SUPERLINEAR SPEEDUP IN PARALLEL
STATE-SPACE SEARCH

V. NAGESHWARA RAO AND VIPIN KUMAR*

AUGUST 1988       AI88-80

* This work was supported by ARO grant #DAAG29-84-K-0060 to the Artificial Intelligence Laboratory, and ONR grant #N00014-86-K-0063 to the Computer Sciences Department at the University of Texas at Austin.
Superlinear Speedup in Parallel State-Space Search

V. Nageshwara Rao and Vipin Kumar*

Department of Computer Sciences,
University of Texas at Austin,
Austin, Texas 78712

May 1988

Abstract

When \( N \) processors perform depth-first search on disjoint parts of a state space tree to find a solution, then the speedup can be superlinear (i.e., \( > N \)) or sublinear (i.e., \( < N \)) depending upon when a solution is first encountered in the space by one of the processors. It may appear that on the average the speedup would be either linear or sublinear. Using an analytical model, we show that if the search space has more than one solution and if these solutions are randomly distributed in a relatively small region of the search space, then the average speedup in parallel depth-first search can be superlinear. If all the solutions (one or more) are uniformly distributed over the whole search space, then the average speedup is linear. This model is validated by our experiments on synthetic state-space trees and the 15-puzzle problem. The same model predicts average superlinear speedup in parallel best-first branch-and-bound algorithms on suitable problems.

*This work was supported by Army Research Office grant # DAAG29-84-K-0060 to the Artificial Intelligence Laboratory, and Office of Naval Research Grant N00014-86-K-0763 to the computer science department at the University of Texas at Austin.
1 Introduction

Consider the problem of finding a solution in a state-space tree containing one or more solutions[3,22,21]. Depth-first search (DFS) is a widely used technique for solving such problems[7,22]. A number of parallel formulations of depth-first search have been developed by various researchers[18,10,6,1,17,11]. In one such formulation, N processors concurrently perform depth-first search in disjoint parts of a state-space tree to find a solution in the search space. The parts of the search spaces searched by different processors are determined dynamically, and are roughly of equal sizes. Since only one solution is needed, the search terminates whenever any of the processors encounters a solution. Depending upon when a solution is first encountered in the space by the processors, the speedup can be superlinear (i.e., \( > N \)) or sublinear (i.e., \( < N \)). This phenomenon of speedup greater than N on N processors in isolated executions of parallel depth-first search has been reported by many researchers [6,18,17,1,25]. The speedup can differ greatly from one execution to another, as the actual parts of the search space searched by different processors are determined dynamically, and can be different for different executions. Hence for some execution sequences, the parallel version may find a solution by visiting fewer nodes than the sequential version (giving superlinear speedup), and for others it may find a solution only after visiting more nodes (giving sublinear speedup). It may appear that on the average the speedup would be either linear or sublinear. Using an analytical model, we show that if the search space has more than one solution and if these solutions are uniformly distributed in a relatively small (randomly located) region of the search space, then the average speedup in parallel depth-first search can be superlinear. If all of the solutions (one or more) are uniformly distributed over the whole search space, then the average speedup is linear. This model is validated by our experiments on synthetic state-space trees and the 15-puzzle problem[21]. For some instances of 15-puzzle with this property, the average speedup was found to be as much as 17 for 9 processors. Our model also predicts average superlinear speedup in parallel best-first branch-and-bound algorithms on suitable problems.

Section 2 gives a brief review of parallel depth-first search. Section 3 presents assumptions and definitions used in the analysis, and introduces a model for the analysis of speedup in parallel depth-first search. Experimental results on synthetic trees and 15-puzzle instances are presented in Section 4. Section 5 discusses the relevance of the model to best-first branch-and-bound. Section 6 discusses related research. Conclusions follow in Section 7.
2 Review of Parallel Depth-First search

There are many different parallel formulations of depth-first search[1,8,18,2,13]. The formulation discussed here is the most commonly used formulation[17,18,11]. In this formulation, the search space is split into many disjoint parts and each part is assigned to one of the processors. More than one part may be assigned to one processor. The splitting of the search space may be done statically before hand [13] or may be performed dynamically [18,17,1]. In dynamic splitting whenever a processor completes search on the part assigned to it, it requests a busy processor for work. The busy processor splits its work into two pieces and gives a piece of its search space to the requesting processor. When a solution is found by any processor, all of them quit. If the search space is finite and has no solutions, then eventually all the processors would run out of work, and the (parallel) search will terminate without finding any solution.

3 A Model for the Analysis of Speedup in Parallel DFS

3.1 Assumptions and Definitions

This model is a simple model of parallel depth-first search of a complete binary tree of M leaf nodes, with static partitioning of the tree among N processors. Each processor gets a contiguous part (of size $\frac{M}{N}$) of the search space for itself. Solutions occur among the leaf nodes. We assume that the solutions are distributed uniformly in a randomly located region. Such assumption is valid in problems for which no good heuristic is available to direct the search. For simplicity we assume that the number of nodes in any subtree of the search tree (generated by DFS) is proportional to the number of leaves in that subtree. Hence amount of work done in a search can be estimated by the number of leaf nodes visited. Sequential and parallel depth-first search stop after finding one solution.

1. Problem size $M$: is the size of the search space (in number of leaf nodes).
2. Number of processors $N$: is the number of processors being used to run parallel DFS.
3. Number of Solutions $\alpha$: is the number of solutions in the search space.
4. Solution region $r$: is the region of the search horizon which contains all the solutions.

We assume that the solutions are uniformly distributed in the region $r$.

\[\text{If a good heuristic is available, then it will be difficult to obtain even linear speedup. The heuristic would cause the solutions to migrate to the left part of the tree, where they will be quickly found by the sequential depth-first search.}\]
5. Nodes expanded $W_N$: is the average of the sum of the number of nodes visited by $N$ processors before one of the processors finds a solution. In our experiments, this average is taken over a large number of executions of parallel depth-first search on a problem instance. In each execution, successors of nodes are ordered randomly. $W_1$ is the average number of nodes expanded by sequential depth-first search.

6. Running time $T_N$: is the average execution time on $N$ processors. $T_1$ is the sequential execution time.

7. Speedup $S$: is the ratio $T_1 / T_N$. For the experiments discussed in this paper the overheads of parallel depth-first search used are negligible. Hence the execution time of a processor (in the sequential as well as parallel case) is proportional to the number of nodes expanded. Therefore,

$$Speedup \ S \approx \frac{W_1}{W_N}$$

8. Efficiency $E$: is the speedup divided by $N$. $E$ denotes the effective utilization of computing resources.

$$E = \frac{S}{N} \approx \frac{W_1}{W_N \times N}$$

In Section 3.2, we derive formulas for work done by sequential depth-first search and by parallel depth-first search. In the Section 3.3, these formulas are used to compute speedup and efficiency. Although, we are able to derive the formulas for $W_N$ only under certain restrictive conditions, we are still able to explain the results of Section 4.

### 3.2 Average work done by depth-first search

#### 3.2.1 Single processor performing DFS

When one processor performs depth-first search on a space with uniform distribution of $\alpha$ solutions over the entire space $M$,

$$Work\ done\ W_1 = \frac{M}{\alpha + 1}$$

This follows from the following analysis. When $\alpha$ out of $M$ nodes are chosen to be goal nodes at random, the mean number of leaf nodes visited by a single processor searching left to right is the expected value of the minimum of the indices of the solution nodes. Expected value of min of the indices

$$= \frac{1}{M} \sum_{\alpha=1}^{m-1} C_{\alpha-1} + 2 \frac{1}{M} C_{\alpha-1} + \cdots + (N - \alpha + 1) \frac{1}{M} \frac{1}{C_{\alpha-1}}$$
We have

\[ \sum_{i=0}^{i=N} C_M = {N+1 \choose M+1} \]

and \( N \choose M = 0 \) if \( N < M \)

Hence

\[
W_1 = \frac{1}{M} \sum_{i=0}^{i=M-1} \frac{C_{\alpha+1} - C_{\alpha+1} + \cdots + C_{\alpha+1}}{\alpha+1} \\
= \frac{C_{\alpha+1}}{\alpha+1} \\
= \frac{M+1}{\alpha+1} \]

\[
\simeq \frac{M}{\alpha+1} \text{ for large } M
\]

Now we consider the more general case in which \( \alpha \) solutions are distributed uniformly in a region \( r \), and the region \( r \) occurs at a random location in the search horizon. (Clearly, if \( r = M \), then we get the previous case in which \( \alpha \) solutions are distributed uniformly in the entire search horizon.) The starting point of the region \( r \) is any where between 0 and \( M - r \). Once search reaches the region, it searches for 1 solution in region \( r \) containing \( \alpha \) solutions. Hence,

\[
W_1 \simeq \frac{1}{2} (M - r) + \frac{r}{\alpha+1}
\]  \hspace{1cm} (1)

### 3.2.2 Multiple processors performing PDFS

Here we consider the case in which \( N \) processors search a space of size \( M \) in parallel. As before, \( \alpha \) solutions are distributed uniformly in a region \( r \), and the region \( r \) occurs at a random location in the search horizon. We have two cases, one when \( r \leq \frac{M}{N} \), and the other when \( r \geq \frac{M}{N} \).

When \( r \leq \frac{M}{N} \), we only compute \( W_N \) for the case in which one of the processors has the solution region entirely in its search space while the other processors do not have any solution in their search spaces. In this case, one of the processors searches a space of size \( \frac{M}{N} \) with a solution region of size \( r \). Assuming that all processors work at uniform speed, the work done by \( N \) processors in parallel equals \( N \) times the work done by a single processor searching on a space of size \( \frac{M}{N} \) with the region \( r \) randomly distributed with in this space.

Therefore

\[
W_N = N \left( \frac{1}{2} \left( \frac{M}{N} - r \right) + \frac{r}{\alpha+1} \right)
\]
When \( r \geq \frac{M}{N} \), we only analyze the simple case in which \( r \) is a multiple of \( \frac{M}{N} \) (i.e., \( r = k\frac{M}{N} \)), and \( k \) processors have \( \alpha \) solutions distributed in their search space, while the rest have no solutions in their search space. If \( k = 1 \), then the work done by \( N \) processors in parallel equals \( N \) times the work done by a single processor searching on a space of size \( \frac{M}{N} \) with \( \alpha \) solutions distributed uniformly in its space.

Therefore

\[
W_{\text{work done}} N \approx N \left( \frac{\frac{M}{N}}{\alpha + 1} \right) = \frac{M}{\alpha + 1}
\]

(3)

The above equation holds even if \( k > 1 \), provided \( \alpha \ll \frac{M}{N} \) (which is generally the case) for the following reason. When \( \alpha \ll \frac{M}{N} \), the probability of more than one processors finding solution at the same time becomes low. In parallel search, \( k \) nodes are visited in each step by the '\( k \) processors' and there are \( \alpha \) solutions in all, uniformly distributed in \( \frac{M}{N} \) steps without repetitions. This situation is similar to a single processor searching for \( \alpha \) solutions uniformly distributed in its search space of size \( \frac{M}{N} \).

### 3.3 Efficiency of Parallel Depth-First Search

We study the efficiency of parallel DFS in the following three cases.

- Uniform distribution of a single solution in the search space.
- Uniform distribution of multiple solutions in the search space.
- Non-uniform distribution of multiple solutions in the search space.

#### 3.3.1 Uniform distribution of a single solution in the search space

In this case, \( \alpha = 1 \) and \( r = M \). Using equations 1 and 3, we have

\[
\text{Average work done by sequential DFS } W_1 = \frac{M}{2}
\]

\[
\text{Average work done by parallel DFS } W_N = \frac{M}{1 + 1}
\]

\[
\text{Efficiency of parallel DFS } E = \frac{W_1}{W_N} = 1
\]

Thus if one solution is uniformly distributed in the search space, then parallel DFS only obtains linear speedup on the average.
3.3.2 Uniform distribution of multiple solutions in the Search Space

In this case again, we have \( r = M \). Using equations 1 and 3, we have

\[
W_1 = \frac{M}{\alpha + 1}
\]

\[
W_N = \frac{M}{\alpha + 1}
\]

\[
E = 1
\]

Thus the speedup remains linear, although the absolute amount of average work done by both sequential and parallel depth-first search goes down by the same amount when \( \alpha \) is increased.

3.3.3 Non uniform distribution of multiple solutions

In section 3.2.2, we have computed \( W_N \) only for two restrictive cases: (i) \( r = k\frac{M}{N} \) for some positive integer \( k \), and \( k \) processors have \( \alpha \) solutions uniformly distributed in their search space; (ii) \( r \leq \frac{M}{N} \), and the solution region falls within one processor’s search space. In the first case, by Equation 3 we have

\[
W_N = \frac{M}{\alpha + 1}
\]

In the second case, by Equation 2 we have

\[
W_N = N\left(\frac{1}{2}(\frac{M}{N} - r) + \frac{r}{\alpha + 1}\right)
\]

As we decrease \( r \) from \( M \) to \( \frac{M}{N} \) (in the multiples of \( \frac{M}{N} \)) and then from \( \frac{M}{N} \) to \( \alpha \) (continuously), we see that the value of \( W_N \) remains at \( \frac{M}{\alpha + 1} \) until \( r = \frac{M}{N} \) and then increases to \( \frac{M}{2} \). On the other hand from Equation 1, the work done in sequential depth-first search increases from \( \frac{M}{\alpha + 1} \) to \( \frac{M}{2} \). Hence we have a region of values for \( r \) where we get superlinear speedup.

We obtain maximum efficiency when \( r = \frac{M}{N} \).

\[
\text{Maximum Efficiency} = \frac{\frac{1}{2}(M - \frac{M}{N}) + \frac{k}{\alpha + 1}}{\frac{M}{\alpha + 1}}
\]

\[
= \frac{\alpha + 1}{2} - \frac{\alpha - 1}{2N}
\]

\[
\approx \frac{\alpha + 1}{2} \quad \text{(for large} \ N)\]

This analysis suggests that superlinear speedup occurs in depth-first search when the region of solutions is comparable to the size of the search space of each processor \( \frac{M}{N} \) and there are multiple solutions. When the region of solutions is very small compared to \( \frac{M}{N} \), the speedup is linear. The speedup is also linear when solutions are uniformly distributed over the entire search space.
4 Experimental Results

We analyzed the performance of parallel DFS in two problems. The first one is the search of an abstract binary tree (of depth 16) and the other is the classical 15-puzzle problem[21]. In both the problems, parallel DFS technique described in [18,19] was used. This scheme uses dynamic splitting of the search space; experiments with both of the problems deviate from the model, as the solution region may be scattered over many processors.

The binary tree search problem models the hacker's problem[24]. It involves searching a complete binary tree in which some of the leaf nodes are solution nodes. This problem was chosen in order to be able to directly manipulate the size of the solution region r and the density of solutions p. We implemented a program on Sequent Balance 21000 and experimented with different cases for up to 16 processors. In order to get mean values of \( T_N \) and \( W_N \), each experiment was repeated at least 100 times. The experiments analyzing superlinear speedup for large \( N \) were repeated more often. Fig. 1 shows a plot of efficiency of parallel depth-first search versus number of processors for the case in which solutions are distributed uniformly in the entire search space. The different plots are for different solution densities. The absolute amount of work done in both sequential and parallel depth-first search varies inversely with \( \alpha \) (the number of solutions) as predicted by equations 1 and 3 (this information is not shown in Fig. 1).

Fig. 2 shows efficiency of parallel depth-first search versus number of processors for the case in which all the solutions are distributed (uniformly) in a region r. In order to randomize the location of the solution region, each experiment involved searching the tree with the solution region starting at a randomly chosen location. Different plots are for different ratios of r to M with fixed \( \alpha \) (=4). As predicted by Equation 1, for small values of r, the sequential depth-first search searches approximately half the space. Notice that when \( \frac{r}{M} = \frac{1}{8} \), peak efficiency is obtained for 8-10 processors and when \( \frac{r}{M} = \frac{1}{16} \), efficiency increases up to 16 processors. Peak efficiency is not as high as predicted by our analysis because the solution region is not confined to one processor (as assumed in the analysis). Since the search space is divided dynamically, the solution region may be shared by many processors. However it appears that some processors were able to get a reasonable distribution of solutions which resulted in the superlinear speedup. Note that the peak speedup is smaller for the ratio \( \frac{r}{M} = \frac{1}{16} \) than for the ratio \( \frac{r}{M} = \frac{1}{8} \). One possible explanation for this is that as \( \frac{r}{M} \) decreases, the solution region becomes narrower which makes it harder for any of the processors to get a reasonable part of the overall solution region. This suggests that if the ratio \( \frac{r}{M} \) is decreased to a very small number (e.g., \( \frac{1}{256} \)), then the peak efficiency would not be much higher than 1. Thus it appears that the superlinear speedup is likely only for a small number of processors.

The 15-puzzle problem was used to experiment with a realistic search space. These experiments were performed on the BBN Butterfly parallel processor for up to 9 proces-
Figure 1: Efficiency vs number of processors for various solutions densities with uniform distribution. The number of solutions is given next to the curves.
Figure 2: Efficiency vs number of processors for different values of region of solutions. An efficiency greater than 1 indicates superlinear speedup.
sors. The experiments involved instances of the 15-puzzle with single solution, uniform distribution of multiple solutions, and non-uniform distribution of multiple solutions. A bounded depth-first search was used to limit the search space. Both the sequential and parallel DFS perform the search in a random order - the alternatives at each node in the search tree are randomly ordered for traversal.\footnote{It might seem that DFS with random ordering of successors would perform much worse than ordered DFS in which successors are ordered according to a heuristic. Experience with the IDA* algorithm on the 15-puzzle problem \cite{5} indicates that the use of the Manhattan distance heuristic (which is the best known admissible heuristic for 15-puzzle) does not make depth-first search any better.} The average timings for sequential and parallel search were obtained by running each experiment 100 times (for every 15-puzzle instance). Fig. 3 shows the average speedups obtained. The instances with uniform distribution of solutions show a near linear speedup. The maximum deviation of speedup is indicated by the banded region. The width of the banded region is is expected to reduce if a lot more repetitions (say a 1000 for every instance) were tried. The instances with non-uniform distribution of multiple solutions give superlinear speedups.

Both of these experiments confirm the predictions of the model. Superlinear speedup occurs in parallel depth-first search whenever many solutions occur within a restricted region of the search space. Linear speedup occurs when one or more solutions occur uniformly distributed in the search space.

5 Superlinear Speedup in Best-First Search

So far we discussed the phenomenon of superlinear speedup in depth-first search. Our analysis also predicts average superlinear speedup in parallel (best-first) branch-and-bound (B&B) for certain kinds of problems. As discussed in \cite{7,20}, the A* algorithm \cite{21} for state-space search is a best-first B&B algorithm.

In B&B/A*, if the heuristic function is consistent\cite{21}, then the cost of the nodes expanded in successive iterations never goes down (it either goes up or stays the same). Let \( V_i \) be the set of nodes expanded by A* after the cost has gone up \( i \)th time but before it has gone up \( i+1 \)th time. Clearly the cost of each node in \( V_i \) (for any \( i \)) is the same, and the heuristic function does not provide any discrimination among different nodes in \( V_i \). \( V_0 \) represents the expanded nodes that have the same cost as the start node. If the cost goes up \( L \) times in the search, then \( V_L \) is the set of nodes expanded whose cost is the same as the optimal solution. Note that the heuristic functions used in most of the problems solved by B&B/A* are consistent. For many such problems (e.g., the Vertex Cover problem, the 15-puzzle problem), \( V_i \) grows very rapidly. For these problems, nodes in \( V_L \) represent a very large fraction of all the nodes expanded by A* (For the instances of VCP and 15-puzzle that we tried, \( V_L \) constitute more than 70% of the total nodes expanded by A*\cite{9}.) Since all the nodes in \( V_L \) have the same cost, the heuristic function does not provide any
Figure 3: Experiments on 15-puzzle.
discrimination between these nodes. When a node in \( V_L \) is selected for expansion, and all of its successors are in \( V_L \) (i.e., they have the same cost as of their parent), then one of them is expanded and the other is put on a list of nodes to be expanded later. Hence A* essentially performs depth-first search of the nodes in \( V_L \). Thus the parallel formulations of best-first B&B/A* presented in [9,16,12] conceptually divide the search space represented by \( V_L \) dynamically and expand nodes in these subspaces in a depth-first fashion. Note that the memory requirement of these formulations still remains exponential.

Hence if the solution nodes in \( V_L \) are located close together, and fall in the search space of one of (or a small number of processors), then again we can experience superlinear speedup on the average.

6 Related Research

Monien, et. al.[17] studied a parallel formulation of depth-first search for solving the satisfiability problem. In this formulation, each processor tries to prove the satisfiability of a different subformula of the input formula. Due to the nature of the satisfiability problem, each of these subformulas leads to a search space with a different average density of solutions. Monien developed an abstract model to show that these differing solution densities are responsible for the average superlinear speedup. Our model is related to this model, as location of solutions in a part of the space indirectly causes difference in the solution densities of the regions searched by different processors.

If the search space is searched in a random fashion, then the number of nodes expanded before a solution is found is a random variable (let's call it \( T(1) \)). One very simple parallel formulation[15,2] of depth-first search is to let the same search space be searched by many processors independently until one of the processors finds a solution\(^4\). The total number of nodes expanded by a processor in this formulation is again a random variable (let's call it \( T(N) \)). Clearly, \( T(N) = \min\{V_1, \ldots, V_N\} \), where each \( V_i \) is a random variable \( T(1) \). If the average value of \( T(N) \) is less than \( \frac{1}{N} \) times \( T(1) \), then also we can expect superlinear speedup\([15,2,5]\). For certain distributions of \( T(1) \), this happens to be the case\([15]\). For example, if the probability of finding a solution at any level of the state-space tree is the same, then \( T(1) \) has this property\([5]\). Clearly, for these cases, our parallel formulation (in which different processors search different parts of the search space) would also exhibit superlinear speedup on the average. But for the distributions discussed in this paper, speedup using this simple parallel formulation is not superlinear (see [2]).

It is also possible to obtain superlinear speedup if simple depth-first search (without its

\(^3\)In some problems, it is possible to further distinguish between these nodes. In that case the superlinear speedup cannot be expected.

\(^4\)Note that our parallel formulation of DFS will always perform better than this simple parallel formulation, as in our formulation there is no duplication of work.
incorporation in the IDA* algorithm) is used to find a shortest path in a state-space graph or tree. In this case, the search continues even after finding the first solution. Whenever a search branch is encountered that has worse cost than the best solution found so far, it is terminated. Search ends when the whole space is exhausted. (Note that this kind of search is very different from that done by IDA*, in which each iteration is a simple depth-first search for any solution in a cost-bounded space.) In this case, the efficiency depends greatly on finding a good solution early in the search tree which could be used later to prune other branches. If many processors search the space simultaneously, then the probability of finding a good solution early in the search increases. This can cause superlinear speedups on the average. This phenomenon was investigated by Imai, et. al.[1]. Here the presence of super-linear speedup is not surprising at all, as for finding an optimal solution, “best-first” search methods (and the iterative-deepening method) are strictly better. The parallel version of depth-first search for this case tries to approximate best-first search, hence does better than the sequential depth-first search. On the other hand, for finding a solution in the state-space tree, depth-first search is not dominated by a “best-first” search (or any other) method.

The possibility of superlinear speedup in parallel best-first B&B algorithms has been investigated by many researchers[12,14,23]. Lai and Sahni[12] show that it is possible to obtain arbitrarily large speedups on certain abstract state-space trees. Based upon their experience with the traveling salesperson problem and the knapsack problem, they conclude that such behavior is unlikely in real problems. Quinn and Deo[23] note that superlinear speedup is unlikely ‘unless there are a large number of subproblems with the same lower bound as the solution cost’ (in our terminology, unless \( V_N \) is large). Our analysis shows that if \( V_N \) is large and if the goals are distributed uniformly, then on the average the speedup will only be linear. But if \( V_N \) is large and the goal nodes are located in a relatively small region, then superlinear speedup should be observed.

7 Concluding Remarks

The research reported in this paper was motivated by our earlier work on the parallel implementation of the IDA* algorithm.[18] Even though, IDA* is the best known sequential algorithm for solving the 15-puzzle (i.e., for finding a shortest sequence of moves between two configurations of the puzzle)[4], we were surprised to obtain superlinear speedups on the average on randomly chosen 15-puzzle instances\(^5\). Our first reaction was to assume that our sample size was too small, and that for big enough sample size, the average speedup would become sublinear. We were later encouraged by Monien’s analysis of the superlinear speedup in the satisfiability problem to perform a similar analysis for the 15-puzzle.

\(^{5}\)the speedup was averaged over many problem instances, and each problem instance was solved many times using the parallel formulation
Although the model presented here is rather artificial, it is adequate to explain the superlinear speedups in parallel depth-first search of search spaces containing a few goal nodes located in a small part of the search space. If all the goal nodes are spread uniformly over the search space, then the model predicts and our experimental results confirm that the average speedup is linear. The model also predicts superlinear speedup on the average even in parallel A*/B&B algorithms for certain kinds of problems. We expect the phenomenon of close location of many goal nodes to occur in many natural problems, as good nongoal nodes in the state space usually lead to many goal nodes (which would be located in a relatively small region of the state-space tree). For example, in the n-queens problem, a good partial assignment of queens leads to many goal nodes, whereas a bad partial assignment leads to no goal nodes[2]. Not surprisingly, researchers have observed superlinear speedup in some n-queen instances[25].

Acknowledgements: The authors would like to thank Rich Korf and Sartaj Sahni for helpful comments on an earlier draft of this paper.

References


