UNCLASSIFIED

ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 3A
7a NAME OF MONITORING ORGANIZATION	
c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code)
a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.

1. TITLE (Include Security Classification)

Distributed Processing on Link Enhancement

2. PERSONAL AUTHOR(S)

Wu, Tsung-Li

3a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1992, September	69

6 SUPPLEMENTARY NOTATION

7 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
CRCS, 0/1-knapsack, UNIX, RPC, Distributed Computing System, Sun Workstation, Distributed Processing.

9 ABSTRACT (Continue on reverse if necessary and identify by block number)

Link enhancement is a classical optimization problem where a military network manager wishes to enhance his network's survivability and routability with a given budget and faces with a multitude of potential architectural configurations from which to choose. This problem is NP-complete and good heuristics do exist. However, heuristics are still computational intensive. Distributed processing of the problem uses multiple workstations to cooperatively solve the problem such that the network manager can make his decision faster than running the algorithms on a single computer. This thesis reports the experiences of using distributed computation and its benefit.

0 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
2a NAME OF RESPONSIBLE INDIVIDUAL ang, Chyan	22b TELEPHONE (Include Area Code) (408)6462081
	22c OFFICE SYMBOL EC/Ya

D Form 1473, JUN 86 *Previous editions are obsolete.*
S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Approved for public release; distribution is unlimited.

Distributed Processing on
Link Enhancement

by

Wu, Tsung-li
Captain, Taiwan R.O.C. Army
B.S., Chung Cheng Institute of Technology, Taiwan R.O.C.

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

ABSTRACT

Link enhancement is a classical optimization problem where a military network manager wishes to enhance his network's survivability and routability with a given budget and faces with a multitude of potential architectural configurations from which to choose. This problem is NP-complete and good heuristics do exist. However, heuristics are still computational intensive. Distributed processing of the problem uses multiple workstations to cooperatively solve the problem such that the network manager can make his decision faster than running the algorithms on a single computer. This thesis reports the experiences of using distributed computation and its benefit.

TABLE OF CONTENTS

ACKNOWLEDGMENT

I am grateful to my advisor: Prof. Yang, Chyan, who suggested me this topic and gave me so much guidance as I prepared my thesis.I am deeply indebted to my second reader, Prof. David Alan Schrady, who offered many valuable advice and suggestions as he reviewed my thesis. My thanks go to the chairman of Electronic Warfare Academic Group for his assistance and worthy opinion -- Prof. Jeffrey B. Knorr. Finally, I thank my wife Pai, Shu-hui for patiently proofreading my thesis.

I. INTRODUCTION

A. BACKGROUND

Network survivability has been studied mainly for the purpose of establishing fault-tolerant networks. Many researchers have studied the subject based on concepts in graph theory that relate to either spanning trees [Ref.1, 2] or cutsets [Ref.3]. To achieve the balance between affordability and survivability, parameterized by the number of circuits required, a linear programming model is also used to minimize total cost for a communication system [Ref.4]. The specific problem of each research may be different and the corresponding optimal solution to that problem can be proven intractable. Realizing the difficulty of the problems, several researchers have proposed heuristic approaches for solving network survivability problems [Ref.5, 6, 7].

It is common that an existing military network consists of many nodes, will contain some nodes that are directly connected with communication links while some of them have to communicate indirectly through intermediate nodes. Sometimes it is desirable to add communication links between nodes of a communication network enhancing the network routability and survivability [Ref.1, 2]. For an existing military network it is important and interesting to ask the

question: what is the optimal link enhancement for a given investment? Our purpose is to maintain a maximum communication network survivability and the performance requirement.

Suppose we have a table for all pairs of nodes which are not yet connected. For each pair of the nodes (i, j) we have the information about the costs (c_{ij}) of establishing a link between them. In addition, we know the value p_{ij} for the performance contribution when the link between the pair of nodes (i, j) is established. In the following discussions, the p_{ij} is called as profit that may simplify the usage of subscripts (since both "cost" and "contribution" start with "c"). The value p_{ij} can be thought of as the contribution of the link connecting node i and j, either in routability or survivability measures [Ref.1, 2]. Now,the question can be stated as follows. Given an investment or budget in dollar amount, B, what is the best strategy of networking link enhancement such that the overall network will have optimal routability and survivability? Managerial considerations usually give a maximum budget figure so that practically we are solving the link enhancement problem with maximum costs.

The link enhancement problem is NP-complete [Ref.7]. Generally speaking, we should find an algorithm that provides near-optimal solution and takes nonexponential time to compute [Ref.4]. In the CRCS (constrained range and reduced candidate set) algorithm [Ref.9], the most important computation is the

combinatorial part. To reduce the long running time it takes, the combinations are separated and distributed to four workstations instead of running it on a single machine. Taking advantage of the technology of Local Area Networks (LANS), this computation can be improved by computer networking and distributed computing.

In the following discussions we assume that each link selection is independent of the others except the total budget B gets reduced. This is a reasonable assumption since usually we select only a small set (from 1 to 6) of links and the assumption will simplify the algorithm to be presented. In reality, after each link selection, we have to update the p_{ij} for the entire network before the next selection takes place. However, the computation of p_{ij} is beyond the scope of this thesis and its computation time should not affect our results.

B. GOALS AND OBJECTIVES

The primary goal of this thesis is to find methods that provide near-optimal solution and take less computing time for the NP-complete link enhancement problem.

C. THESIS OUTLINE

The linear search and CRCS algorithms are discussed in Chapter II. Some techniques and two examples are presented clearly. Chapter III described the methods used to apply to the previous algorithms to reduce the computing time. Issues and techniques in using multiple workstations are also discussed. The two

programs of the host and remote process for distributed approach are in appendices A and B respectively. Chapter IV summarizes conclusions based on this study.

II. LINEAR SEARCH AND CRCS ALGORITHMS

A computer network can be thought of as a graph G(V,E), where V represents the vertices (nodes) and E represents the edges (links) [Ref.5]. Suppose we have a table consisting of tuples of the form (i, j, c_{ij}, p_{ij}) where i and j are the node numbers in the network and c_{ij} is the cost to establish the link between nodes i and node j, the value p_{ij} is the contribution of this link enhancement. We are trying to find a solution for a given investment B such that $\sum c_{ij} \leq B$ and $\sum p_{ij}$ is maximized. We can describe a generic linear search algorithm with the following steps. As we have mentioned above, step 3 will not be included in subsequent discussions.

1. select (remove) a link from the set of candidate links; add this link to the current network.
2. $B = B - c_{ij}$;
3. update the network profile, i.e., compute p_{ij} for the links of the new network
4. stop if $B < c_{ij}$ for all links
5. go to step 1.

A. LINEAR SEARCH ALGORITHMS

Linear search algorithms have been reported and detailed examples can be found in [Ref.7]. Its basic idea is to search the sorted table of the set of candidate

links until the budget is exhausted. Depending on the keys used to address the sorted table there are several variants of linear search. A brief overview is given in this section.

1. One Way Linear Search Algorithms

There are three variations of the one-way linear search algorithm and they differ at step 1 in the ways they select a communication link. We first sort the table in nondecreasing order on the value of c_{ij} and extract the tuples with value of $c_{ij} \leq B$ to form a feasible solution set named FS. Traditional optimal solution for the knapsack problem can be done by adding a field of $r_{ij} = p_{ij} / c_{ij}$ to each tuple and sort the list in nonincreasing order of r_{ij}. We name this new list FS_r that consists of tuples of $(i, j, c_{ij}, p_{ij}, r_{ij})$. Without loss of generality, we can assign one link number to each node pair (i, j) to be considered, i.e., Table 1 shows only the link numbers instead of node pairs. Thus, in subsequent discussion the list FS_r consists of tuples of (k, c_k, p_k, r_k) with k as the link number. Note that the value of r_k effectively measures the contribution per dollar amount. The solution is simply a selection of links from the linear search of the list FS_r until B is exhausted or becomes insufficient. If divisibility is allowed this linear search algorithm, based on r_k, gives the optimal solution [4] for knapsack problems. However, this solution will not give the optimal solution in 0/1-knapsack or link enhancement problems.

Table 1: EXAMPLE OF 20 LINKS TO BE CONSIDERED AND THE CORRESPONDING PROFIT

Link number	Cost	Profit
1	1833	4140
2	1754	3506
3	1246	3819
4	1529	2310
5	2034	3370
6	2568	5276
7	1508	3859
8	1608	4477
9	1691	3269
10	2112	3807
11	1840	3661
12	1960	3560
13	2184	4440
14	2549	2899
15	2254	3643
16	2289	4224
17	1883	4368
18	1682	1922
19	1711	3844
20	1578	3484

From FS we can create two sorted lists FS_c and FS_p. FS_c is sorted by c_k in nondecreasing order while FS_p is sorted by p_k in nonincreasing order. Similar to the linear search we have just described above we can perform a linear search of list FS_c and select links one at a time until the budget is exhausted or becomes insufficient. Likewise, we can do a linear search of list FS_p and obtain the

selections. Since all three one-way linear search methods, based on FS_r, FS_c,or FS_p, are not optimal we can always construct examples that defeat them easily.

2. Two Way and Three Way Linear Search Algorithms

Instead of one-way search methods described above we may make decisions by observing the two lists jointly. For example, we may use FS_r and FS_c together to obtain the selections. We start the linear search separately on these two lists one link at a time. We use the voting scheme to select the link. Whenever we encounter a link such that the link has been visited in FS_r or FS_c the counter associate with the link is increased. When any counter reaches a preset threshold value, e.g. 2, this link is added to the network. The value B is updated by subtracting c_k of the candidate link. We continue the linear search until B is either exhausted or becomes insufficient. This counting method is called a voting algorithm since each link accumulates the votes from different lists until it gets enough votes. The preset threshold in a two-way linear search is set to 2 since each link can get a maximum vote of 2. If the threshold is set to 1, it reduces to a one-way search.

3. BGH - Beat Group of Heuristic

All the heuristic algorithms discussed above are greedy in nature, in that each sorts the FS in a certain order and allocates the available budget accordingly. Since sorting can be done in O(nlogn) time we can achieve our solution in

O(nlogn) time. However, from our study none of them consistently outperforms the others. It is natural to select the best among them, i.e., we can find seven sets of solutions and pick the best one of them. By doing this we select the best solution from a group of heuristic algorithms and we name it BGH. The idea of BGH has been used in Tirumalai and Butler [Ref.8] to select the best of 3 available heuristic algorithms for multiple-valued logic minimization. Unfortunately, even with 6 heuristic algorithms to choose from, the BGH cannot guarantee the optimal solution [Ref.7].

B. IMPROVEMENT BY CONSTRAINED RANGE AND REDUCED CANDIDATE SET

Two major improvements can be done over the linear search algorithms explained above: constrained range (CR) and reduced candidate set (RCS) [Ref.9]. The CR method is to constrain the solution search space in a feasible range which is determined by the available budget and the given costs of links. To constrain the range the method does not compromise the optimality; the method simply tightens the feasible space. The RCS method, however, does compromise in trading computation time for possibility of losing optimality. The combined method of CR and RCS is named CRCS.

1. Constrained Range and Squeezing The Constrained Range

Given a budget B, and costs of candidate links, c_i, we may find the optimum solution within a constrained range hence saving computational costs. Let C_{min} and C_{max} be the minimum and the maximum c_i respectively. Notice that with the given B we can readily compute the constraints: the upper limit, UL = $\lfloor B / C_{min} \rfloor$, and the lower limit LL defined as

$$LL = \lceil B / C_{max} \rceil \text{ if } FS_c(n-j) \leq r$$

$$LL = \lfloor B / C_{max} \rfloor \text{ if } FS_c(n-j) > r$$

where $r = B - \sum_{i=n-j+1}^{n} FS_c(i)$ and $j = \lfloor B / C_{max} \rfloor$.

LL indicates the number of links we can increase when all the budget is used for the links that each requires C_{max}. In a practical sense, this is the minimum number of links we can add. The ceiling in the LL expression represents the possibility that the leftover of B / C_{max} may be sufficient for yet another link. If this possibility is void, the floor option is chosen for conservative computation. UL, however, represents the number of links that we can increase when all of the budget is used for the links that each requires C_{min}. UL is the maximum number of links we can add. The floor in the UL expression represents the impossibility that the leftover of B / C_{min} can be used for any other link since no link costs less than C_{min}. The LL and UL give us the range that the optimum solution should lie:

[LL, UL] instead of [0, n]. In other words, the number of links we can add with the given budget is the constrained range [LL, UL]. In practice, once we obtain the solution from linear algorithms we can squeeze this constrained range further and greatly reduce the computation time.

With the solutions of linear search algorithms we can squeeze the constrained range for both RCS and exact optimal solution. The maximum number k that satisfies $\sum_{i=1}^{k} FS_p(i) < P_{linear}$ may squeeze the LL further: LL=max(LL, k+1). In other words, if the best k choices of $FS_p(i)$ cannot beat a similar scenario, the maximum number k that satisfies $\sum_{i=1}^{k} FS_c(i) <= B$ may squeeze the UL further; UL=min(UL, k). In other words, if the best k choices of $FS_c(i)$ is very close to B such that no more links can be added then we are sure that UL must be no greater than k.

2. Reduced Candidate Set

The philosophy behind the RCS is that if a link should be in the optimal solution set this link must have a high probability to be selected by one of the linear search algorithms. In other words, if we want to improve the result from the linear search algorithm all we need is to examine only those links that have been selected by linear search. This method, the RCS, rejects links that may or may not be in the optimal solution set therefore it may not reach the optimal solution.

Hopefully, the gain in this heuristic is justified by both shorter computation time and higher probability of reaching optimality.

3. Examples

Example 1 :

In Table 1 and 2, the C_{min} = 1246 and B = 7000 while C_{max} = 2568. Using the definitions above, j = [7000 / 2568] = 2 and r = 7000 - 5117 = 1883, $r < FS_c(n-j) = FS_c(18) = 2289$ hence LL=2 and UL=5. The constrained search range is [2,5]. Considering all possible combinations we need to try C(20,2) + C(20,3) + C(20,4) + C(20,5) = 21,679 choices. Since 2^{20} = 1,048,576 and with constrained range in [2,5] we need only 21,679 / 1,048,576 = 2.06% of original exhaustive computation time. This is a tremendous savings! A further squeeze can reduce both LL and UL to 4 therefore only C(20,4) = 4,845 iterations or 0.46% of the original computation time is needed. Note again the methods used to constrain the search range and to squeeze the range do not compromise the optimality.

Example 2 :

$RCS = \cup S_i = S_r \cup S_p \cup S_c \cup S_{rp} \cup S_{pr} \cup S_{rc} \cup S_{cr} \cup S_{pc} \cup S_{cp} \cup S_{rpc} \cup S_{rcp} \cup S_{crp} \cup S_{cpr} \cup S_{pcr}$

$\cup S_{prc} = \{1, 3, 4, 6, 7, 8, 13, 17, 19, 20\}$. The reduced candidate set has 10 links to be considered as opposed to the original n=20. Let's consider the case without squeezing methods first. Instead of 21,679 choices in Example 1 above we now have only C(10,2) + C(10,3) + C(10,4) + C(10,5) = 627 choices. The RCS

improved over the pure constrained range method by performing only 627 / 21,679 = 2.9% of the former computation. Comparing this to the brute force purely exhaustive optimum solution, the combined CRCS takes only 627 / 1,048,576 = 0.06% of the former computation. Considering the squeezing methods we can limit the computation within C(10,4) iterations to find the optimal solution. When both the optimal solution and the CRCS use the squeezing methods the CRCS method uses only C(10,4) / C(20,4) = 210 / 4845 = 4.33% computation time of that used by the optimal solution method. CRCS finds the best solution set of {3, 6, 7, 8} without the exhaustive search since these 4 links are in the reduced candidate set.

Table 2: FS_c, FS_p, FS_r

Link ID	FS_c	Link ID	FS_p	Link ID	FS_r
3	1246	6	5276	3	3065
7	1508	8	4477	8	2784
4	1529	13	4440	7	2559
20	1578	17	4368	17	2320
8	1608	16	4224	1	2259
18	1682	1	4140	19	2247
9	1691	7	3859	20	2208
19	1711	19	3844	6	2055
2	1754	3	3819	13	2033
1	1833	10	3807	2	1999
11	1840	11	3661	11	1990
17	1883	15	3643	9	1933
12	1960	12	3560	16	1845
5	2034	2	3506	12	1816
10	2112	20	3484	10	1803
13	2184	5	3370	5	1657

Table 2, Cont'd

15	2254	9	3269	15	1616
16	2289	14	2899	4	1511
14	2549	4	2310	18	1143
6	2568	18	1922	14	1137

III. DISTRIBUTED APPROACH

In a distributed system, processing activities may be located in more than one computer and the computers communicate over a network. The Host creates several processes to perform work concurrently. When a remote procedure is invoked, the caller is suspended, a message containing the arguments is constructed and passed to the remote machine, and the procedure is executed there. In the UNIX environment, however, a user can explicitly proceed the processes without blocking. This will be shown later in the algorithm flowchart. Workstation users can share information and other resources available in the network. File servers are computers running software to enable workstation users to share information. The Remote Procedure Calls (RPC) is the primary communication mechanism for distributed programs. It allows for accessing remote services and also for passing of parameters from the client to the server. The RPC command remote shell (rsh) in the host computer also uses the original login name in the remote computer. The command line entered as a parameter is sent to the server, at which time rsh connects the UNIX standard input and output channels stdin ,stdout and stderr of the newly initiated command with the process running locally by means of two Transmission Control Protocol (TCP) connections [Ref.11].

The system we use in this experiment consists of four Sun workstations (SPARC station IPX, 28.5 MIPS) and one file server. Each workstation has a 32-bit microprocessor with 16 Mbytes of RAM (see Figure 1). Sun's Network File System (NFS) is an extension of the UNIX operating system which provides a distributed file service base on network UNIX systems.

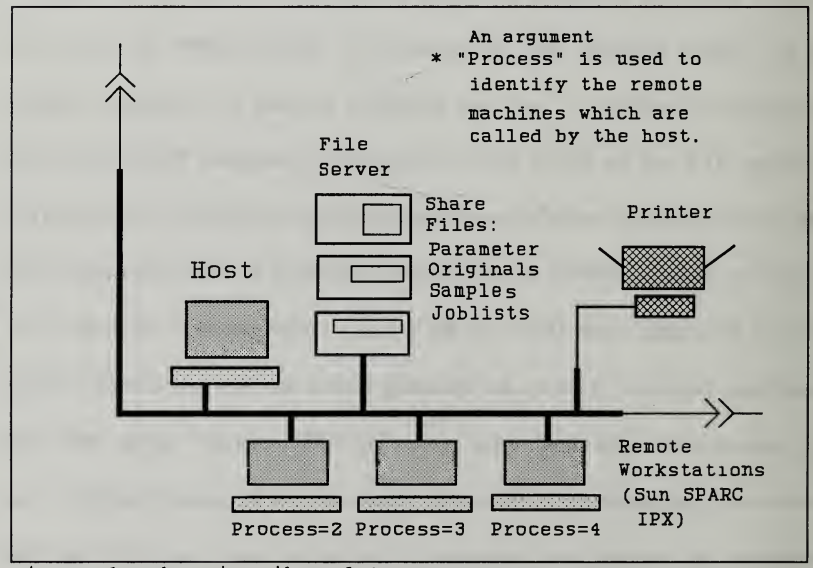

Figure 1 The Distributed System.

A. EXPERIMENT METHODS

Workload balancing is very important to minimize the idle time of each processor and running time of each case. Let N is the number of links, when N is

an even number, the peak of combination C(N, K) occurs at C(N, N/2) and it decays symmetrically. The smallest is C(N, N) or 1. When N is an odd number, the two largest workloads are C(N, (N-1)/2) and C(N, (N+1)/2), and it decays symmetrically from these two values [Ref.10] (see Figure 2 and Figure 3). The algorithm is outlined below.

```
IF    N is an even number
THEN  order[0]=N/2;          /* the largest job */
      j1=j2=1;               /* = C(N,N/2)      */
      FOR (j=1;  j<N-1 ;j++) {
      IF   (j+2) is an odd number{
      THEN order[j]=(N/2)-j1; j1++;
      ELSE order[j]=(N/2)+j2; j2++; }}
order[N-1]=N;  /* C(N,N)=1 is the smallest one */
ELSE
order[0]=(N-1)/2;
order[1]=(N+1)/2;   j1=j2=1;
      FOR (j=2; j<N-1; j++) {
      IF j is an even number {
      THEN order[j]=order[0]-j1;j1++;
```

ELSE order[j]=order[1]+j2;j2++; }}

order[N-1]=N;

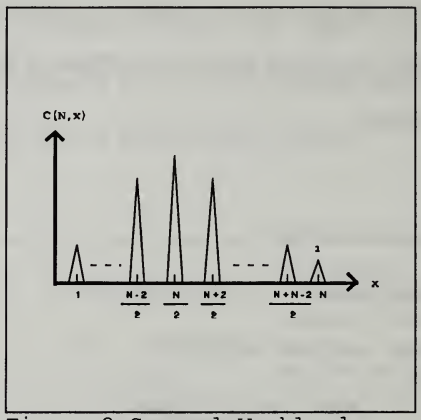

Figure 2 General Workload Distribution When N is an Even Number.

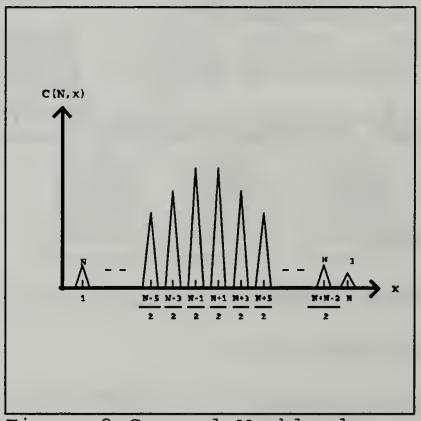

Figure 3 General Workload Distribution When N is an Odd Number.

A job-list is built for the four workstations from C(LN, N_{min}) to C(LN, N_{max}), and is sorted by the value of C(LN, k), where LN is the number of RCS and k is in [N_{min}, N_{max}]. The host does the largest job and the others are assigned in a wraparound order. For example, the fifth one is assigned to the 4th machine while the 8th one is assigned to the host. The rule of job-assignment (see Table 3) and an algorithm for a remote process is shown below. This assignment cannot guarantee the exactly balanced distribution but it is an easy implementation that approximates workload balancing. A remote processor is invoked by the remote shell command in the host program. In the meantime, the RCS, parameters which are necessary for computing and job-list are stored in shared files.

```
j=2;
FOR (a=1; a<=100; a++)  /* a<= some big number */
IF   j <= (Nmax-Nmin+1)
THEN IF LN >= job[j] /* The order of assignment*/
    THEN comb(LN,job[j]);
            IF (a+2) is an odd number
            THEN j=j+5;
            ELSE j=j+3;
ELSE break;
```

When the remote machines are invoked by **rsh**, an argument "process" will be passed from host via system command to identify the i.d. of the different processes, and an output file is created for each process respectively. The remote process executes a program named "share" to find the best solution. Each workstation writes its results to a separate output file (see Figure 1). When the host has handed over the assignment to remote machines, it continues its own job(see Figure 4).

Table 3: JOB ASSIGNMENT FOR THESE FOUR WORKSTATIONS.

Machine i.d.	Machine name	Jobs				
The host	SUN 3	*1	8	9	16	...
Remote-1	SUN10	2	7	10	15	...
Remote-2	SUN 2	3	6	11	14	...
Remote-3	SUN17	4	5	12	13	...

* The 1 means the largest combination job.

To synchronize the completion of these concurrent processes, an integer semaphore [Ref.12] is saved to a file and is used as a signal. This integer, a counting semaphore, is initialized to 0. The semaphore is incremented when a new remote processor is called and is decremented when the remote process

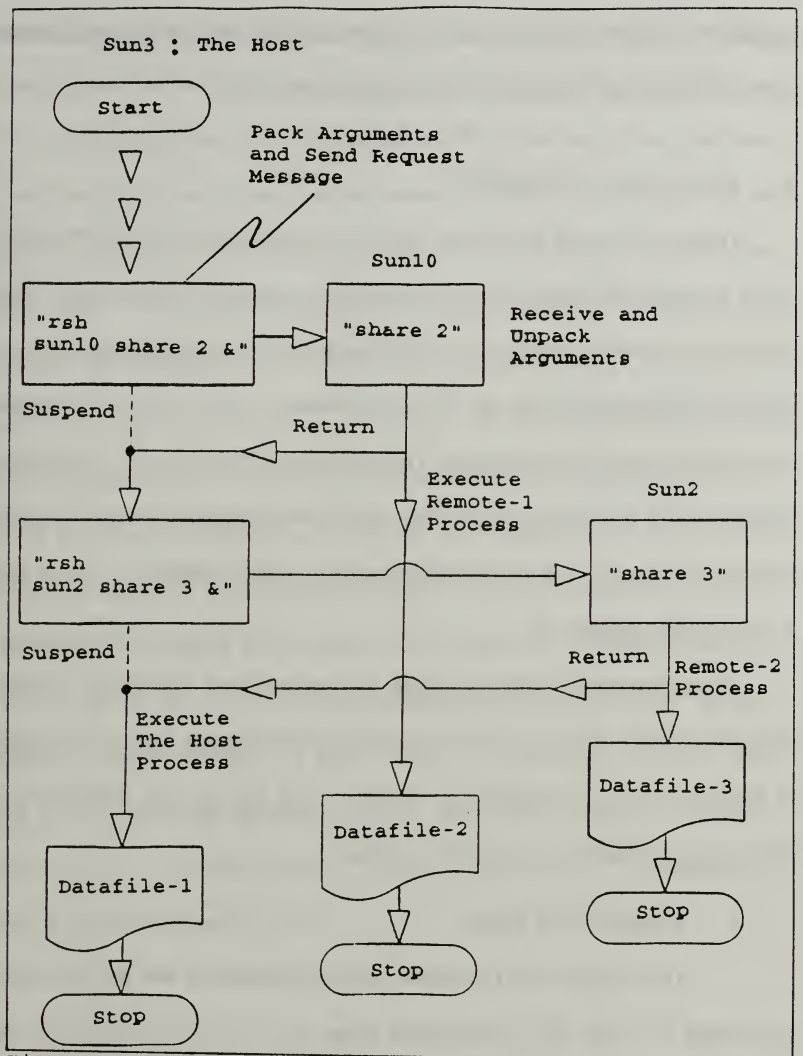

Figure 4 The Flow of Link Enhancement Algorithm of a Three-combination Case.

finishes. The UNIX command "time" is used to measure the user time, system time and real time of the host and the remote process [Ref.13].

B. ANALYSIS OF RESULTS

There are 62 out of the original 180 cases (n=30) tested in [Ref.9,14], and 65 out of the other 180 cases (n=31) which need more than one combination. These 127 (62+65) cases are tested in both distributed and non-distributed approaches in this thesis. When n=30, there are 38 two-combination cases, 22 three-combination cases, and 2 four-combination cases. The remaining 118 cases need 1 combination respectively. In the case of n=31, there are 40 two-combination cases, 19 three-combination cases, and 6 four-combination cases. The remaining 115 cases need 1 combination respectively.

This experiment is tested in a single user environment. We use the computer time (sum of user time and system time) instead of wall-clock time as the measure of comparison. For the distributed approach, each case is done when the host detects the completion of all remote machines.

1. Example of 30 Links

The average time (**computer time**) of distributed and non-distributed approaches for these 38 2-combination cases are 513.56 and 870.25 seconds respectively. The former one saves 41% of the time of the latter's, and about 46%

in some extreme cases (cases 8, 13, 18 and 23). For those 22 3-combination cases, the average times for the distributed and non-distributed approaches are 1058.5 and 2377.3 seconds respectively . We save about 55.5% of the time by the distributed approach and 60.6% in the extreme case (case 83). We need 4 machines for the 142nd case. Because the computing of this case is too simple, the distributed approach takes a longer time than the non-distributed approach (see Table 4 and the reason will be explained later). In the 127th case, the RCS={1, 3, 5, 6, 7, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 23, 25, 26, 27, 28, 30}, it takes 40 seconds by distributed approach or 12.8 seconds by the non-distributed approach to find the solution set {3, 20, 23, 25, 26, 27, 28, 30} and contribution of 48759 profit selected from RCS. To find the optimal solution, we use the entire set of links as the sample space instead of RCS. It takes 4586.3 seconds to reach the best solution set {3, 23, 24, 25, 26, 27, 28, 30} and contribution of 48784 profit by the non-distributed approach. However, it takes only 2478.7 seconds to reach the same results by the distributed approach. Thus we save 45.9% ((4586.3 - 2478.7) / 4586.3 = 45.9%) of the time in the 127th case. Tables 5,6 and 7 summarize the results of 30 links discussed above.

Table 4: THE COMPUTER TIMES (IN SECONDS) OF THE HOST PROCESS FOR DISTRIBUTED AND NON-DISTRIBUTED APPROACH (n=30).

Case	Distributed	Non-distributed
7	4.5	1.6
8	766.6	1423.5
12	6.0	1.9
13	767.8	1423.4
17	7.2	1.8
18	769.2	1423.8
22	8.6	2.1
23	770.2	1424.0
37	13.0	4.0
38	1478.7	2680.5
39	188.1	242.5
42	14.3	4.0
43	1478.6	2663.1
44	190.5	242.6
47	15.2	.9
48	1480.1	2685.3
49	190.4	242.8
52	16.3	.0
53	1481.6	2661.9
54	191.8	242.7
67	20.7	4.2
68	2762.2	4845.2
* 69	195.9	274.7
72	22.0	4.2
73	2760.8	4844.8
* 74	197.5	258.1
* 77	22.6	3.7
* 78	3157.6	7981.7
* 79	198.6	263.1
* 82	26.4	9.4
* 83	3172.2	8054.3
* 84	199.9	263.3
97	32.1	10.9
98	2120.4	3496.0
99	91.2	86.1
102	32.6	11.3
* 103	2769.4	6241.7
104	92.5	86.6

Table 4 (Cont'd)

106	29.3	2.7
107	31.3	6.5
* 108	2767.8	6241.6
* 109	205.9	260.6
111	30.1	3.0
* 112	31.2	5.5
* 113	2769.6	6264.6
* 114	206.5	263.6
126	34.3	.2
127	40.0	12.8
128	1432.4	2216.0
129	53.9	27.7
131	35.3	.3
* 132	38.6	10.0
* 133	2135.2	4313.4
134	55.3	27.7
* 136	36.8	3.3
+ 137	42.1	15.2
* 138	2131.6	4340.8
* 139	101.9	90.8
* 141	39.3	.6
142	43.8	15.1
* 143	2779.7	7061.1
* 144	103.5	91.5

---

* Cases which have 3 combinations.
+ Cases which have 4 combinations.

Average:
1.For "2-combination" cases
  -- 513.56 seconds for distributed approach.
  -- 870.25 seconds for non-distributed approach.
2.For "3-combination" cases
  -- 1058.5 seconds for distributed approach.
  -- 2377.3 seconds for non-distributed approach.

Table 5: DATA OF THE 127th CASE WHEN n=30.

Link number	Cost	Profit
1	1124	2969
2	2062	3568
3	398	3100
4	1183	2858
5	569	3696
6	1170	2907
7	739	2544
8	2377	3219
9	423	2947
10	1266	4024
11	1163	2921
12	1254	4025
13	496	3577
14	1090	2606
15	813	3919
16	913	3969
17	554	3675
18	1701	3877
19	877	2518
20	968	2533
21	1106	2623
22	1180	2871
23	422	2951
24	1027	2558
25	397	3204
26	396	3134
27	762	3887
28	623	2614
29	2023	3609
30	409	3009

Table 6: BEST SELECTION OF THE 127th CASE

	Linear	Optimal
Solution set	30 28 27 26 25 23 20 3	30 28 27 26 25 24 23 3
Contribution	48759	48784

Table 7: THE COMPUTER TIMES (IN SECONDS) OF THE 127th CASE FOR LINEAR AND OPTIMAL SOLUTIONS, USING 2 MACHINES(n=30).

	Linear	Optimal
Distributed	40.0	2478.7
Non-distributed	12.8	4586.3

2. Example of 31 Links

Table 8 shows that the average computation times required for n=31 cases are longer than the n=30 cases (see Table 4).When n=31 the average times of distributed and non-distributed approaches for those 40 two-combination cases are 698.64 and 1137.9 seconds respectively. The former one saves 38.6% of the time of the latter's and it is about 48.3% in one extreme case (case 53) . For those 19 three-combination cases, the average times of the distributed and non-distributed

approach are 2612.3 and 5854.5 seconds respectively . We save about 55.4% of the time by the distributed approach and 58.8% in the extreme case (case 68). For those 6 four-combination cases, the average times of the distributed and non-distributed approach are 474.83 and 932.3 seconds respectively . We save about 49% of the time by the distributed approach and 54.8% in the extreme case (case 143, see Table 8).

Generally, the distributed approach requires overhead in the file accessing, system calls, additional works for workload analysis, job assignment, etc. When computations are intensive, the overhead of the distributed processing, relative to effective computations, is reduced drastically.

Table 8: THE COMPUTER TIMES (IN SECONDS) OF THE HOST PROCESS FOR DISTRIBUTED AND NON-DISTRIBUTED APPROACH (n=31).

Case	Distributed	Non-distributed
37	14.7	.0
42	15.9	6.9
44	615.7	853.9
45	14.1	2.2
46	14.6	1.8
47	17.1	7.0
48	6499.6	12529.7
49	617.0	853.4
50	15.5	2.1
51	15.5	1.9
52	18.5	7.2
53	6487.4	12540.8

Table 8 (Cont'd)

	54	617.9	853.9
	55	16.6	2.1
	63	5618.3	8597.3
	67	22.1	6.9
*	68	6073.1	14743.9
	69	268.1	337.8
	72	23.4	6.9
*	73	6077.7	14712.3
	74	269.3	337.8
	77	24.8	7.1
*	78	6078.4	14688.6
	79	270.9	338.0
	80	22.9	2.5
*	82	28.7	18.1
*	83	6083.3	14657.2
	84	271.8	338.0
	85	24.5	2.7
	93	3632.0	5900.1
	97	72.3	85.9
*	98	5013.8	10899.2
	99	114.5	115.6
*	102	39.9	29.0
*	103	5042.9	10897.2
*	104	277.6	364.0
	105	29.7	3.2
	106	29.7	3.2
	107	41.3	40.0
*	108	5017.9	10915.2
	109	278.4	364.2
	111	30.6	2.9
+	112	42.7	40.1
*	113	5059.3	10897.3
*	114	279.3	364.3
	122	49.9	29.1
	123	848.8	1227.6
	126	35.2	3.3
	127	72.6	68.3
*	128	1435.6	2632.9
	129	60.7	35.2
	131	36.4	3.3
	132	45.4	22.2
*	133	1437.9	2633.0
*	134	124.3	122.4
	135	37.6	3.7
*	136	38.0	3.5

Table 8 (Cont'd)

* 137	46.8	22.5
* 138	1440.0	2631.2
+ 139	287.7	370.5
140	38.6	3.9
* 141	39.1	3.7
+ 142	50.6	33.1
+ 143	2138.6	4729.0
+ 144	288.1	381.1

* Cases which have 3 combinations.
+ Cases which have 4 combinations.

Average:
1.For "2-combination" cases
  -- 698.64 seconds for distributed approach.
  -- 1137.9 seconds for non-distributed approach.
2.For "3-combination" cases
  -- 2612.3 seconds for distributed approach.
  -- 5854.5 seconds for non-distributed approach.
3.For "4-combination" cases
  -- 474.8 seconds for distributed approach.
  -- 932.3 seconds for non-distributed approach.

IV. CONCLUSION

The distributed processing can reduce the computer time to finish the computation of link enhancement problems. A decision maker reaches the solution sooner than if the problem is solved in a single machine, especially when the computation is complicated. In complex cases, the more workstations the better. Optimality of the link enhancement problem may be reached by using the entire set of links as the sample space instead of using the RCS set. The combinations required for optimal solution will be in range [$C(n,N_{min})$, $C(n,N_{max})$] instead of in [$C(LN,N_{min})$, $C(LN,N_{max})$] and it will take longer time than using RCS. In this case, the distributed processing becomes necessary.

APPENDIX A PROGRAM OF THE HOST PROCESS

```
*******************************
  Program of The Host Process
*******************************

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <signal.h>

#define  MAXtry 101
#define  MAXdim 200
#define  MAXNUM 2147483647.00
#define  max(a,b)  (a>=b)?a:b
#define  min(a,b)  (a<=b)?a:b

float U1[MAXdim], U2[MAXdim];
int Rawcost[MAXdim], Rawprofit[MAXdim], Rawratio[MAXdim];

int skip, Idskip, Jdskip, Kdskip,counter=0;
int stop, Idstop, Jdstop, Kdstop;
int n, B, Cmin, Cmax, LN,process,SIGNAL;
long int Plinear;
long now;
char *time_ptr;
int Uc, Up, Vc, Vp;
int out[MAXtry], sample[MAXtry];
struct Original {
                int     index;
                int     cost;
                int     profit;
                int     ratio;
               } TABLE[MAXtry];
struct Sorted_vector {
                    int  value;
                    int  index;
                     };
```

```
struct solution {
                int    CostUsed;
                int    Contribution;
                int    id;
                 } FSopt;
struct solution FS3[7], FS2[7], FS1[4];
struct Sorted_vector Sort[4][MAXtry];

int order[MAXtry];
int job[MAXtry];

int Vote[MAXtry], Candidate[MAXtry];
int ctr, SolOpt[MAXtry], SolCurrent[MAXtry],
   SolFS3[7][MAXtry], SolFS2[7][MAXtry], SolFS1[4][MAXtry];
int currentPi, tmpCi, tmpPi;
char FSstring1[4][2] = {" ", "r", "p", "c"};
char FSstring2[7][3] = {"  ", "rp", "rc", "pr", "pc", "cr", "cp"};
char FSstring3[7][4] = {"   ", "rpc", "rcp", "prc", "pcr", "crp", "cpr"};
char Filename[4][11] = {"          ", "tmpSortedR","tmpsortedP", "tmpSortedC"};
FILE *fraw,*fraw2,*fpara,*fFLAG;
int Id, Jd, Kd;

main(argc, argv)
int argc;
char **argv;

{

float t, x, y, theta;
char command[40];

int NminPrime, NmaxPrime, Ntmp,N,ptr;
int sumPi, sumCi, Nmin, Nmax, i, j, k, I, J, K;
FILE *fin, *fid,*fsamp;

Uc=1000;
Up=3000;

system("rm  -f parameter*");
system("rm  -f sample* ");
```

```
system("rm  -f joblist* ");
system("rm  -f tmpSortedC ");
system("rm  -f tmpSortedP ");
system("rm  -f tmpSortedR");
system("rm  -f original* ");
system("rm  -f core");
system("rm  -f try ");

printf("BEGIN !");
/*  input file format
 Line 1 to n:  index,  cost, profit, ratio
 execution example: distLN 31 try 80 0  (skip 80 iterations for n=31, try is a temp
file, 'distLN' is the object code of distLN.c; 0 at the end means default stop;  if
stop=82 then the   program should run the 81st case since it skips 80 cases and
stop at 82
*/

n = atoi(argv[1]);
/* print the start time -- wall clock */
now=time(NULL);
time_ptr=ctime(&now);
printf(" Distributed ( LN ):TIME Start for  n=%d at %s\n",n, time_ptr);

system("rm -f dLNdata*");
skip = atoi(argv[3]);
stop = atoi(argv[4]);
Idskip = skip/30;    /* Id and Jd each loops 6 times Kd 5 times */
Jdskip=(skip%30) / 5;
Kdskip= skip - (Idskip*30 + Jdskip*5);

Idstop = stop/30 + 1;
Jdstop= (stop%30) / 5 +1;
Kdstop= stop - ((Idstop-1)*30 + (Jdstop-1)*5) +1;

if(stop==0){Idstop=Jdstop=6; Kdstop=5;} /* default */
if(Idstop > 6 || (Jdstop > 6) || (Kdstop > 5))
    {printf("stop error");
     printf("Skips %d %d %d Stops %d %d %d \n", Idskip,Jdskip, Kdskip, Idstop,
Jdstop,Kdstop);
    exit(1);}
```

```
 fraw=fopen("DLNDATA","w");
 fprintf(fraw,"\nIdskip and Jdskip %d %d\n",Idskip, Jdskip);
 fclose(fraw);

 SIGNAL=0;
 FFLAG=fopen("flag","w");
 fprintf(FFLAG,"%d",SIGNAL);
 fclose(FFLAG);

/* generate data: Id, Jd, Kd are loop indexes */

for(Id=1; Id<=6; Id++){
if(Id< 6) Vc=0.1*Id*Uc;
if(Id== 6) Vc=0.01*Id*Uc;

 for(Jd=1; Jd<=6; Jd++){
 if(Jd< 6)  Vp=0.1*Jd*Up;
 if(Jd==6) Vp=0.01*Jd*Up;

  for(Kd=1; Kd<=5; Kd++){
  B=0.1*(2*Kd -1)*n*Uc;

  if(Id==Idstop && (Jd==Jdstop) && (Kd==Kdstop)) {
  printf("\ndone\n");
  now=time(NULL);
  time_ptr=ctime(&now);
  printf("TIME ALL DONE for n=%d at %s\n",n, time_ptr);

exit(0);                                }


system("rm  -f tmpSortedC ");
system("rm  -f tmpSortedP ");
system("rm  -f tmpSortedR");


  counter++;
  fraw=fopen("DLNDATA","a");
  fprintf(fraw,"\n  Vc=%d    Vp=%d    B=%d  ------------------>>>        CASE
%d\n",Vc,Vp,B, counter);
```

```
  printf("\n\n\n\n Vc=%d  Vp=%d  B=%d ------------------>>>    CASE %d\n",
Vc,Vp,B,counter);

  now=time(NULL);
  time_ptr=ctime(&now);
  fprintf(fraw,"This case for n=%d begin at %s\n", n,time_ptr);

  /* generate cost */
  srandom(1);

   for(i=1; i<=n; i++) U1[i] = random()/MAXNUM;
   /* generate profit (contribution) */
   srandom(1);

    for(i=1; i<=n; i++) U2[i] = random()/MAXNUM;
    fin=fopen(argv[2],"w");

     for(i=1; i<=n; i++){
     t = sqrt(-2.0 *log(U1[i]));
     theta = 6.28 *U2[i];
     x= Vc* t* cos(theta) + Uc;
     y= Vp* t* sin(theta) + Up;
     fprintf(fin,"%d %d %d %d\n",i, (int)x, (int)y, (int)(1000*y/x));
   /* print for reference */
     fprintf(fraw,"%d %d %d %d\n",i, (int)x, (int)y, (int)(1000*y/x));
                    }
     fclose(fin);

     if( (Id <=Idskip) ||
       ( (Id==(Idskip+1)) && (Jd <=Jdskip))  ||
       ( (Id==(Idskip+1)) && (Jd==Jdskip+1) && (K <=Kdskip)))
       {
     fprintf(fraw,"Id=%d Jd=%d Kd=%d \n", Id, Jd, Kd);
     fclose(fraw);
     continue;
       }

/*done: above is generateData(argv[2]); */

  now=time(NULL);
```

```
  time_ptr=ctime(&now);
  fprintf(fraw,"Distributed (LN):This case for n=%d begin at %s\n",n,time_ptr);

  strcat(strcpy(command,"preprocess "),argv[2]);
  system(command);
  fin = fopen(argv[2], "r");  /* original data */

  for(i=1; i<= n; i++) {
  fscanf(fin,  "%d  %d  %d  %d",  &TABLE[i].index,  &TABLE[i].cost,
&TABLE[i].profit, &TABLE[i].ratio);
                    }
  fclose(fin);

  for(I=1; I<=3; I++){  /*sorted data 1/2/3=r/p/c */
  fid = fopen(Filename[I], "r");
  for(i=1; i<= n; i++) {
  fscanf(fid, "%d %d", &(Sort[I][i].value), &(Sort[I][i].index));
                    }
  fclose(fid);
                  }

/* ========================================================

   Three one-way search algorithms FSc, FSp, and FSr are performed here

======================================================== */

   for(I=1; I<=3; I++){

   for(i=1; i<=n; i++) SolFS1[I][i]=0;
   tmpCi= tmpPi = 0;
   j=1;

   for(i=1; i<=n; i++) {
   if(tmpCi + TABLE[Sort[I][i].index].cost <= B) {
      SolFS1[I][j++] = Sort[I][i].index;
         tmpCi += TABLE[Sort[I][i].index].cost;
         tmpPi += TABLE[Sort[I][i].index].profit;
                                        }
         FS1[I].CostUsed = tmpCi;
```

```
        FS1[I].Contribution = tmpPi;
                    }


/* ================================================
   Let's do two-way search algorithms for
   FSrp, FSrc, FSpc, FSpr, FScr, FScp
   ================================================ */

  ctr=0;

   for(I=1; I<=3; I++){

    for(J=1; J<=3; J++){
    if(J !=I) {
    ctr++;

     for(i=0; i<=n; i++) {Vote[i]=0; SolFS2[ctr][i]=0;}
     j=1;
     tmpCi= tmpPi =0;

     for(i=1; i<=n; i++) {
     Vote[Sort[I][i].index]++;
     Vote[Sort[J][i].index]++;
     if(Vote[Sort[I][i].index] == 2 || Vote[Sort[J][i].index] == 2){
     if(Vote[Sort[I][i].index]  ==  2  &&
(tmpCi+TABLE[Sort[I][i].index].cost<=B))
       {
      SolFS2[ctr][j++] = Sort[I][i].index;
      tmpPi += TABLE[Sort[I][i].index].profit;
      tmpCi += TABLE[Sort[I][i].index].cost;
       }
      else {
      if(Vote[Sort[J][i].index]  ==  2  &&
(tmpCi+TABLE[Sort[J][i].index].cost<=B))
       {
      SolFS2[ctr][j++] = Sort[J][i].index;
      tmpPi += TABLE[Sort[J][i].index].profit;
      tmpCi += TABLE[Sort[J][i].index].cost;                    }
          }
```

```
                                                    }
                    }

  FS2[ctr].CostUsed = tmpCi;
  FS2[ctr].Contribution = tmpPi;

          }
                  }
                  }  /* I,J,K */


/* ======================================================
    3-way search voting algorithms, 3x2x1 permutations!
====================================================== */

 ctr=0;
  for(I=1; I<=3; I++)  {
   for(J=1; J<=3; J++)  {
   if(J !=I)            {              /* do k */
    for(K=1; K<=3; K++)   {
    if((K !=I) && (K!=J))  {
    ctr++;

     for(i=0; i<=n; i++) {Vote[i]=0; SolFS3[ctr][i]=0;}

    j=1;
    tmpCi= tmpPi =0;

     for(i=1; i<=n; i++) {

      for(k=1; k<=3; k++) Vote[Sort[k][i].index]++;
      if(Vote[Sort[1][i].index]  ==  2  ||
Vote[Sort[2][i].index] == 2 ||
        Vote[Sort[3][i].index] == 2) {
      if(Vote[Sort[I][i].index]==2  &&
(tmpCi+TABLE[Sort[I][i].index].cost<=B))
       {
      SolFS3[ctr][j++] = Sort[I][i].index;
      tmpPi += TABLE[Sort[I][i].index].profit;
      tmpCi += TABLE[Sort[I][i].index].cost;
```

```
        }
       else      {
       if(Vote[Sort[J][i].index]==2  &&
(tmpCi+TABLE[Sort[J][i].index].cost<=B))
         {
       SolFS3[ctr][j++] = Sort[J][i].index;
       tmpPi += TABLE[Sort[J][i].index].profit;
       tmpCi += TABLE[Sort[J][i].index].cost;
         }
       else  {
       if(Vote[Sort[K][i].index]==2  &&
(tmpCi+TABLE[Sort[K][i].index].cost<=B))
         {
       SolFS3[ctr][j++] = Sort[K][i].index;
       tmpPi += TABLE[Sort[K][i].index].profit;
       tmpCi += TABLE[Sort[K][i].index].cost;
         }
            }
              }
                   }

                        }

     FS3[K].CostUsed = tmpCi;
     FS3[K].Contribution = tmpPi;

                         }     /* real work loop */
                       }
                      }
                     }
                    }

/* ===================================
   Find the Best of the Linear Search
   =================================== */

  FS1[0].Contribution = FS2[0].Contribution = FS3[0].Contribution =0;

   for(I=1; I<=3; I++) {
   if(FS1[I].Contribution > FS1[0].Contribution) FS1[0].id =I;
```

```
    FS1[0].Contribution=max(FS1[0].Contribution, FS1[I].Contribution);
                    }

  for(I=1; I<=6; I++) {
  if(FS2[I].Contribution > FS2[0].Contribution)  FS2[0].id =I;
    FS2[0].Contribution=max(FS2[0].Contribution, FS2[I].Contribution);
                    }

  for(I=1; I<=6; I++) {
  if(FS3[I].Contribution > FS3[0].Contribution)  FS3[0].id =I;
    FS3[0].Contribution=max(FS3[0].Contribution, FS3[I].Contribution);
                    }
  I=1;
  if(FS1[0].Contribution < FS2[0].Contribution) I=2;
  if( (I==1) && (FS1[0].Contribution < FS3[0].Contribution))     I=3;
  if( (I==2) && (FS2[0].Contribution < FS3[0].Contribution))     I=3;

   for(i=1;i<=n;i++) SolOpt[i]=0;
   fprintf(fraw,"\n The Best of the Linear Search is:\n");

   switch(I)
    {
   case 1: /* the best is from 1-way */

    for(i=1; i<=n && SolFS1[FS1[0].id][i] !=0 ; i++)
                                    {
    fprintf(fraw,"%d ",SolFS1[FS1[0].id][i]);
          SolOpt[i]=SolFS1[FS1[0].id][i];}
    fprintf(fraw,"\n");
    fprintf(fraw,"CostUsed= %d Contribution= %d from
FS%s\n",FS1[FS1[0].id].CostUsed,
          FS1[0].Contribution, Fsstring1[FS1[0].id]);
    Plinear= FS1[0].Contribution;
    break;

   case 2:
    for(i=1; i<=n && SolFS2[FS2[0].id][i] !=0 ; i++)
                                    {
    fprintf(fraw,"%d ",SolFS2[FS2[0].id][i]);
          SolOpt[i]=SolFS2[FS2[0].id][i];}
```

```
    fprintf(fraw,"\n");
    fprintf(fraw,"CostUsed= %d Contribution= %d from FS%s\n",
                FS2[FS2[0].id].CostUsed,
                FS2[0].Contribution, Fsstring2[FS2[0].id]);
    Plinear= FS2[0].Contribution;
    break;

   case 3: /* from 3-way */
    for(i=1; i<=n && SolFS3[FS3[0].id][i] !=0 ; i++)
                                        {
    fprintf(fraw,"%d ",SolFS3[FS3[0].id][i]);
          SolOpt[i] = SolFS3[FS3[0].id][i];}
    fprintf(fraw,"\n");
    fprintf(fraw,"CostUsed= %d Contribution= %d from
                FS%s\n",FS3[FS3[0].id].CostUsed,
                FS3[0].Contribution, Fsstring3[FS3[0].id]);
    Plinear= FS3[0].Contribution;
    break;

  default: fprintf(fraw,"impossible!\n");
         break;
    }

/* ============================
   Find the union of candidates
   ============================ */

  for(i=1; i<=n; i++)  Candidate[i]=0;

   for(I=1; I<=3; I++) {
    for(i=1; i<=n; i++)        {
    if(SolFS1[I][i] !=0 && Candidate[SolFS1[I][i]] ==0)
    Candidate[SolFS1[I][i]] =1;
    if(SolFS1[I][i] ==0) break;}
                    }

   for(I=1; I<=6; I++) {
    for(i=1; i<=n; i++)        {
    if(SolFS2[I][i] !=0 && Candidate[SolFS2[I][i]] ==0)
    Candidate[SolFS2[I][i]] =1;
```

```
    if(SolFS2[I][i] ==0) break;}
                    }

   for(I=1; I<=6; I++) {
    for(i=1; i<=n; i++)        {
    if(SolFS3[I][i] !=0 && Candidate[SolFS3[I][i]] ==0)
    Candidate[SolFS3[I][i]] =1;
    if(SolFS3[I][i] ==0) break;}
                    }

/* ==============================
   Union of all linear candidates
   ============================== */

  LN=1;
  fsamp=fopen("sample1","w");
   for(i=1;i<=n;i++) if(Candidate[i]==1) {
   fprintf(fsamp,"%d %d\n ",LN,i);
   sample[LN++]=i;
                                    }
  LN--;
  fclose(fsamp);

/* verified above this line */
/* ================================================ */
/* start the phase 2 computation */


  phase2();

  now=time(NULL);
  time_ptr=ctime(&now);
  printf("\n\nLinear Done for  n=%d at %s",n, time_ptr);
  fprintf(fraw,"\nDistributed (LN) :Linear Done for  n=%d at %s\n",n, time_ptr);
  fclose(fraw);

  for(i=1; i<=2; i++)  {
   FFLAG=fopen("flag","r");
   fscanf(FFLAG,"%d",&SIGNAL);
   fclose(FFLAG);
```

```
   if( SIGNAL == 0 ) break;
   i--;                }

  }
 }
}
                /* Id, Jd, Kd */

} /* end of main */

/* * *
 * * *
 * * */


phase2()
{
int Ntmp,Nmin,Nmax,NminPrime,NmaxPrime;
int sumPi,sumCi,i,j,k,j1,j2,s,a;
FILE *fjob1;
char command1[100],tail2[100];



Cmin = Sort[3][1].value;
Cmax = Sort[3][n].value;
Nmin = B/Cmax;
Nmax = B/Cmin;


 printf("\nBefore: Nmin=%d , Nmax=%d , n=%d
\n",Nmin,Nmax,n);
 fprintf(fraw,"\Nbefore: n=%d ,Nmin=%d ,
Nmax=%d\n",n,Nmin,Nmax);


sumPi=0;
 for(i=1; i<=Nmax; i++) {
 if(sumPi+Sort[2][i].value <= Plinear) {
    sumPi = sumPi + Sort[2][i].value;
```

```
                                    }
            else break;}

NminPrime = i;   /* for i-1 best choices of Pi cannot beat Plinear then we are
                    sure the minimum number of links is i */

if(NminPrime > Nmin) Nmin = NminPrime;

/* ========================================================== */

sumCi=0;         /* for i best choices of Ci cannot exceed the B and the UL cannot
                    be more than number of links which consists of these Ci */


 for(i=1; i<=Nmax; i++) {
 if(sumCi+Sort[3][i].value <= B) {
 sumCi = sumCi + Sort[3][i].value;
                                  }
 else break;              }

 NmaxPrime = i -1;
 if(NmaxPrime < Nmax) Nmax = NmaxPrime;
 Nmin=min(Nmin,Nmax);
 Nmax=max(Nmin,Nmax);
 printf("After:  Nmin=%d , Nmax=%d , LN=%d \n", Nmin,  Nmax, LN);
 fprintf(fraw,"After:Nmin=%d  Nmax=%d  LN=%d
Plinear=%d\n",Nmin,Nmax,LN,Plinear);
 currentPi=Plinear;
 for(i=1; i<=n; i++) {SolCurrent[i]=0;}
 if ( LN%2 == 0 ) {           /* workload analysis */
 order[0]= LN/2;              /* the largest one */
 j1=j2=1;
  for (j=1; j<LN-1; j++) {
  if ( (j+2)%2 !=0 ) {
  order[j]=(LN/2)-j1 ;
  j1++;                }
  if ( (j+2)%2 ==0 ) {
  order[j]=(LN/2)+j2;
  j2++;                }
                        }
```

```
order[LN-1]=LN;  }
if ( LN%2 != 0 ) {
order[0]=LN/2;
order[1]=(LN/2)+1;
j1=j2=1;

 for (j=2; j< LN-1; j++) {
 if (j%2 ==0)
    {order[j]=order[0]-j1;j1++;}
 if (j%2 !=0)
    {order[j]=order[1]+j2;j2++;}
                        }
 order[LN-1]=LN;
                }

 printf("\nWe have %d job(s) to do: ",Nmax-Nmin+1);

 k=1;
 fjob1=fopen("joblist1","w");
 for(s=0; s<=LN-1; s++)       {
 if ( (order[s]-Nmin)*(order[s]-Nmax) <= 0 ) {
 job[k]=order[s];
 fprintf(fjob1,"%d %d\n",k,job[k]);
 k++;                                    }

 if ( k>(Nmax-Nmin+1)) break; }
                              fclose(fjob1);

 for(k=1; k<=Nmax-Nmin+1; k++)
 printf("C(%d %d) ",LN,job[k]);

fpara=fopen("parameter","w");
fprintf(fpara,"\n%d %d %d %d %d %d %d %d %d,
Vc,Vp,B,n,Nmax,Nmin,Plinear,LN, counter);
fclose(fpara);

if( (Nmax-Nmin) > 0 )
{
SIGNAL++;
FFLAG=fopen("flag","w");
```

```
fprintf(FFLAG,"%d",SIGNAL);
fclose(FFLAG);

system("cp sample1 sample2");
system("cp joblist1 joblist2");
system("cp try original2");
system("rsh sun10 time share 2 >> dLNdata2  &");

  now=time(NULL);
  time_ptr=ctime(&now);
  fprintf(fraw,")--> Send job(s) to sun10 at %s\n", time_ptr);
}

if( (Nmax-Nmin) > 1 )
{
SIGNAL++;
FFLAG=fopen("flag","w");
fprintf(FFLAG,"%d",SIGNAL);
fclose(FFLAG);

system("cp sample1 sample3");
system("cp joblist1 joblist3");
system("cp try original3");
system("rsh sun2 time share 3  >> dLNdata3 &");

  now=time(NULL);
  time_ptr=ctime(&now);
  fprintf(fraw,")--> Send job(s) to sun2 at %s\n", time_ptr);
}

if( (Nmax-Nmin) > 2 )
{
SIGNAL++;
FFLAG=fopen("flag","w");
fprintf(FFLAG,"%d",SIGNAL);
fclose(FFLAG);

system("cp sample1 sample4");
system("cp joblist1 joblist4");
system("cp try original4");
```

```
system("rsh sun17 time share 4  >> dLNdata4 &");

  now=time(NULL);
  time_ptr=ctime(&now);
  fprintf(fraw,")--> Send job(s) to sun17 at %s\n", time_ptr);
}


 j=1;
 for (a=1; a<= 1000; a++)  {  /* the 1st(host) process */
  comb(LN,job[j],1);
 if ( (a+2)%2 != 0 )
      j=j+7;
 else j=j+1;
 if (j > (Nmax-Nmin+1) )
 break;    /* job[1] [8] [9] [16] [17] ... for sun11 */
                          }

 fprintf(fraw,"B=%d SolOpt=\n",B);
 for(i=1; i<=n && SolOpt[i] !=0 ; i++) {fprintf(fraw,"%d ",SolOpt[i]);}
 fprintf(fraw,"\n\nLN=%d  Nmin=%d  Nmax=%d\n", LN, Nmin, Nmax);
 if(currentPi > Plinear) {
 fprintf(fraw,"Linear(Better) Optimum Contribution: %d\n", currentPi);
 printf("\Nlinear(Better) Optimum Contribution: %d\n", currentPi);
 Plinear=currentPi;
                          }
 else { fprintf(fraw,"Linear(same) Optimum Contribution:
%d\n",currentPi);
        printf("\Nlinear(same)  Optimum  Contribution:
%d\n",currentPi);
      }

 return;
}


/* * *
 * * *
 * * */
```

```
comb(N,K,ptr)
int N,K,ptr;
{
int i, jj, k, local, tmpCi;

local=ptr;  /* local is the index of current candidate */
if (K > N) { printf("error in comb 1 ");return;}
if (K==N) {
    for(i=1; i<=N; i++) out[local++]=i;
    tmpPi= tmpCi= 0;

    for(k=1; k<local; k++)  {
        tmpPi += TABLE[sample[out[k]]].profit;
        tmpCi += TABLE[sample[out[k]]].cost;
        SolCurrent[k]=sample[out[k]];
                            }

    if((tmpPi > currentPi) && (tmpCi <=B)) {
         printf("update"); update();        }
    return;
          }

if(K==1) {
    for(i=1; i <= N; i++) {
            out[local] = i;
            tmpPi= tmpCi= 0;

            for(k=1; k<=local; k++)  {
                tmpPi += TABLE[sample[out[k]]].profit;
                tmpCi += TABLE[sample[out[k]]].cost;
                SolCurrent[k]=sample[out[k]];
                                }

            if((tmpPi > currentPi) && (tmpCi <=B)) {
            update();
                                        }
         }
    return;
  }
```

```
for(jj=1; jj<=2;jj++){
if(jj==1) {out[local]=N;
          comb(N-1, K-1, ++local);
          }
if(jj==2)  {local=ptr; comb(N-1, K, local);
           }
                    }
return;
}


update()
{
int i;

for (i=1; i<=n; i++)
{SolOpt[i] = SolCurrent[i];}
currentPi=tmpPi;
}
```

APPENDIX B PROGRAM OF THE REMOTE PROCESS

```
*********************************
    Program of the Remote process
*********************************

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <signal.h>
#define MAXtry 101

long now;
char *time_ptr;
long int Plinear,currentPi;
int n,LN,Nmax,Nmin,B,process,counter,SIGNAL;
int out[MAXtry],SolCurrent[MAXtry],SolOpt[MAXtry];
int tmpCi,tmpPi;
struct Original {
                int    index;
                int    cost;
                int    profit;
                int    ratio;
               } TABLE[MAXtry];

struct RCS2 {
      int new;   /* sample[ new order of the RCS set] */
      int old;   /*  = old order of the original set. */
          } sample2[MAXtry];

struct RCS3 {
      int new;
      int old;
          } sample3[MAXtry];

struct RCS4 {
      int new;
```

```
        int old;
            } sample4[MAXtry];

struct JOB2 {          /*job[after rearrangement based on[ Nmax-Nmin+1] = order
                       based on n before rearrangement    */
        int after;
        int before;
            } job2[MAXtry];

struct JOB3 {
        int after;
        int before;
            } job3[MAXtry];

struct JOB4 {
            int after;
            int before;
            } job4[MAXtry];

main(argc, argv)
int argc;
char **argv;

{
int a,i,j,k,Vc,Vp;
FILE *fin2,*fin3,*fin4,*fsamp2,*fsamp3,*fsamp4,*fjob2,
     *fjob3,*fjob4,*fpara,*FFLAG;

now=time(NULL);
time_ptr=ctime(&now);
printf("\n\n>>>>>> Distributed (LN) : Time start for this case at %s",time_ptr);
process= atoi(argv[1]);
printf("* Process=%d\n",process);

 fpara=fopen("parameter","r");
 fscanf(fpara,"%d %d %d %d %d %d %d %d %d ",
&Vc,&Vp,&B,&n,&Nmax,&Nmin,&Plinear,&LN,&counter);
 printf("_____< CASE %d >\n, counter);
 printf("\nVc=%d   Vp=%d   B=%d \nn=%d   Nmin=%d   Nmax=%d
Plinear=%d  LN=%d\n ",Vc,Vp,B, n,Nmin, Nmax, Plinear,LN);
```

```
fclose(fpara);

for(i=1; i<=n; i++)     SolCurrent[i]=0;
currentPi=Plinear;


/* ===============
   The 2nd process
   =============== */

if (process == 2)      {

fin2=fopen("original2","r");
for(i=1; i<=n; i++)
{fscanf(fin2, "%d %d %d %d", &TABLE[i].index, &TABLE[i].cost,
            &TABLE[i].profit, &TABLE[i].ratio);}
fclose(fin2);

fsamp2=fopen("sample2","r");
for(i=1; i<=LN; i++)
{fscanf(fsamp2, "%d %d", &sample2[i].new, &sample2[i].old);}
fclose(fsamp2);

printf("\nThe index of RCS members are :\n ");
for(i=1; i<=LN; i++)
{printf("%d ",sample2[i].old);}

fjob2=fopen("joblist2","r");
for(i=1; i<=Nmax-Nmin+1; i++)
 {fscanf(fjob2, "%d %d", &job2[i].after, &job2[i].before);}
fclose(fjob2);

printf("\n\nThis case has  %d combinations:\n ",Nmax-Nmin+1);
for(i=1; i<=Nmax-Nmin+1; i++)
{printf("(%d %d) ",LN,job2[i].before);}
printf("\nFor this process: ");

j=2;                    /*  Host is the 1st process */
 for (a=1; a<= 100; a++)          {
  if (j <= (Nmax-Nmin+1))   {
```

```
   printf("JOB=(%d %d) ",LN,job2[j].before);

     if(LN >= job2[j].before)
     comb(LN,job2[j].before,1);

     if ( (a+2)%2 != 0  )
          j=j+5;
     else j=j+3;              }
   else  break;                    }
                    }


/* ================
   The 3rd process
   ================ */

if (process == 3) {

fin3=fopen("original3","r");
for(i=1; i<=n; i++)
{fscanf(fin3,"%d  %d  %d  %d",  &TABLE[i].index,
&TABLE[i].cost,&TABLE[i].profit, &TABLE[i].ratio);}
fclose(fin3);
fsamp3=fopen("sample3","r");
for(i=1; i<=LN; i++)
{fscanf(fsamp3,"%d %d", &sample3[i].new, &sample3[i].old);}
fclose(fsamp3);

fjob3=fopen("joblist3","r");
for(i=1; i<=Nmax-Nmin+1; i++)
{fscanf(fjob3,"%d %d", &job3[i].after,  &job3[i].before);}
fclose(fjob3);

printf("\Nfor this process: ");

j=3;
for (a=1; a<= 100; a++)          {
if(LN >= job3[j].before)  {
printf("JOB=(%d %d) ",LN,job3[j].before);
comb(LN,job3[j].before,1);}
```

```
if ( (a+2)%2 != 0  )
     j=j+3;
else j=j+5;
if ( j > (Nmax-Nmin+1) )
break;
                              }
                    }

/* ===============
   The 4th process
   =============== */

if (process == 4)                    {

fin4=fopen("original4","r");
for(i=1; i<=n; i++)
{fscanf(fin4,"%d %d %d %d",  &TABLE[i].index,&ABLE[i].cost,
              &TABLE[i].profit, &TABLE[i].ratio);}
fclose(fin4);

fsamp4=fopen("sample4","r");
for(i=1; i<=LN; i++)
{fscanf(fsamp4,"%d %d", &sample4[i].new, &sample4[i].old);}
fclose(fsamp4);

fjob4=fopen("joblist4","r");
for(i=1; i<=Nmax-Nmin+1; i++)
{fscanf(fjob4,"%d %d", &job4[i].after, & job4[i].before); }
fclose(fjob4);

printf("\Nfor this process: ");

j=4;
for (a=1; a<= 100; a++)  {
if(LN >= job4[j].before)  {
printf("JOB=(%d %d) ",LN,job4[j].before);
comb(LN,job4[j].before,1);}
if ( (a+2)%2 != 0  )
     j=j+1;
else j=j+7;
```

```
if (j > (Nmax-Nmin+1))
break;

                    }
                            }

/* ------------------------------------------------ */


printf("\n\nSolOpt= ");
for(i=1; i<=n && SolOpt[i] !=0 ; i++) { printf("%d ", SolOpt[i]);}
printf(" \n\n");

if(currentPi > Plinear)
printf("Distributed (LN) :
   **Better** Optimum Contribution:%d\n\n",currentPi);

else printf("Distributed (LN) :
   ==Same== Optimum Contribution: %d\n\n",currentPi);

now=time(NULL);
time_ptr=ctime(&now);
 printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
       Assigned job done at %s \n\n",time_ptr);


FFLAG=fopen("flag","r");
fscanf(FFLAG,"%d",&SIGNAL);
fclose(FFLAG);

FFLAG=fopen("flag","w");
SIGNAL--;
fprintf(FFLAG,"%d",SIGNAL);
fclose(FFLAG);

}   /* end of main */
```

```
/* * *
 * * *
 * * */

comb(N,K,ptr)
int N,K,ptr;
{
int i, jj, k, local, tmpCi;

local=ptr; /* local is the index of current candidate */
if (K > N) { printf("error in comb 1 ");return;}
if (K==N) {
    for(i=1; i<=N; i++) out[local++]=i;
    tmpPi= tmpCi= 0;

    if (process == 2){
                    for(k=1; k<local; k++){
       tmpPi += TABLE[sample2[out[k]].old].profit;
       tmpCi += TABLE[sample2[out[k]].old].cost;
       SolCurrent[k]=sample2[out[k]].old;
                                          }
                   }
    if (process == 3) {
                    for(k=1; k<local; k++){
       tmpPi += TABLE[sample3[out[k]].old].profit;
       tmpCi += TABLE[sample3[out[k]].old].cost;
       SolCurrent[k]=sample3[out[k]].old;
                                          }
                    }
    if (process == 4){
                    for(k=1; k<local; k++){
       tmpPi += TABLE[sample4[out[k]].old].profit;
       tmpCi += TABLE[sample4[out[k]].old].cost;
       SolCurrent[k]=sample4[out[k]].old;
                                          }
                   }

    if(tmpPi > currentPi && tmpCi <=B) {
        printf("update"); update();    }
    return;
```

```
        }

if(K==1) {
   for(i=1; i <= N; i++) {
          out[local] = i;
          tmpPi= tmpCi= 0;

          if (process == 2){
                        for(k=1; k<=local; k++) {
      tmpPi += TABLE[sample2[out[k]].old].profit;
      tmpCi += TABLE[sample2[out[k]].old].cost;
      SolCurrent[k]=sample2[out[k]].old;
                                                }
                        }
          if (process == 3) {
                        for(k=1; k<=local; k++) {
      tmpPi += TABLE[sample3[out[k]].old].profit;
      tmpCi += TABLE[sample3[out[k]].old].cost;
      SolCurrent[k]=sample3[out[k]].old;
                                                }
                        }
          if (process == 4){
                        for(k=1; k<=local; k++) {
      tmpPi += TABLE[sample4[out[k]].old].profit;
      tmpCi += TABLE[sample4[out[k]].old].cost;
      SolCurrent[k]=sample4[out[k]].old;
                                                }
                        }

          if((tmpPi > currentPi) && (tmpCi <=B))
          update();
                    }
   return;
       }

for(jj=1; jj<=2;jj++){
if(jj==1) {out[local]=N;
         comb(N-1, K-1, ++local);
         }
if(jj==2)  {local=ptr; comb(N-1, K, local);}
```

```
                }
 return;
}


update()
{
int i;

for (i=1; i<=n; i++)
{SolOpt[i] = SolCurrent[i];}
currentPi=tmpPi;
}
```

LIST OF REFERENCES

1. Newport, K.T., and Varshney, P.K., 'On the Design of Performance Constrained Survivable Networks,' Conference Record, 1989 IEEE Military Communications Conference, p.663-670.

2. Schroeder, M.A., and Newport, K.T., 'Enhanced Network Survivability Through Balanced Resource Criticality,' Conference Record,1989 IEEE Military Communications Conference, pp.682-687.

3. Wu, Lin and Varshney, P.K., 'On Survivability Measures for Military Networks,' Conference Record, 1990 IEEE Military Communications Conference, pp. 1120-1124.

4. Rizik, P.D., 'A Model for Survivable, Low Cost Access Area Network Design in a New Europe,' Conference Record, 1990 IEEE Military Communications Conference, pp. 1097-1102.

5. Newport, K.T., Schroeder, M.A., and Wittaker, G.M., 'Techniques for Evaluating the Nodal Survivability of Large Networks,' Conference Record,1990 IEEE Military Communications Conference, pp. 1108-1113.

6. Wittaker, G.M., Schroeder, M.A., and Newport, K.T., 'A Knowledge-Based Approach to the Computation of Network Nodal Survivability,' Conference Record, 1990 IEEE Military Communications Conference, pp. 1114-1119.

7. Yang, C., and Kung, C., 'Networking Link Enhancement with Minimum Costs,' Conference Record, 1990 IEEE Military Communications Conference, pp.1125-1128.

8. Tirumalai, P., and Butler,J.T., 'Minimization Algorithms for Multiple Valued Programmable Logic Arrays,' IEEE Transaction on Computers, Vol.C-40,No.2, February 1991, pp. 167-177.

9. Yang, C., 'Link Enhancement Using Constrained Range and Reduced Candidate Set Searches,' Journal of Computer Communications, Butterworth Heinemann, to appear, Nov., 1992.

10. Coulouris, George F., and Dollimore, Jean, 'Distributed Systems',Addison-Wesley Co.,Inc. 1991.

11. Devore, Jay L., 'Probability and Statistics for Engineering and the Sciences,' Brooks/Cole Publishing Co., 1991.

12. Stevens, W.Richard, 'UNIX Network Programming,' Prentice-Hall, Inc., 1990.

13. Swartz, Ray, 'UNIX Applications Programming:Mastering the Shell,' SAMS, 1990.

14. Yang, C., and Misirlioglu, L., 'A Comparative Study of Network Link Enhancement Algorithms,' IEEE Milcom '92 to appear October 1992.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 — 2
   Naval Postgraduate School
   Monterey, California 93943-5000

3. Professor Jeffrey Knorr, Chairman, Code EC/KO — 1
   Electronic Warfare Academic Group
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Professor Yang, Chyan — 1
   Institute of Management Science
   National Chiao Tung University
   #114 (4F), Chung Shiao West Road, Section 1
   Taipei City, Taiwan R.O.C.

5. Professor David A. Schrady, Code OR/SO — 1
   Department of Operation Research
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Library — 1
   Chung Cheng Institute of Technology
   Tashi, Taoyuan County
   Taiwan R.O.C.

7. Wu, Tsung-li — 3
   6-3 Lane 42, San-min Road, Section 1
   Taoyuan City, Taiwan R.O.C.

Thesis
W9313 Wu
c.1 Distributed processing
on link enhancement.