THE STATUS OF MANIP - A MULTICOMPUTER ARCHITECTURE FOR SOLVING COMBINATORIAL EXTREMUM-SEARCH PROBLEMS

Benjamin W. Wah, Guo-Jie Li and Chee-Fen Yu
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907.

Abstract

In this paper, we report the status of study on MANIP, a parallel computer for solving combinatorial extremum-search problems. The most general technique that can be used to solve a wide variety of these problems on a uniprocessor system, optimally or sub-optimally, is the branch-and-bound algorithm. We have adapted and extended branch-and-bound algorithms with lower-bound tests for parallel processing. Three major problems are identified in the design: interconnection networks for supporting the selection of subproblems, anomalies in parallelism and virtual-memory support. A unidirectional ring network has been shown to be the most cost-effective selection network. We provide sufficient conditions so that anomalies can be eliminated under certain conditions and show that anomalies are unavoidable otherwise. Lastly, we develop a modified branch-and-bound algorithm so that the amount of memory space required under a best-first search is significantly reduced.

Key Words and Phrases: Anomalies, approximations, dominance criteria, heuristic search, NP-hard problems, parallel branch-and-bound algorithms, ring network, virtual memory.

1. INTRODUCTION

MANIP is a parallel computer architecture for processing combinatorial extremum-search problems by branch-and-bound algorithms with lower-bound tests [WAH81]. In this paper, we report the current status of research on MANIP. The optimization problems considered are characterized by an objective function to be minimized or maximized and a set of constraints to be satisfied. A special class is the class of *NP-hard* problems [GAR79]. There are no known optimal algorithms to solve these problems in a time that increases polynomially with the problem size. Many practical problems belong to this class. Examples include traveling salesman, warehouse location, job-shop scheduling, graph partitioning and mathematical programming.

Owing to the apparently exponential running times of algorithms for NP-hard problems, optimal solutions are usually infeasible to obtain. In practice, approximate solutions are often acceptable alternatives. The simplest approach is to solve small problems optimally, and to apply heuristics to solve large problems sub-optimally. However, it is difficult to assess the goodness of results. Polynomial-time approximation algorithms with guaranteed error bounds have been developed for some problems [SAH77]. However, efficient algorithms for slightly different problems are usually different and cannot be generalized. A general algorithm that can solve, optimally or sub-optimally (with guaranteed error bounds), a wide variety of these problems is, thus, preferred. A good candidate is the branch-and-bound algorithm.

A branch-and-bound algorithm is a partitioning algorithm. It decomposes a problem into smaller subproblems and repeatedly decomposes until infeasibility is proved or a solution is obtained [LAW66]. Branch-and-bound algorithms will be discussed in Section 2. A parallel version of the algorithm is presented in Section 3. In Section 4, anomalies of parallel branch-and-bound algorithms with lower-bound tests are studied. Sufficient conditions to avoid some of these anomalies are discussed. The problem on the exponential space requirement will be addressed in Section 5.

2. SERIAL BRANCH-AND-BOUND ALGORITHMS

Many theoretical properties of branch-and-bound algorithms have been developed [NIL80, KOH74, IBA76a, IBA76b,IBA77]. The way in which a problem is repeatedly decomposed into smaller subproblems can be represented as a finite rooted tree $\mathbf{B} = (\mathbf{P}, \mathbf{E})$, where $\mathbf{P}$ is a set of subproblems, and $\mathbf{E}$ is a set of edges. The root of the tree is P_0. If a subproblem P_{i_j} is obtained from P_i by decomposition, then $(P_i, P_{i_j}) \in \mathbf{E}$. The *level number* of a node is the number of edges leading from the root to this node (the root is at level 0). Let $f(P_i)$ be the value of the best solution obtained by evaluating all the subproblems decomposable from P_i, let P_{i_j} be the j-th subproblem decomposable from P_i, and let k_i be the number of such subproblems (i.e., $k_i = |\{(P_i,x): (P_i,x)\in\mathbf{E}\}|$). Then f satisfies:

$$f(P_i) = \min_{j=1,\ldots,k_i} \{f(P_{i_j})\} \quad (1)$$

Each subproblem is characterized by a value that is computed from a lower-bound function g. The lower-bound function satisfies the following properties:

(a) g is a lower-bound estimate of f (2)
(b) g is exact when P_i is feasible (3)
(c) lower bounds of descendant nodes always increase (4)

Branch-and-bound algorithms can be characterized by four constituents: a branching rule, a selection rule, an elimination rule and a termination condition. The first two rules are used to decompose problems into simpler subproblems and appropriately order the search. The last two rules are used to to eliminate generated subproblems that are not better than the ones already known.

Only lower-bound elimination and termination rules are used in MANIP. The elimination of subproblems due to dominance tests are studied elsewhere [LI84]. For simplicity, only the search for a single optimal solution is

This research was supported by National Science Foundation Grant ECS81-05968.

11th Annual Int'l Symposium on Computer Architecture, 1984.

considered here. The computational efficiency for searching all optimal solutions can be investigated similarly.

2.1 Selection Rules

This examines the list of active subproblems and selects one for expansion. If the list is maintained in a first-in/first-out order, the algorithm is called a *breadth-first search*. If the list is maintained in a last-in/first-out order, the algorithm is called a *depth-first search*. Lastly, if the list is maintained in increasing order of lower bounds, the algorithm is called a *best-first search*. Ibaraki mapped these searches into a general form called *heuristic searches* [IBA76b]. A heuristic function is defined which governs the order in which subproblems are selected and decomposed. The algorithm always decomposes the subproblem with the minimum heuristic value. In a best-first search, the lower-bound values define the order of expansion. Therefore, the lower-bound function can be taken as the heuristic function. In a breadth-first search, subproblems with the minimum level numbers are expanded first. The level number can, thus, be taken as the heuristic function. Lastly, in a depth-first search, subproblems with the maximum level numbers are expanded first. The negation of the level number can be taken as the heuristic function. If $\mathbf{U}$ is the current list of active subproblems in the process of expansion and h is the heuristic function, the search function for a subproblem to expand in a serial branch-and-bound algorithm is:

$$s_S(\mathbf{U}) = \{P_i \mid h(P_i) = \min_{P_j \in \mathbf{U}} h(P_j)\} \tag{5}$$

2.2 Lower-bound Elimination and Termination Rules

A lower bound is calculated for a subproblem when it is created. If a subproblem is a feasible solution with the best objective-function value so far, the solution value becomes the *incumbent* z. The incumbent represents the best solution obtained so far in the process. In minimization problems, if the lower bound of a subproblem exceeds the value of the incumbent, this subproblem can be pruned because it will not lead to an optimal solution. The decomposition process continues until all subproblems are either expanded or pruned. During the computation, P_i is terminated if:

$$g(P_i) \geq z \tag{6}$$

Let $\mathbf{L}$ denotes the lower-bound cutoff test, that is, $P_j \mathbf{L} P_i$ means that P_j is a feasible solution and $f(P_j) \leq g(P_i)$.

The above lower-bound test for obtaining an exact optimal solution can be relaxed in order to obtain a suboptimal solution with guaranteed accuracy [LAW66]. Suppose it were decided that a deviation of 10% from the optimum was tolerable. If a feasible solution of 150 is obtained, all subproblems with lower bounds of 136.4 (or $150/(1+0.1)$) or more can be terminated since they cannot lead to a solution that deviates by more than 10% from 150. This technique significantly reduces the amount of intermediate storage and the time needed in order to arrive at a suboptimal solution. Define an *allowance function* $\epsilon(z)$: $\mathbf{R} \rightarrow \mathbf{R}$ (set of reals) such that P_i is terminated if:

$$g(P_i) \geq z - \epsilon(z). \tag{7}$$

The final incumbent value z_F that is obtained by using the modified lower-bound test deviates from the optimal solution value z_O by [IBA76a]:

$$z_F - \epsilon(z_F) \leq z_O \leq z_F \tag{8}$$

Examples of often used allowance functions are:

$\epsilon(z) = \epsilon \geq 0$ (absolute error deviation) and (9)

$$\epsilon(z) = \frac{\epsilon z}{1+\epsilon} \quad \epsilon \geq 0,\ z \geq 0 \text{ (relative error deviation).} \tag{10}$$

Properties of these functions are similar. In this paper, the function for relative error deviation is assumed.

2.3 Serial Branch-and-Bound Algorithm (Single Solution and Lower-bound Tests)

g = lower-bound function; ϵ = allowance function;
s_S = serial search function; z = incumbent value;
$\mathbf{U}$ = set of active subproblems.

(1) (Initialize): $z \leftarrow \infty$; $\mathbf{U} \leftarrow \{P_0\}$;
(2) (Select): If $\mathbf{U} = \emptyset$ then go to Step 7 else let $P_i \leftarrow s_S(\mathbf{U})$, $\mathbf{U} \leftarrow \mathbf{U} - \{P_i\}$; if $g(P_i) \geq z - \epsilon(z)$ then go to Step 2;
(3) (Decompose): Generate sons $P_{i_1}, P_{i_2}, \ldots, P_{i_{k_i}}$ of P_i; $\mathbf{U} \leftarrow \mathbf{U} \cup \{P_{i_1}, \ldots, P_{i_{k_i}}\}$
(4) (Feasibility test): For all $j \in \{1, \ldots, k_i\}$, if P_{i_j} is a feasible solution then $\mathbf{U} \leftarrow \mathbf{U} - \{P_{i_j}\}$ and let $z \leftarrow \min[z, f(P_{i_j})]$;
(5) (Lower-bound test): For all $j \in \{1, \ldots, k_i\}$, if $g(P_{i_j}) \geq z - \epsilon(z)$ then $\mathbf{U} \leftarrow \mathbf{U} - \{P_{i_j}\}$;
(6) (Terminate): Go to Step 2;
(7) (Halt): $f(P_0)$ satisfies $z - \epsilon(z) \leq f(P_0) \leq z$.

In the above algorithm, eliminations are done after branching instead of after selection as in Ibaraki's algorithm [IBA76a]. This reduces the memory space required for storing the active subproblems.

3. PARALLEL BRANCH-AND-BOUND ALGORITHMS

Branch-and-bound algorithms have inherent parallelism. Each of the four rules of branch-and-bound algorithms can be implemented by parallel processing.

(a) *Parallel selection of subproblems*: In the parallel case, a set of subproblems less than or equal in size to the number of processors have to be selected for decomposition in each iteration. The selection problem is especially critical under best-first search because a set of subproblems with the minimum lower bounds must be selected. The selection function (Eq. 5) becomes:

$$s_P(\mathbf{U}) = \begin{cases} \{P_{i_1}, \ldots, P_{i_k}\} \text{ where } h(P_i) < h(P_j), P_i, P_j \in \mathbf{U},\ i \notin \{i_1, \ldots, i_k\},\ j \in \{i_1, \ldots, i_k\} & \text{if } |\mathbf{U}| > k \\ \mathbf{U} & \text{if } |\mathbf{U}| \leq k \end{cases} \tag{11}$$

where k is the number of processors. This returns k subproblems with the minimum heuristic values from $\mathbf{U}$.

(b) *Parallel branch*: The subproblems assigned to the processors can be decomposed in parallel. In order for the processors to be well utilized, the number of active subproblems should be greater than or equal to k.

(c) *Parallel termination test*: Multiple infeasible nodes can be eliminated in each iteration. Further, multiple feasible solutions may be generated, and the incumbent may have to be updated in parallel.

(d) *Parallel elimination test*: The lower-bound test (Eq's 6 or 7) can be sped up by comparing lower bounds of multiple subproblems with the incumbent. However, the execution of the bounding function is problem-dependent, and software implementation is more flexible.

Desai has proposed to use implicit enumeration to find the optimal solution of NP-hard problems [DES78]. The system dedicates one processor to each subproblem, and this processor reports to its parent processor when

the evaluation is completed. Implicit enumeration is time-consuming and wasteful for large problems.

Imai et al. studied parallel branch-and-bound algorithms with a depth-first search in a multiprocessor with shared memory [IMA79]. Depth-first search is used owing to memory limitations. Similarly, Dessouki and Huen studied the same algorithm on a general purpose architecture with a slow communication network [ELD80]. Depth-first search is not effective in minimizing the execution time because a serial best-first search expands the minimum number of nodes when all the lower bounds are distinct [LAW66]. However, a best-first search is space-consuming because the number of active subproblems is large. Currently, with the availability of VLSI technology, larger and inexpensive memories and faster communication media, reducing the execution time becomes an important problem.

MANIP was proposed for implementing parallel branch-and-bound algorithms with a best-first search and lower-bound tests [WAH81,WAH84b]. The system was designed with the following objectives:

(1) it should be modularly expandable to include a very large number of processors;
(2) the design must have high performance, and the cost should be kept low by replicating simple cells;
(3) it should use distributed control so that a controller would not become the bottleneck.

There are multiple subproblem memory controllers connected by a selection network and a secondary-storage redistribution network (Figure 1). Each of the m controllers is connected to n processors and expands n subproblems in parallel. The controller is also responsible for managing the local list of subproblems in ascending order of lower bounds, communicating with other processors through the networks, updating the incumbent and maintaining the virtual memory operating system. Parallel selection is achieved through the selection network. In order to select m·n subproblems with the minimum lower bounds for processing in each iteration, n subproblems with the minimum lower bounds are selected from each controller and sent to the neighboring controller through a ring network. The subproblems received from the neighboring controller is inserted into the local list. The process is repeated m−1 times, and it is proved that each of the controller has n of the m·n subproblems with the minimum lower bounds. However, a complete selection using m−1 shifts is time-consuming. Assuming random distribution of the m·n required subproblems, over 70% of the required subproblems can be selected with only one shift [WAH84b]. The unidirectional ring network is found to be the most cost-effective selection network [WAH84a]. The incumbent is maintained in a global data register. The concurrent update of the incumbent is facilitated by a sequential associative memory that finds the minimum of the feasible solutions when they are shifted out bit-serially and synchronously from the controllers. Since the register containing the incumbent is accessible to all the processors, lower-bound elimination and termination tests can be done in parallel.

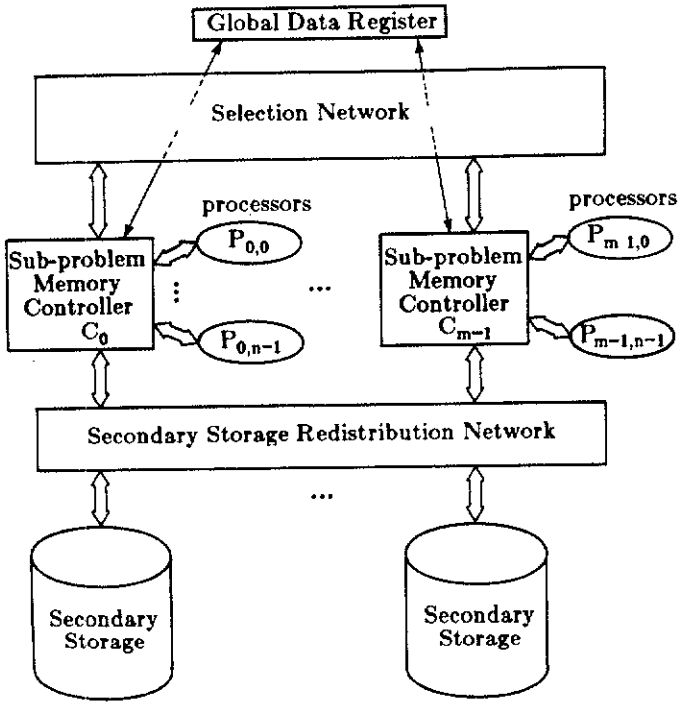

Figure 1. The architecture of MANIP.

4. COMPUTATIONAL EFFICIENCY OF PARALLEL BRANCH-AND-BOUND ALGORITHMS

4.1 Anomalies on Parallelism

From experimental results on vertex-cover, 0-1 knapsack and integer-programming problems, the reduction in computational time of branch-and-bound algorithms with lower-bound tests and best-first search is approximately linear on the average with respect to the number of processors [WAH81,WAH84b]. However, simulations have revealed that the number of iterations for parallel branch-and-bound algorithms using k processors can be (a) more time than that of a single processor--"*detrimental anomaly*" [IMA79,LAI83,MOH83]; or (b) less than one-k'th of the number of iterations of a single processor--"*acceleration anomaly*" [FUL78,WEI82,LAI83, IMA79]. It is desirable to discover conditions that preserve the acceleration anomalies and eliminate the detrimental anomalies.

Let $T^c(k,\epsilon)$ and $T^d(k,\epsilon)$ denote the number of iterations required for expanding a branch-and-bound tree using centralized and k subproblem lists respectively, where k is the number of processors used, and ϵ is the allowance function. Assuming that subproblems are decomposed synchronously, the number of iterations is measured by the number of times that subproblems are decomposed in each processor. The distinction between centralized and multiple lists lies in the memory configuration. When all the processors are connected to a centralized memory, the subproblem list is global to the processors. When each processor has a private memory, only the local subproblem list can be accessed. The configuration of MANIP (Figure 1) with m−1 shifts in each iteration is equivalent to a system with a centralized subproblem list.

An example of a detrimental anomaly is illustrated in Figure 2. In a serial depth-first search, subtree T_2 is terminated owing to the lower-bound test of P_1': $f(P_1')/(1+\epsilon) \leq g(P_2)$ where $\epsilon = 0.1$. In a parallel depth-first search with two processors, a feasible solution, P_4, that terminates P_1 and P_1' is found first. Consequently, subtree T_2 has to be expanded that will eventually terminate subtree T_3. If the size of T_2 is much larger than the size of T_3, the time it takes to expand T_2 using two processors will be longer than the time it takes to expand T_3 using one processor.

An example of an acceleration anomaly is shown in Figure 3. When a single processor with a depth-first

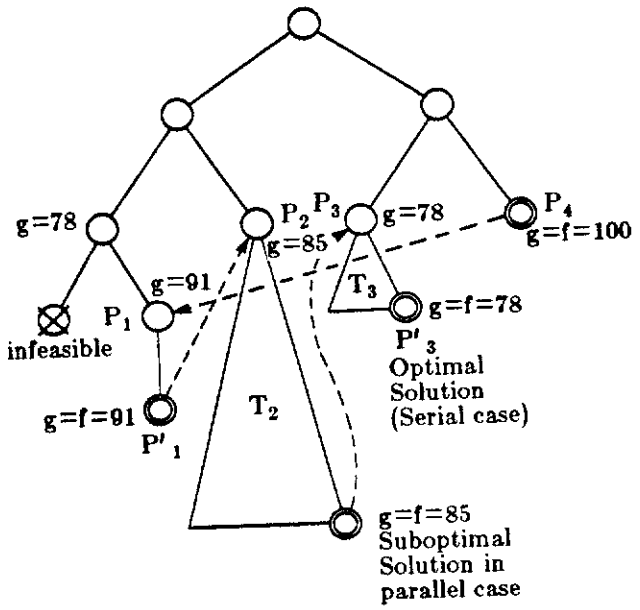

Figure 2. Counterexample for $T^c(2,0.1) \leq T^c(1,0.1)$ using a depth-first search.

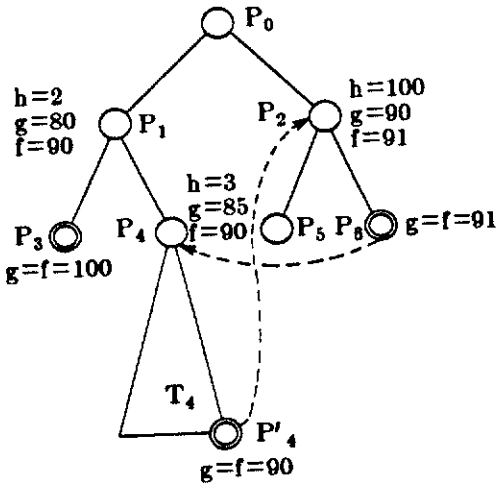

Figure 3. Example of an acceleration anomaly $T^c(1,0.1) > T^c(1,0.1)/2$ using a depth-first search.

search is used, T_4 will be expanded. When two processors are used, P_4 and hence T_4 will be terminated by lower-bound tests. If T_4 is very large, an acceleration anomaly will occur.

The important consideration here is not in knowing that anomalies exist, but in understanding why these anomalies occur. Furthermore, it is desirable to preserve the acceleration anomalies and to avoid the detrimental anomalies. Our objective is to find the sufficient conditions to ensure that $T^c(k,\epsilon) \leq T^c(1,\epsilon)$ and $T^d(k,\epsilon) \leq T^d(1,\epsilon)$, as well as the necessary conditions for $T^c(k,\epsilon) \leq T^c(1,\epsilon)/k$ and $T^d(k,\epsilon) \leq T^d(1,\epsilon)/k$. Note that when there is one processor, only one memory system is used, and $T^c(1,\epsilon) = T^d(1,\epsilon)$.

4.2 Generalized Heuristic Searches to eliminate Detrimental Anomalies in a Single Subproblem List

Recall in Sections 2 and 3 that the selection function uses heuristic values to define the order of node expansions. It was mentioned that breadth-first, depth-first and best-first searches are special cases of heuristic searches. These searches are potentially anomalous when parallel expansions are allowed.

Consider the serial depth-first search. The subproblems are maintained in a last-in-first-out list, and the subproblem with the maximum level number is expanded first. When multiple subproblems have identical level numbers (heuristic values), the node chosen by the selection function depends on the order of insertion into the stack. Suppose the rightmost son is always expanded and inserted first. Then the leftmost son will be the node inserted last and expanded first in the next iteration.

In a parallel depth-first search, the mere extension of the serial algorithm may cause an anomalous behavior. For example, the order of expansion in a serial depth-first search for the tree in Figure 4 is A, B, D, I, J, E, etc.

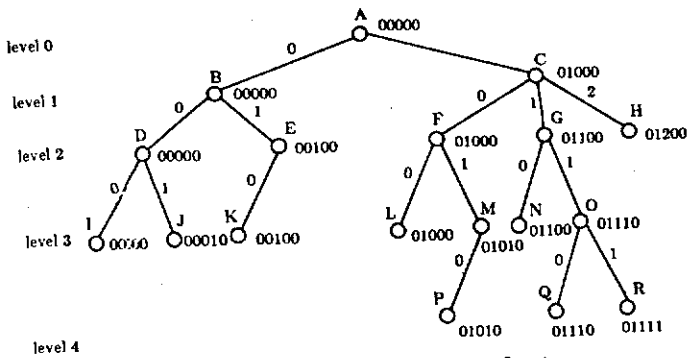

Figure 4. The path numbers of a tree.

When two processors are used, nodes B and C are expanded in the second iteration that result in nodes D, E, F, G and H. Since these nodes have identical level numbers, the search function can choose any two of these nodes for expansion in the next iteration. Suppose the nodes are inserted in the order E, D, H, G and F. Then nodes F and G will be selected and expanded in the third iteration. This may cause unexpected behavior as compared to the serial case and is exactly the reason for the anomalies reported in [LAI83].

To solve this problem, we must define distinct heuristic values for the nodes so that there is no ambiguity on the nodes to be chosen by the selection function. In this paper, a path number is used to uniquely identify a node in a tree. The ***path number*** of a node is a sequence of d+1 integers that represent the path from the root to this node where d is the maximum number of levels of the tree. The path number $E = e_0e_1e_2...e_d$ is defined recursively as follows. The root P_0 exists at level 0 and has a path number of 000...0. A node P_i on level ℓ which is the j-th son (counting from the left) of P_i with path number $E_{P_i} = e_0e_1...e_{\ell-1}000...$ has path number $E_{P_{ij}}$ $= e_0e_1...e_{\ell-1}j00...$. As an example, the path numbers for the nodes in the tree of Figure 4 are shown.

To compare path numbers, the relations ">" and "=" must be defined. A path number $E_1 = e_0^1e_1^1e_2^1 \cdots$ is less than another path number $E_2 = e_0^2e_1^2e_2^2 \cdots$ ($E_1 < E_2$) if there exists $0 \leq j \leq d$ such that $e_i^1 = e_i^2$, $0 \leq i < j$, and $e_j^1 < e_j^2$. The path numbers are equal if $e_i^1 = e_i^2$ for

$0 \le i \le d$. For example, the path number 01000 is less than 01010. Note that nodes can have equal path numbers if they have the ancestor-descendant relationship. Since these nodes never coexist simultaneously in the list of active subproblems, the subproblems in the active list always have distinct path numbers.

The path number is now included in the heuristic function. The primary key is still the lower-bound value or the level number. The secondary or ternary key is the path number and is used break ties in the primary key.

$$h(P_i) = \begin{cases} \text{(level number, path number)} \\ \quad \text{for breadth-first search} \\ \text{(−level number, path number)} \\ \quad \text{for depth-first search} \\ \text{(lower bound, level number, path number) or} \\ \text{(lower bound, −level number, path number)} \\ \quad \text{for best-first search} \end{cases} \quad (12)$$

For a best-first search, two alternatives are defined that search in a breadth-first or depth-first fashion for nodes with identical lower bounds. The heuristic functions defined above belong to a general class of heuristic functions that satisfy the following properties:

(a) $h(P_i) \neq h(P_j)$ if $P_i \neq P_j$ $P_i, P_j \in \mathbf{P}$ (13)
All heuristic values in the active list are distinct.

(b) $h(P_i) < h(P_j)$ if P_j is a descendant of P_i (14)
(heuristic values always increase)

In general, any heuristic function that satisfy Eq's 13 and 14 will not lead to detrimental anomalies. Due to space limitation, the results are stated without proof in the following theorems. The proofs can be found in [LI84].

Theorem 1: Let $\epsilon=0$, i.e., an exact optimal solution is sought. $T^c(k,0) \le T^c(1,0)$ holds for parallel heuristic searches of a single optimal solution in a centralized list using any heuristic function that satisfies Eq's 13 and 14.

When approximations are allowed, detrimental anomalies cannot always be avoided (see Figure 2). The reason for the anomaly is that lower-bound tests under approximation, **L**, are not transitive. That is, $P_i \mathbf{L} P_j$ and $P_j \mathbf{L} P_k$ do not imply $P_i \mathbf{L} P_k$, since $f(P_i)/(1+\epsilon) \le g(P_j)$ and $f(P_j)/(1+\epsilon) \le g(P_k)$ implies $f(P_i)/(1+\epsilon)^2 \le g(P_k)$ rather than $f(P_i)/(1+\epsilon) \le g(P_k)$. Detrimental anomalies can be avoided for best-first or breadth-first searches only.

Theorem 2: $T^c(k,\epsilon) \le T^c(1,\epsilon)$, $\epsilon>0$, holds for parallel best-first or breadth-first searches for a single optimal solution when a heuristic function satisfying Eq's 13 and 14 is used.

Since the lower-bound function is used as the heuristic function in best-first searches, Eq's 13 and 14 are automatically satisfied if all the lower-bound values are distinct. Otherwise, path numbers must be used to break ties in the lower bounds. For breadth-first and depth-first searches, the conditions of Theorem 2 are not sufficient, and the following condition is needed. For any feasible solution P_i, all nodes whose heuristic values are less than $h(P_i)$ cannot be eliminated by the lower-bound test due to P_i, that is, $f(P_i)/(1+\epsilon) \le g(P_j)$ implies that $h(P_i) < h(P_j)$ for any P_j. Generally, this condition is too strong and cannot be satisfied in practice.

4.3 Necessary Conditions to ensure Acceleration Anomalies in a Single Subproblem List

In order to allow the occurrence of acceleration anomalies, it is necessary to expand a node in the parallel case such that a large number of nodes are terminated in the serial case. This is characterized by the complete consistency of heuristic functions. A heuristic function, h, is said to be *not completely consistent* with g if there exist two nodes P_i and P_j such that $h(P_i) > h(P_j)$ and $g(P_i) \le g(P_j)$.

Theorem 3: Let $\epsilon = 0$ and assume that a single optimal solution is sought. The necessary condition for $T^c(k,0) < T^c(1,0)/k$ is that the heuristic function is not completely consistent with g.

The theorem implies that acceleration anomalies may exist and detrimental anomalies can be prevented. It is important to note that the condition in Theorem 3 is not necessary when approximate solutions are sought. An example showing the existence of an acceleration anomaly when h is completely consistent with g is shown in Figure 3. The additional necessary condition is that h is not completely consistent with the lower-bound test with approximation, that is, there exist P_i and P_j such that $h(P_i) > h(P_j)$ and $P_i \mathbf{L} P_j$.

4.4 Multiple Subproblem Lists

When there are multiple subproblem lists, one for each processor, a node with the minimum heuristic value is selected for decomposition from each local list. This node may not belong to the global set of active nodes with the minimum heuristic values, however, the node with the minimum heuristic value will always be expanded by a processor as long as the nodes are selected in a consistent order when there are ties. Since it is easy to maintain the incumbent in a global data register, the behavior of multiple lists is analogous to that of a centralized list. However, the performance of using multiple lists is usually worse than that of a single subproblem list [WAH84b].

4.5 Robustness of Parallel Best-first Searches

Although a parallel best-first search does not always give the best performance, the following performance bounds can be proved [LI84]:

Theorem 4: For a parallel best-first search with k processors, $\epsilon=0$ and $g(P_i) \neq f^*$ if P_i is not an optimal-solution node,

$$\frac{T(1,0)}{k} \le T(k,0) \le \frac{T(1,0)}{k} + \frac{k-1}{k}\ell \quad (15)$$

where ℓ is the maximum number of levels of the branch-and-bound tree to be searched. Since the performance is not affected by using single or multiple subproblem lists, the superscript in T is dropped.

As an example, if $\ell=50$, $T(1,0)=10^6$ (for a typical traveling-salesman problem), and $k=1000$, then $T(1000,0) \le 1049$. This means that almost linear speedup can be attained with 1000 processors.

In this section, we have shown conditions to avoid detrimental anomalies and to preserve acceleration anomalies. The results are summarized in Table 1. These results have to be extended when dominance tests are allowed. Due to space limitations, these results will not be presented here [LI84].

5. VIRTUAL-MEMORY SUPPORT FOR BEST-FIRST SEARCHES

Virtual memory is essential in supporting the large space requirements of best-first searches [WAH82]. In this section, the alternatives for supporting best-first searches on systems with limited main-memory space are discussed. It is found that a direct implementation exhi-

Table 1. Summary of results for the elimination of detrimental anomalies and the preservation of acceleration anomalies in parallel branch-and-bound algorithms.

Allowance function	Subproblem lists	Search strategies	Suff. cond. to eliminate Detrimental Anomaly	Necessary cond. for Acceleration Anomaly
$\epsilon=0$	Single	all	I	II
	multiple		no anomaly	
$\epsilon>0$	single or multiple	breadth-first or best-first	I	exists
		depth-first	anomaly	

Conditions:
I: heuristic function satisfies Eq's 13 and 14.
II: h is not completely consistent with g.
anomaly: the sufficient conditions are impractical.
exists: the necessary conditions are too loose.

bits weak locality (Section 5.1). Modification of the virtual memory to tailor to the characteristics of the algorithm is effective but inflexible (Section 5.2). Lastly, the modification of the search algorithm in order to enhance the amount of locality exhibited is found to be the most viable approach (Sections 5.3).

5.1. Direct Implementation on a Conventional Virtual Memory

A direct implementation uses a list of subproblems and a priority queue[1] of pointers to these subproblems. In each iteration, the subproblem with the minimum lower bound is deleted from the list. This generates new subproblems that are inserted back. The priority queue of pointers is used to maintain the ordering required for a best-first search as restoring a sorted list of pointers is less costly than maintaining a sorted list of subproblems.

As subproblems are not ordered by lower bounds in the list, the subproblem chosen for expansion is equally likely to be in any slot in the list. This implies weak locality, and a direct implementation is inefficient.

5.2. Modified Virtual Memory

The proposed virtual-memory system is depicted in Figure 5 [YU83]. Subproblems in the secondary storage are organized as a B^+-tree[2]. Each leaf of the tree is a page and contains one or more subproblems. Since the non-terminal nodes of the tree are pointer nodes that are much smaller than leaf nodes, a substantial portion of the pointer tree may be kept in main memory. This reduces the number of disk accesses required to access a subproblem in the secondary storage.

The main memory contains a partial list of subproblems and a heap of pointers to subproblems in the partial

[1] The priority queue can be implemented by a heap which is a complete binary tree such that the value of each node is at least as small as that of its descendants. Although hardware implementation such as systolic arrays can be used, the problem of mapping subproblems onto the secondary memory is not solved.

[2] A B-tree of order m is a search tree which is either empty or satisfies the following properties [COM79]: (i) the root node has at least two children; (ii) each node contains at least m keys and m+1 pointers; and (iii) each node contains at most 2m keys and 2m+1 pointers. A B^+-tree is a variant of the B-tree in which all records reside in the leaves. The upper levels are organized as a B-tree and serve as an index to locate a record. The leaf nodes are linked from left to right for easy sequential processing.

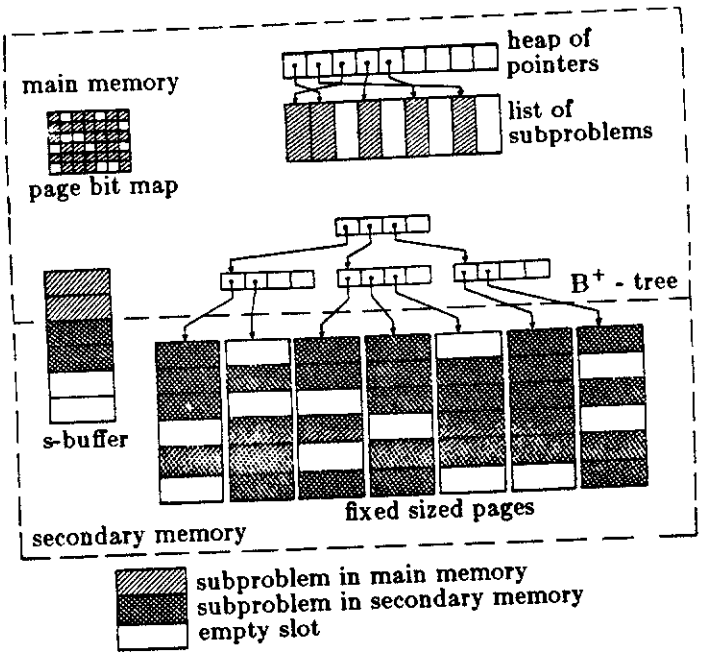

Figure 5. Modified virtual-memory system.

list. Newly generated subproblems are inserted into the memory list. When the main-memory space is exhausted, subproblems are moved to the B^+-tree in the secondary storage. In a best-first search, the subproblem with the smallest lower bound is always expanded. It is prohibitively expensive to compare the smallest subproblem in the memory list with the smallest subproblem in the B^+-tree each time a subproblem is expanded. This can be solved by keeping a portion of the first B^+-tree page in the main memory.

In inserting a subproblem into a normal B^+-tree, the page into which the subproblem is inserted is read into the main memory. The subproblem is inserted into the page image in main memory, and the page image is written back onto disk. A more efficient scheme may be obtained by setting the block size to an integral number of disk sectors greater than or equal to the size of a subproblem and using a bit map in main memory that shows the status of blocks of all the pages. Inserting a subproblem into a page, therefore, consists of searching for an empty block in the bit map and writing the block when the disk head is properly positioned.

The design of the virtual-memory operating system depends on the replacement algorithm and the page size. An efficient replacement algorithm should maximize the number of subproblems inserted into each page and avoid replacing subproblems that will be expanded in the immediate future. Analysis and simulation results have shown that for integer-programming problems, between 70% to 90% of the subproblems in main memory with the largest lower bounds should be replaced each time. For knapsack problems, between 30% to 50% of the subproblems should be replaced. The page size should be chosen to minimize the disk traffic. A suitable page size for both integer-programming and knapsack problems is found to be between 60% to 90% of the maximum size of the subproblem list in main memory.

5.3. Modified Branch-and-Bound Algorithms

One flaw of the modified virtual-memory system is that whenever a page is full, it has to be split into two. This copying of subproblems between pages introduces a

substantial overhead that may be avoided by allowing the logical pages to vary in size.

The range of possible lower bounds is partitioned into a set of disjoint regions, each of which is implemented as a variable-sized stack. An active subproblem generated is pushed into the appropriate stack. Subproblems within a stack are not ordered by lower bounds. Thus, it is prohibitively expensive to access the subproblem with the least lower bound in a stack. Instead, the top of the first non-empty stack is loaded into the main memory and expanded in a depth-first fashion. The effect of this relaxation on the order of expansion is small. Suppose the optimal solution is in stack m. At worst, all the subproblems in stack m have to be expanded before the process terminates.

The important parameter in this scheme is the width of each stack. A simple argument shows that if the optimal-solution value is known, the optimal number of stacks is one. This stack accepts all subproblems with lower bounds less than or equal to the optimal solution. Subproblems with lower bounds greater than the optimal solution are discarded. This stack can be implemented entirely in main memory as a last-in-first-out stack, i.e., a depth-first search is used. In practice, the optimal solution value cannot be estimated accurately beforehand. A second stack must be used which accepts subproblems with lower bounds greater than the estimated value. When the optimal solution resides in the second stack, all subproblems there must be examined. To limit the number of subproblem examined when this happens, the second stack should further be partitioned into smaller stacks. The scheme called *modified depth-first search* implements an approximation of a best-first search and is illustrated in Figure 6.

5.4 Comparison of Various Schemes

In this section, we compare the different virtual-memory-support schemes for branch-and-bound algorithms by deriving the lower-bound estimates on paging overhead and presenting the simulation results on integer-programming problems.

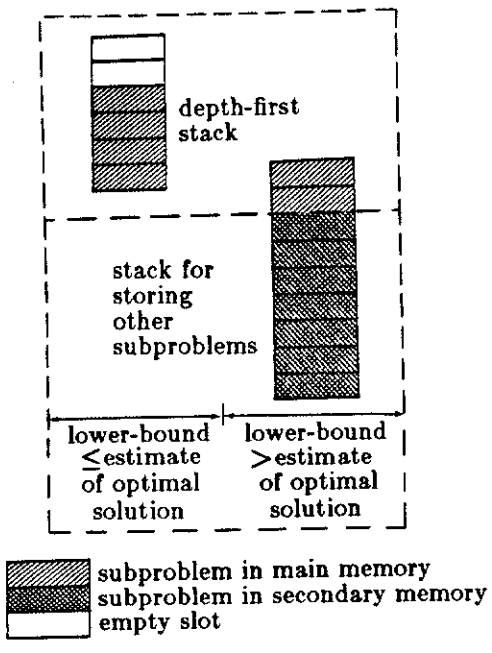

Figure 6. Modified branch-and-bound algorithm with a depth-first search in the first stack.

Let x be the number of subproblems expanded and y be the number of subproblems generated but never expanded (i.e., the number of terminated subproblems). Assume that x and y are the same for all the schemes, and the space required by the active subproblems under a best-first search is much larger than the main-memory size. For the three schemes, let a_i and h_i, $i \in \{DI$ (direct implementation), MVM (modified virtual memory), MDF (modified depth-first search)$\}$, be the page size and the ratio of page size to subproblem size respectively. Further, let the sum of the average seek and rotational delays of a disk be s, and the disk transfer overhead be t seconds per byte.

For direct implementation, assume that the heap of pointers resides in main memory. Let g_{DI} be the probability that a subproblem chosen for expansion resides in the secondary storage. g_{DI} is not one because a subproblem to be expanded may reside in main memory in a space vacated by an expanded subproblem. Therefore, an integral number of pages containing $g_{DI}x$ subproblems are read from disk. On the other hand, $g_{DI}x+y$ subproblems are written to disk. Out of these subproblems, $g_{DI}x$ subproblems are written into slots vacated by subproblems being expanded, thus giving rise $g_{DI}x\lceil 1/h_{DI}\rceil$ page writes. The remaining y subproblems that occupy y/h_{DI} pages must be first read from disk and later written out. The disk traffic is:

$$p_{DI} = 2\left(g_{DI}x\left\lceil\frac{1}{h_{DI}}\right\rceil + \frac{y}{h_{DI}}\right)\left(s+a_{DI}t\right) \qquad (16)$$

For the modified virtual-memory scheme, assume that the page containing the set of subproblems with the smallest lower bounds resides completely in main memory, and let g_{MVM} be the probability that a subproblem being expanded is in this page. Then $(g_{MVM}x+y)$ subproblems are written to disk, and $g_{MVM}x$ subproblems are read from disk. On the average, each B^+-tree page is three-quarters full. The average number of pages read in is $\frac{g_{MVM}x}{((3/4)h_{MVM})}$. Let f_{MVM} be the average number of subproblems inserted into a page during a replacement, and assume that no pages are split in a replacement. The disk traffic is:

$$p_{MVM} = \left(\frac{4g_{MVM}x}{3h_{MVM}} + \frac{g_{MVM}x+y}{f_{MVM}}\right)\left(s+a_{MVM}t\right) \qquad (17)$$

For the modified branch-and-bound algorithm, let the boundary between the first and the second stacks be the value of the optimal solution. Then y subproblems are written to disk. The disk traffic is:

$$p_{MDF} = \frac{y}{h_{MDF}}\left(s+a_{MDF}t\right) \qquad (18)$$

When all the h_i's and a_i's, $i \in \{DI, MVM, MDF\}$, are equal, a comparison of the different equations shows that the modified depth-first search has the best lower-bound performance. p_{MDF} is always smaller than p_{MVM} as f_{MVM} must be smaller than $h_{MVM}/2$ in order to avoid a page split. Likewise, p_{MDF} is always smaller than p_{DI} because g_{DI} is non-zero.

Simulation results on 17-variable 17-constraint integer-programming problems are shown in Figure 7. Each subproblem requires a memory size of 1476 bytes. A main-memory size of 128 Kbytes (10% of the virtual space) is assumed. Cases with small and large pages are shown, and the improvement is smaller for small pages. Only the results with the optimal page size is shown for

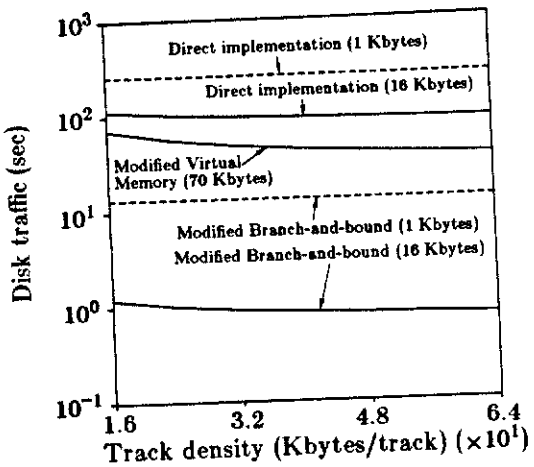

Figure 7. Simulation results for various virtual-memory schemes to support branch-and-bound algorithms on 17 by 17 integer-programming problems (main memory size is 128 Kbytes, page size is shown on each curve).

the modified virtual-memory scheme. With large pages, there is a 100-times reduction of paging overhead for the modified depth-first search as compared to a direct implementation, and a 60-times reduction as compared to a modified virtual-memory scheme. The improvement is within 20 times for small pages. From these results, we can also infer that when the memory space required by a subproblem is small (such as knapsack problems), the improvement will be even better.

6. CONCLUSION

In this paper, we have shown some research results on the parallel execution and virtual-memory support of (approximate) branch-and-bound algorithms. Sufficient conditions are proposed so that parallel execution of branch-and-bound algorithms under lower-bound tests always results in reduction in execution time. It is shown that the improvement for parallel execution can be greater than the number of processors under some very loose necessary conditions. A best-first search is found to be a good search strategy that can guarantee little degradation in performance with increasing degrees of parallelism. We have also presented alternatives for the virtual-memory support. It is found that by relaxing the requirements of branch-and-bound execution, over 100-times reduction in paging overhead can be attained for integer-programming problems as compared to a direct implementation on a conventional virtual-memory system.

REFERENCES

[COM79] Comar, D., "The Ubiquitous B-tree," *Computing Surveys*, Vol. 11, No. 2, pp. 121-137, June 1979.

[DES78] Desai, B. C. "The BPU, A Staged Parallel Processing System to Solve the Zero-One Problem", *Proc. of ICS'78*, Taipei, Taiwan, pp. 802-817, Dec. 1978.

[ELD80] El-Dessouki and W. H. Huen, "Distributed Enumeration on Network Computers," *IEEE Trans. on Computers*, Vol. C-29, No. 9, pp. 818-825, Sept. 1980.

[FUL78] Fuller, S.H. et al, "Multi-Microprocessors: A Overview and Working Example," *Proc. of IEEE*, Vol. 66, No. 2, pp. 216-228, Feb. 1978.

[GAR79] Garey, M. R., and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[IBA76a] Ibaraki, T., "Computational Efficiency of Approximate Branch-and-Bound Algorithms," *Math. of Oper. Research*, Vol. 1, No. 3, pp. 287-298, 1976.

[IBA76b] Ibaraki, T., "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," *Int'l Jr. of Comp. and Info. Sci.*, Vol. 5, No. 4, pp. 315-344, 1976.

[IBA77] Ibaraki, T., "The Power of Dominance Relations in Branch-and-Bound Algorithms," *JACM*, Vol. 24, No. 2, pp. 264-279, 1977.

[IMA79] Imai, M., T. Fukumura and Y. Yoshida, "A Parallelized Branch-and-Bound Algorithm Implementation and Efficiency," *Systems, Computers, Controls*, Vol. 10, No. 3, pp. 62-70, 1979.

[KOH74] Kohler, W. and K. Steiglitz, "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems," *JACM*, Vol. 21, No. 1, pp. 140-156, 1974.

[LAI83] Lai, T.H. and S. Sahni, "Anomalies in Parallel Branch-and-Bound Algorithms," *Proc. 1983 Int'l Conf. on Parallel Processing*, pp. 183-190, Aug. 1983.

[LAW66] Lawler, E. L., and D. W. Wood, "Branch-and-Bound Methods: A Survey," *Operations Research*, Vol. 14, pp. 699-719, 1966.

[LI84] Li, G.-J., and B. W. Wah, "Computational Efficiency of Parallel Approximate Branch-and-Bound Algorithms," Technical Report TR-84-6, School of Electrical Engineering, Purdue University, February 1984.

[MOH83] Mohan, J., "Experience with Two Parallel Programs Solving the Traveling-Salesman Problem," *Proc. of the 1983 Int'l Conf. on Parallel Processing*, pp. 191-193, 1983.

[NIL80] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, California, 1980.

[SAH77] Sahni, S., "General Techniques for Combinational Approximation," *Operations Research*, Vol. 25, No. 6, pp. 920-936, 1977.

[WAH81] Wah, B. W. and Y. W. Ma, "MANIP - A Parallel Computer System for Implementing Branch-and-Bound Algorithms," *Proc. 8th Annual Symposium on Computer Architecture*, pp. 239-262, 1982.

[WAH82] Wah, B. W., and C. F. Yu, "Probabilistic Modeling of Branch-and-Bound Algorithms," *Proc. COMPSAC*, pp. 647-653, Nov. 1982.

[WAH84a] Wah, B. W. and K.-L. Chen, "A Partitioning Approach to the Design of Selection Networks," *IEEE Transactions on Computers*, Vol. C-33, No. 3, March 1984.

[WAH84b] Wah, B. W. and Y. W. Ma, "MANIP - A Multicomputer Architecture for solving Combinatorial Extremum-Search Problems," *IEEE Trans. on Computers*, Vol. C-33, No. 5, May 1984.

[WEI82] Weide, B. W., "Modeling Unusual Behavior of Parallel Algorithms," *IEEE Trans. on Comp.* Vol. C-31, No. 11, pp. 1126-1130, Nov. 1982.

[YU83] Yu, C. F. and B. W. Wah, "Virtual-Memory Support for Branch-and-Bound Algorithms," *Proc. of COMPSAC*, pp. 618-626, Nov. 1983.