1993

A parallel branch and bound algorithm for the multi-facility capacity expansion problem using networked RISC workstations

Margaret King Mayer
Lehigh University

AUTHOR:

Mayer, Margaret King

TITLE:

A Parallel Branch and Bound Algorithm For The Multi-Facility Capacity Expansion Problem Using Networked RISC Work-Stations

DATE: May 30, 1993

A PARALLEL BRANCH AND BOUND ALGORITHM FOR THE MULTI-FACILITY CAPACITY EXPANSION PROBLEM USING NETWORKED RISC WORKSTATIONS

by

Margaret King Mayer

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

May 20, 1993

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

5/20/93

Date

Thesis Advisor

Co-Advisor

Chairperson of Department

TABLE OF CONTENTS

	Page
LIST OF TABLES	iv
LIST OF FIGURES	v
ABSTRACT	1
CHAPTER 1 - INTRODUCTION	2
CHAPTER 2 - THE MODEL	4
CHAPTER 3 - NETWORK FLOW REPRESENTATION	7
CHAPTER 4 - SOLUTION APPROACH	10
CHAPTER 5 - BRANCH AND BOUND SEARCH STRATEGIES	17
5.1 Depth First Search	17
5.2 Breadth First Search	18
CHAPTER 6 - IMPLEMENTATION OF BRANCH AND BOUND APPROACH	19
6.1 Sequential Implementation	19
6.2 Parallel Implementation	20
CHAPTER 7 - PERFORMANCE MEASURES	25
CHAPTER 8 - RESULTS	30
CHAPTER 9 - FUTURE RESEARCH	40
REFERENCES	43
APPENDIX A	44
VITA	45

LIST OF TABLES

Table		Page
7.1	Sequential vs. Parallel DFS	27
7.2	Cycle Variation Performance	27
7.3	Solution Time for No Reinitialization of Processors	29
8.1	Total Solution Time vs. Cycle Length	32
8.2	BFS vs. DFS Reinitialization	32
8.3	Maximum Number of Nodes Solved	39

LIST OF FIGURES

Figure		Page
3.1	CEP as Network Flow Problem	9
4.1	Concave Cost Function	11
4.2	Concave and Linear Cost Functions	11
4.3	Branch and Bound Tree	12
4.4	Concave Cost Function with Linear Underestimator	15
6.1	Branch and Bound Tree in Parallel	21
7.1	Processor Workload	29
8.1	Solution Time vs Processor, Variation 1	33
8.2	Solution Time vs Processor, Variation 2	33
8.3	Solution Time vs Processor, Variation 3	34
8.4	Solution Time vs Processor, Variation 2, T=18	37
8.5	Solution Time vs Processor, Variation 2, T=20	37
8.6	Speed Up for EX3, Variation 1, T=15	38
8.7	Speed Up for EX3, Variation 2, T=5	38
8.8	Speed Up for EX3, Variation 3, T=15	39

ABSTRACT

A parallel branch and bound algorithm for the multi-facility capacity expansion problem(CEP) is examined as a means of extending real-time feasibility. The CEP addresses the minimum cost expansion of facilities over time. There are two or more facilities that may require expansion to meet demands over the length of a planning horizon. Each facility is ranked in order of decreasing quality so that a higher quality facility may convert excess capacity to a lower facility. The decision to meet demand through expansion or conversion must be made at each stage of the planning horizon. Historically, both exact solution approaches and heuristic approaches have been applied to this problem. An exact solution is more desirable, but the exact solution approach quickly becomes real-time infeasible on a sequential machine. Performance of the algorithm is measured in solution time, and in terms of load balancing. Since the parallel algorithm is implemented over a network of IBM RISC workstations, efficient message passing is critical to good performance. Several strategies are developed and implemented to reduce message passing and improve performance. The results of the implementation are discussed.

CHAPTER 1

INTRODUCTION

The purpose of the capacity expansion problem(CEP) is to determine a minimum cost expansion plan given inputs of a planning horizon and cost information. The three central issues of the CEP are: expansion sizes, expansion times, and locations[4]. In a multi-facility type CEP, there are two or more facilities that may require expansion to meet demands. Each facility is ranked in order of quality so that a higher quality facility may convert excess capacity to satisfy a lower quality facility's demands. The decision to expand a facility to meet demand or to convert excess from another facility must be made at each stage in the planning horizon. The multi-type CEP tries to resolve this issue. A.S. Manne[6] pioneered much of this work in 1967 using data from the Indian heavy process industries, and his results are still used today as performance benchmarks for new heuristics.

This thesis reports the performance of a branch and bound programming approach using a network representation of the problem to solve the CEP in sequential and parallel configurations. The solution space for the CEP as a network problem is exponential, 2^{NT}, where N=number of facilities and T=number of time periods. Since the number of distinct solutions grows exponentially, the problem quickly becomes real-time infeasible, i.e., results cannot be determined in time to be useful. Heuristic approaches to achieve near optimal solutions were developed by Luss[2,4,5] using pseudo-polynomial algorithms. A near optimal solution is sufficient for applications where the extra expense or time required in solving to optimality renders the problem real-time infeasible. In other applications the requirements of exact solution justify the extra cost or time. A parallel programming approach is examined as a means of extending real-time feasibility for an exact solution approach to the problem.

The performance of the sequential and parallel programs is critical in analyzing real-time feasibility. Performance is measured not only in solution time, but also in the storage requirements necessary to attain that solution. The decrease in total solution time by converting an algorithm can be calculated by a speed up ratio. The parallel algorithm approach is a natural path to take, given the repeated calculations necessary in combinatorial problems to achieve optimization. Although speed-up ratios greater than 1 can be attained by converting sequential programs to parallel programs, communication overhead between the processors contribute to solution time.

Besides determining optimal expansion of facilities, the CEP model can be applied to communications networks. Luss[3] describes a cable sizing application in which 2 cable types are available to use. Cable type 1 can meet demands for cable type 2, but the type 2 cable cannot be substituted for cable type 1. The demands are known for a horizon of T periods, and an optimal policy can be determined to meet demands and minimize the cost of the cable. The CEP model can also be applied to inventory and production problems. An example is a multi-product inventory and production model, where one product type can be substituted for another product. Since the cost to produce either product typically involves fixed and variable costs, it may be optimal to perform substitutions to meet customer demands.

CHAPTER 2

THE MODEL

The model for the multi-facility type CEP was formulated by Luss[4] with the following notation:

$$\min_{x_{it}, y_{ijt}} \left(\sum_{t=1}^{T} \sum_{i=1}^{N} [c_{it}(x_{it}) + h_{it}(I_{i,t+1})] \right)$$

such that

$$I_{it+1} = I_{it} + x_{it} + \sum_{k=1}^{i-1} y_{kit} - \sum_{k=i+1}^{N} y_{ikt} - r_{it}$$

$$I_{it} \geq 0$$

$$I_{i1} = I_{iT+1} = 0$$

$$x_{it} \geq 0, \; y_{ijt} \geq 0$$

$$t = 1..T \quad i = 1..N, \; j = i+1..N$$

i,j,k,l,m	=	Indices for facilities. N facilities are available and ordered from 1 to N in order of decreasing quality.
t,u,v	=	Indices for the time line. The time line consists of T periods.
r_{it}	=	Demand for facility i in period t. Assumed to be integer.
x_{it}	=	The expansion size of facility i in period t. Assumed to be integer.
y_{ijt}	=	The amount of capacity converted from facility i to facility j in period t, where i < j. Assumed to be integer.
I_{it}	=	The excess capacity for facility i at the beginning of period t. Integer assumption and no shortages are allowed.
$c_{it}(x_{it})$	=	Capacity expansion cost function.
$h_{it}(I_{i,t+1})$	=	The holding cost function associated with having excess capacity at facility i for 1 period, from period t to t+1.

This model assumes a finite horizon of T periods, and N facility types. All demand increments, expansions, and conversions occur instantaneously at the beginning of each period. Luss[5] adopted the policy regarding converted capacity that any converted capacity in this formulation becomes a physical part for the new facility, and cannot be changed back to the original facility.

The cost function in any CEP is typically nondecreasing and concave to represent economies of scale. Generally, the cost functions consist of fixed and variable costs associated with the expansion of the facility. Luss[3] reasoned that operating costs are generally not considered since they are assumed to be independent of any chosen expansion policy. The cost function should consider the time value of money, although it is difficult to choose an appropriate discount rate(*p*) for a long horizon. Manne[6] noted that the value of *p* is often subjective, and vary as to whether it is private or government business. Once *p* has been chosen, the typical cost function has the following form:

$$c_{it}(x_{it}) = p^{t-1}(A_i + B_i x_{it})$$

where:

A = the fixed charged cost of expansion for facility i

B = the variable cost per unit of expansion

p = the discount factor.

Although this is the "popular" form for a cost function in a CEP, it is by no means the only one. Luss[4] points out that a piecewise concave function could be used. An example for doing so would be when different technologies are used to expand/add facilities.[4] In this case, it is concave in the range of each technology used.

The demands over time, r_{it}, are often represented in the form of a function. Luss[4] defines 3 typical demand functions:

$$R_i(t) = \begin{pmatrix} \mu + \delta t \\ \mu e^{\delta} t \\ \beta[1 - e^{\delta} t] \\ \\ \textit{where } \mu,\ \delta,\ ,\beta\ \geq 0 \textit{ constant.} \end{pmatrix}$$

The linear function assumes demand has a constant growth rate, while the exponential function makes demand growth proportional to the volume. In certain markets, demand may have a saturation point, which is what the last function captures. If the problem consists of a short horizon, other functions can also be considered.

CHAPTER 3

NETWORK FLOW REPRESENTATION

The multi-type CEP can be represented as a network flow problem(Fig. 3.1). There is a single source node with supply of R, where R is the sum of all demands. There will be NT additional nodes, for each facility in each time period. At each of these nodes, the demand r_{it} exists. In Luss[3], the flows are defined as follows:

- Expansion Flow, x_{it}, from the source node to every node;
- Excess Capacity Flow, $I_{i,t+1}$, from each node (i,t) to node (i,t+1);
- Conversion Flow, y_{ijt}, from each node (i,t) to node (j,t) where $i > j$.

The beginning flows, I_{i1}, and ending flows, $I_{i,T+1}$, are defined to equal 0. The costs associated with each flow are:

y_{ijt} has cost 0;

$I_{i,t+1}$ has holding cost function $h_{it}(I_{i,t+1})$;

x_{it} has the concave cost function $c_{it}(x_{it})$.

A feasible flow and optimal solution in the network is found through an extreme point solution[3]. Zangwill[8] defines extreme flow in a network as having at most one positive incoming flow into each node. In addition, the network must have a single source, which the CEP network does. In Luss[2], he notes that more than one optimal solution may exist, and it may not be an extreme point solution. Any extreme point solution for the CEP will satisfy the following properties as defined by Luss[2]:

$I_{it}x_{it} = 0$, $t=1,2..T$;

$I_{it}y_{ijt} = 0$, for $i,j = 1,2..N$, $i < j$.

The above states that no extra capacity can exist if a facility is to be expanded in a period t.

$x_{jt}y_{ijt} = 0,$

$y_{ijt}y_{kjt} = 0$ for $k = 1,2,..N$, $k < j$, $i = k$.

This states that either an expansion or capacity conversion can occur, but not both. It is not optimal to expand a facility to meet only a part of the demand with that expansion.

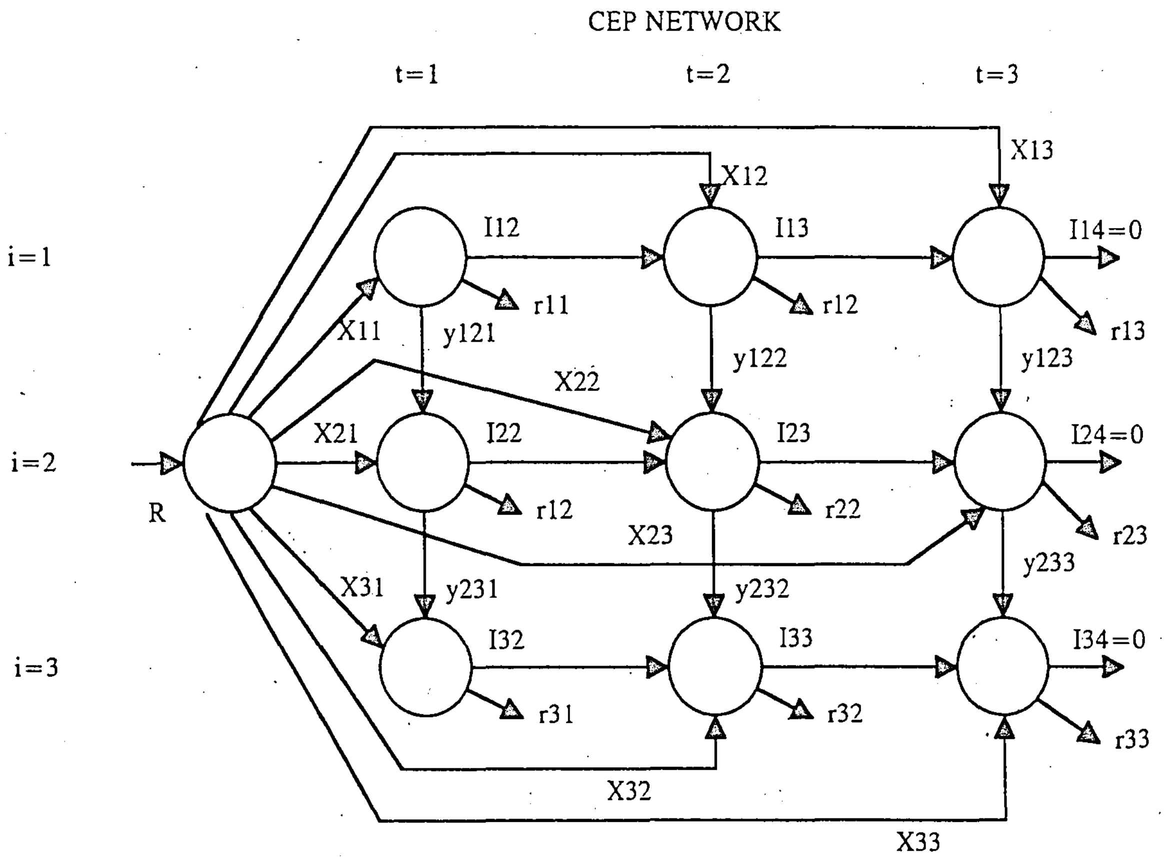

Figure 3.1 - CEP as Network Flow Problem

CHAPTER 4

SOLUTION APPROACH

It would appear that the network flow representation of the problem could be easily solved, but the expansion cost functions are concave(Fig. 4.1) and, in particular, non-linear. In this case there is no way to transform this cost function to a purely unit cost function that network solution methodologies require. It is desirable to use a network flow approach since the number of arcs is small compared to the number of distinct solutions. Basic network simplex could be used to solve this network if only the arc costs were purely linear. A transformation of the concave cost into a linear one would satisfy the unit cost restriction.

It is straightforward to convert the function when an arc does not produce. If arc x_{it} does not produce, a linear function(Fig 4.2) which is parallel to the concave cost but passes through the origin is equivalent to the concave cost function since zero flow costs nothing. If an arc were to produce, the true concave cost function should be used. In this case, the unit portion, B_i, is passed to the network flow problem, and the fixed cost is added to the solution cost afterward. The decision to produce or not becomes the central issue to solve the CEP as a network since a unit cost can be found for either case. Once a decision to expand has been made, the appropriate unit cost is then assigned to the arcs and a minimum cost network flow (MCNFS) problem is solved. A branch and bound algorithm can be utilized to determine the optimal arc expansion plan.

Branch and bound is a powerful tool that allows a difficult combinatorial problem to be solved without enumerating the solution space. At each node in the branch and

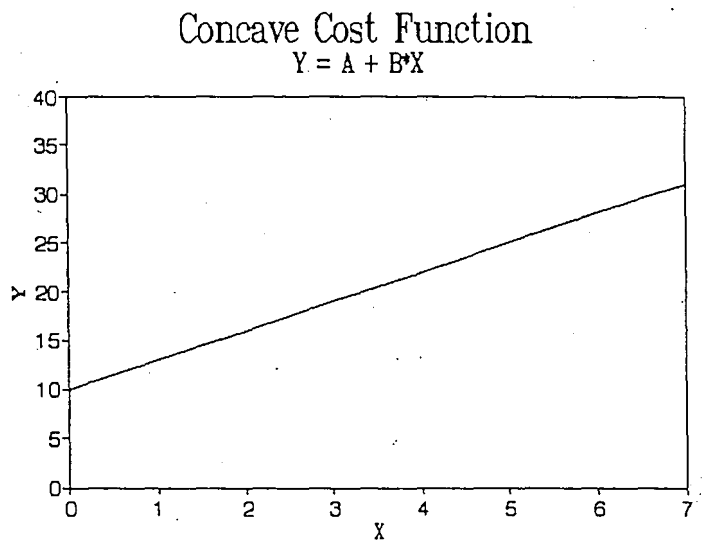

Figure 4.1 - Concave Cost Function

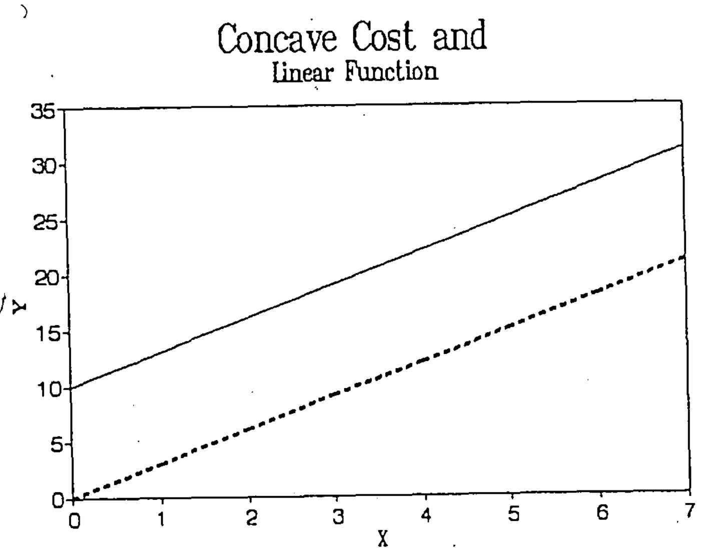

Figure 4.2 - Concave and Linear Cost Function

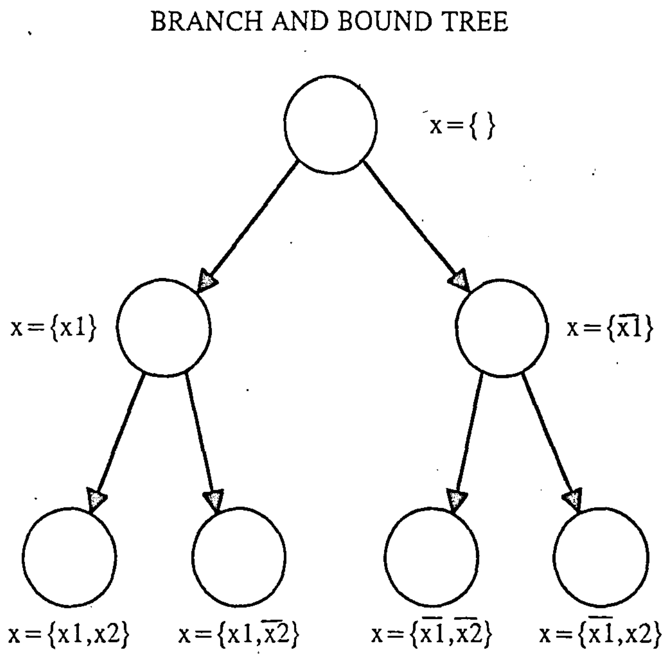

Figure 4.3 - Branch and Bound Tree

bound tree(Fig 4.3), a certain concave flow, x_{it}, is picked or "branched on" to be fixed to 0(closed) or allowed to be greater than 0(open) and placed into the set **X**. Any variable not yet branched on is unassigned. By systematically branching on each concave arc, the solution space will eventually be enumerated. However, branch and bound has the ability to implicitly enumerate the tree by fathoming(eliminating) sub-optimal branches. To do this, a bound, Z_B, which is the best solution found to the problem so far, is compared to the bound at a particular node, Z_A. If $Z_A >= Z_B$, continuation on that branch will only provide a sub-optimal solution, so that node is fathomed.

To return Z_A for each node, the MCNFS is solved by separating the objective function as follows:

$$Z=\min\sum_{t=1}^{T}\sum_{i=1}^{N}p^{t-1}A_i(X_{it})+\sum_{t=1}^{t}\sum_{i=1}^{N}p^{t-1}B_i(x_{it})$$

The variable portion of the cost function, $B_i()$, is the only cost passed to the MCNFS. To enforce the "fixing" of variables, the actual variable cost on the concave arcs changes according to its status in the branch and bound node:

$$B(x_{it})=\begin{pmatrix} B_i & if\ x_{it} & open \\ M & x_{it} & closed \\ e_{it} & x_{it} & unassigned \end{pmatrix}$$

The M cost(analogous to Charnes' Big M) is an infinite cost assigned to closed arcs. If a feasible solution exists only with this arc included it will be reflected in the value of Z_A.

The function, $e_{it}(x_{it})$, is calculated using the capacity, K_{it}, associated with the arc

x_{it}. Although this is an uncapacitated problem, the minimization of the objective function will drive each facility i in each time period u to at most meet demand for period u to T, for itself and facilities j to N, j > i. This value is calculated as follows:

$$K_{jt}=\sum_{j=i}^{N}\sum_{t=u}^{T} r_{jt}$$

The underestimate cost function, e_{it}, for unassigned variables is defined as:

$$e_{it}(x_{it})=(B_i+\frac{A_i}{K_{it}})x_{it}$$

This approximation(Fig. 4.4) provides a linear estimator for the unassigned arcs within the branch and bound scheme. Since this function incorporates the implicit capacity and the fixed cost into the estimate, it will provide the MCNFS with a better picture of choosing this arc to expand. MCNFS will return an approximate solution, Z_A since only the variable portion of the objective function is passed to it. The MCNFS is solved at each node in this branch and bound approach.

Once a Z_A is returned by MCNFS, it is compared against Z_B, and the branch and bound proceeds as follows:

1. If $Z_A < Z_B$, a new value Z_F is calculated which adjusts Z_A by incorporating true variable and fixed information:

```
$Z_F=Z_A$;
for each unassigned $x_{it}$ with $x_{it}$>0:
    $Z_F=Z_F+A_i+(B_i-e_{it})*x_{it}$;
```

   If $Z_F < Z_B$, replace Z_B with Z_F.
   Branch on the next unassigned variable.

2. If $Z_A >= Z_B$, fathom the node.

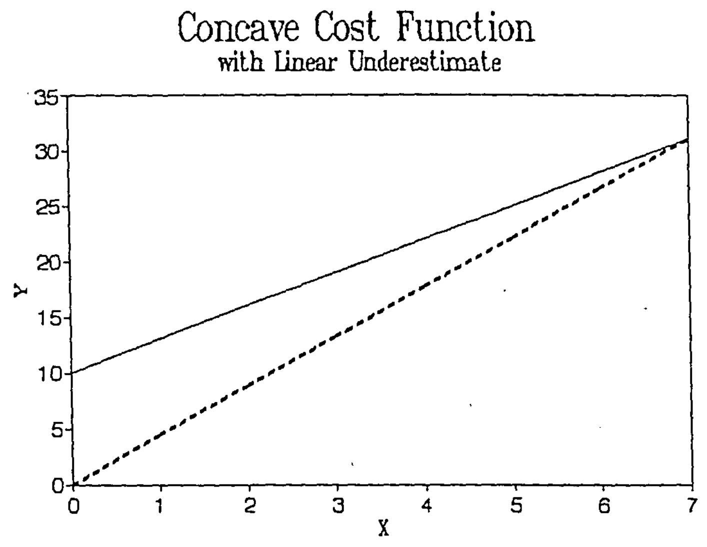

Figure 4.4 - Concave Cost Function with Linear Underestimator

Notice that a node can only be fathomed for step 2's condition. The adjusted Z_F cannot fathom the node since the costs on the arcs change as the branch and bound tree expands. Z_F is a valid solution as all costs are adjusted for the unassigned variables based on the flow returned by MCNFS, so it can replace Z_B. However, further branching from this node may yield different flows on previously unassigned arcs due to cost variability.

CHAPTER 5

BRANCH AND BOUND SEARCH STRATEGIES

When a branch and bound procedure is implemented on a computer, it can be quite memory intensive, since the number of distinct solutions for any problem that exist is 2^M, wher M=number of variables. For the CEP, there are N*T such variables, which are all the concave arcs. The number of total branch and bound nodes(BBNODES) that could be solved during the procedure is:

$$\sum_{m=1}^{NT} 2^m$$

To generate the BBNODES, there are two rules typically used: depth first search(DFS) and breadth first search(BFS).

5.1 Depth First Search

A DFS generates both sides of the branch nodes BBNODE1 {x0=0} and BBNODE2 {x0=1}. It picks one side, BBNODE2, solves the problem, and generates 2 new nodes with an additional arc fixed. For example, suppose arc x1 is picked, the 2 new nodes are: BBNODE3{x0=0,x1=0} and BBNODE4 {x0=0,x1=1}. In DFS, the last node created, BBNODE4, is evaluated and then 2 new nodes may be generated, etc. This continues until a BBNODE is fathomed and the procedure backs up a level. If DFS picked BBNODE4 to evaluate and it was fathomed, we would next solve BBNODE3. It may branch to include more variables, and thus create more BBNODES. If BBNODE3 was fathomed also, the only node left to evaluate is BBNODE1. Once evaluated, it may generate BBNODE5 {x0=0,x1=1} and BBNODE6 {x0=0,x1=0}. BBNODES that have been created but not evaulated(active nodes) may be fathomed if the associated Z_A exceeds Z_B at some point. Although a BBNODE has not been evaluated, it inherits its parent Z_A value as a lower bound to its eventual solution. When no more nodes are left to be evaluated a solution has been found.

5.2 Breadth First Search

BFS works the same way but before branching down a level both BBNODE1 and BBNODE2 are evaluated to pick a winner with the "best" Z_F. Two new problems, BBNODE3 and BBNODE4, are generated from the winner, say BBNODE1, and now BBNODE2, BBNODE3, and BBNODE4 are compared to decide a new winner. Z_A is continually evaluated against Z_B in case it can be fathomed, and excluded from the comparison. The difference between DFS and BFS is the status of the active BBNODES. Active BBNODES in BFS have already been solved and are used to determine which node to branch on. DFS active nodes have not been solved and branching is done with the most recently solved node unless fathomed. If fathomed, the next BBNODE is chosen according to a LIFO rule. BFS strategy will tend to generate more BBNODES on the active list, since it continually searches the active list to determine which problem to solve next. Both rules have their merits, and many researchers use a combination of the two to achieve fast results.

CHAPTER 6

IMPLEMENTATION OF BRANCH AND BOUND APPROACH

6.1 Sequential Implementation.

The sequential implementation of branch and bound is solved using three different approaches: DFS only, BFS only, and a combination of BFS and DFS. Each node contains the complete information to solve the MCNFS. It is necessary to keep some redundant information since costs change relative to the status of x_{it} (unassigned, open, closed). By retaining all the information at each node, the whole branch and bound tree does not need to be saved. Each time two new nodes are created, the parent node is destroyed, since the children inherit complete information.

The DFS and BFS only approaches uses the strategies as described in Chapter 5. At worst, N*T nodes will be on the active list for DFS, where the worst case is going to the bottom of the tree before fathoming branches. Since the amount of information retained for each node is not trivial, the DFS approach allows large problems to be solved without exhausting computer memory. The BFS approach will generate more nodes and larger problems cannot be solved sequentially.

To utilize the features of BFS and extend the size of the problem solved, a hybrid of the two strategies was developed. The algorithm solves DFS until the size of its list is twice the number equal to N*T. It then switches to BFS until the list size is strictly less than that. This hybrid method helps DFS to stop branching and producing new problems on what may be a sub-optimal part of the tree. By turning to BFS, a better bound may be found, and consequently fathom out nodes on the active list.

The sequential branch and bound approaches were implemented on an IBM RISC workstation using a combination of C and C++. The MCNF algorithm used in the sequential and parallel implementations is proprietary software written by Dr. Louis J. Plebani of Lehigh University. The location of all source code is given in Appendix A.

6.2 Parallel Implementation.

The branch and bound procedure in parallel operates asynchronously using an multiple instruction multiple datastream(MIMD) approach. This approach allows each processor to operate on different data sets and to perform a different set of instructions. Although each processor will solve a branch and bound problem, each one operates asynchronously, so that each processor could be at different stages in the solution. There was a fixed number of processors, MPROC, available to solve the problem, and the branch and bound tree was parsed accordingly. Each processor is initialized with some concave arcs assigned to open or closed, which is dependent on the number of processors. For a problem with 4 processors, each machine would be initialized as shown in Figure 6.1. Using the properties of CEP, an additional concave arc, x_{11}, the expansion of facility 1 in period 1, is fixed to open only, as no other facility can meet its demand through a conversion. All feasible solutions must include $x_{11}=1$ due to this property.

A process manager exists to control MPROC processes and provide communication among them. Ideally, each processor solves one BBNODE on its active list and reports its results. MPROC processes asynchronously report results such as

PARALLEL BRANCH AND BOUND TREE

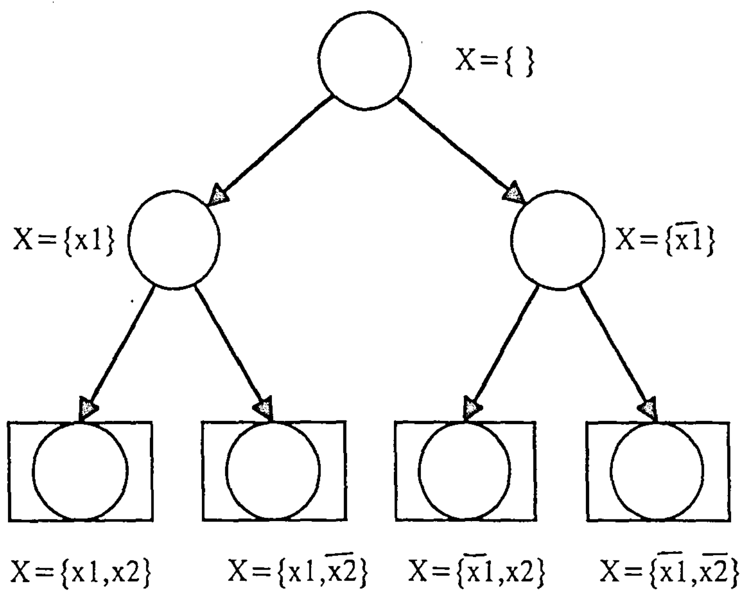

Figure 6.1 - Branch and Bound Tree in Parallel

BUSY, IDLE, and NEWLB to the manager and receives instructions such as CONTINUE, NEWLB, and NEWBBNODE. A BUSY process is one that has active BBNODES to solve, whereas an IDLE process has an empty BBNODE list. A process may also return a new lower bound, NEWLB, in addition to the process status. If the manager receives NEWLB, it then broadcasts the NEWLB to each process. Each process will then manage its active BBNODE list to see if any node exceeds NEWLB.

If a manager receives an IDLE message, it will attempt to find a BBNODE from a BUSY process to pass to the IDLE process. The manager will query the BBNODE that reported the last NEWLB on the assumption that that part of the tree may yield the global solution. This effectively transfers part of a branch and bound tree from one processor to another. The queried BBNODE will take a problem off its active list to give to the idle processor. It will fail if the queried BBNODE does not have more than two nodes left to solve. This condition exists because empirical results show the time to solve one branch and bound node is quite small, approximately 0.007 seconds for problems sampled in this research. Transfer of a new problem and reinitialization of the idle processor will take considerably more time than the above. It is more efficient to leave the processor idle than force a transfer. If unable to find a BBNODE, the process remains IDLE until a BBNODE can be found for it to solve. When all nodes are IDLE, the optimal solution has been found.

To implement the parallel strategy described above, the software LINDA from Computer Science Associates was used. The branch and bound code and process manager was written in ANSI C and C++. LINDA provides the TCP/IP protocols needed to remotely execute processes across a network of IBM RISC workstations. Since the processors are distributed across this network, there is considerable overhead in communication such as reporting processor status. A processor only reports NEWLB and IDLE to the manager which is dedicated to process management. The processors only read NEWLB once every few cycles instead of every cycle, where a cycle is a branch and bound node. When trying to write a new NEWLB, the processor will read in the latest NEWLB so that it will only report a true NEWLB. There exists a tradeoff between communication overhead and knowing perfect information, but the time to solve a node is so small that checking every few cycles provides a performance improvement. This is discussed more in Chapter 7.

As with the sequential implementation, three variations of the branch and bound strategy at each node were used: DFS only, BFS only, and DFS-BFS and compared against the sequential implementation. In addition to that, re-initialization of processors can also use BFS or DFS to determine which problem to give to the idle processor. Re-initialization strategy was also varied along with the problem selection.

Using only a DFS search strategy effectively implements a hybrid of DFS and BFS. Each processor gets the NEWLB from the manager on a timely basis and can fathom its own active list from it. Since MPROC nodes are being solved at one iteration, the process manager is employing a BFS-like strategy by passing the best known Z_B to all processes. If the number of cycles is lengthened before reading the NEWLB, the greater the chance that it would be solving many sub-optimal nodes and lose all advantages of parallel processing.

With the BFS only strategy for the parallel implementation, the same limitations apply as with the sequential approach. The Linda software uses shared memory distributed across all processors so that less memory is available to each processor than normal. Due to this, some problems may be able to be solved sequentially with BFS, but not in parallel. However, since parts of the tree are distributed on different processors which do communicate NEWLB, the active list size may be small enough to compensate for the loss in memory. Since the size of the list is still critical in terms of memory, the hybrid approach was also implemented in parallel. Due to communication overhead and a cycle length > 1, each processor may not have the latest NEWLB, and could be generating and solving sub-optimal problems. The same rules for switching from DFS to BFS and back still apply.

CHAPTER 7

PERFORMANCE MEASURES

To judge the performance of the branch and bound algorithm in sequential and parallel configurations, several statistics were necessasry besides total time to solution. Measures such as load balancing of each processor are considered important to identify areas of performance gain and loss. In a distributed parallel environment, knowledge of the frequency of communication among processors is also necessary to judge and tune performance. Preliminary results from sample problems can be used to determine the configuration of the branch and bound parallel algorithm to yield good results over a larger set of problems.

Time to solution was measured for both configurations and a speed-up S, was computed to determine the increase in performance from sequential to parallel:

$$S = 1 - \frac{t_p}{t_s}$$

t_s is the time to solution sequentially, and t_p time to solution in parallel. It is possible for S to be negative if the parallel implementation is not efficient.

In both configurations, the number of total nodes and bottom nodes solved was kept. Bottom nodes solved is the number of times the algorithm completely fixed all arcs to open or closed before fathoming the branch. In a DFS strategy, it is expected that this number will be greater than with BFS. As the number of processors increases, a BFS-like strategy is implemented, and will fathom a branch before reaching a bottom node so that less bottom nodes will be solved. Table 7.1 shows a sample problem where each processor uses a DFS strategy to select nodes to solve. Parallel-I uses DFS also to reinitialize, and Parallel-II uses a BFS reinitialization strategy. The number of total nodes solved increases due not only to the BFS-like strategy, but also to other factors such as communication delay and unbalanced loads.

In the parallel configuration, the frequency with which a processor reads the new lower bound and checks messages affects performance tremendously. When processors read messages after each branch and bound node solved, the performance gains are completely obscured. In effect, the processors are continually competing to gain access to the NEWLB and spending more time checking messages than solving nodes. To fix this problem, each processor will read messages after a number of cycles C, where a cycle is 1 node that has been solved. Total solution time was used to measure the difference in performance for different cycles, and was used to determine the "optimal" cycle length for a problem. If any strategies were varied, the "optimal" cycle length was re-investigated before running a suite of problems. Table 7.2 exhibits the difference in performance for a sample problem of 2^{24}, where BFS reinitialization was used. Although the total time does not strictly decrease as the number of cycles is increased beyond 15, this can be attributed to network noise. Using the results of Table 7.2, a cycle length of 20 would be used.

Another factor that affects the speedup of the solution is the load balance of each processor. Each processor is randomly assigned a node in the tree to begin the alogrithm, where the number of variables fixed is dependent upon the number of

Sequential vs. Parallel DFS

MPROC	Total Nodes	Bottom Nodes	Max Nodes
1	560	19	560
2 -PI	914	0	648
3	862	0	366
4	614	0	260
5	784	0	354
6	751	0	329
7	674	0	303
8	808	0	328
9	693	0	216
2-PII	1008	0	510
3	877	0	332
4	809	0	237
5	760	0	178
6	729	0	200
7	645	0	205
8	666	0	207
9	783	0	186

Table 7.1 - Sequential vs. Parallel DFS

Total Solution Time(sec) for Cycle Variation

	Cycle						
MPRO	1	5	10	15	20	25	30
2	8.01	4.69	3.94	4.35	3.84	4.14	4.36
3	5.32	3.10	2.92	2.35	2.56	2.69	2.48
4	4.06	1.82	1.38	1.87	1.86	2.36	1.69
5	4.07	2.50	2.00	2.03	1.52	1.36	1.47
6	3.03	1.88	1.16	1.15	1.25	1.15	1.98
7	2.56	1.34	1.91	1.10	1.50	0.73	0.83
8	2.94	1.64	1.75	1.51	1.38	1.66	0.53
9	2.11	1.84	1.44	1.55	1.09	0.81	1.06

Table 7.2 - Cycle Variation Performance

processors available. Although the processors are randomly assigned, the variables that are fixed are in order x_1 first, then x_2, x_3, etc. If the problems do not keep the processor busy, processors become idle while other processors are busy. To correct this unbalanced load, idle processors can request a NEWPROB. This extra communication overhead of passing problems to idle processors affects the total solution time as shown in Table 7.3. For small problems $< 2^{20}$, it is better to leave the load unbalanced since average node solution time is 0.007 seconds, and the number of total nodes solved will be small. In large problems, as shown for a 2^{36}, not reinitializing an idle processor takes away any performance gains.

To help identify unevenly loaded processors, the total number of nodes solved on each processor can be used. A set of problems can be run without reinitialization of processors to show how the tree was initially distributed with respect to work load. Figure 7.1 shows the workload of 7 processors solving a 2^{36} problem. The reinitialization strategies can then be varied to compare improvement in workload. The maximum number of nodes solved by one processor out of MPROC processors for 1 problem can be used to judge the improvement.

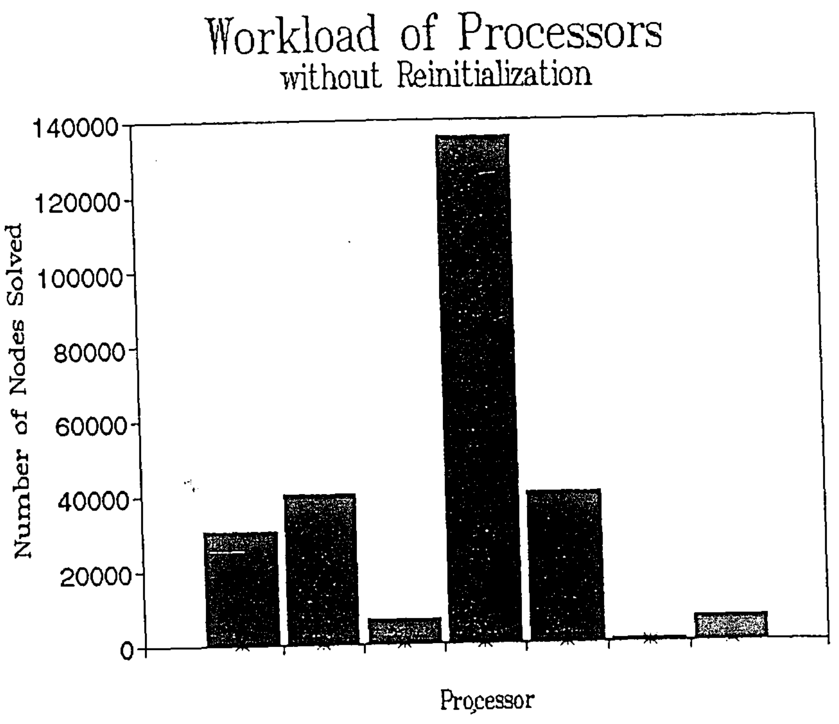

Figure 7.1 - Processor Workload

Total Solution Time(sec),
No Reinitialization

MPRO	NT=2	NT=3
Seq	0.68	258.1
2	0.39	495.7
3	0.4	332.4
4	0.33	397.1
5	0.4	443.5
6	0.28	454.1
7	0.14	457.3
8	0.24	443.5
9	0.31	431.2

Table 7.3 - Solution Time for No Reinitialization of Processors

CHAPTER 8

RESULTS

To test the various strategies proposed, a two facility problem was used where the planning period was varied from T=5 to T=20. Luss[2] developed a set of test problems for a heuristic and his parameters for the concave cost function and demands were used here. Fixed costs for both facilities equaled 4000, and the respective variable costs were 33 and 24. Three variations on demand were used:

1. $r_{1t}=r_{2t}=50$;
2. $r_{1t}=50,\ r_{2t}=30$;
3. $r_{1t}=50,\ r_{2t}=75$.

MPROC processors were varied from 2 to 9, and included a dedicated system manager.

Five experiments were set up which varied the strategies in choosing what problem to solve and reinitializing idle processors:

EX1: DFS to select a problem; No reinitialization of idle processors.
EX2: DFS to select and reinitialize.
EX3: DFS to select; BFS to reinitialize.
EX4: BFS to select and reinitialize.
EX5: Hybrid BFS/DFS to select; BFS to reinitialize.

Since the sequential implementation does not need a reinitialization strategy, only experiments of DFS, BFS, and BFS/DFS selection strategy were necessary.

As discussed in Chapter 7, a test set of N=2,T=12 for demand variation 1, was run on each experiment to determine optimal cycle length. By lengthening the cycle, the number of sub-optimal nodes solved increases since a processor does not read a NEWLB after each BBNODE solved. The number of bottom nodes solved increased for the DFS strategy-based experiments. Table 8.1 exhibits the total solution time and nodes solved for MPROC=5 versus a cycle length of 1 and the optimal cycle length chosen for each experiment.

Reinitializing the idle processor with a new problem involves some overhead which is not specifically quantifiable as the LINDA software does not provide this capability. However, results from total solution time show the decrease in performance as processors are reinitialized which is consistent for all problems. It is necessary to reinitialize the processor with a problem which will keep the processor busy for a long period of time. Unfortunately, this involves more work than simply popping the first problem of a busy processor's list as in the DFS approach of LIFO as done in EX2. To improve performance, the BFS reinitialization was implemented in EX3. When a processor is queried for a problem, it will employ a BFS problem selection strategy in picking out a problem for the processor. In this way, a processor has a better chance of staying busy longer, although there is no guarantee of that. Table 7.1 also demonstrates the maximum number of nodes solved by 1 processors for the 2^{24} with a BFS reinitialization. BFS reinitialization redistributes the load better and tends to minimizes the maximum number of nodes solved on 1 processor. Table 8.2 summarizes the results of this switch for MPROC=4. With these results, the DFS approach of re-initialization was abandoned. Although DFS reinitialization can provide a positive speed up for some values of MPROC, BFS consistently provides a better ratio for all values.

The remaining experiments, EX3-5, vary the selection strategy in choosing the next problem to solve within each processor. Figures 8.1-3 depict the total solution time

Cycle Length

	EX1		EX2		EX3		EX4		EX5	
	1	Opt=15	1	Opt=20	1	Opt=30	1	Opt=20	1	Opt=25
Total Time	6.23	3.72	3.46	2.32	4.07	1.47	4.71	1.62	4.79	0.95
Total Nodes	779	991	580	941	760	1011	460	549	503	609
Bottom Solved	0	9	0	9	0	14	0	1	0	0

Table 8.1 - Total Solution Time vs. Cycle Length

BFS vs DFS Reinitialization

	NT=20		NT=36	
	T	S	T	S
Seq	0.68	-	258.05	-
No Reinit	0.33	0.51	397.12	-0.53
DFS	0.7	-0.02	298.28	-0.15
BFS	0.41	0.39	122.12	0.52

T=total solution time(sec)

S=speed up ratio

Table 8.2 - BFS vs DFS Reinitialization

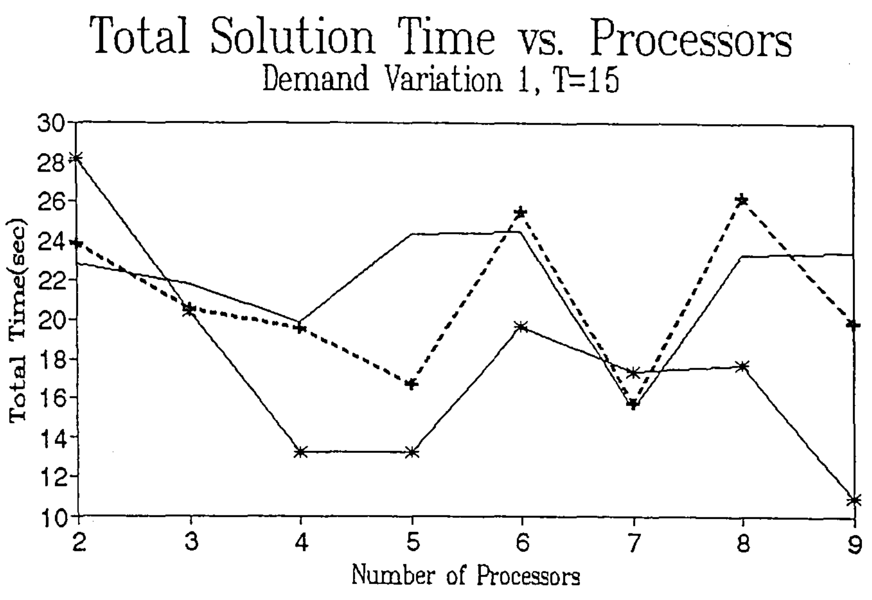

Figure 8.1 - Solution Time vs Processor, Variation 1

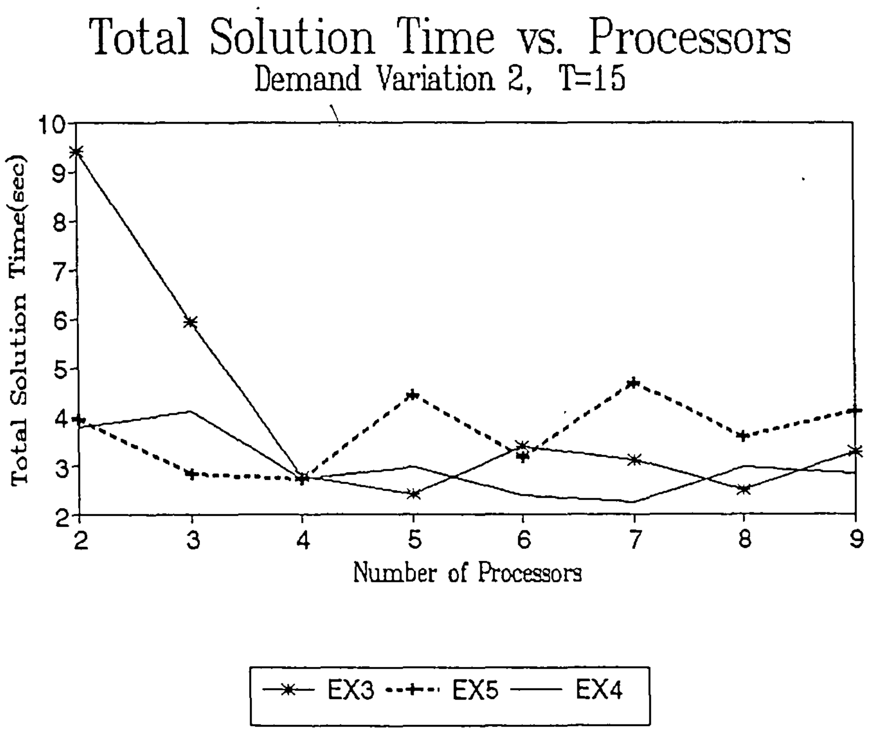

Figure 8.2 - Solution Time vs Processor, Variation 2

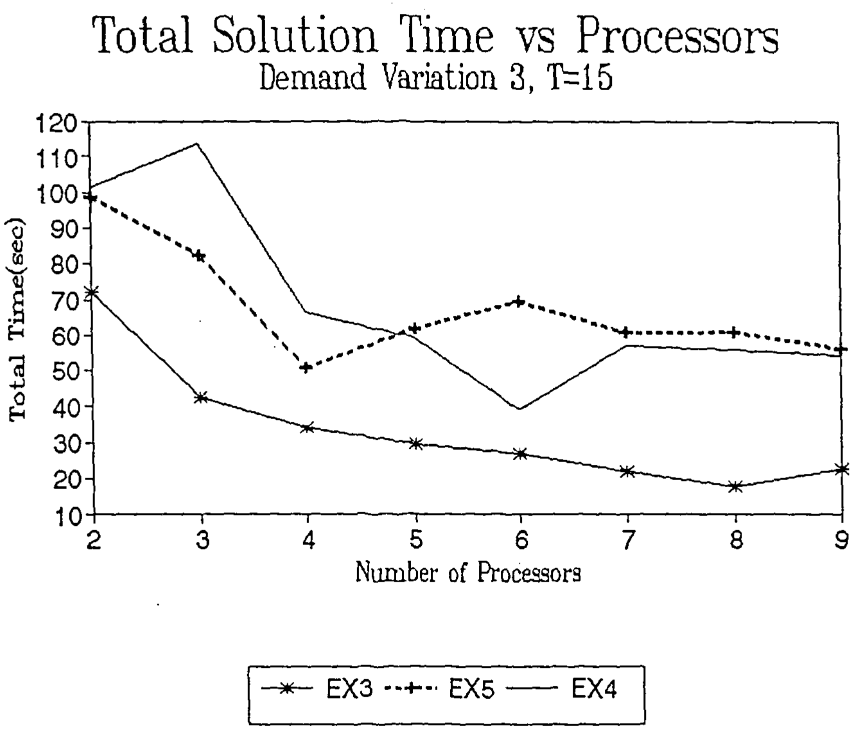

Figure 8.3 - Solution Time vs Processor, Variation 3

of the 3 strategies versus the number of processors for each demand variation when the planning horizon equals 15. EX3 outperforms the other 2 strategies for demand variations 2 and 3. For variation 1, EX4 does better overall. When the horizon is increased to 18 and 20 as in Figures 8.4-5, EX3 performs considerably better for MPROC > 5. In reviewing all figures, EX3 provides a solution quicker than EX4 or EX5 when the number of processors is not small. By looking at the maximum number of nodes solved by a processor for each demand variation, it can be shown why EX3 does not perform well sometimes. Table 8.3 shows that EX3 solves 3 times as many nodes than EX4 or EX5 when MPROC=2. As the number of processors working on a problem increases, the maximum nodes solved decreases for EX3. EX4 and EX5 do not decrease due to cycle lengths and the initialization of the processors.

Speed up ratios were calculated for EX3 versus the sequential time to solution. Figure 8.6-8.8 exhibit the ratio versus the number of processors assigned to the problem. When there are two processors to solve the problem, the communication overhead causes the parallel process to take longer which results in a negative speed up. Given the communication overhead involved in adding another processor and the problem with which it is initialized, it may not contribute positively to speed up. This can be seen in Figure 8.6 where adding a sixth processor decreases the speed up ratio. The speed-up then increases for additional processors from that point, and eventually surpasses the ratio when MPROC=5. In general, network noise can also contribute to decrease in performance, but cannot be measured by the Linda software.

The speed up curves are not expected to be strictly increasing. At some point, adding more processors to a specific problem may not yield any performance improvement. Since each processor must communicate it will decrease speed up. Since there were only 10 processors available for experimentation, a saturation point was not reached. The speed up curves cannot determine the optimal number of processors to assign to a problem. To do this, the network load must be monitored closely. It would not necessarily be correct to factor out the load from the speed up if these problems were to be run when the network has other traffic. The speed up ratio should reflect the conditions of the network when the problem is run.

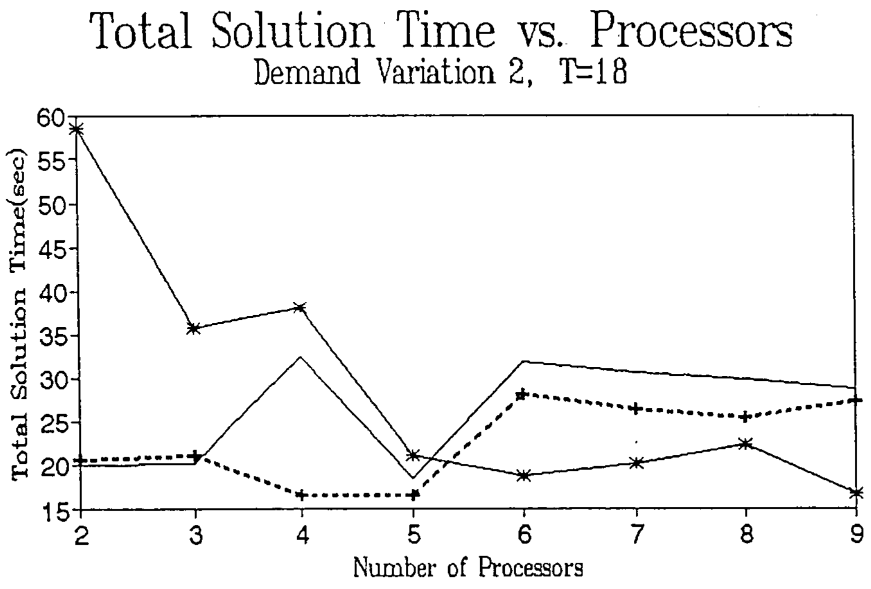

Figure 8.4 - Solution Time vs Processor, Variation 2, T=18

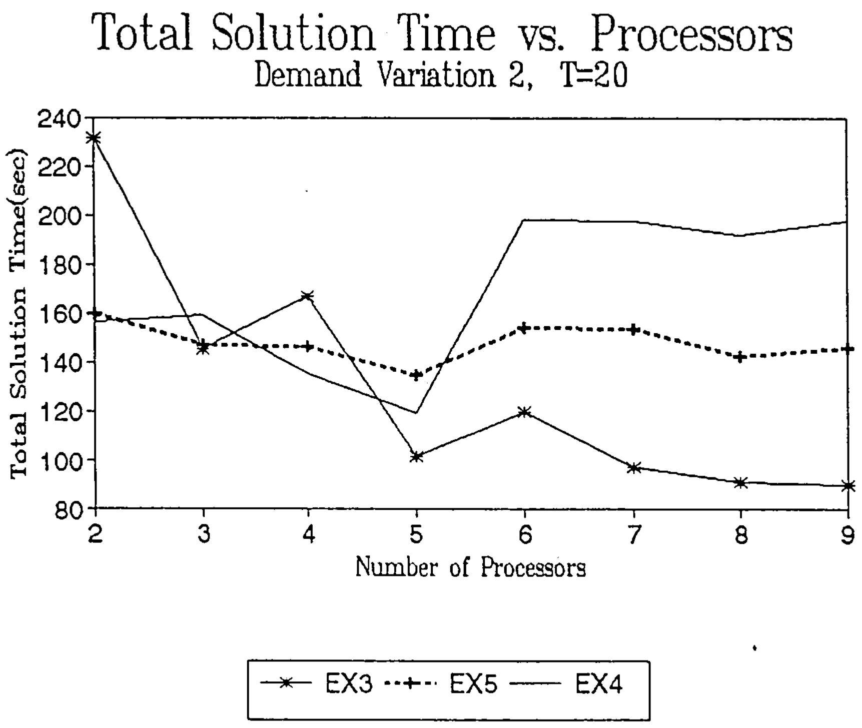

Figure 8.5 - Solution Time vs Processor, Variation 2, T=20

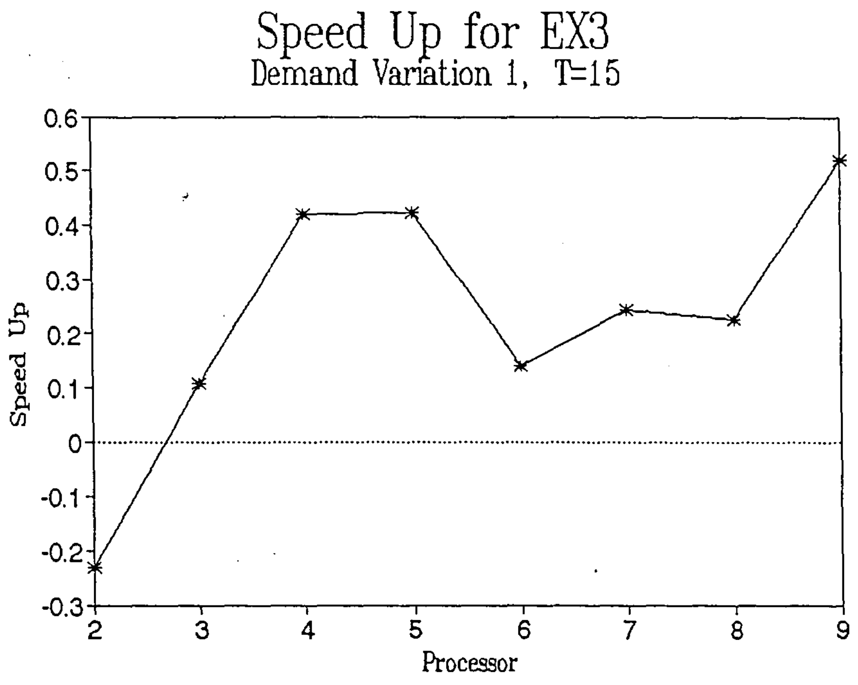

Figure 8.6 - Speed Up for EX3, Variation 1, T=15

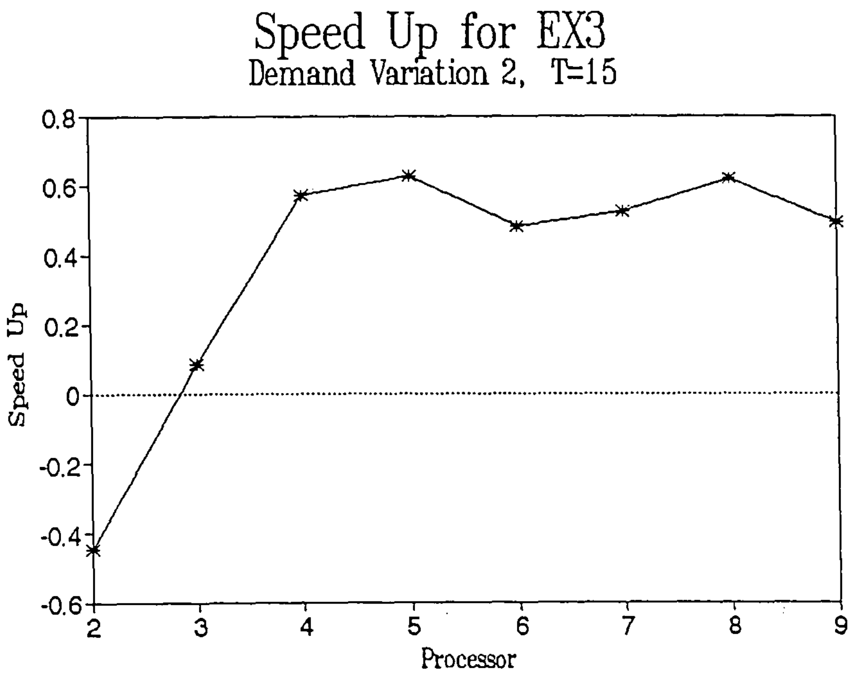

Figure 8.7 - Speed Up for EX3, Variation 2, T=15

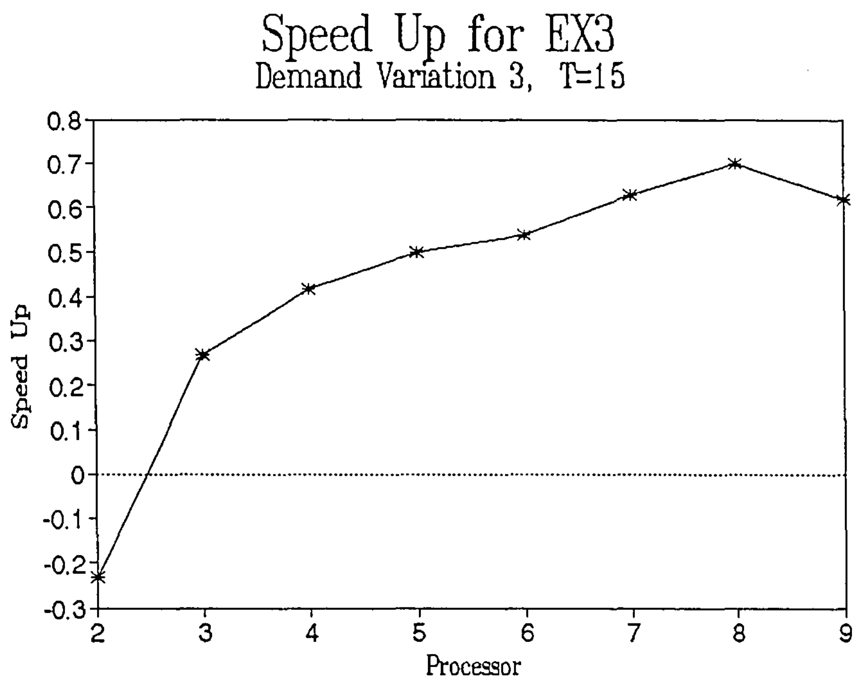

Figure 8.8 - Speed Up for EX3, Variation 3, T=15

Maximum Nodes Solved

MPRO	EX3	EX4	EX4
2	6197	2116	2053
3	4051	1982	2196
4	2574	2029	2132
5	2283	2118	2141
6	2417	2153	2140
7	1569	2153	2113
8	1580	2153	2141
9	1422	2153	2110

Table 8.3 - Maximum Number of Nodes Solved

CHAPTER 9

FUTURE RESEARCH

The results of this research using networked workstations are very promising. Future research will include writing network code tailored specifically for the problem of interest. Since the Linda software is generic as to the application, an unknown amount of unnecessary overhead may exist for a specific problem. Network code which is specific to a problem should improve performance over the Linda software. Solution times vary widely for a problem due to other traffic on the network, and it is difficult to judge the true application overhead with other traffic without a measure of network traffic. Specific code will gather statistics on network load at run time and the amount of time spent passing messages.

To improve performance at the processor level for the CEP, a better branching strategy should be investigated. Currently, the algorithm picks the next variable off the list to fix, but a more thorough analysis of the problem parameters could yield a more efficient strategy. For example, a sample problem where demand for facility 1 is less than for facility 2 requires more nodes to be solved than a problem where the opposite is true. Changing the branching strategy for this case based on demand may present better results.

Problems larger than 2^{40} should be run to better determine the speed up in a networked environment. In this research, sequential problem solution time for this size of problem over different strategies took no more than one hour, and small problems were solved in a matter of seconds. Nine processors assigned to a problem only provided the power of three processors due to communication overhead and network traffic. Since communication overhead clearly affects speed up, problems which take hours to solve sequentially may have a better speed up ratio, even for a small number of processors.

The load balancing of the processors should be examined as a way to make the algorithm more efficient. Load balancing can be looked at in two different places: the initialization of the processors, and the re-initialization of idle processors. The current algorithm simply fixes the variables in order based on the number of processors. The same research done for branching could be used here to determine which variables should be fixed first to minimize the number of nodes solved. If this is possible, the load may be better balanced. Initialization strategies should be varied against search and reinitialization strategies(EX1-EX5) to optimize the algorithm. Reinitialization of processors could be more efficient if it was passed a number of problems to be solved. This would increase the communication overhead, but if that number of problems could be varied as to the length of the list on the queried processor then the overall system would benefit. This is a fairly straightforward investigation, and once the overhead statistics are in place, performance could be measured accurately.

It is also possible that a problem may benefit from redundant processors. Redundant processors are initialized with the same part of the tree, at the same time. They should employ different branching and/or search strategies so that they are not solving the same nodes at the same time. Although the tree is parsed into smaller sections due to redundancy, it may fathom the tree quicker due to variations in strategies.

For example, one processor could perform DFS to get a good solution quickly while the other redundant one performs BFS to help fathom out sub-optimal solutions. By communicating frequently, these processors may find a good solution quickly and limit the number of sub-optimal nodes solved. A good tandem strategy should yield less overall nodes solved than by using BFS only, and limit the number of times it goes to the bottom of the tree as in a DFS strategy.

REFERENCES

[1] Lee, S-B., and Luss, H., "Multifacility-type Capacity Expansion Planning: Algorithms and Complexities", *Operations Research,* **35**, 249-253(1987).

[2] Luss, H., "A Capacity Expansion Model for Two Facility Types", *Naval Research Logistics Quarterly,* **26**, 291-303(1979).

[3] Luss, H., "Operations Research and Capacity Expansion Problems: A Survey", *Operations Research*, **30**, 907-947(1982).

[4] Luss, H., "A Multifacility Capacity Expansion Model with Joint Expansion Set-Up Costs", *Naval Research Logistics Quarterly*, **30**, 97-111(1983).

[5] Luss, H., "A Heuristic for Capacity Expansion Planning With Multiple Facility Types", *Naval Research Logistics Quarterly,* **33**, 685-701 (1986).

[6] Manne, A., *Investments for Capacity Expansion: Size, Location and Time-Phasing,* George Allen and Unwin, London.

[7] Verter, V., and Dincer, M.C., "An integrated evaluation of facility location, capacity acquisition, and technology selection for designing global manufacturing strategies", *European Journal of Operational Research,* **60**, 1-18 (1992).

[8] Zangwill, W., "Minimum Concave Cost Flows in Certain Networks", *Management Science*, **14**, 429-50 (1968).

APPENDIX A

The source code for the parallel and sequential branch and bound algorithms is on file with:

Dr. Louis J. Plebani
Lehigh University
Dept of Industrial Engineering
200 W. Packer Ave.
Bethlehem, PA 18015
215-758-4038

To obtain a copy of the source code, contact Dr. Plebani.

VITA

Margaret K. Mayer
8 Sixth Street
Frenchtown, NJ 08825
908-996-4912

DOB: 7/6/66
Place of Birth: New Haven, CT

EDUCATION

DOCTOR OF PHILOSOPHY CANDIDATE Expected December 1994
MASTERS OF SCIENCE DEGREE Expected May 1993
Department of Industrial Engineering
Lehigh University, Bethlehem, Pennsylvania

BACHELOR OF SCIENCE DEGREE May 1988
School of Operations Research and Industrial Engineering
Cornell University, Ithaca, New York

WORK EXPERIENCE

Instructor
Lehigh University January 1992 to May 1992
Taught a Software Tools course which presented the basics of Quattro Pro and C language. Responsibilites included preparation of lectures and creation of homework assignments and projects.

Grader
Lehigh University September 1991 to December 1991
Created and graded homework assignments for Engineering Economy and Inventory Control. Assisted professor in creation of exams.

Assistant Database Administrator
Greenwich Capital Markets, Greenwich, CT December 1988 to July 1991
Provided technical and analytical support to mortgage backed sales and trading efforts. Preparation for analysis included converting raw data to summary reports, using Sybase database, C language, and Unix operating system. Maintained system integrity of existing databases and designed new databases.

PUBLICATIONS

<u>STANDARD HANDBOOK OF PLANT ENGINEERING</u> Summer 1992
Edited Chapter 8, Materials Handling, as an independent consultant to K.W. Tunnell Co. Responsible for updating the text and graphics.

END

OF

TITLE