dominance relation is consistent with the lower-bound function[2]. Let $T(u_1)$ and $T(u_2)$ be the computation time to find the optimal solution by $A(u_1)$ and $A(u_2)$, respectively. Then $T(u_1) \geq T(u_2)$.

Theorem 1 holds for any search strategies. However, a smaller initial upper bound does not always guarantee a reduction in computation time when approximate B&B algorithms are employed. Anomalies may happen, although they do not happen frequently.

## 4. ITERATIVE REFINING A* SEARCH

Having realized that the initial upper bound has a vital influence on $T_f$ in depth-first searches, we need to answer the next question: how to obtain a small initial upper bound with the minimum overhead. This can be found by an efficient heuristic algorithm, which finds a feasible solution to the problem. This approach is not taken in this paper as it is problem dependent. Our goal is to develop a general problem-independent scheme for finding initial upper bounds efficiently.

Note that the initial upper bound cannot be assigned arbitrarily. If the assigned initial upper bound is less than $f_o$, then it is not a true upper bound but a threshold. Thresholds have been used in IDA* search [2] and can be selected optimally to minimize the total search overhead [7]. However, the criterion for their selection is not based on minimizing the overhead for finding an initial upper bound.

An approximate B&B algorithm can be used to find suboptimal solutions of COPs. During an approximate B&B procedure, a node $P_i$ generated will be terminated if

$$g(P_i) \geq \frac{u(P_i)}{1+\varepsilon} \qquad \varepsilon \geq 0, \tag{4.1}$$

where $u(P_i)$ is a global upper bound for $P_i$ and $g(P_i)$ is its lower bound. $u^f$, the final suboptimal solution obtained by the approximate B&B algorithm, deviates from $f_o$, the optimal solution value, by

$$\frac{u^f}{1+\varepsilon} \leq f_o \leq u^f \qquad \varepsilon \geq 0. \tag{4.2}$$

Usually, $\varepsilon_f$ (= $(u^f - f_o)/f_o$), the final error achieved, is much smaller than the allowed error $\varepsilon$. Since $f_o$ is unknown in advance, $g^f$, the global minimum lower bound, instead of $f_o$, is often used to get $\varepsilon_g$, the upper bound of the error of the suboptimal solution obtained. Note that $\varepsilon_g = (u^f - g^f)/g^f$. If $\alpha$, the worst-case accuracy of the solution, is defined as $f_o/u^f$, then $\alpha = 1/(1+\varepsilon)$.

Approximations significantly reduce the amount of time needed to arrive at suboptimal solutions. Let $A(\varepsilon_1)$ and $A(\varepsilon_2)$ be B&B algorithms with $\varepsilon_1$ and $\varepsilon_2$, respectively, and $\varepsilon_1 < \varepsilon_2$. If a suboptimal solution $u^f$ is found by both algorithms, then in most cases, $A(\varepsilon_2)$ finds $u^f$ earlier than $A(\varepsilon_1)$ does, though anomalous behavior may appear when a depth-first search is employed.

The above claim has been verified experimentally and is demonstrated in Figure 1. A symmetric TSP is solved by a

1. In a guided depth-first search, the children of a node are searched in monotonic order of lower bounds.

2. A dominance relation D is said to be consistent with the lower-bound function if $P_i \, D \, P_j$ implies that $g(P_i) \leq g(P_j)$, where $g(P_i)$ and $g(P_j)$ are nodes in the B&B search tree. More detailed discussion on dominance relations can be found in the reference [4]

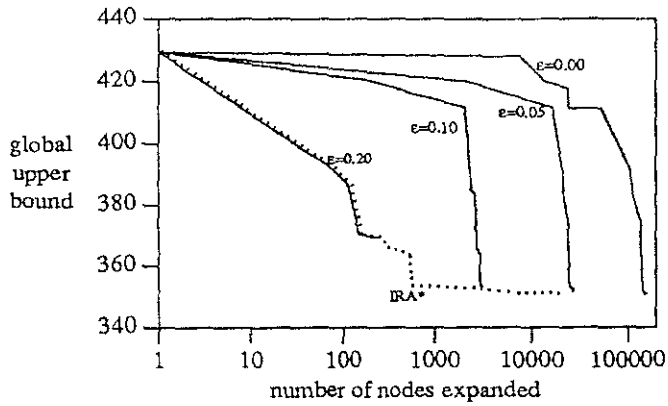


Figure 1. The effect of error allowance on global upper-bound for guided depth-first searches. (TSP of 20-cities, initial seed = 4)

guided depth-first search with four different error allowances, 0.2, 0.1, 0.05 and 0. The computational behaviors are measured in terms of the rates of decrease of the global upper bounds. Figure 1 shows that the larger the error allowance is, the faster the global upper bound decreases. For instance, when $\varepsilon = 0.2$, 240 nodes are expanded to reach the global upper-bound value 369.2, while 138,000 nodes are expanded in order to reach the same global upper-bound value when $\varepsilon = 0$.

Comparing the four curves and considering the effect of the initial upper bounds, we can arrange the depth-first searches in such a way that the approximation search with a large $\varepsilon$ is implemented first; the suboptimal solution obtained is then transferred to the next iteration as the initial upper bound, in which a smaller $\varepsilon$ is used. As a result, the time for solving a COP can be reduced significantly!

For the above example, the same problem can be solved in four iterations with the error allowances decreasing from 0.2 to 0. It takes 241 node expansions to finish the first iteration; the second iteration with $\varepsilon = 0.1$ and initial incumbent 369.2 takes 491 node expansions; the third iteration with $\varepsilon = 0.05$ and initial incumbent 353.3 takes 2,419 node expansions; finally, the fourth iteration with initial incumbent 351.8 takes 14,229 node expansions to find the optimal solution 351.0 and complete the search. The computational behavior of this implementation is reflected by the dotted curve in Figure 1, which shows an order-of-magnitude improvement in performance (see Case 4 in Tables 2 and 3). Since the suboptimal solution is refined iteratively, we call this scheme an *Iterative Refining A** (in short, IRA*) search.

The IRA* search scheme is outlined in Figure 2. The stopping criterion can be the number of iterations, the final error allowance, or other conditions. An upper-bound function can be used to compute $u_1^0$, the initial incumbent of the root. The key of the IRA* search scheme is in *refining*, *i.e.*, choosing the proper strategy to reduce the error allowance. The goal of refining the computation except the last iteration is to find an optimal or desired suboptimal solution as early as possible. Once the desired solution is found, the remaining time is only used to verify optimality.

```
INPUT:  initial error allowance ε_1;
        initial incumbent u_1^0 if available;
        problem parameters;
OUTPUT: Optimal or suboptimal solution;
BEGIN
  WHILE (stopping-criterion is not satisfied) {
    DO approximate depth-first search from the root of the
       tree with ε_i and initial incumbent u_i^0
    UNTIL no active node remains in the stack or the time
       constraint is exceeded;
    IF (time constrain is exceeded) THEN
       report the achieved solution and the error bound ε_g;
    ELSE {
       (a) record the suboptimal solution and ε_g;
       (b) set new error allowance ε_{i+1} < ε_i;
       (c) set u_{i+1}^0, the initial incumbent of the next iteration,
           equal to u_i^f, the final incumbent in this iteration;
    }
  }
END
```

Figure 2. The IRA* search scheme

Table 2. Performance comparison of various search algorithms for 5 20-cities TSPs. ($T_g$: time for guided depth-first search; $T_r$: time for IRA* search; and $T_b$: time for best-first search)

| | Initial Seeds | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $T_g$ | 67787 | 57626 | 223957 | 156937 | 135852 |
| $T_r$ | 20828 | 51043 | 88864 | 17380 | 54889 |
| $T_b$ | 16235 | 23491 | 77292 | 13867 | 48062 |
| $T_r/T_b$ | 1.28 | 2.17 | 1.15 | 1.25 | 1.14 |

Table 3. Time of each iteration in IRA* for 5 20-cities TSPs. $T(\varepsilon)$ is the time needed in an iteration of IRA* to achieved an approximation degree $\varepsilon$.

| Approx. Searches | Initial Seeds | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| T(0.2) | 769 | 210 | 407 | 241 | 303 |
| T(0.1) | 1071 | 2291 | 2329 | 491 | 1279 |
| T(0.05) | 1886 | 9055 | 8836 | 2419 | 5245 |
| T(0) | 17102 | 39487 | 77292 | 14229 | 48062 |
| T(0)/T(0.05) | 9.17 | 4.36 | 8.75 | 5.89 | 9.16 |

The sequence of refinements must be chosen properly. If the difference in error allowances between successive iterations is too large, then the final upper bound found in each iteration may have little influence on the next iteration. In contrast, if the difference is too small, then the global upper bound may remain unchanged in some iterations; hence, these iterations do not contribute to improving the solution. From our experience, four iterations are sufficient for the problems we have tested, and either the optimal or a good suboptimal solution is found in the iteration with $\varepsilon = 0.05$.

Table 2 compares the performance among a guided depth-first search, IRA*, and a best-first search[3] for 5 20-city symmetric TSPs.[4] Due to space limitations, other results which exhibit similar behavior are not shown. The results show that the performance of IRA* is very nice. In all instances except when the initial seed is 2, the search times required by IRA* are only 14 to 28 percent more than that for the best-first search, and the space required is linear. To our knowledge, no other general search scheme has achieved this high performance.

We now explain the reason why IRA* is efficient and analyze its performance. Note that TSP is a strict NP-hard problem, whose computation time grows exponentially when the accuracy of the solution is increased. Table 3 demonstrates this claim. The time required in an approximation search with $\varepsilon = 0.05$ is only about 20 percent of $T_b$. On the other hand, since the lower-bound function for TSP based on the spanning-tree method is not very tight, it is quite possible that a node provides a new incumbent but its lower bound is relatively loose. Thus, an optimal or a good suboptimal solution is usually found before the last iteration, and the overhand of IRA* is quite small.

The computational behavior of the approximate depth-first searches are related to the sequences of incumbents, which are problem-instance dependent. Therefore, a theoretical analysis of the performance of IRA* is very difficult. The following theorem shows the minimum time required by IRA*.

Throughout this paper, all analysis are based on the assumption that *the number of nodes whose lower bound is less than g increases exponentially with g*. Empirically, we have verified this assumption for strict NP-hard COPs, such as symmetric TSPs for fully connected cities [6]. This assumption holds true in a large range of lower-bound values, from the minimum lower-bound value to some value larger than the optimal solution. Let $n_0$ and $B_0$ be, respectively, the numbers of nodes whose lower bounds are minimum and are less than the optimal-solution value. $n_0$ is one for most COPs, *i.e.*, only the root of the B&B search tree has the minimum lower bound.

**Lemma 1.** Assume that n(g), the number of nodes whose lower bounds are less than g in the B&B tree, increases exponentially with g within the range $g_r < g < L$, and that $n(g_r) = n_0$, then

$$n(g) = n_0 \left(\frac{B_0}{n_0}\right)^{\left(\frac{g-g_r}{f_o-g_r}\right)}, \tag{4.3}$$

where $f_o$ and $g_r$ are the optimal-solution and minimum lower-bound values, respectively, and $L > f_o$.

*Proof*: According to the assumption, we can denote n(g) as an exponential function.

$$n(g) = M \cdot A^g, \tag{4.4}$$

where M and A are constants. From the following conditions,

$$M \cdot A^{g_r} = n_0 \tag{4.5}$$

$$M \cdot A^{f_o} = B_0, \tag{4.6}$$

we get $A = \left(\frac{B_0}{n_0}\right)^{\left(\frac{1}{f_o-g_r}\right)}$ and $M = n_0 \left(\frac{B_0}{n_0}\right)^{\left(\frac{-g_r}{f_o-g_r}\right)}$ □

In terms of the above result, we can analyze the best-case behavior of IRA*.

**Theorem 2.** Let $\varepsilon_i$ be a series of error allowances, $\varepsilon_i > \varepsilon_{i+1}$, $1 \le i < k$, and $\varepsilon_k \ge 0$. Let $T_r$ be the computation time required by IRA* for solving a COP and $n_0 = 1$. Then $T_r \ge \sum_{i=1}^{k} B_0^{r_i}$, where $r_i$ satisfies $\frac{f_o}{1+\varepsilon_i} - g_r = r_i(f_o - g_r)$.

*Proof.* Consider iteration $i$ in which the error allowance is $\varepsilon_i$. From Eq. (4.1), all nodes whose lower bound is less than $\frac{f_o}{1+\varepsilon_i}$ have to be expanded; thus, at least $n\left(\frac{f_o}{1+\varepsilon_i}\right)$ nodes are expanded in iteration $i$. From Lemma 1,

$$n\left(\frac{f_o}{1+\varepsilon_i}\right) = B_0^{\left(\frac{\frac{f_o}{1+\varepsilon_i}-g_r}{f_o-g_r}\right)} \tag{4.7}$$

Note that, in the above equation, $n(\cdot)$ is a function notation. Let $\frac{f_o}{1+\varepsilon_i} - g_r = r_i(f_o - g_r)$. Eq. (4.7) shows that at least $B_0^{r_i}$ nodes have to be expanded in iteration $i$. The theorem is proved by considering $k$ iterations. □

Only a loose upper bound of $T_r$ can be derived due to its anomalous behavior. However, our results in Table 3 demonstrate that most of the overhead of IRA* is incurred in the last iteration. If the optimal solution is found before this, then $T_r$ is usually close to $T_b$, since all nodes expanded in the last iteration have to be expanded in a best-first search as well. In general, this is not the worst case. We now discuss the worst-case behavior of IRA*.

**Theorem 3.** Let $k$ be the number of iterations of an IRA*, $\varepsilon_k$ be 0, and the optimal solution be found in the last iteration. Let $t_j$ be the computation time required in the $j$-th iteration, and $n_0$ be 1. Then $t_k < B_0^{r_{k+1}}$, and $t_{k-1} < B_0^{r_{k-1}+1}$, where $r_k = \frac{\varepsilon_{k-1} f_o}{f_o - g_r}$ and $r_{k-1} = \frac{(\varepsilon_{k-2}-\varepsilon_{k-1})\cdot f_o}{(1+\varepsilon_{k-1})(f_o-g_r)}$.

*Proof.* Since the optimal solution is found in the last iteration and using the approximate lower-bound test defined in Eq. (4.1), we can claim that $u_{k-1}^0 = u_{k-2}^f < (1+\varepsilon_{k-2}) f_o$; otherwise, the optimal solution must be found before the $(k-1)$-th iteration. Similarly, $u_{k-1}^f < (1+\varepsilon_{k-1})\cdot f_o$, and $g_{k-1}^f < u_{k-1}^0/(1+\varepsilon)$. The worst case happens when the incumbent is unchanged until the last node is expanded in the $(k-1)$-th iteration. This implies that

$$t_{k-1} < n\left(\left(\frac{1+\varepsilon_{k-2}}{1+\varepsilon_{k-1}}\right)\cdot f_o\right)$$

From Eq. 4.4, we have

$$t_{k-1} < M \cdot A^{\left(\frac{(1+\varepsilon_{k-2})\cdot f_o}{1+\varepsilon_{k-1}}\right)}$$

3. $T_b$ is obtained in the last iteration of IRA*, since $u_k^0 = f_o$. It is very difficult to solve a large symmetric TSP by a best-first search due to the space constraint.

4. For asymmetric TSPs and TSPs on incompletely connected graphs, problems with several hundred cities can be solved. The solvable size of TSPs on symmetric and fully connected graphs is quite small. Even for 20 cities, the number of nodes expanded in the search tree may exceeds one million.

Hence,

$$t_{k-1} < B_0^{\left\lceil 1+\frac{(\varepsilon_{k-2}-\varepsilon_{k-1})\cdot f_o}{(1+\varepsilon_{k-1})(f_o-g_r)}\right\rceil} \quad (4.8)$$

Applying the above analysis to the $k$-th iteration, we get

$$t_k < B_0^{\left\lceil 1+\frac{\varepsilon_{k-1} f_o}{f_o-g_r}\right\rceil} \quad (4.9)$$

The theorem follows from Eq's (4.8) and (4.9) and the definition of $r_i$, $i = k-1, k$. □

Theorems 2 and 3 are of importance only in verifying the performance of IRA* and not in predicting it, as $B_0$ is problem-instance dependent and is difficult to estimate. Some parameters in the above theorems can be estimated. For example, $g_r$ is easy to obtain by abstracting the information of the root node. From our experience, $f_o$ is close to the mean of the root's lower-bound and upper-bound values. More accurate estimate of $f_o$ can be obtained by using statistical extreme-value distribution theorem. According to this theorem, TSP can be viewed as $(n-1)!/2$ samples and each upper bound as a local minimum, and the optimal solution satisfies the Weibull distribution.

To verify this upper bound, we examine a 20-city TSP with $u_k^0 = 398.792$, $g_r = 303.165$, $f_o = 395.792$, $T_b = 552101$, and $t_k = 756385$. Since $B_0$ is not available, we use $T_b$ instead of $B_0$ in order to derive a loose upper bound of $t_k$. In this example, $u_{k-1}^f = u_k^0 = 398.792$; hence, the achieved $\varepsilon_{k-1}$ is $(u_k^0 - f_o)/f_o = 3/f_o$, and $r_k$ is 0.033. From Theorem 3, we get $t_k = 756385 < T_b^{1+rk} = 854093$.

The sequence of ε used in IRA* is called a *refinement schedule*. A good refinement schedule should be determined in terms of the distribution of nodes by lower bounds and the accuracy of the initial incumbent. The design of an optimal refinement schedule is open at this time. In this paper, a static refinement schedule is adopted. We have examined a number of dynamic refinement schedules and found that the search performance of IRA* for solving TSPs is not sensitive to changes in the refinement schedule, since the computation time of the last iteration is dominant and is close to $T_b$ in most cases.

## 5. PARALLEL ITERATIVE REFINING A*

In studying parallel IRA*, we focus on improving its speedup with respect to the *best* sequential algorithm. To simplify the analysis, we assume that the overheads of load balancing and communication are negligible. We assume that the computational model is a loosely coupled multicomputer, and that the B&B tree is split into a number of subtrees that are searched in parallel. We found earlier that when multiple lists are used, a parallel search may risk searching a large number of extra nodes [4]. Our objective here is to reduce this risk. We further assume that the parallel computation is synchronized by iterations. Note that the search trees in different iterations have different sizes.

Before presenting the theoretical analysis, we need to explain the representation of the search tree. The search trees used in the literature are usually depicted by depth. For the sake of understanding IRA*, the search trees used here are depicted by cost (see Figure 3). Note that nodes in the same cost level do not mean that they are located in the same depth of the original tree.

In iteration $i$, the global upper bound decreases from $u_i^0$ to $u_i^f$ (it is possible that $u_i^f = u_i^0$). When a parallel IRA* is implemented, the speedup is related to the interval $\frac{u_i^0 - u_i^f}{1+\varepsilon_i}$. Similar to the discussion in Section 3, we define $r_i$ as $\frac{u_i^0 - u_i^f}{(1+\varepsilon_i)(f_o - g_r)}$. Since $(u_i^0 - u_i^f) < (f_o - g_r)$, we have $0 \le r_i \le r_{\max} \le 1$. $r_{\max}$ reflects the maximum threshold interval among all iterations. Define $C_i = \frac{n\left\lceil \frac{u_i^0}{1+\varepsilon_i}\right\rceil}{n\left\lceil \frac{u_i^f}{1+\varepsilon_i}\right\rceil}$. From Lemma 1, we have $C_i = A^{r_i(f_o-g_r)}$. Let $C_{\max} = \max_i(C_i)$, namely, $C_{\max} = A^{r_{\max}(f_o-g_r)}$. $C_{\max}$ reflects the ratio between the numbers of nodes expanded with different thresholds and is an important parameter for characterizing the behavior of IRA* searches. In a conventional depth-first search, $r_{\max} = 1$; hence, $C_{\max} = B_0$ if $n_0 = 1$.

In the following discussion, notations (1) and (P) are used to denote the parameters used in serial and in parallel computing.

**Theorem 4.** Assume that the search tree is decomposed into multiple subtrees, that the processors are synchronized in node expansions, that $u_i^0(P) = u_i^0(1)$, and that $u_i^f(P) = u_i^f(1)$. Let $T_r(1)$ and $T_r(P)$ be the computation times required for the IRA* algorithms using one and P processors, respectively. The speedup, $\frac{T_r(1)}{T_r(P)}$, are limited within the following bounds.

$$\max\left(1, \frac{P}{C_{\max}}\right) \le \frac{T_r(1)}{T_r(P)} \le C_{\max} \cdot P \quad (5.1)$$

*Proof.* The proof is in two parts. Due to space limitation, we show only the part for finding the lower-bound speedup; the part for finding the upper-bound speedup is similar.

In finding the lower-bound speedup, since the parallel IRA* algorithm is synchronized by iterations, $T_r(P) = \sum_{i=1}^{k} t_i(P)$. For the $i$-th iteration in both serial and parallel cases, the nodes whose lower bounds are less than $\frac{u_i^f}{1+\varepsilon_i}$ must be expanded. These nodes are called *essential nodes*, while other nodes expanded are called *nonessential nodes*. In the worst case, no nonessential node is expanded in the serial case but all nonessential nodes are expanded in the parallel case. This situation is possible when $u_i^f$, the final incumbent of iteration $i$, is not found until $\frac{1}{P} \cdot n\left\lceil \frac{u_i^0}{1+\varepsilon_i}\right\rceil$ essential nodes have been expanded in the serial case, and all these nodes are assigned to processor 1 in the parallel case if $P > C_i$. In addition to essential nodes, all nonessential nodes are expanded in the parallel case, and these nonessential nodes are assigned to processor 2 to processor P. Figure 3 shows the worst case for parallel IRA* in which $P > C_i$ and

$$t_i(1) = n\left\lceil \frac{u_i^f}{1+\varepsilon_i}\right\rceil, \quad t_i(P) = \frac{1}{P} \cdot n\left\lceil \frac{u_i^0}{1+\varepsilon_i}\right\rceil$$

Without loss of generality, the initial steps for generating $P$ nodes in the parallel IRA* search and the ceiling function is

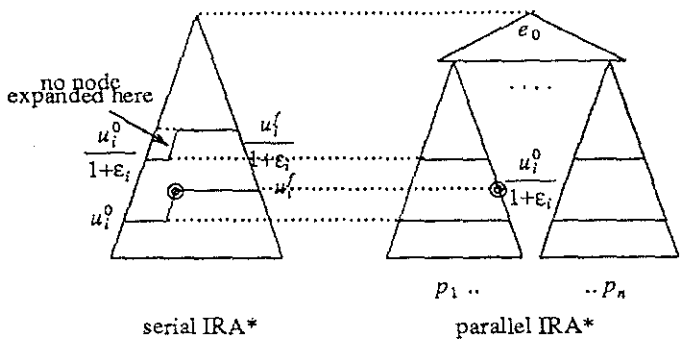


Figure 3. The worst case behavior of parallel IRA*.

omitted in this proof. This means that the overhead of synchronization is ignored. To derive the upper bound of minimum speedup, *i.e.*, the maximum speedup in the worst case, we neglect the overhead of load balancing. Accordingly,

$$\frac{t_i(1)}{t_i(P)} = \frac{n\left\lceil \frac{u_i^f}{1+\varepsilon_i} \right\rceil}{\left\lceil \frac{1}{P} \right\rceil \cdot \left\lceil n\left\lceil \frac{u_i^0}{1+\varepsilon_i} \right\rceil \right\rceil} \quad \text{if } P > C_i \tag{5.2}$$

From Eq.(5.2) and the definition of $C_i$, we have

$$\frac{P}{C_i} \le \frac{t_i(1)}{t_i(P)} \quad \text{if } P > C_i. \tag{5.3}$$

When $P \le C_i$, it is possible that in the parallel case processor 1 expands the same nodes as expanded in the serial case, and all nodes expanded by other processors are unnecessary. This implies that

$$1 \le \frac{t_i(1)}{t_i(P)} \quad \text{if } P \le C_i. \tag{5.4}$$

The lower bound is proved by combining Eq's (5.3) and (5.4), and using the definition of $C_{\max}$. □

Note that the speedup is dominated by the last iteration in which the threshold interval is restricted within $\varepsilon_{k-1} f_o$, and oftentimes $u_k^0 = u_k^f$. This implies that $C_k$ is very small and is very likely equal to 1. Consequently, in practice, the bounds of speedup are tighter than those given in Theorem 4, especially when load balancing is applied.

The above theorem shows the robustness of parallel IRA*. Since $C_{\max}$ is small in IRA*, the condition that $P > C_{\max}$ is generally satisfied. Parallel IRA* also avoids the detrimental anomalies and poor slowdown often found in parallel depth-first searches; that is, at least $\frac{P}{C_{\max}}$ times speedup can be gained. The cost of this gain is in the loss of large super-linear speedups, although super-linear speedup less than $C_{\max} P$ is still achievable. In contrast, the possible speedups in conventional parallel depth-first search may range from 1 to $B_0 \cdot P$.

Table 4. Simulation results on parallel IRA* and PGDFS for solving 5 20-city TSPs using 16 processors.

| Init. Seed | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Sequential Best-First Search | | | | | |
| $T_b$ | 16235 | 23491 | 77292 | 13867 | 48062 |
| IRA* and Parallel IRA* | | | | | |
| $T_r$ | 20828 | 51043 | 88864 | 17380 | 54889 |
| $T_r^1$ | 26879 | 48404 | 87739 | 177783 | 55025 |
| $T_r(16)$ | 1697 | 3043 | 5501 | 1127 | 3457 |
| $R_r$ | 0.78 | 0.46 | 0.87 | 0.80 | 0.88 |
| $H_r$ | 0.77 | 1.05 | 1.01 | 0.98 | 1.00 |
| $U_r$ | 15.84 | 15.91 | 15.95 | 15.78 | 15.92 |
| $S_r$ | 12.27 | 16.77 | 16.15 | 15.42 | 15.88 |
| $SP_r$ | 9.57 | 7.72 | 14.05 | 12.30 | 13.90 |
| GDFS and Parallel GDFS | | | | | |
| $T_g$ | 67787 | 57626 | 223957 | 156937 | 135852 |
| $T_g^1$ | 122035 | 49340 | 219463 | 42032 | 85028 |
| $T_g(16)$ | 7643 | 3099 | 13732 | 2643 | 5340 |
| $R_g$ | 0.24 | 0.41 | 0.35 | 0.09 | 0.35 |
| $H_g$ | 0.56 | 1.17 | 1.02 | 3.73 | 1.60 |
| $U_g$ | 15.97 | 15.92 | 15.98 | 15.90 | 15.92 |
| $S_g$ | 8.87 | 18.60 | 16.31 | 59.38 | 25.44 |
| $SP_g$ | 2.12 | 7.58 | 5.63 | 5.25 | 9.00 |

Table 5. Summary of various speedups for 10 TSP instances using 16 processors and random seeds ranging from 1 to 10. ($T_b$: time for best-first search; $T_g$: time for guided depth-first search; $T_r$: time for IRA* search.)

| average speedup | $\frac{T_b(1)}{T_r(16)}$ | $\frac{T_b(1)}{T_g(16)}$ | $\frac{T_g(16)}{T_r(16)}$ | $\frac{T_g(1)}{T_g(16)}$ | $\frac{T_r(1)}{T_r(16)}$ | $\frac{T_g(1)}{T_r(16)}$ |
|---|---|---|---|---|---|---|
| Method 1 | 11.42 | 5.06 | 2.26 | 14.34 | 14.69 | 32.38 |
| Method 2 | 11.37 | 5.67 | 3.37 | 20.26 | 14.96 | 64.45 |

Method 1 computes the speedup as the ratio of the total computation times of the 10 problem instances using different search strategies. Method 2 computes the speedup as the average speedup of individual problem instances.

## 6. SIMULATION RESULTS OF PARALLEL IRA*

To verify the theoretical results derived above, we have simulated the performance of parallel IRA* (PIRA*) and parallel guided depth-first searches (PGDFS) using 16 processors. Results for 5 symmetric TSPs of 20 fully-connected cities are shown here. Other results are not shown due to space limitation.

The parallel search is implemented as follows. Initially, a serial best-first search is carried out to produce 16 active nodes, which are distributed into 16 separated lists. A parallel search is then carried out with the 16 nodes as the roots of subtrees. In each step, a processor expands one node, provided there are 16 or more active nodes among the lists. If a list is empty, it gets a node from another list. Since we are focusing on the speedup ($SP$) of the parallel search, we use the number of steps as our performance measure. The refinement schedule adopted in the parallel IRA* search is the same as that in the serial one.

The simulation results for PIRA* and PGDFS are shown in Table 4. The sequential search times, $T_b$, $T_r$, and $T_g$, are taken from Table 2. $T_g(P)$ and $T_r(P)$ are the times required by PGDFS and PIRA*, respectively. Since the sets of the nodes expanded in a serial search and the corresponding parallel search are not identical, we denote the total number of nodes expanded in all processors in PGDFS (resp. PIRA*) by $T_g^1$ (resp. $T_r^1$).

Let $H$ be the ratio of the number of the nodes expanded in a serial search and that in the corresponding parallel search. $H$ reflects the amount of extra work or savings in a parallel search as compared to the serial one. During a parallel search, especially in the first and last few steps, some processors may be idle as there are insufficient active nodes. The parameter $U$ characterizes the speedup of $T_r^1$ (resp. $T_g^1$) over $T_r(P)$ (resp. $T_g(P)$). The simulation results show that in almost every step all processors are busy. That is, $U$ is very close to 16. Using the above definitions and recalling R and $S$ defined in Section 2, we have the following relations among these parameters.

$$SP = R \times S = R \times (H \times U) \tag{6.1}$$

Tables 4 demonstrates that PIRA* is much more efficient and robust than PGDFS. For the 5 TSP instances, the speedups ($SP$) range from 7.72 to 14.05 in PIRA*, with an average speedup[5] of 12.03. For PGDFS, the speedups range from 2.12 to 9.00, with an average speedup of 5.51. In PIRA*, the number of nodes expanded is close to that of the serial IRA*. As a result, poor slowdown in performance is avoided, and $S$, the speedups with respect to the same search scheme, are close to the number of processors (ranging from 12.27 to 16.77). Obviously, all these speedups obtained by simulations are within the bounds derived in Section 5. In PGDFS, the values of parameter $H$ is more diverse than those in PIRA*; hence, anomalies occur more frequently ($S$ ranges from 8.87 to 59.38). Even in the case when $S$ equals 59.38, the real speedup ($SP$) is only 5.25, since $R$ is too small.

Table 5 compares different methods for computing speedups. We focus in this paper on $T_b(1)/T_r(P)$, namely, the speedup of the parallel algorithm as compared to the best sequential algorithm. This is shown in column 2 in Table 5. We see that PIRA* has a speedup of around 11 as compared to a sequential best-first search.

For research on parallel algorithms, what is important is the speedup based on the same search strategy. This is illustrated by the results in columns 5 and 6 in Table 5, which show that the speedups are closer to linear and can even be superlinear. This linear or superlinear speedup has been observed frequently by researchers in this area. However, these speedups do not represent the merits of parallel processing in actually solving the original problem.

The speedups achieved by PIRA* as compared to traditional sequential methods based on guided depth-first search are more dramatic. These are shown in the last column in Table 5, which show that a speedup of over 32 can be achieved.

5. The average speedup is computed by dividing the summation of $T_b$'s by the summation of $T_r(16)$'s (or $T_g(16)$'s) for the 5 TSPs.

## 7. CAVEATS

The effectiveness of the IRA* algorithm depends on the complexity of the COP to be solved. For problems with fully polynomial-time approximation algorithms, such as the knapsack problem, the IRA* algorithm may not be effective. A necessary condition for IRA* to be effective is that the computation time of the problem instance to be solved must be reduced substantially by approximate depth-first searches. Second, since IRA* is based on depth-first searches, it acquires the drawbacks and anomalous behavior of approximate depth-first searches. Overall, anomalies happen much less frequently in IRA* than in depth-first searches because we are performing a sequence of depth-first searches, each of which may compensate the anomalous behavior of another.

## ACKNOWLEDGEMENTS

We would like to thank Mr. Lon-chan Chu for his helpful discussions and for providing WISE, an integrated Search Environment which implements a B&B search of the traveling-salesman problem. We are also indebted to Dr. Akiko Aizawa for her technical assistance.

## REFERENCES

[1] V. Kumar, et al. (ed.), in *Parallel Algorithm for Machine Intelligence and Vision*, Springer-Verlag, New York, 1990.

[2] R. E. Korf, "Depth-First Iterative Deepening: An Optimal Admissible Tree Search," *Artificial Intelligence*, vol. 27, pp. 97-109, North-Holland, 1985.

[3] V. Kumar, K. Ramesh, and V. N. Rao, "Parallel Best-First Search of State-Space Graphs: A Summary of Result," *Proc. National Conf. Artificial Intelligence*, pp. 122-127, AAAI, 1988.

[4] G. J. Li and B. W. Wah, "Computational Efficiency of Combinatorial OR-Tree Searches," *Trans. on Software Engineering*, vol. 16, no. 1, pp. 13-31, IEEE, Jan. 1990.

[5] V. N. Rao, V. Kumar, and K. Ramesh, "A Parallel Implementation of Iterative-Deepening-A*," *Proc. of National Conf. Artificial Intelligence*, pp. 878-882, AAAI, 1987.

[6] B. W. Wah and L.-C. Chu, "TCA*--A Time-Constrained Approximate A* Search Algorithm," *Proc. Int'l Workshop on Tools for Artificial Intelligence*, pp. 314-320, IEEE, Nov. 1990.

[7] B. W. Wah, *MIDA*: An IDA* Search with Dynamic Control*, Research Report CRHC-91-09, Center for Reliable and High Performance Computing, Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL 61801, March 1991.