EFFICIENT BRANCH-AND-BOUND ALGORITHMS ON A TWO-LEVEL MEMORY SYSTEM

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

Abstract:
In this paper we have investigated the efficient evaluation of branch-and-bound algorithms with a best-first search on a two-level memory hierarchy. Three alternatives are compared here. First, the algorithm is studied with respect to a conventional virtual memory with a LRU replacement algorithm. Second, a specialized virtual memory is designed to execute the algorithm exactly. Lastly, the requirements of the algorithm are relaxed to enhance the locality of accesses, and the data structure is implemented as multiple stacks. It is found that there is over a 100-time reduction in disk overhead for integer-programming problems with the last approach as compared to a direct implementation on a conventional virtual memory; and there is a 60-time improvement over the second approach. The improvement is related to the memory size for storing a subproblem. For problems that have smaller subproblems, the improvement is even greater.

KEYWORDS AND PHRASES: Branch-and-bound, best-first search, two-level memory hierarchy, virtual-memory, disk overhead, modified branch-and-bound.

1. INTRODUCTION

The search for solutions in a combinatorially large problem space is important in artificial intelligence and operations research. Search problems may be classified as either decision or optimization problems. In a decision problem one attempts to determine the existence of at least one solution that satisfies a given set of constraints. Examples include theorem-proving, expert systems and some permutation problems. An optimization problem is characterized by an objective function to be minimized or maximized and a set of constraints to be satisfied. Examples include the traveling-salesman, warehouse-location, job-shop-scheduling, knapsack, vertex-cover and integer-programming problems.

A general technique for solving combinatorial-search problems is the branch-and-bound (B&B) algorithm [LAW66]. A B&B algorithm is a partitioning algorithm that decomposes a problem into smaller subproblems and repeatedly decomposes until infeasibility is proved or a solution is found. Many theoretical properties of serial B&B algorithms have been developed [NIL80, KOH74, IBA76a, IBA76b, IBA77, IBA78]. It has also been recognized that the B&B algorithm is a generalization of many heuristic search procedures [KUM83] such as A* [NIL71], AO* [MAR78], SSS* [STO83], B* [BER79], alpha-beta [KNU75] and dynamic programming [DRE77].

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

B&B algorithms are characterized by four constituents: branching rule(s), selection rule(s), elimination rule(s) and termination condition(s). 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.

In a B&B algorithm let P_i be a subproblem, and let $f(P_i)$ be the value of the best solution obtained by evaluating all the subproblems decomposable from P_i. 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;
(b) g is exact when P_i is feasible;
(c) lower bounds of descendant nodes always increase.

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, P_i, exceeds the value of the incumbent, P_i can be pruned because it will not lead to an optimal solution. The decomposition process continues until all subproblems are either expanded or eliminated. In other words, P_i is terminated during the computation if:

$$g(P_i) \geq z \tag{1}$$

The above elimination rule for obtaining an exact optimal solution can be relaxed in order to obtain a suboptimal solution with a 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 time needed 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:

$$g(P_i) \geq z - \epsilon(z). \tag{2}$$

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]:

$$z_F - \epsilon(z_F) \leq z_O \leq z_F \tag{3}$$

Examples of often used allowance functions are:

$$\epsilon(z) = \epsilon \geq 0 \text{ (absolute error deviation) and} \tag{4}$$

$$\epsilon(z) = \frac{\epsilon z}{1+\epsilon} \quad \epsilon \geq 0,\ z \geq 0 \text{ (relative error deviation).} \tag{5}$$

The selection rule 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***. The B&B algorithm may be summarized as:

```
incumbent = 0;
list of subproblems = {P_0};
while list of subproblems is not empty do [
    apply selection rule to list of subproblems;
    expand selected subproblem by the branching rule;
    for P ∈ set of son subproblems generated [
        if subproblem P is a feasible solution then [
            update incumbent;
            apply elimination rule ]
        else if value < z−ε(z) then
            insert subproblem P into the list of subproblems
    ]
]
```

To illustrate the B&B algorithm, the evaluation of an integer-programming problem [MIT70] is shown here. Integer-programming problems may be expressed as a constrained optimization:

$$\begin{array}{ll} \text{Minimize} & CX \\ \text{subject to} & AX \geq B \\ & X^T = (x_1, x_2, \ldots, x_n) \\ & x_i\text{: non-negative integer, } i=1,2,\ldots n; \\ & A, B, C \text{ are constant matrices.} \end{array} \qquad (6)$$

These problems differ from ordinary linear-programming problems in that the variables are restricted to non-negative integer values.

One approach to the problem is the following. Apply the dual simplex method to a subproblem and solve it as a linear program. If the optimal solution is integral, a feasible solution has been generated; otherwise, create two new subproblems as follows. Choose a variable that has a non-integral value (say $x_i = 4.4$) and restrict that variable to the next lower integral value for one problem ($x_i \leq \lfloor 4.4 \rfloor$ or $x_i \leq 4$) and to the next higher integral value ($x_i \geq \lceil 4.4 \rceil$ or $x_i \geq 5$) for the other. The variable chosen is the one with the greatest up or down penalty. The up penalty for a variable x_i having a value of a_i is the estimate of the amount by which the solution to the current subproblem would increase if the integral constraint $x_i \geq \lceil a_i \rceil$ was introduced. The down penalty is similar, except that it is associated with the constraint $x_i \leq \lfloor a_i \rfloor$. The lower bound of a new subproblem is the sum of the optimal simplex solution and the associated penalty. This entire process is repeated on the new subproblems.

Figure 1(b) shows the B&B tree for the problem in Figure 1(a). The dual simplex method gives an optimal solution of 14.2 for the original problem. Since the variables are not integral, a feasible solution has not been generated. Up and down penalties are calculated for the variables. x_1 has the greatest penalty ($U = 1.8$). Two new subproblems are then created, one with $x_1=0$, and the other with $x_1 \geq 1$. The lower bounds are calculated as shown in Figure 1(b). The dual simplex method is

$$\min x_0 = 7x_1 + 3x_2 + 4x_3$$

$$x_1 + 2x_2 + 3x_3 \geq 8$$

$$3x_1 + 2x_2 + x_3 \geq 5$$

$$x_1, x_2, x_3 \geq 0, \text{ integer}$$

(a)

Optimal dual simplex solution

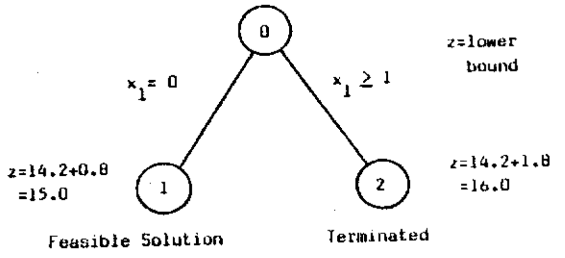
$x_0 = 14.2$

$x_1 = 0.4$

$x_2 = 3.8$

$x_3 = 0$

Variable	Down Penalty	Up Penalty
x_1	0.8	1.8
x_2	0.3	0.13



Optimal dual simplex solution

$x_0 = 15.0$

$x_1 = 0$

$x_2 = 5$

$x_3 = 0$

(b)

Figure 1 (a) An example of an integer-programming problem and (b) the corresponding B&B solution

applied again to the subproblem with the smallest lower bound, and a feasible solution is generated with all variables having integral values. This constitutes an optimal solution since the lower bound of the remaining subproblem is greater.

It has been shown that a best-first search has a predictable and the best time efficiency if $g(P_i) \neq f^*$ when P_i is not an optimal-solution node and f^* is the optimal-solution value [LI84]. This result has been derived in conjunction with a computer model that consists of a CPU and a main memory. In many computers the main memory is limited in size and has to be extended in size by a secondary memory. Although the best-first search requires the minimum execution time, it needs an exponential amount of memory space and may use the secondary memory for extended storage. In this case there is a mismatch in access efficiency and hence a degradation in performance.

In a depth-first search the list of subproblems is stored in a last-in/first-out stack. The size of this stack is the maximum depth of the search tree. For many optimization problems, this is equal to the number of variables in the problem and is reasonably small. A depth-first search can, thus, be run entirely in main memory and avoid the delays of the secondary storage. However, the saving in time may be offset by the increased number of subproblems to be examined. A breadth-first search has a similar behavior. On the other hand, a best-first search requires a large memory space and is generally efficient in the number of subproblems expanded. Hence if an efficient memory management system is available so that all the secondary-storage accesses are overlapped with the computations, a best-first search will be more efficient.

Several researchers have addressed the problem of improving the performance of non-search algorithms in a computer with a two-level memory hierarchy managed by a paged virtual-memory system. These include rules for enhancing locality [MOR66], automatic restructuring of programs [BAE76,HAT71,FER74,FER75,FER76], matrix manipulation [MCK69,FIS79], sorting [BRO70,ALA84] and database operations [SMI78].

In this paper we have studied the efficient evaluation of best-first searches in a computer model with a two-level memory hierarchy that comprises of a CPU, a main memory, a slower secondary memory and the corresponding secondary-memory controller. The CPU and the secondary-memory controller overlap in processing. The time requirement of an algorithm is measured by the expected completion time on the two-level memory-computer model when the search process is run in a single-user mode.

2. DIRECT IMPLEMENTATION OF BEST-FIRST SEARCHES

One implementation of the best-first search 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 since 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 anywhere in the list. There will be a wait if the chosen subproblem is in the secondary memory. On the other hand, once the main memory is full, subproblems must be moved to the secondary storage before the process can continue. This introduces another wait. To avoid these waits, we have investigated two approaches to support best-first searches: modification of the virtual memory that manages the distribution of subproblems between the main memory and the secondary memory, and modification of the search algorithm. In this section we present the implementation of the algorithm in a modified virtual memory.

The proposed virtual-memory system is depicted in Figure 2. Subproblems in the secondary storage are

[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.

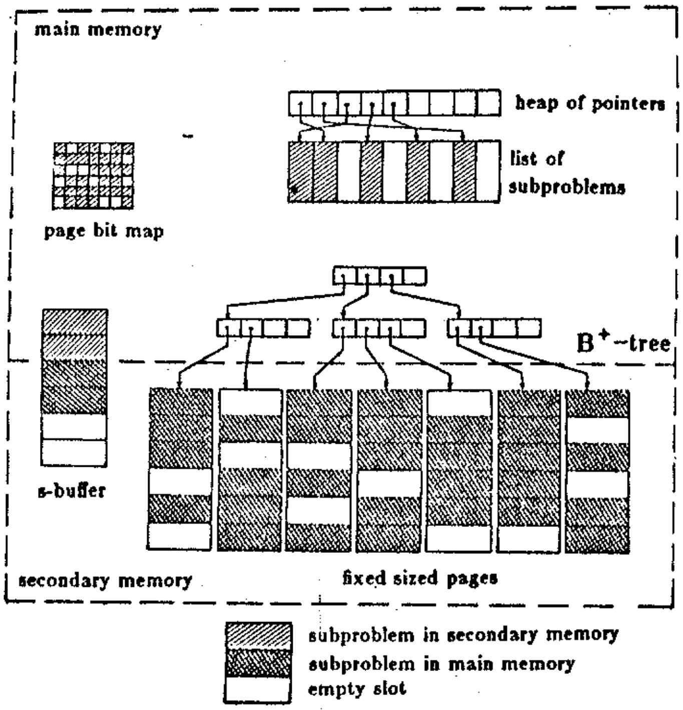


Figure 2 Modified virtual-memory system

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 the 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 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 subproblem with the smallest lower bound in the memory list with the subproblem having the smallest lower bound 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 is to set the block size to an integral number of disk sectors that is greater than or equal to the size of a subproblem and to

[2] A B-tree of order m is a search tree that 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.

use a bit map in main memory to show 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 [YU83]. 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.

3. MODIFIED BEST-FIRST B&B ALGORITHM

Although effective, modification of the virtual-memory system is problem dependent and is thus inflexible. On the other hand, the modification of the search strategy is easier to be managed by the designers. In this section we describe a modified B&B algorithm that improves the locality of access to the secondary storage without sacrificing the performance.

3.1. Description of the Algorithm

Let $g(P_0)$ be the lower bound of the initial problem. For any subproblem P_i, let $\delta(P_i) = g(P_i) - g(P_0)$. As depicted in Figure 3, the range of possible δs is partitioned into n disjoint regions, $\{[u_0,u_1), [u_1,u_2), \ldots, [u_{n-1},u_n)\}$ where $u_0 = 0$ and $u_n = \infty$. Each region $[u_i,u_{i+1})$, $0 \le i \le n-1$, is implemented as a stack. The list in each stack is usually large and may have to be stored partially in the secondary memory. It is thus inefficient to access a subproblem with the smallest lower bound in a list due to high overhead for secondary storage accesses. To solve this problem, subproblems within a list are not ordered by lower bounds and are expanded in a depth-first manner. To minimize the access overhead, the top portion of each stack resides in a buffer in the main memory while the remaining part resides in the secondary memory. To increase the overlap between the main-memory and secondary-storage accesses, there are disk buffers for storing temporary data.

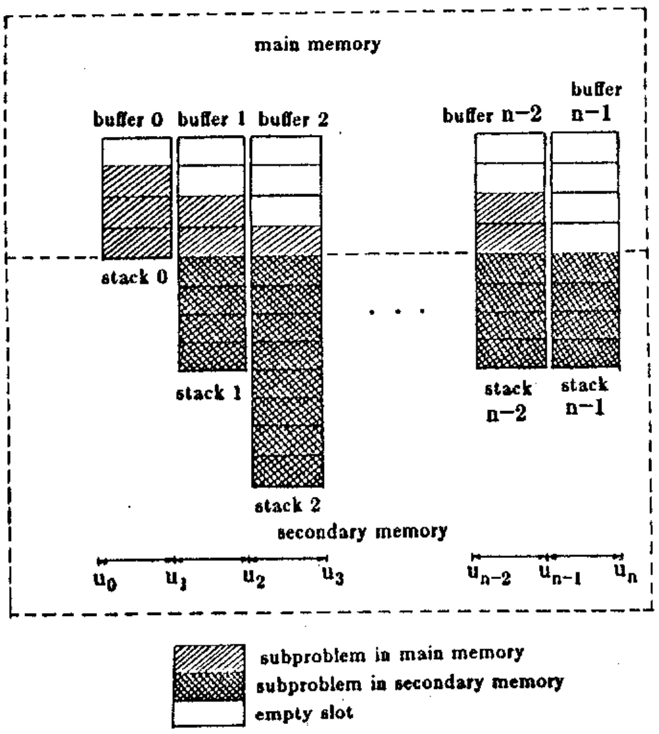


Figure 3 Modified B&B algorithm

The *modified selection rule* chooses the subproblem on top of the non-empty buffer with the smallest index for expansion. This is the buffer that contains the subproblem in main memory with the smallest lower bound. Since subproblems within a list are not sorted, the *lower-bound elimination rule* also has to be modified. Let the value of the incumbent be z and the allowance function be $\epsilon(z)$. Suppose that $z - \epsilon(z)$ lies in the range of stack k, then all stacks with indices greater than k are eliminated. There may be subproblems in stack k whose lower bounds are greater than $z - \epsilon(z)$ but are not eliminated immediately. These subproblems will be eliminated when they are moved to the main memory during the expansion of list k. As a result, it will be necessary to carry out the lower-bound test on each selected subproblem before it is expanded. The modified B&B algorithm is summarized as follows:

```
incumbent = 0;
list of subproblems = {P_0};
while number of subproblems > 0 do [
    let k be the smallest index of the non-empty lists;
    if buffer k is empty and no load request is outstanding then
        initiate load from stack k into buffer k;
    if all buffers are empty then
        wait until load from stack k into buffer k is completed;
    apply modified selection rule;
    if lower bound of selected subproblem < z−ε(z) then [
        if all buffers are full then
            wait until subproblems stored;
        expand selected subproblem;
        if expanded subproblem is a feasible solution then [
            update incumbent;
            apply modified elimination rule ]
        else if value < z−ε(z) then
            push subproblems into corresponding stacks;
        for i ∈ non-empty buffer with index > k
            if buffer i is full
                then initiate store from buffer i to stack i
    ]
]
```

More subproblems may be expanded in the modified B&B algorithm than in a direct implementation. Let the value of the optimal solution falls within the range of stack k. Since subproblems are not ordered by lower bounds in a stack, subproblems in stack k with lower bounds greater than the value of the optimal solution may be expanded if they are smaller than the value of the incumbent. Some subproblems from stacks with indices greater than k may also be expanded due to the modified selection rule. These subproblems are never expanded in a best-first search.

The increase in the number of subproblems expanded depends on the assignment of boundaries between adjacent stacks. Suppose we have an estimate of the optimal solution, v_E, a lower limit, a_L, and an upper limit, a_U, on the accuracy of the estimate. A simple argument shows that there should be at least three stacks. When three stacks, i = 0, 1, 2, are used, the stack boundaries should be $u_0 = 0$, $u_1 = v_E/a_U - \epsilon(v_E/a_U)$, $u_2 = v_E/a_L - \epsilon(v_E/a_L)$ and $u_3 = \infty$. The optimal solution resides in Stack 1, and all subproblems in it must be examined. To limit the number of subproblems examined when this happens, Stack 1 should further be partitioned into smaller stacks. In the next section we derive an analytical procedure for determining the boundaries of stacks that will minimize the average number of subproblems examined for any given number of stacks. This will also serve as a guide in determining the number of stacks to be used. We will only consider the case when te allowance function $\epsilon(z)$ is zero. The results for the case when $\epsilon(z)$ is not zero can be obtained by replacing the terms v_E/a_U and v_E/a_L in the derived equations by $v_E/a_U - \epsilon(v_E/a_U)$ and $v_E/a_L - \epsilon(v_E/a_L)$ respectively.

3.2 Analysis of Stack Boundaries

Let the main memory allocated to the modified B&B algorithm be n_p, and the size of a buffer be n_b. Then the number of buffers (and hence the number of stacks), n, will be $\lfloor n_p/n_b \rfloor$. As a simplification, when a stack is terminated, its buffer is not reallocated to another stack. When a buffer is full, its contents are copied to a disk buffer of size n_b and subsequently transferred to the stack on disk as a single disk cluster. It is assumed that there is always a free disk buffer.

Let v_A be the value of the optimal solution. Assume that v_A is uniformly distributed over the range $(v_E/a_U, v_E/a_L)$. Let $(u_i | v_A = v)$, $0 \leq i \leq n$, be the optimal assignment of u_i when $v_A = v$, and let $(w_i | v_A = v) = (u_{i+1} | v_A = v) - (u_i | v_A = v)$ be the width of stack i. Suppose that v falls within the range of stack k. Let M_i be the total number of subproblems written to stack i in the secondary storage for the duration of the B&B process. By assuming that:

(i) the increase in lower bound when a subproblem is expanded is a random variable that is exponentially distributed with parameter λ (the distribution has been verified experimentally to be gamma with $\alpha < 1$ for integer-programming and knapsack problems [WAH82]; the exponential distribution is used here as an approximation),
(ii) each subproblem expansion generates s subproblems,
(iii) all expanded subproblems are independent of each other,

the expected value of M_i given that $v_A = v$ has been shown in Appendix A (Eq. (A-12)) to be equal to:

$$E[M_i | v_A = v] \simeq \begin{cases} \left(1 - e^{-\lambda(w_i | v_A = v)}\right) s\, e^{\lambda(s-1)(u_i | v_A = v)} & \text{for } 1 \leq i \leq k \\ \left(1 - e^{-\lambda(w_i | v_A = v)}\right) s\, e^{-\lambda(u_i | v_A = v)}\, e^{\lambda s v} & \text{for } n-1 \geq i > k \end{cases} \tag{7}$$

The boundaries u_i, i=1, ..., n-2, should be chosen so that the disk overhead for each stack between $(v_E/a_U, v_E/a_L)$ when the process terminates is approximately the same. This will minimize the expected number of subproblems expanded when v_A is uniformly distributed over the range $(v_E/a_U, v_E/a_L)$. A subproblem in stack i, $i \leq k$, undergoes two disk transfers since the subproblem has to be written from buffer i into stack i and later read from stack i into buffer i. A subproblem in stack i, i >k, undergoes only one disk transfer since the subproblem is never read from stack i into buffer i. The disk overhead for the stacks will be the same if:

$$\begin{aligned} E[M_1 | v_A = v] &= \cdots = E[M_k | v_A = v] \\ &= 2E[M_{k+1} | v_A = v] = \ldots = 2E[M_{n-1} | v_A = v] \end{aligned} \tag{8}$$

This will be satisfied if the assigned boundaries satisfy:

$$\begin{aligned} \frac{1 - e^{-\lambda(w_{i+1} | v_A = v)}}{1 - e^{-\lambda(w_i | v_A = v)}} &= e^{\lambda(s-1)(w_{i+1} | v_A = v)} \\ &\quad \text{for } i \leq k \\ \frac{1 - e^{-\lambda(w_{i+1} | v_A = v)}}{1 - e^{-\lambda(w_i | v_A = v)}} &= 2\, e^{-\lambda(w_{i+1} | v_A = v)}\, e^{\lambda s(v - (u_i | v_A = v))} \\ &\quad \text{for } i = k+1 \\ \frac{1 - e^{-\lambda(w_{i+1} | v_A = v)}}{1 - e^{-\lambda(w_i | v_A = v)}} &= e^{-\lambda(w_{i+1} | v_A = v)} \\ &\quad \text{for } i \geq k+2 \end{aligned} \tag{9}$$

In addition, since $u_1 = v_E/a_U$, $u_{n-1} = v_E/a_L$, we have:

$$\sum_{i=1}^{n-1} (w_i | v_A = v) = \frac{v_E}{a_L} - \frac{v_E}{a_U} \tag{10}$$

An iterative scheme may be used to compute the values of $(w_i | v_A = v)$, i = 1, ..., n−2, and $(u_i | v_A = v)$, i = 2, ..., n−2, that will satisfy Eq's (9) and (10).

Given the values of $(u_i | v_A = v)$, i = 2, ..., n-2, the assumption on the distribution of v_A can then be used to determine the values of u_i.

$$u_i = \int_{v = v_E/a_U}^{v_E/a_L} (u_i | v_A = v) \frac{1}{(v_E/a_L - v_E/a_U)}\, dv \tag{11}$$

It is not possible to obtain a closed form for Eq. (11). Instead, a discrete approximation may be used in which v_A is chosen from $v_i = \frac{v_E}{a_U} + \frac{i\, v_E}{M}\left(\frac{1}{a_L} - \frac{1}{a_U}\right)$, i = 1, ..., M, with equal probability. Then,

$$u_i = \sum_{i=1}^{M} (u_i | v_A = v_i) \Pr(v_A = v_i) = \frac{1}{M} \sum_{i=1}^{M} (u_i | v_A = v_i) \tag{12}$$

The expected disk overhead due to the assignment $\{u_1, u_2, \ldots, u_{n-1}\}$ may be evaluated. Let $E[T_D | v_A = v]$ be the expected disk overhead given that $v_A = v$. Suppose that $u_k \leq v \leq u_{k+1}$. Let $E[P_i | v_A = v]$, i = 2, ..., n-1, be the number of disk clusters that make up stack i.

$$E[P_i | v_A = v] = \left\lfloor \frac{E[M_i | v_A = v]}{n_b} \right\rfloor \tag{13}$$

Let the average seek time of the disk be t_s, the average rotational latency be t_r, and the average transfer time be t_t. Then the average disk overhead to transfer one buffer of subproblems, t_C, is given by:

$$t_C = t_s + t_r + t_t \times n_b \tag{14}$$

and

$$\begin{aligned} & E[T_D | v_A = v] \\ & = \sum_{i=0}^{k} 2E[P_i | v_A = v]\, t_C + \sum_{i=k+1}^{n} E[P_i | v_A = v]\, t_C \end{aligned} \tag{15}$$

The assumption on the distribution of v_A can then be used to determine the expected disk overhead based on the assignment $\{u_1, \ldots, u_{n-1}\}$.

$$E[T_D] = \int_{v=v_E/a_U}^{v_E/a_L} E[T_D \mid v_A=v] \frac{1}{(v_E/a_L - v_E/a_U)} dv \qquad (16)$$

which may be approximated by:

$$E[T_D] = \sum_{i=1}^{M} E[T_D \mid v_A=v_i] \Pr(v_A=v_i)$$

$$= \frac{1}{M} \sum_{i=1}^{M} E[T_D \mid v_A=v_i] \qquad (17)$$

By computing the expected disk overhead $E[T_D]$ for different values of n, it will be possible to determine the value of n that should be used.

3.2 Analysis of Stack Boundaries for Discrete Valued B&B

A similar analysis may be carried out for discrete valued B&B algorithms (say for vertex-cover problems). It is assumed that the difference in lower bounds between a son sub-problem and its parent, J, is an independent negative binomial random variable with distribution:

$$\Pr(J=k) = f(k;r,p) \qquad (18)$$

$$= \binom{r+k-1}{k} p^r (1-p)^k \qquad k = 0,1,\ldots$$

This has been verified for the vertex-cover problems. The other assumptions made in Section 3.2 remain unchanged.

Let N(d) be the number of expanded subproblems that have lower bounds less than or equal to $g(P_0) + d$. Utilizing the boundary condition $N(d_0) = m_0$, it can be shown that:

$$E[N(d)] = \frac{m_0(s-1)+1}{s-1} \left(\frac{1-p}{1-ps^{1/r}} \right)^{(d-d_0)} - \frac{1}{s-1} \qquad (19)$$

$$\text{for } d = d_0, d_0+1, \ldots$$

For any $h \le d$, the number of expanded subproblems with lower bound h is $Q(h)=N(h)-N(h-1)$, and its expectation, $E[Q(h)]$, is given by:

$$E[Q(h)] = \frac{m_0(s-1)+1}{s-1} \left(\frac{1-p}{1-ps^{1/r}} \right)^{h-1} \frac{p(s^{1/r}-1)}{1-ps^{1/r}} \qquad (20)$$

Subproblem P_k, with $u_{i-1} \le \delta(P_k) < u_i$, will be written to stack i. As in Section 3.2, let M_i be the total number of subproblems written to stack i in the secondary storage prior to the termination of the B&B process. By assuming that the expanded subproblems are independent and the optimal solution is $v_A=v$, $E[M_i \mid v_A=v]$ can be approximated by:

$$E[M_i \mid v_A=v] \simeq \sum_{h=d_0}^{\min(v,u_{i-1})-1} E[Q(h)] \left(\sum_{j=u_{i-1}-h}^{u_i-h-1} f(j;r,p) \right) \qquad (21)$$

Eq. (21) is an approximation as it ignores the first m_0 expanded subproblems. The stack boundaries should be assigned such that:

$$E[M_1 \mid v_A=v] \simeq \ldots \simeq E[M_k \mid v_A=v] \qquad (22)$$

$$\simeq 2E[M_{k+1} \mid v_A=v] \simeq \ldots \simeq 2E[M_{n-1} \mid v_A=v]$$

Exact equality is not possible due to the discrete nature of the problem. An iterative scheme is used to solve for the the above equation subject to the constraint of Eq. (10). The boundaries can then be computed by:

$$u_i = \frac{1}{v_E/a_L - v_E/a_U + 1} \sum_{v=v_E/a_U}^{v_E/a_L} E[u_i \mid v_A=v] \qquad (23)$$

As in Section 3.2, the expected disk overhead when $v_A=v$, $E[T_D \mid v_A=v]$, may be computed for $v = v_E/a_U, v_E/a_U + 1, \ldots, v_E/a_L$, and the expected disk overhead, $E[T_D]$ is:

$$E[T_D] = \frac{1}{v_E/a_U - v_E/a_L + 1} \sum_{v=v_E/a_U}^{v_E/a_L} E[T_D \mid v_A=v] \qquad (24)$$

3.4 Simulation Results

We have carried out simulations for the modified B&B algorithm with integer-programming and vertex-cover problems. The effectiveness of the schemes developed in Sections 3.2 and 3.3 may be evaluated by observing M_i, the actual number of subproblems written to stack i in the secondary storage when the B&B process terminates. When n stacks are used and stack k contains the optimal solution, the values of M_i should satisfy:

$$M_1 \simeq \cdots M_k \simeq 2M_{k+1} \simeq \cdots \simeq 2M_{n-2} \qquad (25)$$

Some results are shown in Table 1 for integer-programming problems and in Table 2 for vertex-cover problems. We have presented these results in the form of the mean, standard, maximum and minimum deviations of the set of values $M_1, \ldots, M_k, 2M_{k+1}, \ldots, 2M_{n-2}$. It can be seen that the deviations from the mean are reasonable. In contrast, if a simple scheme such as equal-width stacks had been used, the values would show an exponential increase.

Our simulations suggest that for both the integer-programming and vertex-cover problems, three stacks should be used to minimize the disk overhead. This can be clearly seen from Table 3(a) that shows the variation of disk overhead with the number of stacks for five vertex-cover problems. The disk overhead calculated is based upon a disk with an average delay of 36.333 msec

Table 1. Statistical measures for the number of subproblems written to the stacks for five 20-variables, 20-constraints integer programming problems.

Problem	Measure	4 stacks	5 stacks	6 stacks	7 stacks	8 stacks	9 stacks
1	mean	579.0	424.0	312.3	240.0	194.2	167.0
	std dev	248.0	75.6	50.3	67.3	74.1	74.4
	max dev	248.0	98.0	75.7	88.0	136.8	164.0
	min dev	248.0	12.0	14.7	23.0	3.167	17.0
2	mean	423.5	302.0	227.5	177.6	147.2	126.3
	std dev	60.5	45.1	79.5	93.8	97.3	97.2
	max dev	60.5	58.0	130.5	181.4	211.8	232.7
	min dev	60.5	6.0	2.5	0.6	5.2	7.3
3	mean	216.0	144.3	110.5	84.6	64.0	55.0
	std dev	89.0	16.8	14.2	27.7	27.9	27.2
	max dev	89.0	23.3	23.5	35.4	38.0	37.0
	min dev	89.0	7.7	2.5	10.6	6.0	9.0
4	mean	482.0	325.7	240.0	193.2	171.0	231.7
	std dev	304.0	112.8	64.6	45.5	46.1	51.2
	max dev	304.0	148.7	91.0	55.8	83.0	107.3
	min dev	304.0	24.3	31.0	16.2	6.0	8.7
5	mean	192.0	227.7	168.2	134.0	149.8	144.3
	std dev	57.0	82.3	24.4	19.9	21.4	27.0
	max dev	57.0	105.3	38.3	33.0	29.8	55.7
	min dev	57.0	9.7	4.3	3.0	0.2	4.3

Table 2. Statistical measures for the number of subproblems written to the stacks for five 80-node vertex-cover problems.

Problem	Measure	4 stacks	5 stacks	6 stacks	7 stacks	8 stacks	9 stacks
1	mean	224.5	605.0	1078.0	625.6	820.2	591.6
	std dev	153.5	161.9	445.6	223.7	355.5	252.1
	max dev	153.5	228.0	705.0	283.6	498.2	355.6
	min dev	153.5	96.0	25.0	165.4	10.2	106.6
2	mean	519.0	313.7	346.0	1041.8	892.2	803.3
	std dev	64.0	103.9	116.6	455.4	499.4	479.0
	max dev	64.0	141.3	172.0	696.8	664.8	753.7
	min dev	64.0	35.7	42.0	87.2	59.8	84.3
3	mean	493.5	329.0	352.5	282.0	650.0	557.1
	std dev	8.5	122.4	73.8	144.3	230.2	297.3
	max dev	8.5	173.0	110.5	242.0	388.0	497.1
	min dev	8.5	83.0	5.5	63.0	9.0	44.1
4	mean	418.5	279.0	1103.0	921.8	791.2	678.1
	std dev	272.5	291.4	483.3	466.3	460.0	338.3
	max dev	272.5	412.0	682.0	762.2	792.8	468.9
	min dev	272.5	2.0	186.0	82.8	47.8	152.1
5	mean	285.0	915.7	1201.3	1254.2	1085.0	966.9
	std dev	252.0	284.9	520.2	570.4	571.1	598.0
	max dev	252.0	396.7	710.3	801.2	798.0	903.1
	min dev	252.0	137.3	284.3	39.8	97.0	169.9

Table 3. Effect of the number of stacks on (a) the actual and predicted disk overheads and (b) the number of subproblems expanded for vertex-cover problems.

Number of Stacks	Disk overhead (msec)							
	Problem 1		Problem 2		Problem 3		Problem 4	
	Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual	Predicted
3	303.9	303.9	455.9	303.9	455.9	455.9	911.8	911.8
4	2130.1	2762.9	1127.7	3621.2	1754.2	2596.6	1127.7	3508.5
5	5503.4	4480.1	1853.2	6317.7	1740.9	3768.8	1403.9	4356.2
6	5465.8	7635.9	12649.5	9448.1	7183.7	5471.6	7808.7	8180.2
7	9667.0	9824.8	11787.8	12562.1	8236.7	6324.1	9568.3	10322.2
8	9665.3	11903.1	14188.5	15364.5	10093.8	8850.6	11046.1	11916.7
9	12763.7	16979.9	16555.7	18049.5	11052.6	9783.4	12393.7	13384.7

(a)

Number of Stacks	Number of subproblems expanded			
	Problem 1	Problem 2	Problem 3	Problem 4
3	4642	4747	3221	3806
4	4581	4399	2824	3534
5	4002	4171	2824	3534
6	2757	5681	3538	3561
7	3609	4065	3538	3431
8	2936	3997	3196	3329
9	3420	3975	3196	3329

(b)

and a capacity of 20 Kbytes per track. Table 3(b) shows the number of expanded subproblems for the same vertex-cover problems. As expected, the maximum number of subproblems are expanded when three stacks are used. However, the number is not excessively large as compared to those when more stacks are used. Similar results are shown in Tables 4(a) and 4(b) for five integer-programming problems. Tables 3(a) and 4(a) also compare the predicted disk overhead with the actual simulated disk overhead. For the vertex-cover problems, the predicted results are fairly close to the actual simulated results in spite of the many simplifying assumptions made. For the integer-programming problems, the predicted results are much smaller than the actual simulated results. One possible reason is the use of an exponential distribution instead of the more accurate gamma distribution for the increase in lower bounds when a subproblem is expanded. However, it may be seen that most of the predicted values differ from the actual values by factors that do not vary greatly for each problem. In addition, both the predicted and actual values increase with the number of stacks. This allows us to use the analytical expressions to predict the number of stacks to be used and their boundaries.

Table 4. Effect of the number of stacks on (a) the actual and predicted disk overheads and (b) the number of subproblems expanded for integer-programming problems.

Number of Stacks	Disk overhead (msec)							
	Problem 1		Problem 2		Problem 3		Problem 4	
	Predicted	Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual
3	2454.1	6095.9	219.3	840.8	276.1	5325.2	286.9	2706.7
4	4581.2	6588.2	343.2	1058.8	683.8	6058.8	614.4	6333.6
5	6806.2	16243.2	552.0	1362.3	931.9	7492.8	865.3	7458.0
6	7577.9	9492.6	621.7	1573.9	1085.7	8705.7	965.5	8797.0
7	8122.1	10562.3	862.6	1808.0	1140.7	9705.9	1176.4	9460.7
8	8976.2	11952.6	1038.5	2022.8	1405.4	10941.3	1494.5	9671.6
9	10409.6	13886.3	1298.6	2307.0	2138.6	15820.7	1969.6	9767.0

(a)

Number of Stacks	Number of subproblems expanded			
	Problem 1	Problem 2	Problem 3	Problem 4
3	3821	563	2981	2056
4	3471	563	2780	1765
5	3134	563	2782	1633
6	3402	563	2792	1616
7	3435	563	2789	1564
8	3455	563	2777	1434
9	3454	563	2556	1283

(b)

Our simulations also show that the performance of the modified B&B algorithm appears to be relatively insensitive to the uncertainity in the estimate of the optimal solution value for the integer-programming and vertex-cover problems. This increases the robustness of our method because the optimal solution value may be difficult to estimate before the algorithm terminates.

The modified B&B algorithm for the integer-programming and vertex-cover problems is easy to implement when three stacks are used. Only an estimate of the value of the optimal solutions and the bounds of accuracy of this estimate are necessary. For other problems, it will be necessary to study sample problems to obtain an indication of the number of stacks that should be used. If more than three stacks are to be used, then it will be necessary to actually run the algorithm for a short period (say, a few hundred iterations) to obtain an estimate on the distribution of the increase in lower bounds and the branching factor. The stack boundaries may then be computed and the problem solved by the modified B&B process.

4. COMPARISON OF VARIOUS SCHEMES

Simulation results for two 17-variable, 17-constraint integer-programming problems are shown in Figures 4 and 5. Each subproblem requires a memory size of 1476 bytes. A main-memory size of 128 Kbytes (10% of the virtual space) is assumed. Only the results with the optimal page sizes are shown for the modified virtual-memory scheme and the direct implementation. With large pages, there is a 100-time reduction of disk overhead for the modified B&B search as compared to a direct implementation, and a 60-time reduction as compared to a modified virtual-memory scheme.

Some simulation results for 80-node vertex-cover problems are shown in Figures 6 and 7. Each subproblem requires 330 bytes, and a main memory of 128 Kbytes is assumed. The modified B&B algorithm exhibits close to a 100-time reduction of the disk overhead as compared to a direct implementation and close to a 10-time reduction as compared to a modified virtual-memory scheme.

In conclusion, modification of the virtual memory to tailor to the characteristics of the search algorithm is effective but inflexible. The modification of the search algorithm to enhance the amount of locality exhibited is found to be a viable approach.

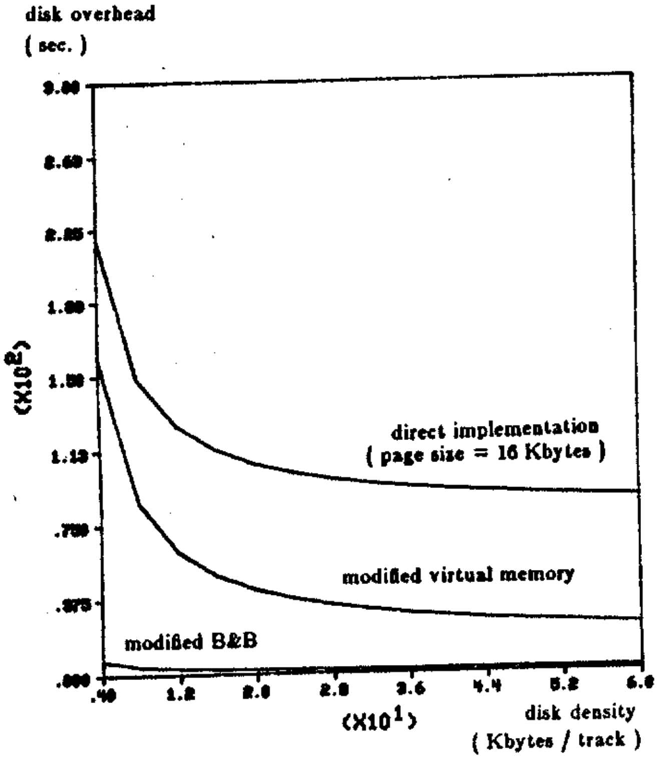


Figure 4 Simulation results for B&B algorithms on a 17 by 17 integer-programming problem (main memory size is 128 Kbytes).

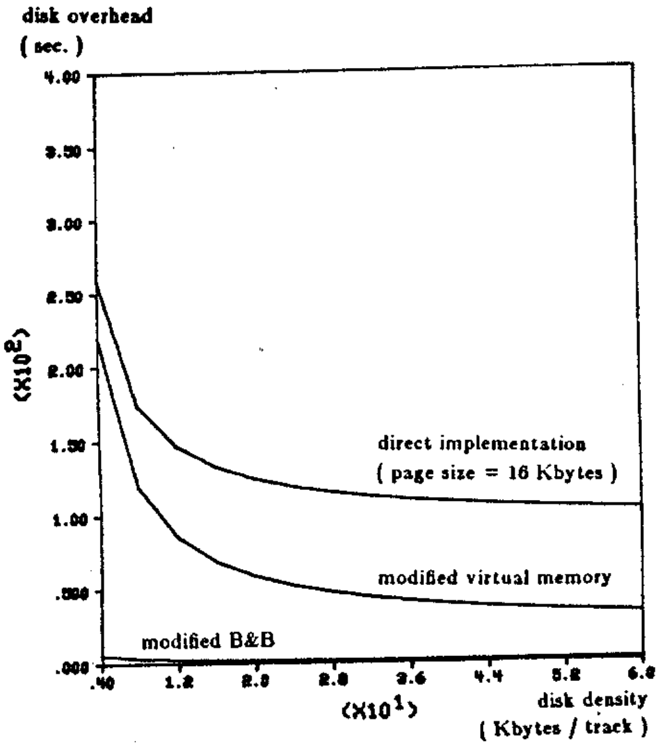


Figure 5 Simulation results for B&B algorithms on a 17 by 17 integer-programming problem (main memory size is 128 Kbytes).

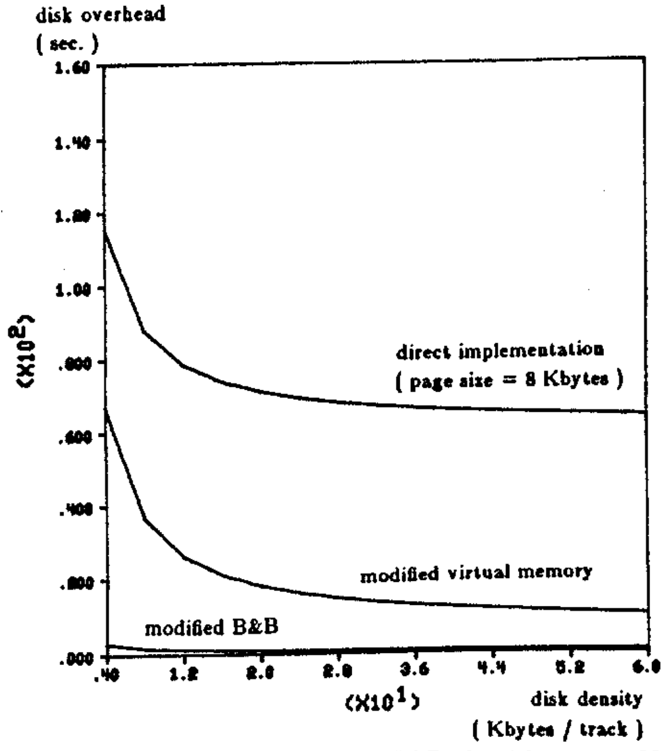


Figure 6 Simulation results for B&B algorithms on a 80 node vertex-cover problem (main memory size is 128 Kbytes).

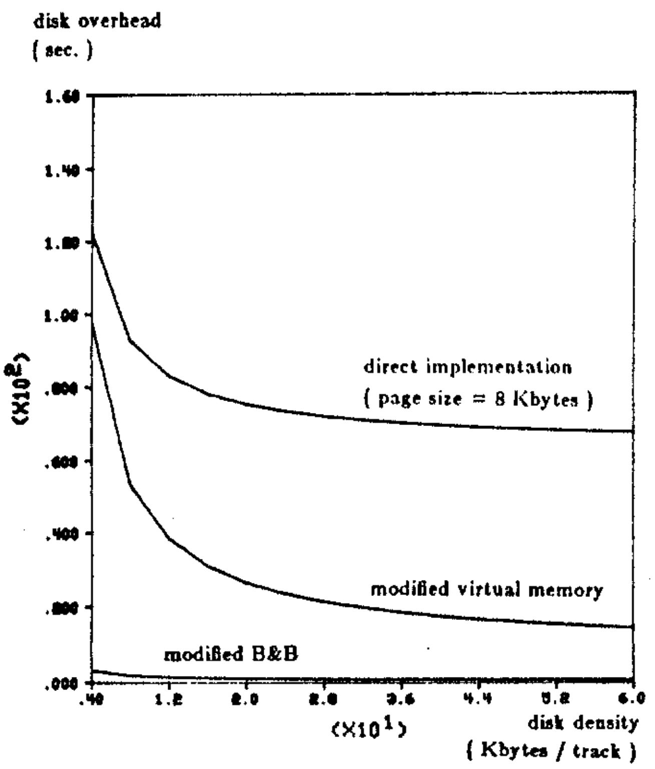


Figure 7 Simulation results for B&B algorithms on a 80 node vertex-cover problem (main memory size is 128 Kbytes).

APPENDIX A: EXPECTED NUMBER OF SUBPROBLEMS IN A STACK

This is based upon the wall model of a best-first search [WAH82] as depicted in Figure A-1. In this model a best-first search process is represented by two walls moving towards each other. The front wall represents the minimum lower-bound value of the subproblems, and the back wall represents the value of the incumbent. Let P_{i_j} be the j-th subproblem expanded, and $\delta(P_{i_j})$ be the lower-bound value of P_{i_j}. Initially, the front wall is at 0 and the back wall is at infinity. Since the lower bound of a descendant subproblem is always greater than that of the corresponding ancestor subproblem, the front wall always moves to the right in a best-first search.

When a subproblem P_s becomes a feasible solution, $\delta(P_s)$ is compared with the position of the back wall. If the position of the back wall (incumbent) is greater than $\delta(P_s)$, the back wall is set to this value. Successive expansions of subproblems cause the front and back walls to approach each other, and the process terminates when the two walls meet.

Let N(x) be the number of subproblems expanded and M(x) be the number of subproblems generated when the front wall is at x. It is assumed that the increase in lower bound between a descendant subproblem and the parent subproblem is a random variable satisfying the exponential density function, $f_y(y) = \lambda\, e^{-\lambda y}$. Also, each parent subproblem is assumed to be expanded into s descendant subproblems. Using these assumptions, it has

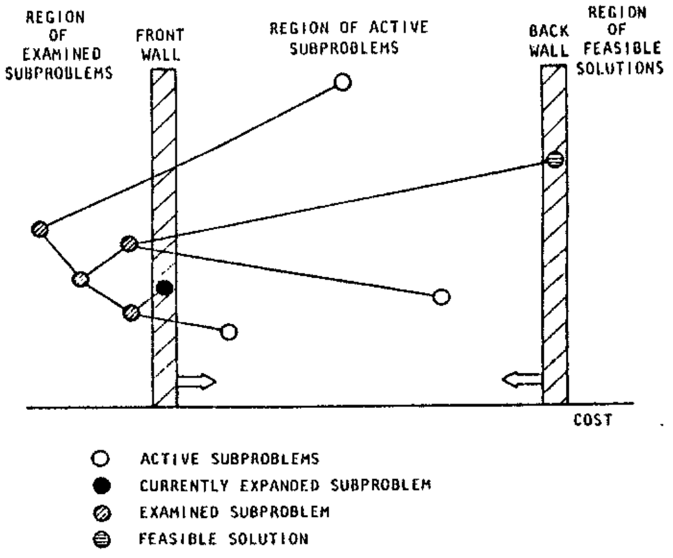

REGION OF EXAMINED SUBPROBLEMS
FRONT WALL
REGION OF ACTIVE SUBPROBLEMS
BACK WALL
REGION OF FEASIBLE SOLUTIONS
COST
○ ACTIVE SUBPROBLEMS
● CURRENTLY EXPANDED SUBPROBLEM
⊘ EXAMINED SUBPROBLEM
⊖ FEASIBLE SOLUTION

Figure A-1 The model of the B&B process under best-first search.

been shown in [WAH82] that the expected values of N(x) and M(x) are:

$$E[N(x)] \simeq \frac{s}{s-1}\, e^{\lambda(s-1)x} - \frac{1}{s-1} \qquad \text{(A-1)}$$

and

$$E[M(x)] \simeq s\, E[N(x)] + 1 \qquad \text{(A-2)}$$

Let subproblem P_q be the optimal solution, and $\delta(P_q) = v$. The total number of subproblems expanded is N(v). Let $Z_1, \ldots, Z_{N(v)}$ be identically and independently distributed random variables with distribution $F_z(z \mid v, N(v))$. Suppose that Z_i, $1 \le i \le N(v)$, has the value z_i. The expected number of random variables Z_i with value $z_i \le x$ given v and N(v), $E[K(x) \mid v, N(v)]$, is:

$$E[K(x) \mid v, N(v)] = N(v)\, F_z(x \mid v, N(v)) \qquad \text{(A-3)}$$

Without loss of generality, let $z_1 \le \ldots \le z_{N(v)}$. Assuming that the expanded subproblems are independent of each other, $\delta(P_{i_k})$ may be approximated by z_k provided that:

$$E[K(x) \mid v, N(v)] = E[N(x)] \qquad \text{(A-4)}$$

Substituting Eq's (A-1) and (A-3) into Eq. (A-4) yields:

$$F_z(x \mid v, N(v)) \simeq \frac{E[N(x)]}{N(v)} \qquad \text{(A-5)}$$

$$= \frac{1}{N(v)} \left[\frac{s}{s-1}\, e^{\lambda(s-1)x} - \frac{1}{s-1} \right]$$

Differentiating Eq. (A-5) yields the density function, $f_z(z \mid v, N(v))$:

$$f_z(x \mid v, N(v)) = \frac{1}{N(v)}\, \frac{s}{s-1}\, \lambda(s-1)\, e^{\lambda(s-1)x} \qquad \text{(A-6)}$$

Let the front wall be at position $x \le u_i$. The expected number of subproblems generated will be E[M(x)], and the expected number of subproblems expanded will be E[N(x)]. Let P_k be one of the E[M(x)] subproblems generated and $Pr(u_i \le \delta(P_k) < u_{i+1} \mid v, N(v))$ be the probability that $\delta(P_k)$ lies within $[u_i, u_{i+1})$. Out of the

E[M(x)] generated subproblems, the expected number of subproblems in the region $[u_i, u_{i+1})$ given v and N(v), $E[W(x)|v,N(v)]$, is given by:

$$E[W(x)|\,v,N(v)] = E[M(x)]\ Pr(u_i \le \delta(P_i) < u_{i+1}|\,v,N(v))$$

$$= (s\,E[N(x)] + 1)\ Pr(u_i \le \delta(P_i) < u_{i+1}|\,v,N(v))$$

$$\simeq s\ E[N(x)]\ Pr(u_i \le \delta(P_i) < u_{i+1}|\,v,N(v)) \qquad \text{(A-7)}$$

Let P_k be a subproblem generated by the expansion of P_h and $I(P_k)$ be the increase in lower bound due to this expansion.

$$\delta(P_k) = \delta(P_h) + I(P_k) \simeq Z_h + Y_k \qquad \text{(A-8)}$$

Since $I(P_k)$ has been assumed to be an exponentially distributed random variable,

$$Pr(u_i \le \delta(P_k) < u_{i+1}|\,Z_h = z, v, N(v)) \qquad \text{(A-9)}$$

$$= \int_{y=u_i-z}^{u_{i+1}-z} \lambda\, e^{-\lambda y}\, dy = e^{-\lambda u_i} - e^{-\lambda u_{i+1}}$$

Unconditioning and applying Eq. (A-6), we have:

$$Pr(u_i \le \delta(P_k) < u_{i+1}) \qquad \text{(A-10)}$$

$$= \int_{z=0}^{x} (\, e^{-\lambda u_i} - e^{-\lambda u_{i+1}}\,)\ \frac{1}{N(v)}\ \frac{s}{s-1}\ \lambda\,(s-1)\,e^{\lambda\,(s-1)z}\, dz$$

Substituting Eq. (A-10) into Eq. (A-7) yields:

$$E[W(x)|\,v,N(v)] \simeq \int_{0}^{x} (e^{-\lambda u_i} - e^{-\lambda_{i+1}}\,)\ s^2\, \lambda\, e^{\lambda s z}\, dz$$

$$= s\ e^{-\lambda u_i}\,(\,1 - e^{-\lambda w_i}\,)\,(\,e^{\lambda s x} - 1\,) \qquad \text{(A-11)}$$

Eq. (A-11) may be used to derive approximate expressions for the number of subproblems written into stack i, M_i, in the modified B&B algorithm. If $u_i \le v$ then M_i is the number of subproblems in stack i when it becomes the current stack (the first non-empty stack with the minimum index) since at this point, subproblems in stack i are expanded in a depth-first fashion and no new subproblems will be written to stack i in the secondary memory. Thus, $E[M_i|\,v] \simeq E[W(u_i)|\,v,N(v)]$ if the number of subproblems chosen for expansion from buffer i before stack i becomes the current stack is small. If $u_i > v$ then M_i is the number of subproblems in stack i when stack i is terminated. If no feasible solution is generated before the optimal solution is found, then $E[M_i|\,v] \simeq E[W(v)|\,v,N(v)]$. For problems such as integer programming and vertex cover, it has been found that the number of feasible solutions generated is small, and hence the above approximations are good. When the number of feasible solutions generated before the process terminates is significant as in the knapsack problems, then $E[M_i|\,v] \le E[W(v)|\,v,N(v)]$ and the approximations do not hold. Thus,

$$E[M_i|\,v] \simeq \begin{cases} E[W(u_i)|\,v,N(v)] & u_i \le v \\ E[W(v)|\,v,N(v)] & u_i > v \end{cases}$$

$$= \begin{cases} s\,e^{-\lambda u_i}(1 - e^{-\lambda w_i})(e^{\lambda s u_i} - 1) & u_i \le v \\ s\,e^{-\lambda u_i}(1 - e^{-\lambda w_i})(e^{\lambda s v} - 1) & u_i > v \end{cases}$$

$$\simeq \begin{cases} (1 - e^{-\lambda w_i})\,s\,e^{\lambda(s-1)u_i} & u_i \le v \\ (1 - e^{-\lambda w_i})\,s\,e^{-\lambda u_i}\,e^{\lambda s v} & u_i > v \end{cases} \qquad \text{(A-12)}$$

REFERENCES

[ALA84] T. O. Alanko, H. H. A. Erkio and I. J. Haikala, "Virtual Memory Behavior of some Sorting Algorithms," *IEEE Trans. on Software Engineering,* Vol. SE-10, No. 4, pp. 422-431, July 1984.

[BAE76] J. L. Baer and G. R. Sager, "Dynamic Improvement of Locality in Virtual Memory Systems," *IEEE Trans. on Software Engineering,* Vol. SE-2, No. 1, pp. 54-62, March 1976.

[BER79] H. Berlinger, "The B* Tree Search Algorithm: A Best-First Proof Procedure," *Artificial Intelligence,* Vol. 12, pp. 23-40, 1979.

[BRO70] B. S. Brown, F. G. Gustavson and E. S. Mankin, "Sorting in a Paging Environment," *CACM,* Vol. 13, No. 8, pp. 438-494, August 1970.

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

[DRE77] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming,* Academic Press, New York, 1977.

[FER74] D. Ferrari, "Improving Locality by Critical Working Sets," *CACM,* Vol. 17, No. 11, pp. 614-620, Nov. 1974.

[FER75] D. Ferrari, "Tailoring Programs to Models of Program Behavior," *IBM Journal of Research and Development,* Vol. 19, No. 3, pp. 244-251, May 1975.

[FER76] D. Ferrari, "The Improvement of Program Behavior," *Computer,* pp. 39-47, Nov. 1976.

[FIS79] P. C. Fisher and R. L. Probert, "Storage Reorganization Techniques for Matrix Computation in a Paging Environment," *CACM,* Vol. 22, No. 7, pp. 405-415, July 1979.

[HAT71] D. J. Hatfield and J. Gerald, "Program Restructuring for Virtual Memory," *IBM Systems Journal,* Vol. 10, No. 3, pp. 168-192, 1971.

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

[IBA76b] T. Ibaraki, "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] T. Ibaraki, "The Power of Dominance Relations in Branch-and-Bound Algorithms," *JACM,* Vol. 24, No. 2, pp. 264-279, 1977.

[IBA78] T. Ibaraki, "Branch-and-bound Procedure and State-Space Representation of Combinatorial Optimization Problems," *Information and Control,* Vol. 36, pp. 1-27, 1978.

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

[KNU75] D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence,* Vol. 6, pp. 293-326, 1975.

[KUM83] V. Kumar and L. Kanal, "A General Branch-and-Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures," *Artificial Intelligence,* Vol. 21, pp. 179-198, 1983.

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

[LI84] G. J. Li and B. W. Wah, "Computational Efficiency of Parallel Approximate Branch-and-Bound Algorithms," *Proc. Int'l Conf. on Parallel Processing,* August 1984.

[MCK69] A. C. McKellar and E. G. Coffman, Jr., "Organizing Matrices and Matrix Operations for Paged Memory Systems," *CACM,* Vol. 12, No. 3, pp. 153-164, 1969.

[MAR78] A. Martelli and U. Montanari, "Optimizing Decision Trees Through Heuristically Guided Search," *Comm. ACM,* Vol. 21, pp. 1025-1039, 1978.

[MIT70] L. Mitten, "Branch and Bound Methods: General Formulation and Properties," *Operations Research,* Vol. 18, pp. 24-34, 1970.

[MOR66] J. E. Morrison, "User Program Performance in Virtual Storage Systems," *IBM Systems Journal,* Vol. 5, No. 2, pp. 216-237, 1966.

[NIL71] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill Book Company, New York 1971.

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

[POH70] I. Pohl, "Heuristic Search Viewed as Path Finding in Graph," *Artificial Intelligence,* Vol. 1, pp. 193-204, 1970.

[SMI78] A. J. Smith, "Sequentially and Prefetching in Database Systems," *ACM Trans. on Database Systems,* Vol. 3, No. 3, pp. 223-247, Sept. 1978.

[STO83] G. C. Stockman and L. N. Kanal, "Problem Reduction Representation for the Linguistic Analysis of Waveforms," *IEEE Trans. on PAMI,* Vol. 5, No. 3, pp. 287-298, 1983.

[WAH82] B. W. Wah and C. F. Yu, "Probabilistic Modeling of Branch-and-Bound Algorithms," *Proc. of COMPSAC,* pp. 647-623, 1982.

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