Randomized Load-balancing for Parallel Branch-and-bound Algorithms for Multi-level Network Design

F. R. B. Cruz[1]*, G. R. Mateus[2], J. MacGregor Smith[3]

[1] Departamento de Estatística, [2] Departamento de Ciência da Computação,
Universidade Federal de Minas Gerais, Belo Horizonte – MG – Brazil
[3] Department of Mechanical and Industrial Engineering,
University of Massachusetts, Amherst – MA – USA
{fcruz@est,mateus@dcc}.ufmg.br, jmsmith@ecs.umass.edu

Abstract—
This paper is concerned with parallel implementations of the classical branch-and-bound algorithm for a multi-level network optimization problem. Multi-level network optimization problems arise in many contexts such as telecommunication, transportation, or electric power systems. A new model for multi-level network design is introduced and formulated as a mixed-integer program. The formulation is innovative because it integrates in the same model aspects of discrete facility location, topological network design, and dimensioning. We propose implementations which are suitable for MIMD (*multiple instruction stream, multiple data stream*) parallel computation systems. Thus, the implementations are very convenient for use in networks of workstations which nowadays has become so popular. We have tested two versions of the branch-and-bound algorithm as well as different load balancing strategies. The results are very encouraging indicating a gain over sequential computations in terms of execution time.

Keywords— **Parallel branch-and-bound, parallel computing, load balancing, network optimization, network design problems**

I. Introduction

A. *Motivation and Problem Statement*

In order to guarantee quality of service (QoS) and performance at minimum cost, network design and planning in engineering systems requires policy decisions, analysis of investment strategies, and technical development plans. Network planning must satisfy the expected demand for new services, upgrading, and improvements on the existing network. The aim is to explore the hierarchical organization of each network and to propose integrated network models as decision support systems. In this context, we have focused solutions for basic urban mapping data capture and data analysis using a Geographic Information System (GIS) and network optimization systems [MPL96, MCL94]. The multi-level network optimization (MLNO) problem treated here is a network design model that raises optimization aspects of dimensioning, topological design, and facility location. In this sense, the model can be applied for network planning to explore design aspects at different levels in a modeling approach that integrates several hierarchical levels.

We define the MLNO problem on a multi-weighted digraph $\mathcal{D} = (N, A)$, where N is the set of nodes and A is the set of arcs. Figure 1 shows a m-level network example containing candidate supply nodes, demand nodes, and transshipment nodes at each level. The objective is to determine an optimum combination of supply nodes and arcs to provide the required flow type to all demand nodes respecting rules of flow conservation.

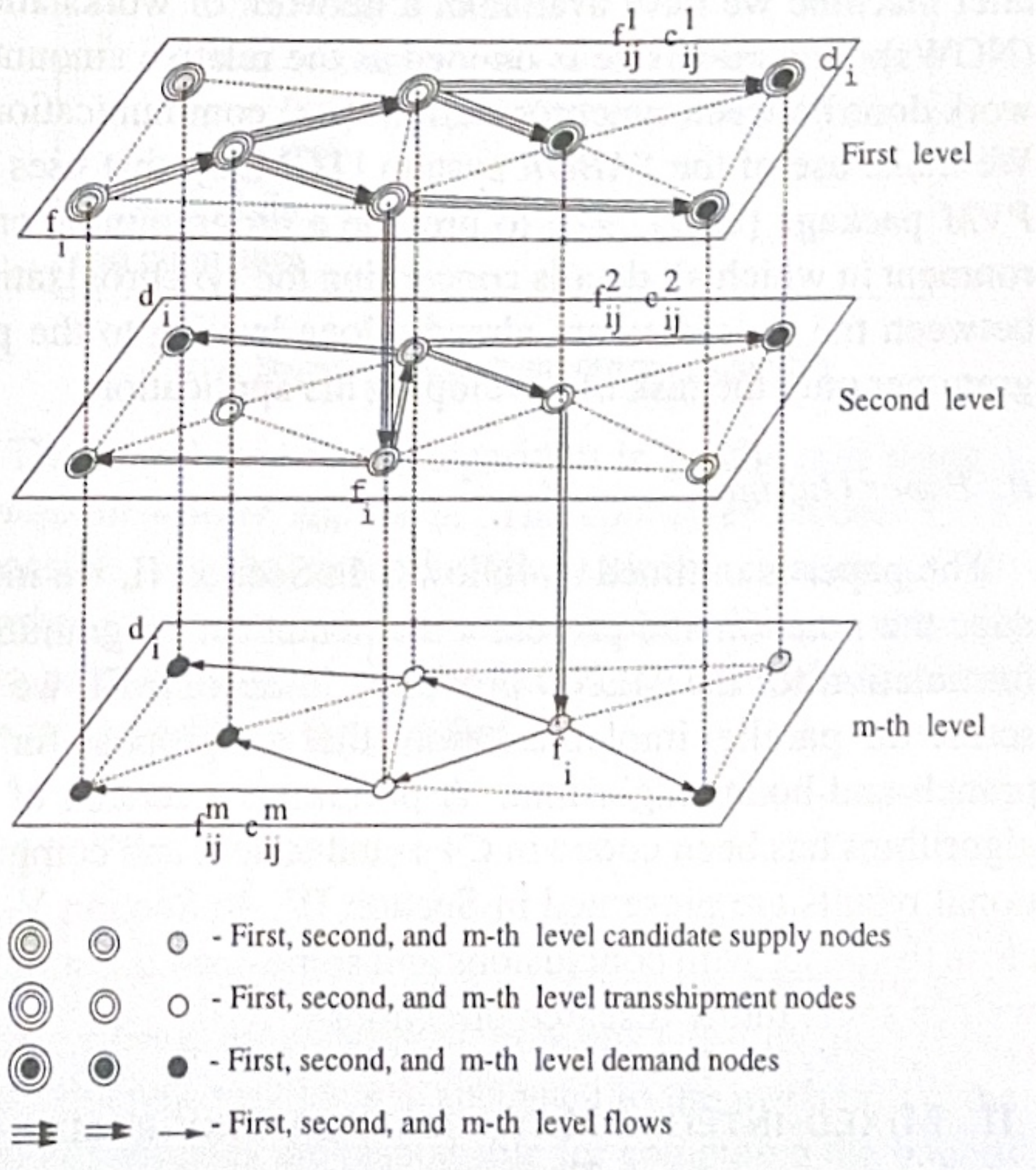

f^1_{ij} c^1_{ij}
d_i
First level
f_i
d_i
f^2_{ij} c^2_{ij}
Second level
f_i
d_i
m-th level
f_i
f^m_{ij} c^m_{ij}
- First, second, and m-th level candidate supply nodes
- First, second, and m-th level transshipment nodes
- First, second, and m-th level demand nodes
- First, second, and m-th level flows

Fig. 1. The MLNO Problem

The MLNO problem is $\mathcal{NP}$-hard since it generalizes other

*To whom all correspondences should be addressed at: Caixa Postal 702, Departamento de Estatística - UFMG, 30123-970 - Belo Horizonte - MG, Brazil. E-mail: fcruz@est.ufmg.br. Phone: (+55 31) 499 5929. Fax: (+55 31) 499 5924.

$\mathcal{NP}$-hard optimization problems, such as the Steiner problem in graphs [GJ79], the telephonic switching center problem [LZC87], or the uncapacitated location problem [Erl78]. Little research has been done on the MLNO problem. In some recent papers, models for multi-level network design have appeared, but not as done here. Some papers do not consider the integration of location aspect [CRC86, DV89], others do not consider dimensioning aspects [BMM94]. This paper considers a modeling issue that is to integrate discrete location aspects, topological network design, and network dimensioning in the same model. Somebody might argue that the MLNO problem may not be able to capture all complexities existing in the actual network design problem. However, its solutions may provide insights and a starting point for further and more accurate analyses.

An exact approach to solve any $\mathcal{NP}$-hard optimization problem normally is to implicitly enumerate all solutions. Although time consuming, branch-and-bound is a well-known technique largely applied to many similar problems and the idea of reducing the computational time of branch-and-bound algorithms by means of parallelism is very promising [Lau93, GC94]. Possible approaches range from those using parallel exploration of the branch-and-bound search tree to those using parallelism to compute the lower and upper bounds themselves. In this work, we are more concerned about the first view which represents a more coarse grain parallel application more convenient for the parallel machine we have available, a network of workstations (NOWs). The grain size is defined as the relative amount of work done between synchronizations (*i.e.*, communications). We make use of the *SABOR* system [TCM95], that uses the *PVM* package [GBD+94], to provide a programming environment in which all details concerning the synchronizations between the processes are already done leaving to the programmer only the task of developing his application.

B. Paper Outline

The paper is outlined as follows. In Section II, we introduce the notation and present a mathematical programming formulation for the MLNO problem. In Section III, we describe the parallel implementations that we propose for the branch-and-bound algorithm. A preliminary version of the algorithms has been coded in *C++* and tested, and computational results are presented in Section IV. In Section V, we close the paper with conclusions and some open questions as well as some future research directions.

II. Mixed-integer Mathematical Programming Formulation

In formulating the MLNO problem, we made some assumptions concerning the settings which are made explicit below:

1. The arcs have cost parameters that include a fixe cost of using the arc and a variable cost per-uni of flow. There is a discontinuity in the zero flo values, so the total cost is a nonlinear function o the amount of flow.
2. The supply capacity of the first-level candidat supply nodes equals the sum of all demands in al levels.
3. The candidate supply nodes of the other levels are really "transformation" nodes. Without loss of generality, they receive flows from one level and convert them to another in a 1:1 ratio. Example transformations include copper cable service being transformed 1:1 into optical fiber service.
4. There is a cost for transforming flows from one level to another. We model here possible hardwares that must be present to interconnect the different networks.

A. Notation

We now define the notation used.

m	- number of levels;
R^l	- set of l-th level candidate supply nodes;
D^l	- set of l-th level demand nodes.
d_i	- l-th level demand node $i \in D^l$;
T^l	- set of l-th level transshipment nodes, defined as follows: $T^l = N \setminus (R^l \cup D^l \cup R^{l+1})$ for $l = 1, 2, \ldots, (m-1)$, and $T^m = N \setminus (R^m \cup D^m)$;
c^l_{ij}	- non-negative per-unit cost for l-th level flow on arc $(i,j) \in A$;
x^l_{ij}	- l-th level flow through arc $(i,j) \in A$;
f^l_{ij}	- non-negative fixed cost for using arc $(i,j) \in A$ to support l-th level flow;
y^l_{ij}	- boolean variable which assumes the value 1 or 0 depending on whether or not the arc (i,j) is being used to support l-th level flow;
f_i	- non-negative allocation cost for the l-th level candidate supply node $i \in R^l$;
z_i	- boolean variable which is set to 1 or 0 depending on whether or not the node $i \in R^l$ is being selected to provide l-th level flow;
M^l	- capacity on all arcs in the l-th level, but relaxed in this paper and considered a *big* enough number, *i.e.* $M^l = \sum_{L=l}^{m} \sum_{i \in D^L} d_i$;
s^l	- capacity on all l-th level candidate supply nodes, but also relaxed in this paper, *i.e.* $s^l = M^l$;
$\delta^+(i)$	- set $\{j \mid (i,j) \in A\}$;
$\delta^-(i)$	- set $\{j \mid (j,i) \in A\}$.

B. Formulation

The mathematical programming formulation describing the MLNO problem is presented as a flow-based mixed-integer programming (MIP) model:

Model (M):

$$\min \sum_{l=1}^{m} \left[\sum_{(i,j)\in A} (c_{ij}^l x_{ij}^l + f_{ij}^l y_{ij}^l) + \sum_{i\in R^l} f_i z_i \right], \quad (1)$$

s.t.:

$$\sum_{j\in\delta^+(i)} x_{ij}^l - \sum_{j\in\delta^-(i)} x_{ji}^l = -\left(\sum_{j\in\delta^+(i)} x_{ij}^{l-1} - \sum_{j\in\delta^-(i)} x_{ji}^{l-1} \right), \quad \forall \; \underset{l=2,3,\ldots,m,}{i \in R^l,} \quad (2)$$

$$\sum_{j\in\delta^+(i)} x_{ij}^l - \sum_{j\in\delta^-(i)} x_{ji}^l = 0, \quad \forall \; \underset{l=1,2,\ldots,m,}{i \in T^l,} \quad (3)$$

$$\sum_{j\in\delta^+(i)} x_{ij}^l - \sum_{j\in\delta^-(i)} x_{ji}^l = -d_i, \quad \forall \; \underset{l=1,2,\ldots,m,}{i \in D^l,} \quad (4)$$

$$\sum_{j\in\delta^+(i)} x_{ij}^l - \sum_{j\in\delta^-(i)} x_{ji}^l \le s^l z_i, \quad \forall \; \underset{l=1,2,\ldots,m,}{i \in R^l,} \quad (5)$$

$$x_{ij}^l \le M^l y_{ij}^l, \quad \forall \; \underset{l=1,2,\ldots,m,}{(i,j) \in A,} \quad (6)$$

$$x_{ij}^l \ge 0, \quad \forall \; \underset{l=1,2,\ldots,m,}{(i,j) \in A,} \quad (7)$$

$$y_{ij}^l \in \{0,1\}, \quad \forall \; \underset{l=1,2,\ldots,m,}{(i,j) \in A,} \quad (8)$$

$$z_i \in \{0,1\}, \quad \forall \; \underset{l=1,2,\ldots,m.}{i \in R^l,} \quad (9)$$

The objective function (1) minimizes three terms: (i) the first accounts for the variable cost for all flow types, (ii) the second accounts for the fixed cost associated with the use of the arcs (the overhead cost), and (iii) the last considers the total cost resulting from the use of the supply nodes.

Constraints (2) ensure the network flow conservation between adjacent levels at each supply candidate node. Constraints (3) and (4) are the usual network flow conservation equalities at each transshipment node and demand node. For example, from the point of view of level 1 all nodes $i \in N \setminus (R^1 \cup D^1 \cup R^2)$ are transshipment nodes (see Figure 1). Constraints (5) ensure there is no flow transformation in a candidate supply node if it is not selected, and constraints (6) express the fact that the flow through an arc must be zero if this arc is not included in the design.

III. Algorithms

We propose parallel implementations based on the sequential branch-and-bound algorithm depicted in Figure 2. In that description, U_{BEST} is the global upper bound and $\mathcal{L}$ is a list of unexplored problems $(M)^i$, each of which is of the form $Z_M^i = \min\{\mathbf{cx} \text{ s.t.: } \mathbf{x} \in S^i\}$, where $S^i \subseteq S$ and S is the set of feasible solutions. Associated with each problem in $\mathcal{L}$ are a lower bound $L^i \le Z_M^i$ and an upper bound $U^i \ge Z_M^i$. The bounds L^i and U^i are computed according to a Lagrangean relaxation based procedure described previously [CMM98]. Before creating its children $(M)^{2i+1}$ and $(M)^{2i+2}$, the problem $(M)^i$ is reduced by a Lagrangean relaxation based algorithm [CMM99]. The branching variable selected is the first free variable found that does not form cycles with the remaining previously fixed variables [CMM99]. For memory economy purposes, the search rule applied was *last-in-first-out* which yields a *depth-first* search strategy.

```
algorithm Branch-and-Bound
    $U_{\text{BEST}} \leftarrow +\infty$
    $\mathcal{L} \leftarrow \{(M)^0\}$
    while $\mathcal{L} \neq \emptyset$ do
        /* search rule */
        select and delete a problem $(M)^i$ from $\mathcal{L}$
        /* bound rule */
        Compute_Lower_and_Upper_Bounds($L^i$,$U^i$)
        update $U_{\text{BEST}}$
        /* branch rule */
        if $L^i < U_{\text{BEST}}$ and $(M)^i$ is not a leaf then
            $\mathcal{L} \leftarrow \mathcal{L} \cup \{(M)^{2i+1}\} \cup \{(M)^{2i+2}\}$
        end if
    end while
end algorithm
```

Fig. 2. Sequential Branch-and-Bound Algorithm

The branch-and-bound algorithm is parallelized using a *controller-worker* approach. The *controller* process is responsible for the general initializations, creation of the *worker* processes, and their coordination. We now describe two parallel versions for the branch-and-bound algorithm, the centralized and the distributed.

A. Centralized version

The high level algorithm for the centralized version are presented in Figure 3. In such a version, the *controller* manages the list $\mathcal{L}$ of unexplored problems $(M)^i$ but the task of expanding the problems is attributed to the *workers*, *i.e.* the *worker* processes are responsible for computing the bounds and generating the respective children.

In other words, the *controller* keeps itself in the main loop while there are problems $(M)^i$ to be solved in the list $\mathcal{L}$ or else there is still some *worker* in expansion work. If there is

```
$U_{\text{BEST}} \leftarrow +\infty$
$\mathcal{L} \leftarrow \{(M)^0\}$
while there is work do
    if $\mathcal{L} \neq \emptyset$ then
        select a problem $(M)^i$ from $\mathcal{L}$
        send $(M)^i$ and $U_{\text{BEST}}$ to worker
    end if
    if there is an answer then
        receive $U'_{\text{BEST}}$ and $\mathcal{L}'$ from worker
        update $U_{\text{BEST}}$
        $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}'$
    end if
end while
terminate workers
```

a) Controller Algorithm

```
$U'_{\text{BEST}} \leftarrow +\infty$
while not terminate do
    receive problem $(M)^i$ and $U_{\text{BEST}}$
    Compute_Lower_and_Upper_Bounds($L^i$,$U^i$)
    update $U'_{\text{BEST}}$
    if $L^i < U'_{\text{BEST}}$ and $(M)^i$ is not a leaf then
        $\mathcal{L}' \leftarrow \{(M)^{2i+1}\} \cup \{(M)^{2i+2}\}$
    else
        $\mathcal{L}' \leftarrow \emptyset$
    end if
    send $U'_{\text{BEST}}$ and $\mathcal{L}'$ to controller
end while
```

b) Worker Algorithm

Fig. 3: Centralized Parallel Branch-and-Bound Algorithm

any problem in the list $\mathcal{L}$, the *controller* choose one of them according to the *last-in-first-out* strategy and sends it to the first *worker* free. Having any expansion concluded, the *controller* receives the partial best upper bound U'_{BEST} and the list $\mathcal{L}'$ of recently generated children. It updates its own best upper bound U_{BEST} and its list $\mathcal{L}$. Otherwise, the *controller* sends a terminate message to all *workers*. By themselves, the *workers* keep in a main loop receiving problems $(M)^i$, solving them, and sending back the results to the *controller* until they receive the terminate sign.

Because this implementation demands frequent synchronizations, it may cause a bottleneck in the *controller* and results in sub-utilization of the *worker* processes. However, it may be efficient if the granularity of the problem solved is coarse enough, *i.e.* if the problem expansion is computationally intensive compared to the communication costs. The distributed version we present tries to overcome this possible drawback.

B. Distributed version

The high level algorithm for the distributed version is illustrated in Figure 4. The *controller* is responsible for the initial load distribution and the algorithm finalization. Each *worker* implements the sequential branch-and-bound algorithm with slight modifications that permit load exchanges with other *workers*.

The *controller* expands the problem $(M)^0$ up to the point it has as many children $(M)^{i_k}$ as it has *workers*. So, the *controller* distributes the load and keeps itself in a loop waiting the k-th *worker* to explore completely the problem $(M)^{i_k}$ received. Thus, the *controller* terminates the *workers*, receives all partial best solutions U'_{BEST} and chooses the best among all received.

According to the load balancing policy in use, the *workers* are entitled to solve the sub-problems they received by exchanging load between themselves. Concerning this issue, we have tested three different load balancing policies. We shall describe them now.

Static balancing

This is the simplest load balancing procedure which means no dynamic balancing at all. Each *worker* receives an initial assignment, problem $(M)^{i_k}$, and has to solve it alone no matter how long it takes. An immediate possible drawback on this approach has to do with those problem instances with unbalanced search trees. We shall talk more about this in the following section.

Zhang balancing

This balancing policy is a contribution of Karp and Zhang [KZ93]. The policy is also very simple. Here, the *workers* are entitled to exchange loads. Each time a *worker* expands a problem or it keeps the children to itself or sends them to some neighbor choosing it by a random distribution. In such a case, the *controller* communicates merely with *one* of the *workers* and sends to it the initial problem $(M)^0$. The remaining *workers* will receive their loads as long as somebody already working chooses them to send its children.

Modified balancing

We think we could improve Zhang's algorithm in practice by including the following modification. First, a *worker* should not be entitled to send the list $\mathcal{L}'$ out if this action would result in a null load. Moreover, the *workers* should be more *active* and ask for load when they were running out of problems. Of course, this is not a simpler approach but may result in some gain in terms of processing time. We shall now present computational results where all possibilities are tested.

```
$U_{\text{BEST}} \leftarrow +\infty$
generate problems $(M)^{i_k}$
for all $k$ do
    send problem $(M)^{i_k}$ to worker $k$
end for
while somebody is working do
    receive current status of workers
end while
terminate workers
for all $k$ do
    receive $U'_{\text{BEST}}$ from worker $k$
    update $U_{\text{BEST}}$
end for
```

a) Controller Algorithm

```
receive problem $(M)^{i_k}$
$U'_{\text{BEST}} \leftarrow +\infty$
$\mathcal{L} \leftarrow \{(M)^{i_k}\}$
while not terminate do
    if $\mathcal{L} \neq \emptyset$ then
        select a problem $(M)^i$ from $\mathcal{L}$
        Compute_Lower_and_Upper_Bounds($L^i$,$U^i$)
        update $U'_{\text{BEST}}$
        if $L^i < U'_{\text{BEST}}$ and $(M)^i$ is not a leaf then
            $\mathcal{L}' \leftarrow \{(M)^{2i+1}\} \cup \{(M)^{2i+2}\}$
        else
            $\mathcal{L}' \leftarrow \emptyset$
        end if
        send $U'_{\text{BEST}}$ and $\mathcal{L}'$ to other workers
    end if
    receive $U'_{\text{BEST}}$ and $\mathcal{L}'$ from other workers
    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}'$
    send current status to controller
end while
send results to controller
```

b) Worker Algorithm

Fig. 4: Distributed Parallel Branch-and-Bound Algorithm

IV. Computational Experience

We shall present the performance measures employed to evaluate the implementations. There are various different quality measures for parallel algorithms [Qui87]. We shall use only two of them. The first is the *speedup*, $s(p)$, which is defined as follows:

$$s(p) = t_{\text{seq}}/t_{\text{par}}(p), \qquad (10)$$

where t_{seq} is the time spent by the best known sequential algorithm and $t_{\text{par}}(p)$ is the time spent by the parallel algorithm with p processors.

The other measure is the processor usage, u, defined below:

$$u = 100\% \times t_{\text{calc}}/t_{\text{total}}, \qquad (11)$$

where t_{total} is the total execution time and t_{calc} is the time really spent within useful calculations, *i.e.* the total time minus the time the processor idles waiting for load.

All randomly generated testing problems came from a procedure similar to that one presented previously [CMM99], which has been extensively applied to create test problem instances of the *Steiner* problem in graphs, a special case of the MLNO problem. According to this procedure, node positions, arc extremities, basic arc weights Ω_{ij}, candidate supply nodes, and demand nodes are chosen by uniform distribution. The problems actually solved were the directed version of the graph generated, each edge being substituted by two opposite arcs with the same weight. All demands were considered unitary. The costs f^l_{ij} and c^l_{ij} were derived from the weights Ω_{ij} using constant factors.

A. Homogeneous Network

The programs were executed in a homogeneous network with five *Sun* SPARC SLC machines, an Ethernet network, 10 MBits, connected by a TCP/IP protocol bus [Tan81], and NFS file system server. In our analysis, for sake of simplicity, we concentrate in two one-level random problem instances, T_1 ($|N| = 20$, $|A| = 58$, and $|D| = 4$) and T_2 ($|N| = 35$, $|A| = 98$, and $|D| = 5$), using the costs $f_{ij} = c_{ij} = \Omega_{ij}$.

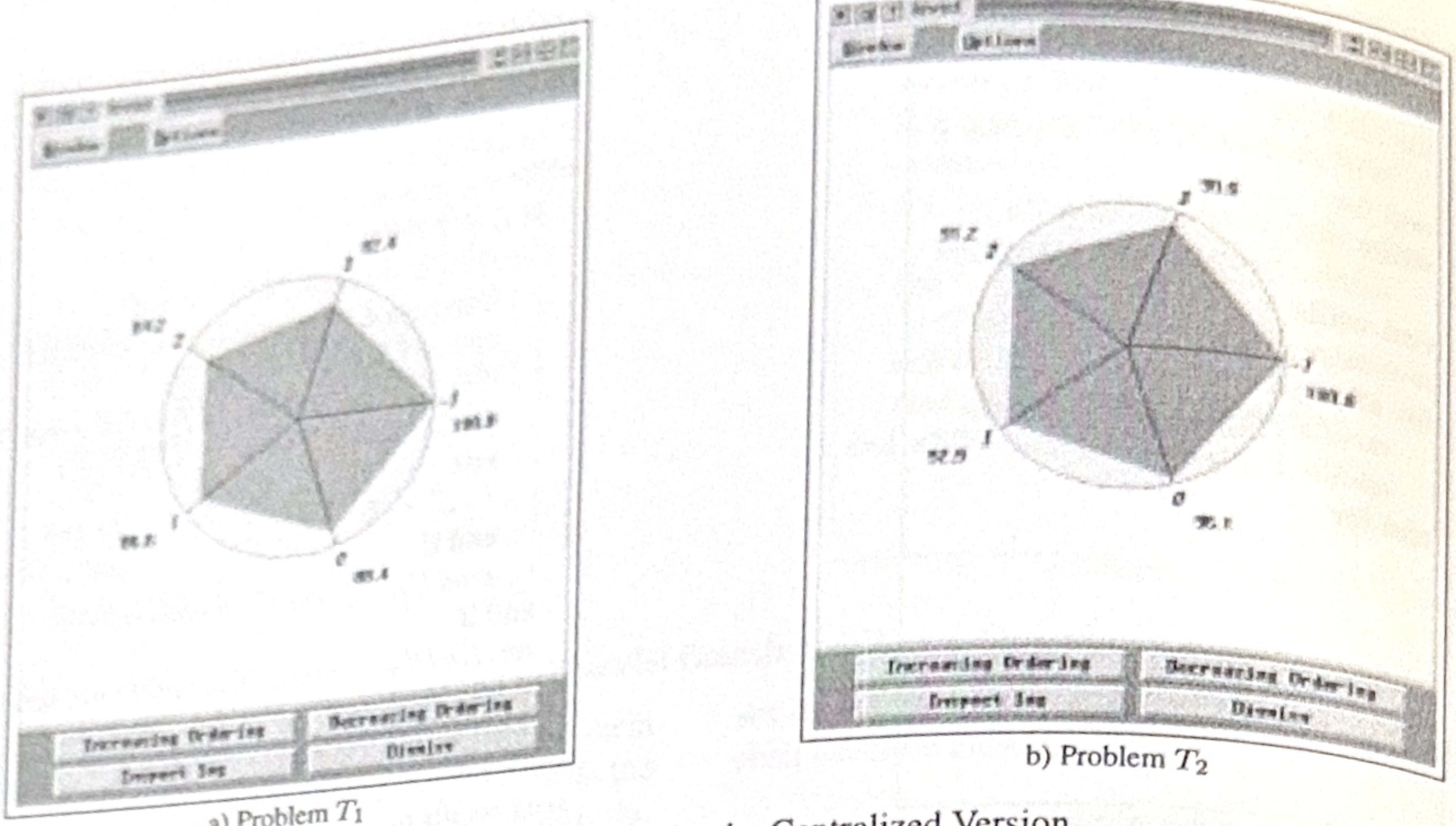
TABLE I
AVERAGE* SPEEDUP FOR ALL IMPLEMENTATIONS

	Algorithms			
		Distributed		
Problem	Centralized	Static	Zhang	Modified
T_1	4.9	1.5	1.8	2.1
T_2	3.1	1.8	1.7	1.7

*over 5 experiments

In Table I, we show the average *speedup* $s(p)$ obtained for all tested versions, made over five experiments in a low load period. For problem T_1, it is noticeable that we almost reached the theoretical ideal *speedup*, *i.e.* 5, applying the centralized algorithm. Considering the distributed algorithm, the best results were those with the modified load balancing approach. However, in this case, the reached *speedup* equals 2.08 which is smaller than the *speedup* of the centralized version. For problem T_2, the centralized algorithm also shows superiority, but in this case the *speedup* equals 3.05. In the distributed algorithm, the best *speedup* reached was by using the static load balancing policy. Nevertheless, for all three balancing policies, the *speedup* was roughly the same.

In Figure 5, we show the processor usage rates and the



a) Problem T_1

b) Problem T_2

Fig. 5: Kiviat Diagrams for the Centralized Version

load balance for the centralized version. The *controller* process is identified by label '-1'. The remaining processors run *worker* processes. Looking at these pictures, we can conclude that the usage is quite good. The minimal usage value is 82.4%. For the problem T_2, the results are even better with a minimal usage around 90.6%! This could help to explain the centralized algorithm superiority. Once the grain seems to be coarse, *i.e.* they are problems in which the lower and upper bound calculations are relatively intensive, the communication existing in this algorithm does not degenerate it performance.

Figure 6 shows results for the distributed version. We can note improvements on the usage throughout the three balancing policies. The worst minimal usage value is obtained with the static balancing (23%) and the better is with the modified balancing (approximately 74%). We see that the better usage values obtained with the distributed algorithm is less than that obtained with the centralized version. This fact explains the better results reached for the *speedup*. Nevertheless, we cannot conclude definitively that the centralized algorithm is always better, since only three balancing policies were tested.

Figure 7 presents the behavior for the three versions of the distributed algorithm using problem T_2. The results are quite similar to those previously shown. Here, comparing to the other alternative policies, the modified strategy also produced a more uniform load balancing in association with considerable better usage rates.

B. Heterogeneous Test

For the sake of simplicity, all instances tested had 8 node, 14 arcs, only 1 supply node, and 1 demand node in the first-level. For the second-level, we have worked with 1 candidate supply node as well as 4 demand nodes. The costs $f_{ij}^2 = 2\Omega_{ij}$ and $c_{ij}^2 = 128\Omega_{ij}$ were considered higher than $f_{ij}^1 = \Omega_{ij}$ and $c_{ij}^1 = 8\Omega_{ij}$, which is a reasonable assumption in practical multi-level networks. Usually the higher (*i.e.* first) levels take advantage of higher demands and are allowed to use special media with lower per-unit cost. On the other hand, if demand is not high enough, as it is in lower (*i.e.* second) levels, less efficient media with higher per-unit costs may be required in order to guarantee lower overall cost.

TABLE II

HARDWARE DETAILS OF THE MACHINES USED

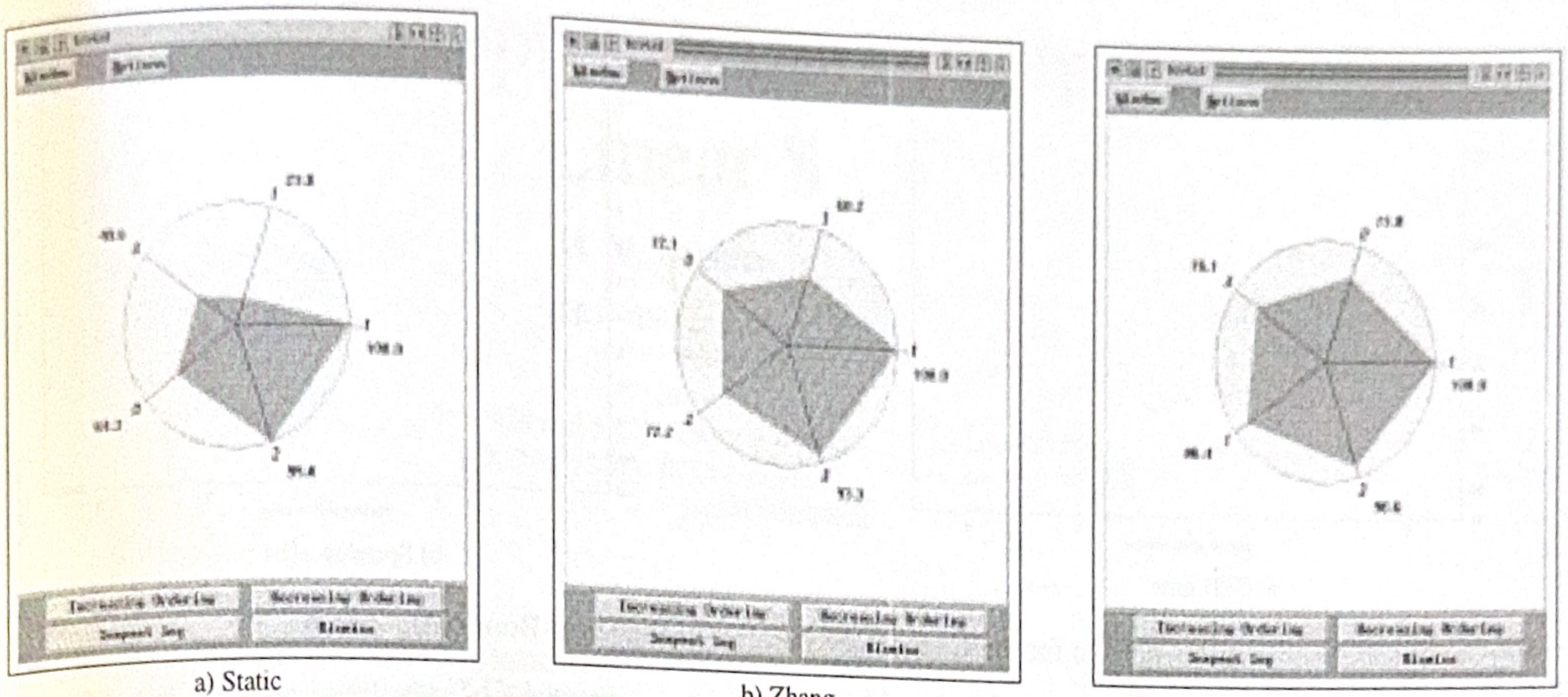
Name	SunOS	CPU Type	System Model	RAM	"Speed"*
aroeira	Rel. 5.5.1	Model 140 UltraSPARC	Ultra 1	128 MB	1.00
caviuna	Rel. 5.5	110 MHz microSPARC II	SPARCstation 4	64 MB	0.74
turmalina	Rel. 5.5.1	Model 140 UltraSPARC	Ultra 1	160 MB	0.71
diamante	Rel. 5.5.1	Model 61 SuperSPARC	Axil 320	256 MB	0.63
cello	Rel. 5.5.1	50 MHz microSPARC I	SPARCclassic	16 MB	0.24
fluorita	Rel. 5.5.1	50 MHz microSPARC I	SPARCclassic	16 MB	0.24

* (60 sec) ÷ (average sequential time to solve the hardest problem)

The parallel results reported were executed in the network of workstations (NOWs), an Ethernet network, 10 Mbits, connected by a TCP/IP protocol bus [Tan81], and NFS file system server. More information about the hardware is shown in Table II. These machines were used as a fully connected parallel *heterogeneous* computer.

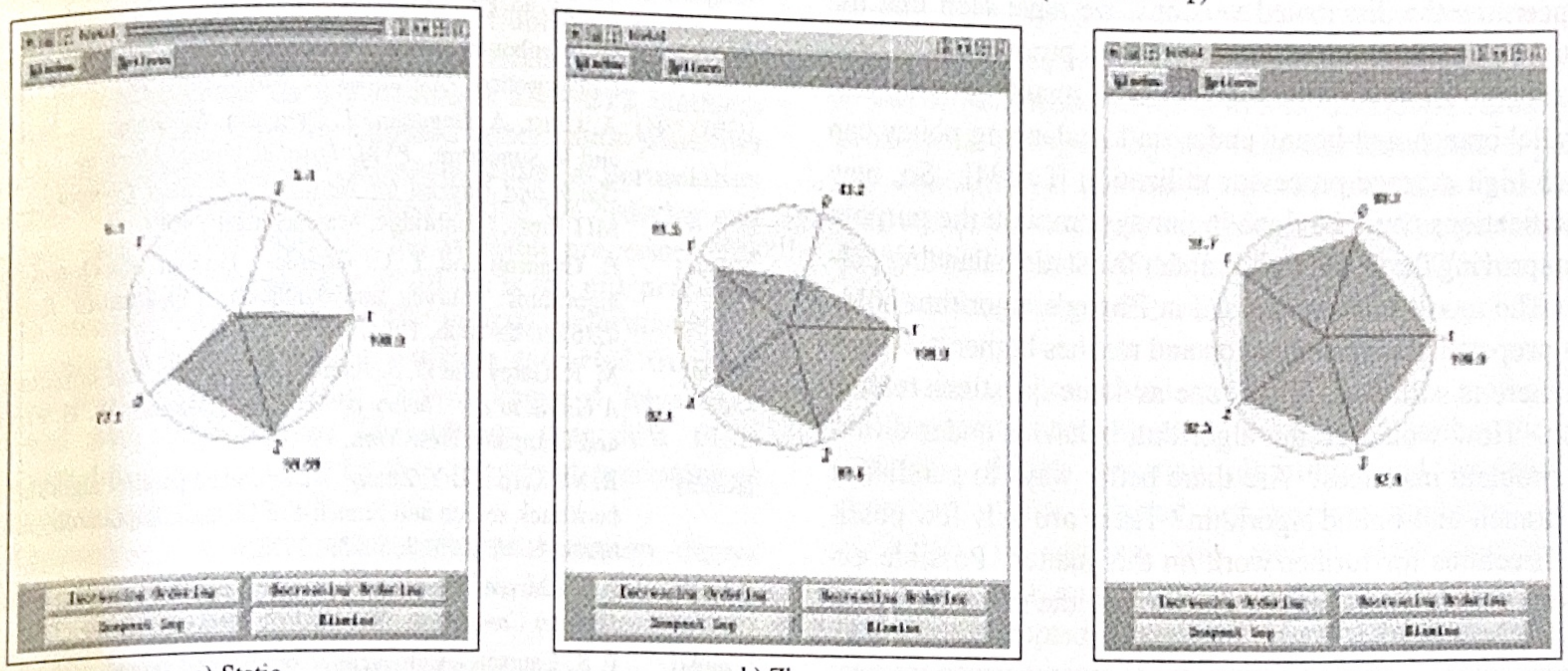
Figure 8 shows the results we have obtained with the centralized parallel branch-and-bound implementation, using up to 6 processors (machines). All CPU times reported are the elapsed time in order to solve the hardest instance out of 5 different ones which were tested.

We compare the speedup with the linear (ideal) speedup. Even considering that we have computed the speedup based on the sequential time of the fastest machine (machine *aroeira*, Table II), in the average case, the parallel algorithm solved the hardest problem in the set as much as twice faster.



a) Static b) Zhang c) Modified

Fig. 6: Kiviat Diagrams for the Distributed Version (Problem T_1)



a) Static b) Zhang c) Modified

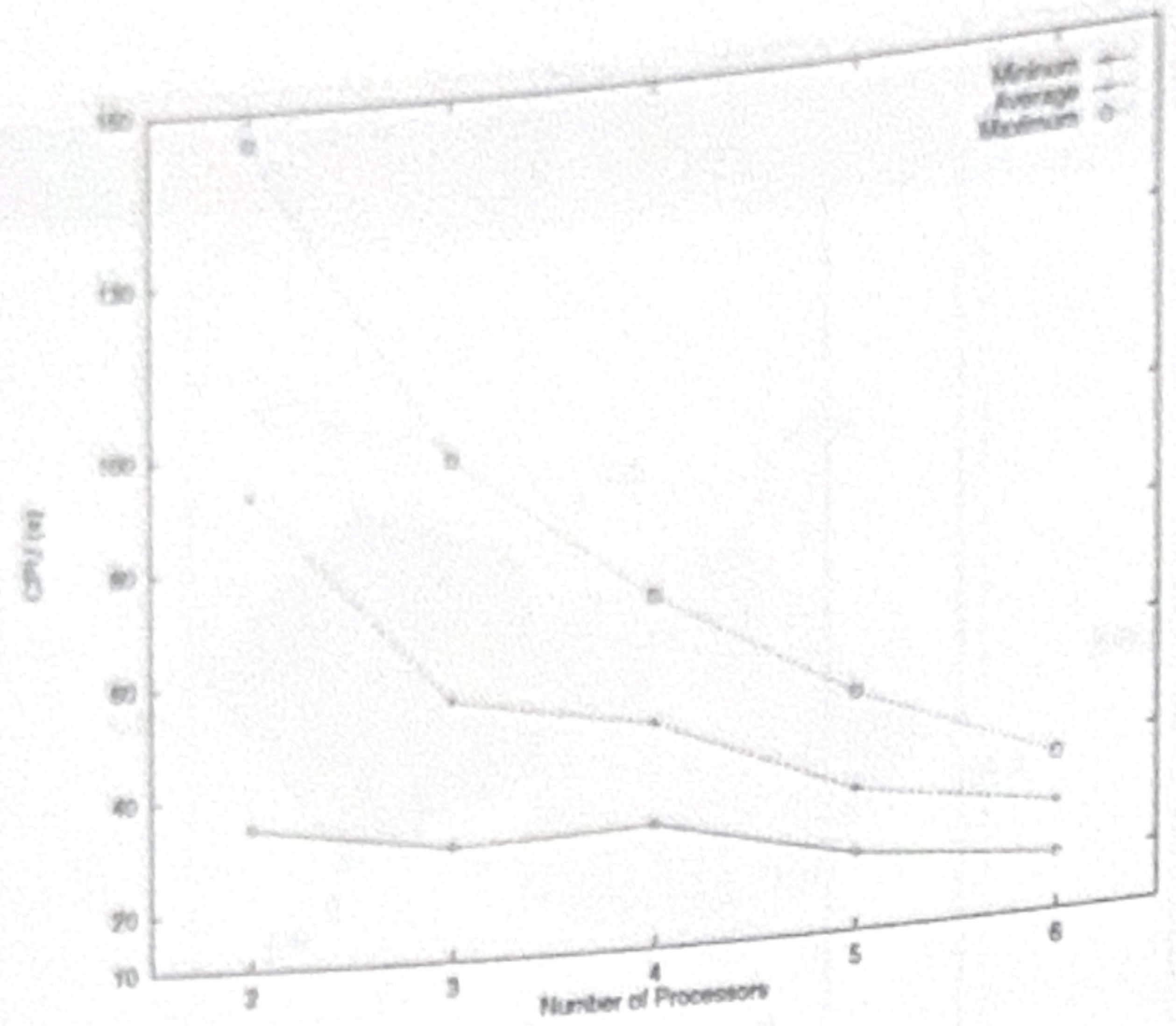
Fig. 7: Kiviat Diagrams for the Distributed Version (Problem T_2)

It is important to mention that the machines were not dedicated to run only the parallel algorithm. The average times presented were taken over several days in a low load period since it was impossible to isolate all machines used to conduct the parallel experiments.

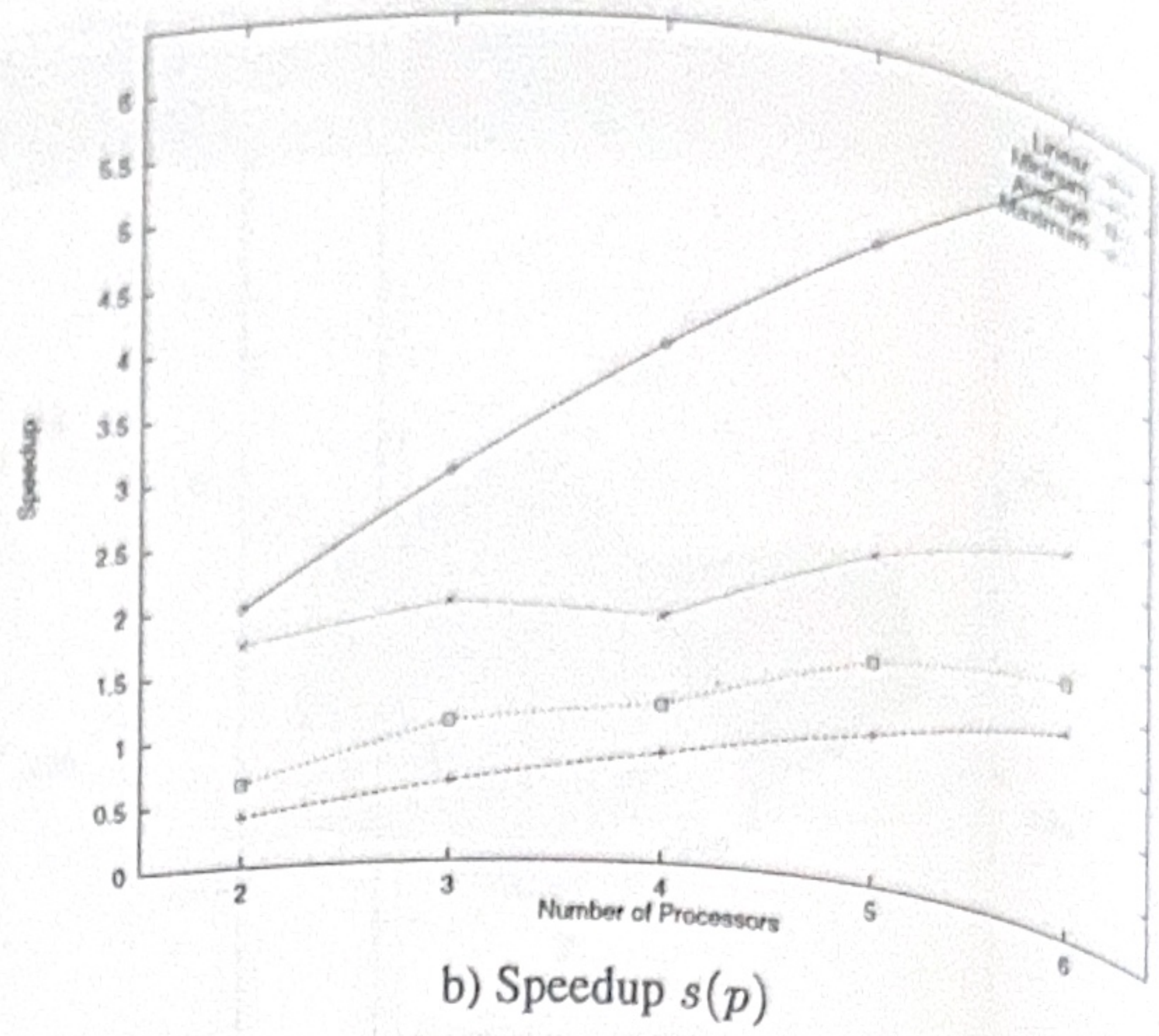
V. Conclusions and Final Remarks

A multi-level network optimization (MLNO) problem was defined and its importance was discussed. The MLNO problems integrates location, topological network design and dimensioning aspects. One possible mathematical programming formulation for the problem was proposed and branch-and-bound algorithms based on this formulation was developed. We have focused on two parallel implementations for the branch-and-bound algorithm. The parallel implementations have followed a *controller-worker* approach. In the centralized version, the *controller* manages the list of unsolved problems and distributes jobs to *workers* all around the time. The distributed version corresponds to a more coarse grain parallel application where the *controller* is responsible only for initial assignments and algorithm finalization. For the distributed version, we have tested three different load balancing policies, one of them is completely original. In Table I, all implementations were compared in terms of average *speedup*. The results seem to indicate a gain over the sequential computation. Therefore, we should conclude that the parallelization of branch-and-bound algorithms applied to this network design problems is very promising.

Along all computational experiments with the centralized version, better load balancing procedures might be considered. For instance, the applied strategy was to assign a node expansion to the first free *worker*. Are there better alter-



a) CPU time



b) Speedup $s(p)$

Fig. 8: Results for the Centralized Parallel Branch-and-Bound Algorithm

natives? Moreover, the *controller* is clearly a bottleneck in this version. Would it be possible to minimize this effect? Concerning the distributed versions, we have seen that the static version does not work well since it produces poor load balancing. However, there are research results proving that parallel branch-and-bound under static balancing policy can reach high average processor utilization [Lau94]. So, new modifications could be done in our system with the purpose of improving the performance under the static balancing policy. The modification proposed in Zhang's algorithm holds this property of high utilization and reaches higher *speedups* but there is still work to be done as some questions remain open. How would be the algorithm behavior under different problem instances? Are there better ways to parallelize the branch-and-bound algorithm? These are only few possible directions for further work on this matter. Possible extensions of this work might also include the investigation of these questions.

Acknowledgment

The research of Frederico Rodrigues Borges da Cruz and Geraldo R. Mateus is supported by grants from *Conselho Nacional de Desenvolvimento Científico e Tecnológico*, CNPq, *Fundação de Amparo à Pesquisa do Estado de Minas Gerais*, FAPEMIG, and *Pró-Reitoria de Pesquisa*, PRPq, UFMG.

References

[BMM94] A. Balakrishnan, T. L. Magnanti, and P. Mirchandani. A dual-based algorithm for multi-level network design. *Management Science*, 40(7):567–581, 1994.

[CMM98] F. R. B. Cruz, J. MacGregor Smith, and G. R. Mateus. Solving to optimality the uncapacitated fixed-charge network flow problem. *Computers & Operations Research*, 25(1):67–81, 1998.

[CMM99] F. R. B. Cruz, J. MacGregor Smith, and G. R. Mateus. Algorithms for a multi-level network optimization problem. *European Journal of Operational Research*, 118(1):165–181, 1999.

[CRC86] J. R. Current, C. S. ReVelle, and J. L. Cohon. The hierarchical network design problem. *European Journal of Operational Research*, 27:57–66, 1986.

[DV89] C. W. Duin and A. Volgenant. Reducing the hierarchical network design problem. *European Journal of Operational Research*, 39:332–344, 1989.

[Erl78] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978.

[GBD+94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts, 1994.

[GC94] B. Gendron and T. G. Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[KZ93] R. M. Karp and Y. Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computation. *Journal of the ACM*, 40(3):765–789, 1993.

[Lau93] P. S. Laursen. Simple approaches to parallel branch-and-bound. *Parallel Computing*, 19:143–152, 1993.

[Lau94] P. S. Laursen. Can parallel branch-and-bound without communication be effective? *SIAM Journal on Optimization*, 4(2):143–152, 1994.

[LZC87] H. P. L. Luna, N. Ziviani, and R. M. B. Cabral. The telephonic switching centre network problem: Formalization and computational experience. *Discrete Applied Mathematics*, 18:199–210, 1987.

[MCL94] G. R. Mateus, F. R. B. Cruz, and H. P. L. Luna. An algorithm for hierarchical network design. *Location Science*, 2(3):149–164, 1994.

[MPL96] G. R. Mateus, C. I. P. S. Pádua, and H. P. L. Luna. Integrated network models for local access network design. In *Proceedings of the International Telecommunications Symposium 1996*, pages 6–10, Acapulco, Mexico, 1996.

[Qui87] M. J. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill Book Company, New York, first edition, 1987.

[Tan81] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, New York, 1981.

[TCM95] A. I. Tavares, M. L. B. Carvalho, and G. R. Mateus. Aided design and analysis of distributed branch-and-bound algorithms. In *Annals of XV International Conference of the Chilean Society of Computer Science*, pages 448–458, Arica, Chile, 1995. Chilean Society of Computer Science.