BIT 26 (1986), 35–43

AN UPPER BOUND FOR THE SPEEDUP OF PARALLEL BEST-BOUND BRANCH-AND-BOUND ALGORITHMS[†]

MICHAEL J. QUINN and NARSINGH DEO

Department of Computer Science
University of New Hampshire
Durham, New Hampshire 03824 USA

Computer Science Department
Washington State University
Pullman, Washington 99164 USA

Abstract.

This paper derives an upper bound for the speedup obtainable by any parallel branch-and-bound algorithm using the best-bound search strategy. We confirm that parallel branch-and-bound can achieve nearly linear, or even super-linear, speedup under the appropriate conditions.

CR Categories: F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory.
Keywords and Phrases: branch-and-bound, parallel algorithm, traveling salesman problem.

1. Introduction

Backtrack is a form of exhaustive search used to find an optimal solution to a problem. Branch-and-bound is a variant of backtrack that can take advantage of information about the optimality of partial solutions to avoid considering solutions that cannot be optimal. The enduring popularity of branch-and-bound algorithms is a testament to their usefulness. A sampling of the multitude of references in the literature indicates that branch-and-bound methods have been applied to computationally intensive problems in artificial intelligence [9] as well as in the solution of many *NP*-complete combinatorial optimization problems, including flow-shop and job-shop sequencing problems [4, 11, 12], traveling salesman problems [2, 7], general quadratic assignment problems [10], and integer programming problems [1]. The advent of VLSI technology has made possible the construction of large parallel computers. These computers can help satisfy the demands for higher performance that improvements in circuit speed alone cannot give. Thus, it is not surprising that work has been done to implement a branch-and-bound algorithm on a parallel computer.

Received February 1985. Revised August 1985.

[†] This work was supported by U.S. Army Research Office grant DAAG29-82-K-0107.

Mohan [8] has parallelized Little, Murty, Sweeney, and Karel's branch-and-bound algorithm [7] to solve the traveling salesman problem on the Cm* multiprocessor. His implementation exhibits good speedup, where speedup is defined to be the time required by the sequential algorithm to solve the problem divided by the time required by the parallel algorithm to solve the same problem. In fact, once architecture-dependent inefficiencies are removed, Mohan estimates that parallel branch-and-bound for the traveling salesman problem could exhibit nearly linear speedup; that is, the speedup could be nearly equal to the number of processors used.

This paper derives an upper bound for the speedup obtainable by *any* parallel branch-and-bound algorithm using the best-bound search strategy. We confirm that parallel branch-and-bound can achieve nearly linear, or even super-linear, speedup under the appropriate conditions.

Lai and Sahni [5] and Li and Wah [6] have also discussed parallel branch-and-bound algorithms. Our approach differs from theirs, however, in that it can be combined with experimental results from a problem domain to determine an upper bound for the expected speedup obtainable by executing a parallel form of the algorithm on a multiprocessor.

2. Branch-and-Bound.

We formally define branch-and-bound using the notation of Ibaraki [3]. Let P_0 denote an optimization problem and f denote some objective function to be minimized. (Of course, branch-and-bound can also be used to maximize the value of some objective function.) The decomposition process applied to P_0 can be represented by a rooted tree $\mathscr{B} = (\mathscr{P}, \mathscr{E})$, where $\mathscr{P}$, the set of nodes of $\mathscr{B}$, corresponds to the decomposed problems, and $\mathscr{E}$, the set of arcs of $\mathscr{B}$, corresponds to the decomposition process. The original problem P_0 is the root of $\mathscr{B}$. Given P_i and $P_j \in \mathscr{P}$, the arc $(P_i, P_j) \in \mathscr{E}$ if and only if P_j is generated from P_i by a decomposition. The set of terminal nodes of $\mathscr{B}$, denoted $\mathscr{T}$, are those partial problems which are solved without further decomposition. The level of $P_i \in \mathscr{B}$, denoted $L(P_i)$, is the length of the path from P_0 to P_i in $\mathscr{B}$. P_0 has level 0. Like Ibaraki, we assume that $\mathscr{B}$ is a finite tree.

A branch-and-bound algorithm attempts to solve P_0 by examining a small number of elements in $\mathscr{P}$. A lower-bounding function $g : \mathscr{P} \to E \cup \{\infty\}$ is calculated for each decomposed problem as it is created, where E represents the set of nonnegative real numbers. This lower bounding function g satisfies three conditions:

(a) $g(P_i) \leqslant f(P_i)$ for $P_i \in \mathscr{P}$

(b) $g(P_i) = f(P_i)$ for $P_i \in \mathscr{T}$

(c) $g(P_j) \geqslant g(P_i)$ if P_j is a child of P_i in $\mathscr{B}$.

At any point during the execution of a branch-and-bound algorithm there exists a set $\mathscr{A}$ of problems that have been generated but not yet examined. The best-bound search strategy always chooses the problem from $\mathscr{A}$ with the smallest lower bound. In the case when two or more problems have the same smallest lower bound g', we assume that the best-bound search chooses problem $P_i \in \mathscr{A}$ such that $g(P_i) = g'$ and for all $P_j \in \mathscr{A}$, $g(P_j) = g' \Rightarrow L(P_j) \leqslant L(P_i)$, i.e., from the problems with the smallest lower bound, it chooses one that is as deep as any other in the search tree. The first solution found is an optimal solution, because no unexamined problems have a smaller lower bound. Hence the algorithm can terminate – the bound on the unexamined problems precludes them from being searched.

We now introduce some notation of our own. Let $\mathscr{P}' \subseteq \mathscr{P}$ denote the set of problems actually examined by a branch-and-bound algorithm using the best-bound strategy. Let P_s denote the problem in $\mathscr{T}$ whose solution terminates the algorithm. Clearly $P_0 \in \mathscr{P}'$ and $P_s \in \mathscr{P}'$. Note that $g(P_s) = f(P_0)$. Let $U_i = \{P_j | P_j \in \mathscr{P}' \text{ and } L(P_j) = i\}$, for all $i \geqslant 0$. Note that $\mathscr{P}' = \cup_{i=0}^{\infty} U_i$. Let $W_i = \{P_j | P_j \in U_i \text{ and } g(P_j) < f(P_0)\}$, for all $i \geqslant 0$. Let $W = \cup_{i=0}^{\infty} W_i$. In other words, W is the set of subproblems whose lower bounds are less than the cost of the optimal solution. Note that $U_i - W_i$ is the set of subproblems at level i of the state space tree whose lower bounds are equal to the cost of the optimal solution. Let $u_i = |U_i|$ and $w_i = |W_i|$. Let $t(i)$ denote the amount of time needed to examine and decompose a subproblem at level i of the search tree $\mathscr{B}$.

The execution time of a branch-and-bound algorithm can be expressed in terms of u_i and $t(i)$. The execution time of a branch-and-bound algorithm using the best-bound search strategy is $\sum_{i=0}^{\infty} u_i t(i)$.

Theorem 1 proves that if at some level of the state space tree a best-bound branch-and-bound algorithm examines a number of nodes whose lower bounds are equal to the cost of the optimal solution, and these nodes are closer to the root of the state space tree than the node representing the optimal solution, then one of these nodes is an ancestor of the node representing the optimal solution. This theorem is used when we want to determine the minimum number of such nodes that must be examined.

THEOREM 1. *If $0 \leqslant i < L(P_s)$ and $u_i > w_i$, then there exists $P_j \in U_i - W_i$ such that P_j is an ancestor of P_s in $\mathscr{B}$.*

PROOF BY CONTRADICTION. Assume there exists an i, $0 \leqslant i < L(P_s)$, such that $u_i > w_i$, yet there is no node $P_j \in U_i - W_i$ that is an ancestor of P_s. This implies that there exists a node $P_k \in W_i$ that is the ancestor of P_s, and $g(P_k) < g(P_j)$, for all $P_j \in U_i - W_i$. Since the best-bound search strategy always chooses the active problem with the smallest lower bound, and favors a problem deeper in the search tree when two active problems have the same lower bound, the problems in $U_i - W_i$ cannot be examined before P_s. Thus they are never examined, which contradicts the definition of U_i. Therefore, the ancestor of P_s at level i must be in the set of problems $U_i - W_i$.

3. Parallel Branch-and-Bound.

In our analysis we assume that branch-and-bound is parallelized in the manner described by Mohan [8]. Let p denote the number of processors executing a parallel branch-and-bound algorithm using the best-bound search strategy. The list of active problems, arranged so that the unexamined nodes of the state space tree are available in nondecreasing order of their lower bounds, is accessible to all p processors. Each processor repeatedly removes the element with the smallest lower bound, decomposes the problem into several subproblems, determines the lower bounds of these subproblems, and inserts them into the list in such a way that the list of active elements remains sorted. Each processor must have exclusive possession of the list of active elements while performing insertions and deletions. However, since we are determining an upper bound for the speedup achievable, we assume that there is no processor contention for access to this list. The processors continue to iterate until no unexamined subproblems have lower bounds less than $f(P_0)$.

When execution of the algorithm begins with consideration of the original problem, there are not enough problems in $\mathscr{A}$ to keep all the processors busy (assuming that $p > 1$). In this case some of the processors are idle. Even when there are enough active problems to keep all the processors busy, *only those processors that examine problems in $\mathscr{P}'$ are doing useful work.* Examining, decomposing, and finding lower bounds for problems in $\mathscr{P} - \mathscr{P}'$ cannot lead to a solution, and only serves to increase the size of $\mathscr{A}$ needlessly.

Unlike the sequential algorithm, in which examination of P_s leads to the termination of the algorithm, the execution of the parallel algorithm may have to continue on beyond this point. This is because taking the p best problems one "iteration" (admittedly a loosely-defined concept in an asynchronous algorithm) is not the same as taking the single best problem p iterations. It is for this reason that the parallel algorithm may examine problems in $\mathscr{P} - \mathscr{P}'$. Problem P_s may actually be examined too soon; problems with smaller lower bounds may not yet have been generated. Thus the parallel algorithm must continue until there exist no problems with lower bounds less than $f(P_0) = g(P_s)$.

The fact that one processor iterating p times may not examine the same active problems as p processors iterating once could cause the speedup of the parallel algorithm to be greater than p. Consider the sequential algorithm for a moment. Until the problem P_s is examined, problems with lower bounds equal to $g(P_s)$ may be examined. However, once P_s has been examined, the problems in $\mathscr{A}$ with lower bounds equal to $g(P_s)$ are never examined. Instead, the algorithm terminates. An analogous situation arises in the execution of the parallel algorithm. Once P_s has been examined, only those subproblems with a smaller

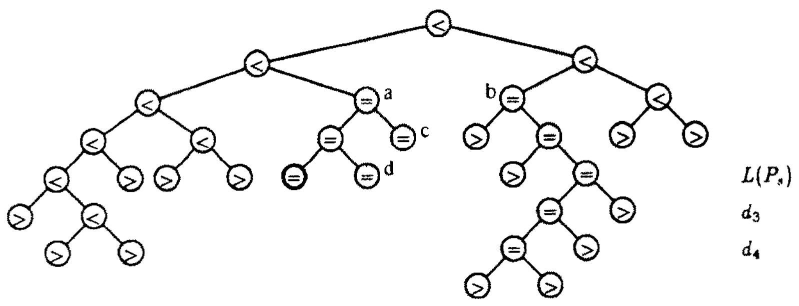

Fig. 1. Problems generated by a sequential branch-and-bound algorithm.

lower bound are examined. Thus if there are a large number of problems in $U-W$, if there is enough work involved in examining the nodes of W plus P_s and the ancestors of P_s (Theorem 1) to keep p processors busy, and if the structure of the subtrees in $U-W$ is such that the ancestors of P_s can be explored without bringing elements of $U-W$ to the front of the active list, then the parallel algorithm could exhibit a faster than linear speedup.

Assume that the tree in Fig. 1 represents the set of problems generated by a sequential branch-and-bound algorithm. $\mathscr{P}'$ consists of the interior nodes of this tree and the solution node – the heavily outlined leaf node. The interior of each node is marked '<', '=', or '>' to indicate whether the lower bound of the corresponding problem is less than, equal to, or greater than the cost of the solution. The first problems to be encountered are those whose lower bounds are less than the solution's cost. After these problems have been examined, the sequential algorithm has a choice between the problems associated with the nodes marked *a* and *b*, since they have equal depth in the search tree. In this example the sequential algorithm chooses node *b* – otherwise, *b* would never be searched, and it would be a leaf node. Choices in favor of the left subtree cause nodes *c* and *d* to be leaf nodes. The parallel algorithm could end up doing much less work than the sequential algorithm, if it happens to choose node *a* rather than node *b*. In this case the subtree of which *b* is the root may never be searched, if there is enough work examining *a*'s subtree to keep all the processors busy until the solution is discovered. Since it is possible for each of p processors to do less than $1/p$ of the work of the sequential algorithm, speedup may be greater than p.

An expression for the minimum execution time for parallel branch-and-bound requires the definition of a few more terms.

$$\text{Let } v_i = \begin{cases} \min\{w_i+1, u_i\}, & \text{for } 0 \leqslant i < L(P_s) \qquad \text{(Theorem 1)} \\ w_i+1, & \text{for } i = L(P_s) \\ w_i, & \text{for } i > L(P_s) \end{cases}$$

$$\text{Let } d_1 = \begin{cases} \infty, \text{ if } p \geqslant v_j \text{ for all } j \geqslant 0 \\ \max \{i | 0 < j < i \Rightarrow v_j = p\}, \text{ otherwise} \end{cases}$$

$$\text{Let } d_2 = \begin{cases} 0, \text{ if } d_1 = \infty \\ \min \{i | j > i \Rightarrow v_j = p\}, \text{ otherwise} \end{cases}$$

$$\text{Let } d_3 = \min \{i | j > i \Rightarrow v_j = 0\}$$

$$\text{Let } d_4 = \min \{i | j > i \Rightarrow u_j = 0]$$

$$\text{Let } R_1 = \sum_{i=d_1}^{d_2-1} (v_i - p)t(i)$$

$$\text{Let } R_2 = \sum_{i=d_2}^{d_3-1} (p - v_i)t(i).$$

To elaborate, v_i is the minimum number of nodes that the parallel algorithm examines at level i. Level d_1 is the first level in the search tree $\mathscr{B}$ in which there are enough problems in $\mathscr{P}'$ to keep all the processors busy. If $d_1 = \infty$, then there will never be enough useful work (problems in $\mathscr{A} \cap \mathscr{P}'$) to keep all the processors busy. Similarly, $d_2 - 1$ is the last level in the tree that contains more problems in $\mathscr{P}'$ than p. From level d_1 through $d_2 - 1$, there are more problems from $\mathscr{P}'$ accumulating in $\mathscr{A}$ than can be removed by the processors. Once the search continues beyond level d_2, the processors begin to catch up, and the number of problems in $\mathscr{A} \cap \mathscr{P}'$ begins to decline. The parallel algorithm may not have to examine any problems beyond d_3. The sequential algorithm does not examine any problems beyond level d_4. R_1 represents the backlog of work that accumulates when the search is between levels d_1 and d_2. R_2 represents the amount of time available to processors to eliminate the backlog while other processors examine problems from levels d_2 and beyond. If $R_1 \leqslant R_2$, then the parallel algorithm may terminate in as few as $d_3 + 1$ "iterations", representing the time it takes to generate and explore the problems deepest in $\mathscr{P}'$.

The minimum execution time of a parallel branch-and-bound algorithm using the best-bound search strategy is presented below.

CASE 1: $R_1 \leqslant R_2$ (Figure 2).
Execution time $\geqslant \sum_{i=0}^{d_3-1} t(i)$.

CASE 2: $R_1 > R_2$ (Figure 3).

Execution time $\geqslant \sum_{i=0}^{d_1-1} t(i) + \left(\sum_{i=d_1}^{d_3} v_i t(i)\right) \Big/ p$.

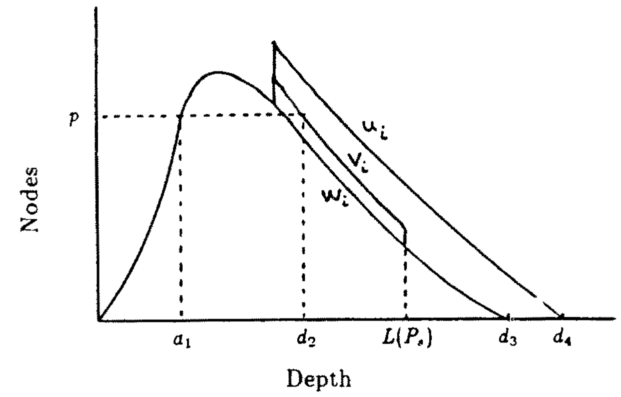

Fig. 2. $R_1 \leqslant R_2$. Backlog of unexamined subproblems is eliminated by time solution node is examined.

The speedup of a parallel algorithm is the ratio between the excecution time of the sequential algorithm and the execution time of the parallel algorithm. The speedup of the parallel branch-and-bound algorithm is given below.

CASE 1: $R_1 \leqslant R_2$

$$\text{Speedup} \leqslant \left(\sum_{i=0}^{d_4} u_i t(i)\right) \Big/ \sum_{i=0}^{d_3} t(i).$$

CASE 2: $R_1 > R_2$

$$\text{Speedup} \leqslant \left(\sum_{i=0}^{d_4} u_i t(i)\right) \Big/ \left(\sum_{i=0}^{d_1-1} t(i) + \left(\sum_{i=d_1}^{d_3} v_i t(i)\right) \Big/ p\right).$$

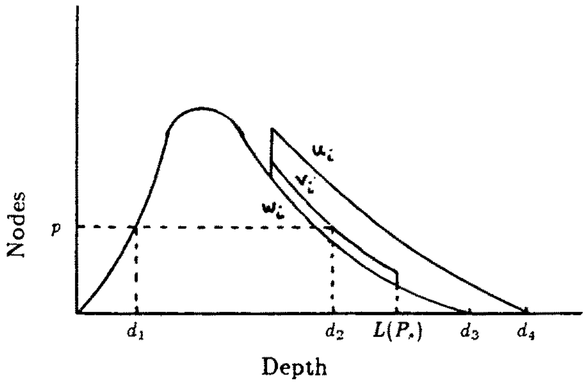

Fig. 3. $R_1 > R_2$. Backlog of unexamined subproblems is not eliminated by time solution node is examined. Algorithm requires additional iterations to clear backlog.

THEOREM 2. *Given p processors executing a parallel branch-and-bound algorithm using the best-bound search strategy, the optimum speedup $> p$ only if*

$$R_1 \leqslant R_2 \Rightarrow \sum_{i=0}^{d_4} u_i t(i) > p \sum_{i=0}^{d_3} t(i)$$

$$R_1 > R_2 \Rightarrow \sum_{i=0}^{d_4} (u_i - v_i) t(i) > \sum_{i=0}^{d_1-1} (p - v_i) t(i).$$

PROOF. Since we have an expression for the upper bound on speedup, we can determine those conditions that are necessary to speedup to be greater than p, given p processors.

CASE 1: $R_1 \leqslant R_2$

$$\sum_{i=0}^{d_4} u_i t(i) / \sum_{i=0}^{d_3} t(i) > p$$

$$\Rightarrow \sum_{i=0}^{d_4} u_i t(i) > p \sum_{i=0}^{d_3} t(i)$$

CASE 2: $R_1 > R_2$

$$\sum_{i=0}^{d_4} u_i t(i) \Bigg/ \left(\sum_{i=0}^{d_1-1} t(i) + \left(\sum_{i=d_1}^{d_3} v_i t(i) \right) \Big/ p \right) > p$$

$$\Rightarrow \sum_{i=0}^{d_4} u_i t(i) > p \sum_{i=0}^{d_1-1} t(i) + \sum_{i=d_1}^{d_3} v_i t(i)$$

$$\Rightarrow \sum_{i=0}^{d_4} u_i t(i) > \sum_{i=0}^{d_1-1} (p - v_i) t(i) + \sum_{i=0}^{d_3} v_i t(i)$$

$$\Rightarrow \sum_{i=0}^{d_4} u_i t(i) > \sum_{i=0}^{d_1-1} (p - v_i) t(i) + \sum_{i=0}^{d_4} v_i t(i) \text{ (Since } v_i = 0 \text{ for all } i > d_3)$$

$$\Rightarrow \sum_{i=0}^{d_4} (u_i - v_i) t(i) > \sum_{i=0}^{d_1-1} (p - v_i) t(i):$$

4. Discussion.

Given the formulas we have presented in this paper, a person can use measurements taken from the execution of a sequential best-bound branch-and-bound algorithm operating in a particular problem domain to determine an upper bound on the expected speedup of a parallel branch-and-bound algorithm operating in the same domain. The speedup achievable is likely to vary from one problem domain to another. However, some general comments can be made.

For sufficiently large problems, it is unlikely that a small number of decompositions will produce a subproblem with a lower bound greater than the cost of a solution. Thus if the decomposition of a single problem results in an average of k subproblems being generated, the first i levels of $\mathscr{B}$ are likely to contain nearly k^i subproblems in $\mathscr{P}'$. This implies that for a reasonable number of processors p, no processor should have to spend more than about $\log_k p$

iterations idling before it can begin examining problems in $\mathscr{P}'$. Hence the parallelization of branch-and-bound using the best-bound search strategy has a high potential for achieving nearly linear speedup until the Amdahl Effect becomes a consideration (i.e., until there is not enough work for the number of available processors). Theorem 2 indicates that speedup is unlikely to exceed the number of processors used, unless there are a large number of subproblems with the same lower bound as the solution cost.

An important consideration in the design of a branch-and-bound algorithm is how much time should be spent determining the lower bound on the solution of a problem. Finding a tight lower bound requires more computation time, but results in the examination of fewer nodes. Settling for a looser lower bound enables less time to be spent evaluating each node, at the cost of examining more problems. This consideration has a special significance in a parallel processing environment, because it is important that there always be enough unexamined subproblems to keep all the processors busy. However, since the number of unexamined nodes is likely to be an exponential function of the depth of the search (as discussed in the previous paragraph), the parallel algorithm designer should be careful not to swing too far toward the second extreme.

Acknowledgements.

We thank the referees for their helpful suggestions.

REFERENCES

1. A. M. Geoffrion and R. E. Marsten, *Integer programming algorithms: a framework and state of the art survey*, Management Science, 18, 9 (May 1972), 465–491.
2. M. Held and R. Karp, *The traveling salesman problem and minimum spanning trees: Part II*, Mathematical Programming, 1, 1 (October 1971), 6–25.
3. T. Ibaraki, *Theoretical comparisons of search strategies in branch-and-bound algorithms*, Internat. Journal of Computer and Information Sciences, 5, 4 (1976), 315–344.
4. E. Ignall and L. Schrage, *Application of branch and bound technique to some flow-shop scheduling problems*, Operations Research, 13, 3 (May–June 1965), 400–412.
5. T.-H. Lai and S. Sahni, *Anomalies in parallel branch-and-bound algorithms*, Proc. 1983 Internat. Conf. on Parallel Processing, 183–190.
6. G.-J. Li and B. W. Wah, *Computational efficiency of parallel approximate branch-and-bound algorithms*, Proc. 1984 Internat. Conf. on Parallel Processing, 473–480.
7. J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, *An algorithm for the traveling salesman problem*, Operations Research 11,6 (November-December 1963), 972–989.
8. J. Mohan, *Experience with two parallel programs solving the traveling salesman problem*, Proc. 1983 Internat. Conf. on Parallel Processing, 191–193.
9. N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA, 1980, Section 2.4.
10. J. F. Pierce and W. B. Crowston, *Tree-search algorithms for quadratic assignment problems*, Naval Research Logistics Quarterly, 18, 1 (March 1971), 1–36.
11. L. Schrage, *Solving resource-constrained network problems by implicit enumeration-nonpreemptive case*, Operations Research, 18, 2 (March-April 1970), 263–278.
12. L. Schrage, *Solving resource-constrained network problems by implicit enumeration-preemptive case*, Operations Research, 20, 3 (May-June 1972), 668–677.