Parallel Algorithms and Applications, Vol. 18 (3) September 2003, pp. 121–137

PARALLEL ALGORITHMS FOR A MULTI-LEVEL NETWORK OPTIMIZATION PROBLEM

F.R.B. CRUZ[a,*] and G.R. MATEUS[b,†]

[a]*Departamento de Estatística, Universidade Federal de Minas Gerais, 31270-901–Belo Horizonte-MG, Brazil;* [b]*Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, 31270-901–Belo Horizonte–MG, Brazil*

(Received 15 February 2000; In final form 12 March 2001)

Multi-level network optimization (MLNO) problems arise in many contexts such as telecommunication, transportation, or electric power systems. This paper is mainly concerned with parallel implementations of the classical branch-and-bound algorithm for multi-level network design. A model for such a problem is presented and formulated as a mixed-integer program. The formulation is appealing because it integrates in the same model aspects of discrete facility location, topological network design, and dimensioning. We propose implementations that are suitable for *multiple instruction stream, multiple data stream* (MIMD) 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

Computing Reviews Categories: G. 1.6. [Numerical Analysis]: Optimization-integer programming; G.2.2. [Discrete Mathematics]: Graph Theory-network problems; G.4.[Mathematics of Computing]: Mathematical Software-parallel and vector implementation

INTRODUCTION

Motivation and Problem Statement

In engineering systems, network design and planning requires policy decisions, analysis of investment strategies and technical development plans, in order to guarantee quality of service (QoS) and performance at minimum cost. 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

*Corresponding author. Tel.: +55-31-3499-5929. Fax: +55-31-3499-5924. E-mail: fcruz@est.ufmg.br
†E-mail: mateus@dcc.ufmg.br

ISSN 1063-7192 print/ISSN 1029-032X online © 2003 Taylor & Francis Ltd
DOI: 10.1080/10637190310001603630

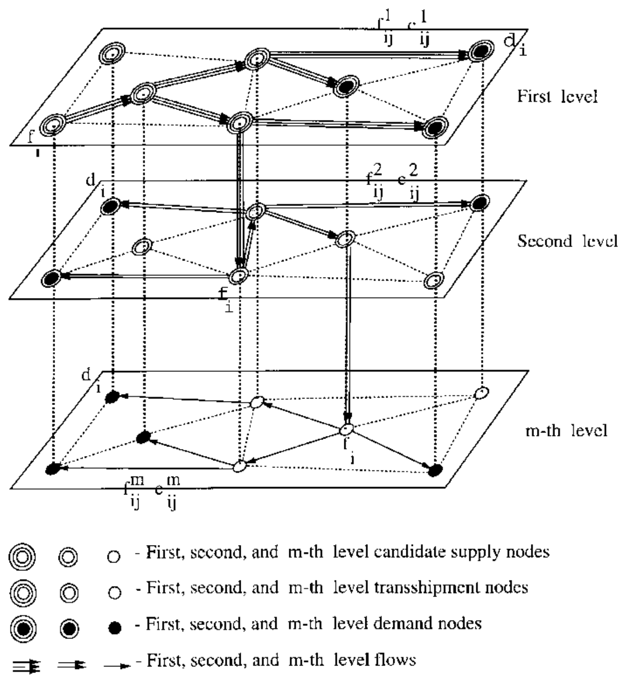

$f_{ij}^1 \; c_{ij}^1$
d_i
First level
f_i
d_i
$f_{ij}^2 \; c_{ij}^2$
Second level
f_i
d_i
m-th level
f_i
$f_{ij}^m \; c_{ij}^m$
- First, second, and m-th level candidate supply nodes
- First, second, and m-th level transshipment nodes
- First, second, and m-th level demand nodes
- First, second, and m-th level flows

FIGURE 1 The MLNO problem.

(GIS) [1] and network optimization systems [2]. The multi-level network optimization (MLNO) problem treated here was introduced recently [3] and it 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.

The MLNO problem is $\mathcal{NP}$-hard since it generalizes other $\mathcal{NP}$-hard optimization problems, such as the fixed-charge network flow problem [4], the Steiner problem in graphs [5], the telephonic switching center problem [6], or the uncapacitated location problem [7]. Models for multi-level network design have appeared in the literature. Some papers do not consider the integration of location aspects [8,9] while others do not consider dimensioning aspects [10,11]. Some research has been done on the MLNO problem that considers a modeling issue that is to integrate discrete location aspects, topological network design, and network dimensioning in the same model [3]. 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 [12,13].

Parallel processing has been widely studied as an additional source of improvement in search efficiency in discrete optimization [14]. From a parallel computing point of view, the challenge is how to use a set of processors to improve the search efficiency of branch-and-bound algorithms, given that an attractive feature is that disjoint subproblems can be decomposed simultaneously and independently. The principle of this (high level) parallelization is based on this observation. It consists of concurrently decomposing several subproblems at each iteration [15–26]. Other sources of parallelism exist, as shown in Refs. [26,27]. In general terms, the potential to be explored consists of a linear—in the number of processors—reduction of the number of iterations and, as a consequence, a linear improvement on the search efficiency of branch-and-bound algorithms. However, the reduction of the number of iterations can deviate considerably from linear due to possible *speedup anomalies* [20,21].

In this work, we are more concerned about this high level parallelization which represents a more coarse grain parallel application more convenient for the parallel machine available for us, 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 [28], that uses the *PVM* package [29], 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.

Paper Outline

The paper is outlined as follows. In the second section, we introduce the notation and present a mathematical programming formulation for the MLNO problem. In the third section, 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 the fourth section. In the fifth section, we close the paper with conclusions and some open questions as well as some future research directions.

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 fixed cost of using the arc plus a variable cost per-unit of flow. There is a discontinuity in the zero flow values, so the total cost is a nonlinear function of the amount of flow.
(2) The supply capacity of the first-level candidate supply nodes equals the sum of all demands in all levels.
(3) The candidate supply nodes of the other levels are really “transformation” nodes. 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.

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: $T^l = N \backslash (R^l \cup D^l \cup R^{l+l})$ for $l = 1, 2, \ldots, (m-1)$, and $T^m = N \backslash (R^m \cup D^m)$;
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. $M^l = \sum_{L=1}^{m} \sum_{i \in D^L} d_i$;
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 $\{j \vert (i,j) \in A\}$
$\delta^-(i)$	set $\{j \vert (j,i) \in A\}$

Formulation

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

Model (M):

$$\min \sum_{l=1}^{n} \left[\sum_{(i,j) \in A} (c^l_{ij} x^l_{ij} + f^l_{ij} y^l_{ij}) + \sum_{i \in R^l} f_i z_i \right], \tag{1}$$

s.t:

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

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

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

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

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

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

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

$$z_i \in \{0,1\}, \quad \forall\ i \in R^l,\ \ l = 1,2,\ldots,m, \tag{9}$$

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 $i \in N \backslash (R^1 \cup D^1 \cup R^2)$ are transshipment nodes (see Fig. 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.

ALGORITHMS

We propose parallel implementations based on the sequential branch-and-bound algorithm depicted in Fig. 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 $Z_M^i = \min\{cx \text{ s.t. : } x \in S^i\}$, where $S^i \subseteq S$ and S is the set of feasible solutions. Associated with each problem in $\mathcal{L}$ are a lower bound $L^i \leq Z_M^i$ and an upper bound $U^i \geq Z_M^i$. The bounds L^i and U^i are computed according to a Lagrangean relaxation based procedure described in Ref. [4]. Before creating its children $(M)^{2i+1}$ and $(M)^{2i+2}$, the problem $(M)^i$ is reduced by a Lagrangean relaxation based algorithm [3]. The branching variable selected is the first free variable found that does not

```
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
```

FIGURE 2 Sequential branch-and-bound algorithm.

form cycles with the remaining previously fixed variables [3]. For memory economy purposes, the search rule applied was *last-in-first-out* which yields a *depth-first* search strategy.

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.

Centralized Version

The high level algorithm for the centralized version are presented in Fig. 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 any problem in the list $\mathcal{L}$, the *controller* chooses one of them according to the *last-in-first-out* strategy and sends it to the first free *worker*. 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.

Distributed Version

The high level algorithms for the distributed version are shown in Fig. 4. The *controller* is responsible for the initial load distribution and the algorithm finalization.

```
$U_{\text{BEST}} \leftarrow +\infty$
$\mathcal{L} \leftarrow \{(M)^0\}$
while there is work do
    if $\mathcal{L} \neq \emptyset$ then
        select a problem $(M)^i$ from $\mathcal{L}$
        send $(M)^i$ and $U_{BEST}$ 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 $U_{BEST}$
    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

FIGURE 3 Centralized parallel branch-and-bound algorithm.

```
$U_{\text{BEST}} \leftarrow +\infty$
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

FIGURE 4 Distributed parallel branch-and-bound algorithm.

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 for 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 of this approach has to do with those problem instances with unbalanced search trees. We shall talk more about this issue in the next section.

Zhang Balancing

This balancing policy is a contribution of Karp and Zhang [30]. 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 uniform 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 $\mathcal{L}^l$ 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.

COMPUTATIONAL EXPERIENCE

Before showing the computational results, we shall present the measures employed to evaluate the implementations. There are various different quality measures for parallel algorithms [31]. We shall use only two of them. The first is the *speedup*, $s(p)$, which is defined as follows:

$$s(p) = t_{\mathrm{seq}}/t_{\mathrm{par}}(p), \tag{10}$$

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

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

$$u = 100\% \times t_{\mathrm{calc}}/t_{\mathrm{total}}, \tag{11}$$

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

All randomly generated testing problems came from a procedure similar to that one presented in Ref. [32], 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_{ij}^l and c_{ij}^l were derived from the weights Ω_{ij} using constant factors.

Homogeneous Network

In this part of the experiments, the programs were executed in a homogeneous network with five *Sun* SPARC SLC machines, an Ethernet network, 10 MBits, connected by TCP/IP [33], and NFS server.

In our analysis, for sake of simplicity, we concentrate in 2 (two) one-level random problem instances, $T_1(|N| = 20,\ |A| = 58,$ and $|D| = 4)$ and $T_2(|N| = 35,\ |A| = 98,$ and $|D| = 5)$. Other instances were tested and the results (not shown) do not differ quite much from those presented here. The costs $f_{ij} = c_{ij} = \Omega_{ij}$ were used, in which Ω_{ij} is the distance between nodes i and j. For a detailed study involving different ratios between fixed f_{ij} and variable c_{ij} costs, see Ref. [4]. Basically, problems T_1 and T_2 differ in the number of branch-and-bound nodes that must be explored before complete solution of the problem (around 100 and 500 nodes,

TABLE I Average *Speedup* for all implementations

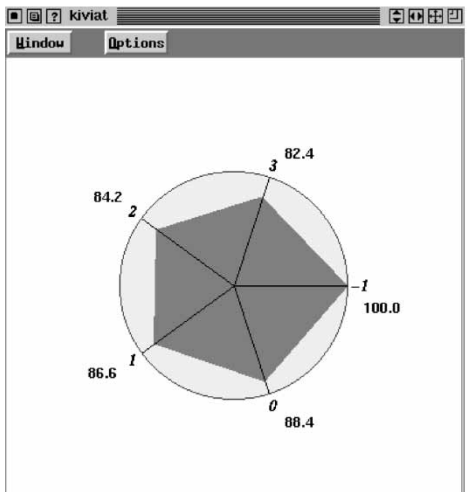
	Algorithms			
		Distributed		
Problem	*Centralized*	*Static*	*Zhang*	*Modified*
T_1	4.9	1.5	1.8	2.1
T_2	3.1	1.8	1.7	1.7

Over 5 experiments.

respectively) and in the balancing of the search tree (T_1 has a well balanced search tree unlikewise T_2).

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 *speedup* reached 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* was by using the static load balancing policy. Nevertheless, for all three balancing policies, the *speedup* was roughly the same.

The *speedup* reached could be explained by the grain size of this application. Since the SPARC SLC's are old and very slow machines, the communication delays were despicable for instance T_1, leading for a close to maximum *speedup*. For T_2 which presents a broader branch-and-bound tree, the communication factor becomes important and the *speedup* is not as good. In both cases, the bottleneck effect of the centralized version is not felt and its performance is superior.



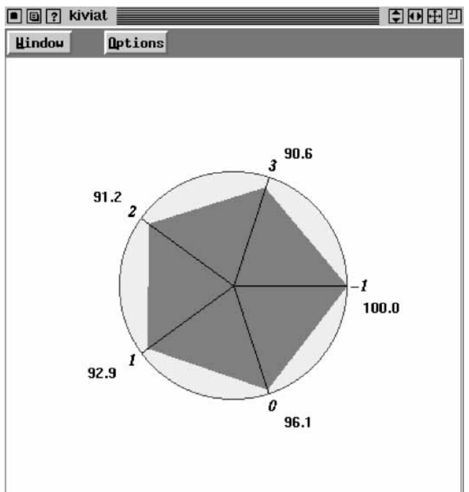


FIGURE 5 Kiviat diagrams for the centralized version.

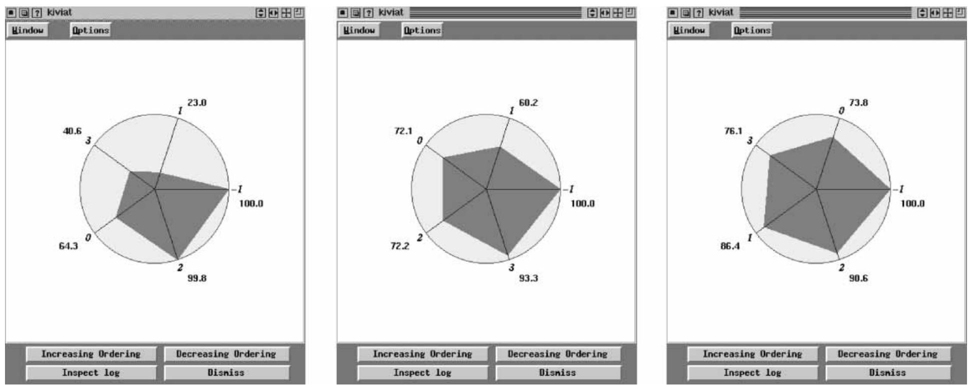


FIGURE 6 Kiviat diagrams for the distributed version (Problem T_1).

In Fig. 5, we show the processor usage rates and the 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. For instance T_1, the minimal usage value is 82.4%. For T_2, the results are even better, with a minimal usage around 90.6%. However, both usages are close enough to assert the algorithm robustness. This high usage (close to 100%) could help to explain the superiority of the algorithm. Again, the granularity of the application was coarse enough to reduce the bottleneck effect of the centralized version.

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 definitely 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, indicating the robustness of the results. Here, comparing to the other alternative policies, the modified strategy also produced a more uniform load balancing in association with considerably better usage rates.

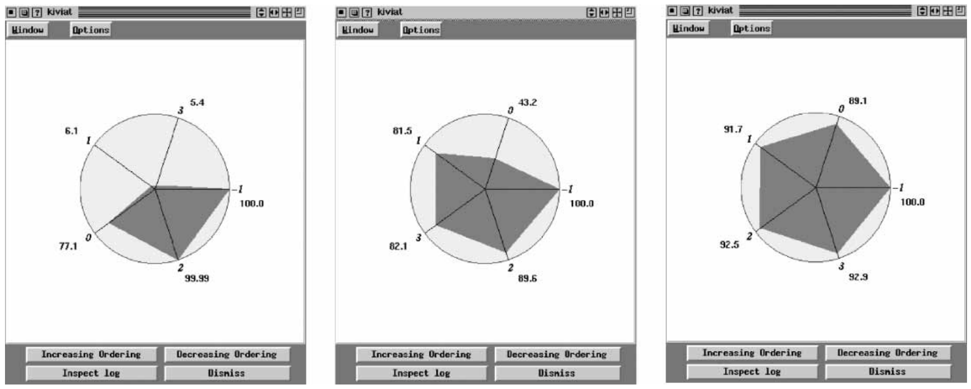


FIGURE 7 Kiviat diagrams for the distributed version (Problem T_2).

TABLE II Hardware details of the machines used in the parallel experiments

					Sequential CPU time (*s*)		
Name	*Operating system*	*CPU type*	*System model*	*RAM* (*MB*)	*B*-3*	*B*-11*	2-*Level*[†]
Aroeira	SunOS Release 5.5.1	Model 140 UltraSPARC	Ultra 1	128	78.0	110.0	60.0
Caviuna	SunOS Release 5.5.1	110 MHz microSPARC II	SPARCstation 4	64	140.0	210.0	81.0
Candeia	SunOS Release 5.5.1	110 MHz microSPARC II	SPARCstation 4	64	140.0	210.0	n/a
Diamante	SunOS Release 5.5	Model 61 SuperSPARC	Axil 320	256	130.0	240.0	95.0
Turmalina	SunOS Release 5.5.1	Model 140 UltraSPARC	Ultra 1	160	180.0	300.0	85.0
Cello	SunOS Release 5.5	50 MHz microSPARC I	SPARCclassic	16	400.0	590.0	250.0
Fluorite	SunOS Release 5.5	50 MHz microSPARC I	SPARCclassic	16	400.0	590.0	250.0
Azurita	SunOS Release 5.5	50 MHz microSPARC I	SPARCclassic	16	400.0	590.0	n/a
Aruak	SunOS Release 5.5	50 MHz microSPARC I	SPARCclassic	48	790.0	1200.0	n/a

* Beasley's networks [34], with $f_{ij} = \Omega_{ij}$ and $c_{ij} = 10\Omega_{ij}$.
[†] Randomly generated two-level networks.

As a final remark, the dependence of the static balancing upon the problem instance which is undesirable is to be noted. The search trees (not shown) could help to explain such a relationship. Better load balances correspond to balanced search trees. On the other hand, the modified balancing seems to be independent on these problem instances, although the static balancing produces better *speedup*, for instance, T_2.

Heterogeneous Test

The parallel experiments reported were performed in the same network used in the above section. Details about the hardware are shown in Table II. These machines were used as a fully connected parallel heterogeneous computer.

Beasley's Networks

In this section, we solve two one-level problems derived from the Beasley's instances [34], B-3 ($|N| = 50$, $|A| = 126$ and $|D| = 24$) and B-11 ($|N| = 75$, $|A| = 300$ and $|D| = 18$), used to test algorithms to solve Steiner problems in graphs [35,36] and fixed-charge network flow problems [4]. In order to adapt these instances to our problem, a non-Steiner node must be chosen as supply node and the others must be considered as demand nodes. Unitary demand was assumed, fixed costs $f_{ij} = \Omega_{ij}$, and variable costs $c_{ij} = 10\Omega_{ij}$.

The *speedup* reported are based on the smallest sequential CPU time presented in Table II. All machines were running only the sequential algorithm and all I/O operations were disregarded. In Fig. 8, the plots of the average speedup are presented. Since the machines could not be isolated for the experiments, all the five time measures were taken during low load period, preferably at night and weekends. It is noticeable that the average *speedup* grows with the number of machines. If we had more machines available, we could have had smaller times. In fact, for this configuration, the parallel algorithm does not provide a real advantage over sequential computation. Since we are using faster machines than the SPARC SLC's, now the communication time becomes a large proportion of the total computation time. Additionally, we are taking a fast machine (*aroeira*, Table II) as a reference for the *speedup*. As it will be seen, the use of a considerable slow machine in the settings (*aruak*, Table II) jeopardizes the overall performance.

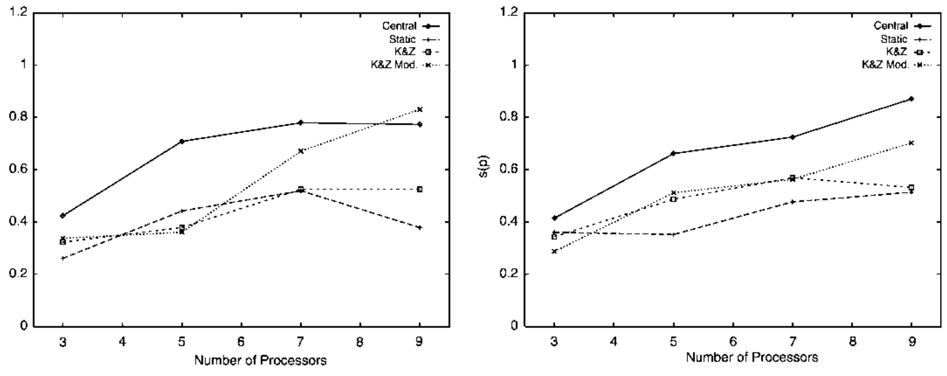


FIGURE 8 Average *speedup* for all instances.

Figure 9 presents the Kiviat diagram for the centralized version, considering nine processors. The usage is low and irregular, depending on the instance tested, which is undesirable. This version needs a coarse grain and not surprisingly, works better for the instance *B*-11 for which the exploration of each node spends more time. Although we have less synchronizations in the static balancing, the average *speedup* is poor, Fig. 8. The load unbalance is remarkable, as seen in Figs. 10 and 11, and the dependence on the instance tested is also present. Figure 12 shows the search trees and makes it clear that the search tree unbalance plays a key role in the load balance. It must be taken into consideration that Zhang and the modified balancing algorithms improve the average *speedup*, besides being instance independent.

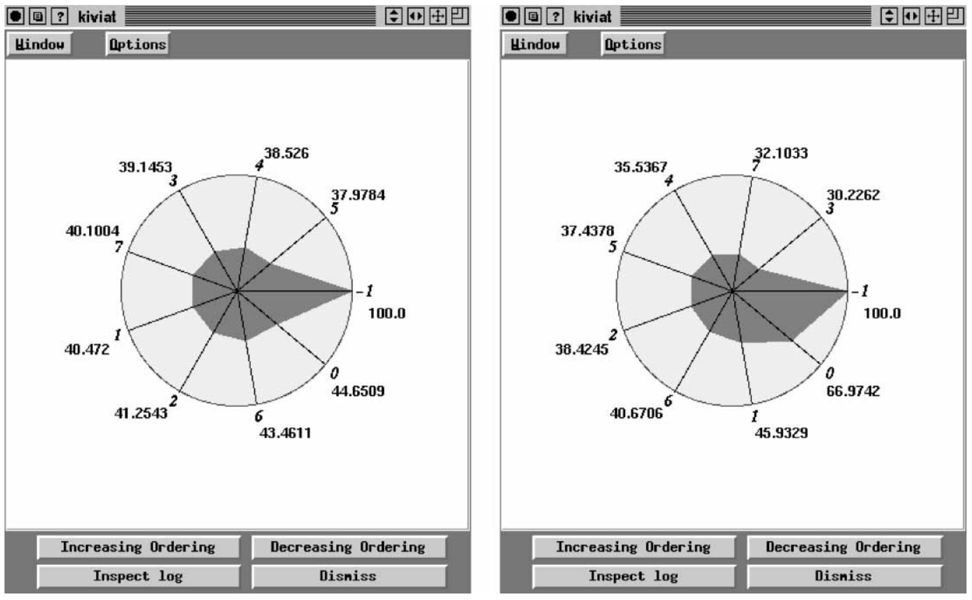


FIGURE 9 Kiviat diagrams for the centralized version.

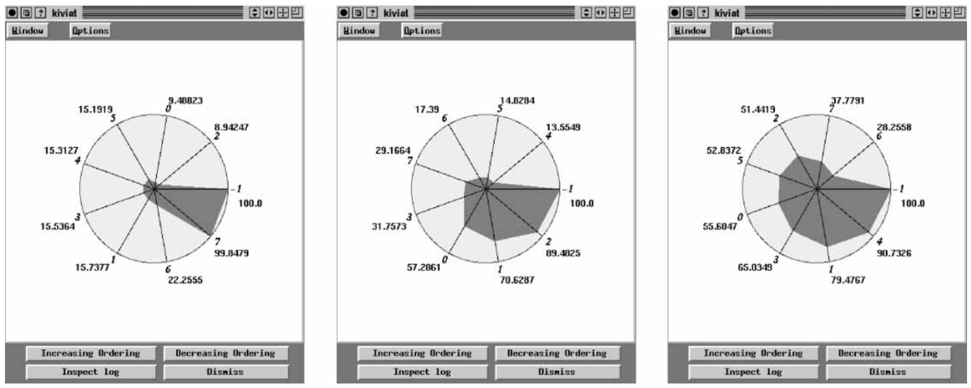


FIGURE 10 Kiviat diagrams for the distributed version (Problem B_3).

Two-level Networks

In these experiments, we discharged three machines (*candeia*, *azurita* and *aruak*). The machine *candeia* was not available for these experiments and the other two were too slow (Table II). As it will be seen, in spite of losing a fast machine, the results improved.

For the sake of simplicity, all two-level instances tested had 8 nodes, 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 and 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.

Figure 13 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 five different instances 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*,

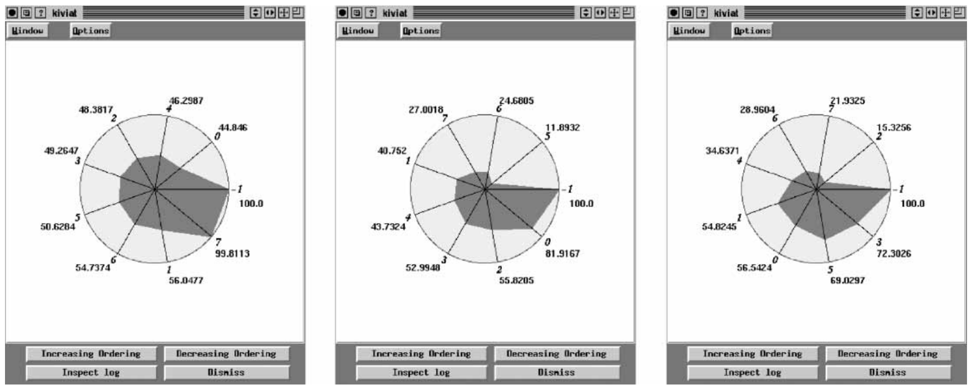
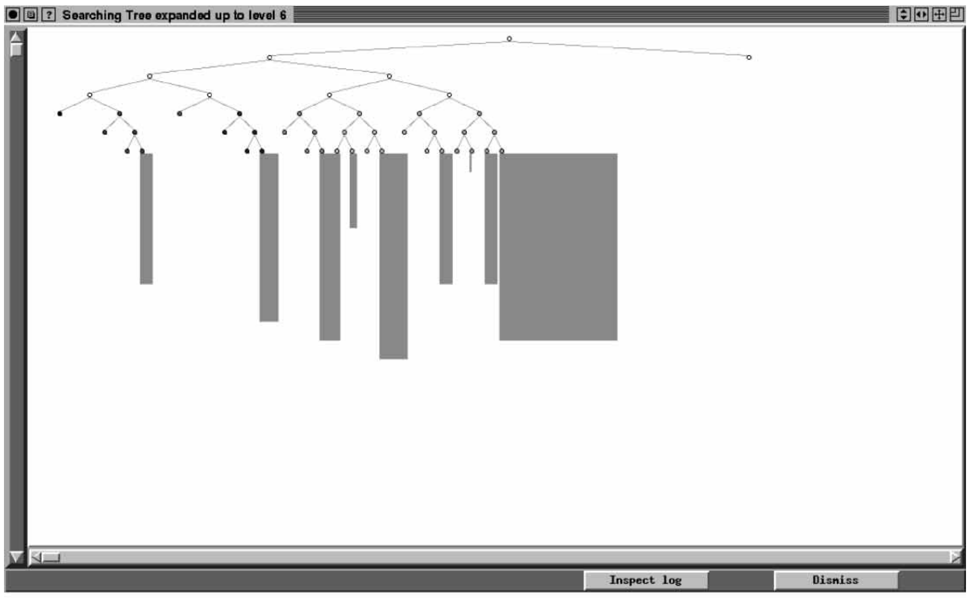


FIGURE 11 Kiviat diagrams for the distributed version (Problem B_{11}).



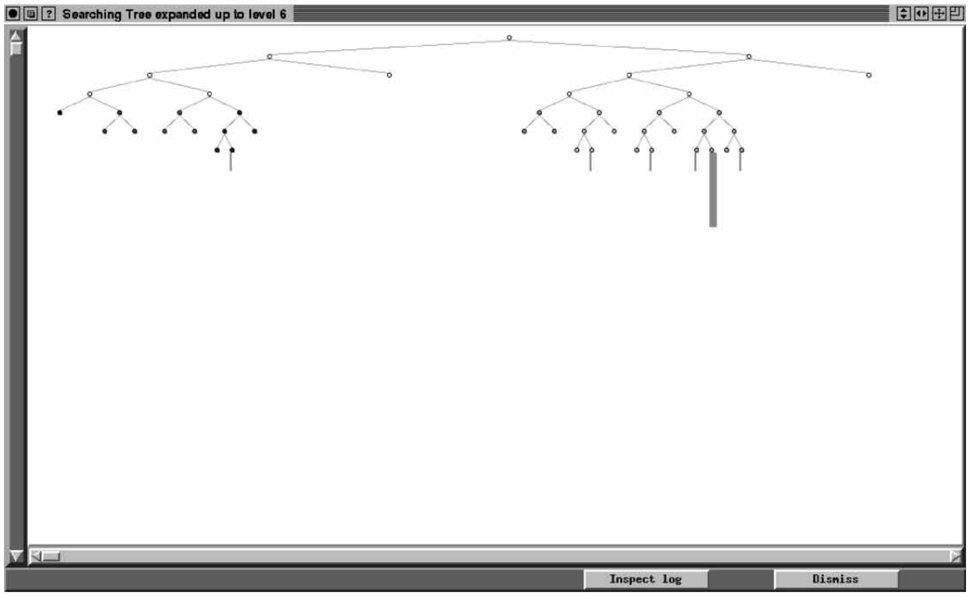


FIGURE 12 Branch-and-bound search trees.

Table II), in the average case, the parallel algorithm solved the hardest problem in the set as much as twice faster. 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.

As a final remark, it is possible to infer that if we had a faster connection, the *speedup* could be higher, as observed for the homogeneous tests. The *speedup* could also be higher, for those instances for which the node exploration is computationally intensive, as it is the case for large sized problem instances.

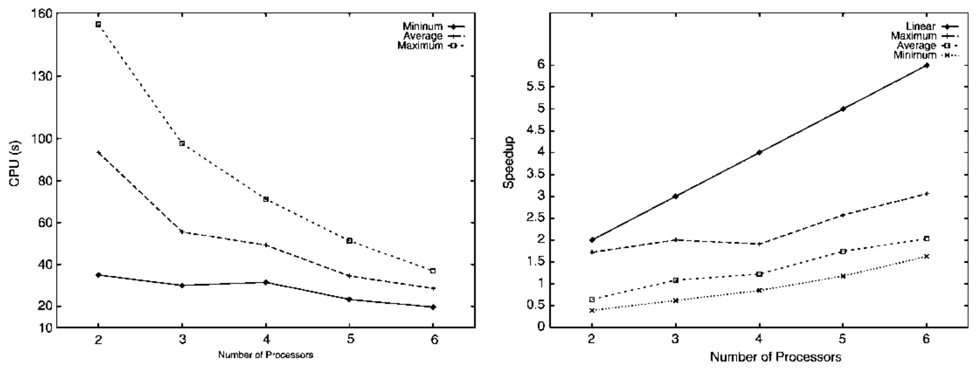


FIGURE 13 Results for the centralized parallel branch-and-bound algorithm.

CONCLUSIONS AND FINAL REMARKS

The importance of the MLNO problem was discussed. The MLNO problem integrates location, topological network design and dimensioning aspects. One possible mathematical programming formulation for the problem was presented and branch-and-bound algorithms based on this formulation were 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.

Concerning 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 alternatives? 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 [37]. 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.

Acknowledgements

The authors wish to thank Prof. J. MacGregor Smith and Prof. Afonso Ferreira for their careful reading and valuable comments on an earlier version of this paper. The research of Frederico Rodrigues Borges da Cruz and Geraldo Robson Mateus is supported in part by grants from the Brazilian agencies *Conselho Nacional de Desenvolvimento Científico e Tecnológico*, CNPq, grant 301809/96–8, and *Fundação de Amparo ã Pesquisa do Estado de Minas Gerais*, FAPEMIG, grant CEX-855/98.

References

[1] Mateus, G.R., Pádua, C.I.P.S. and Luna, H.P.L. (1996) "Integrated network models for local access network design", Proceedings of the International Telecommunications Symposium (Acapulco, Mexico), pp 6–10.

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

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

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

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

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

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

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

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

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

[11] Balakrishnan, A., Magnanti, T.L. and Mirchandani, P. (1994) "Modeling and heuristic worst-case performance analysis of two-level network design problem", *Management Science* **40**(7), 846–867.

[12] Laursen, P.S. (1993) "Simple approaches to parallel branch-and-bound", *Parallel Computing* **19**, 143–152.

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

[14] Kumar, V., Grama, A., Gupta, A. and Karypis, G. (1994) Introduction to Parallel Computing: Design and Analysis of Algorithms (The Benjamin/Cummings Publishing Company, Inc.).

[15] Ferreira, A. and Pardalos, P., eds (1996) Solving Combinatorial Optimization Problems in Parallel: Methods and Techniques, volume 1054 of LNCS State-of-the-Art Surveys (Springer-Verlag).

[16] Corrêa, R. and Ferreira, A. (1995) "Parallel best-first branch-and-bound in discrete optimization: a framework", In: Ferreira, A. and Pardalos, P., eds, Solving Combinatorial Optimization Problems in Parallel, volume 1054 of LNCS State-of-the-Art Surveys (Springer-Verlag), pp 171–200.

[17] Corrêa, R. and Ferreira, A. (1995) "A distributed implementation of asynchronous parallel branch and bound", In: Ferreira, A. and Rolim, J., eds, Solving Irregular Problems in Parallel: State of the Art (Kluwer Academic Publisher, Boston (USA)), pp 157–176.

[18] Corrêa, R. and Ferreira, A. (1995) "Modeling parallel branch-and-bound for asynchronous implementations", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **22**, 45–56.

[19] Dehne, F., Ferreira, A. and Rau-Chaplin, A. (1990) "Parallel branch and bound on fine grained hypercube multiprocessors", *Parallel Computing* **15**(1–3), 201–209.

[20] Corrêa, R. and Ferreira, A. (1995) "On the effectiveness of parallel branch and bound", *Parallel Processing Letters* **5**(3), 375–386.

[21] Corrêa, R., Ferreira, A. and Porto, S. (1998) "Selected algorithmic techniques for parallel optimization", In: Du, D. and Pardalos, P., eds, Handbook of Combinatorial Optimization (Kluwer Academic Publisher, Boston (USA)) Vol. **3**, pp 407–456.

[22] Lai, T. and Sahni, S. (1984) "Anomalies in parallel branch-and-bound algorithms", *Communications of the ACM* **27**, 594–602.

[23] Lai, T. and Sprague, A. (1985) "Performance of parallel branch-and-bound algorithms", *IEEE Transactions on Computers* **C-34**(10), 962–964.

[24] Li, G. and Wah, B. (1986) "Coping with anomalies in parallel branch-and-bound algorithms", *IEEE Transactions on Computers* **C-35**(6), 568–573.
[25] Pardalos, P. and Li, X. (1990) "Parallel branch-and-bound algorithms for combinatorial optimization", *Super-computer*, 23–30.
[26] Trienekens, H., (1990) "Parallel Branch and Bound Algorithms", Ph.D. Thesis, Erasmus University (Rotterdam).
[27] Clausen, J. and Traff, J.L. (1994) Do inherently sequential branch-and-bound algorithms exist? *Parallel Processing Letters*. **4**(1–2), 3–14.
[28] Tavares, A.I., Carvalho, M.L.B. and Mateus, G.R. (1995) "Aided design and analysis of distributed branch-and-bound algorithms", Annals of XV International Conference of the Chilean Society of Computer Science (Chilean Society of Computer Science, Arica, Chile), pp 448–458.
[29] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V. (1994) PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing (The MIT Press, Cambridge, Massachusetts).
[30] Karp, R.M. and Zhang, Y. (1993) "Randomized parallel algorithms for backtrack search and branch-and-bound computation", *Journal of the ACM* **40**(3), 765–789.
[31] Quinn, M.J. (1987) Designing Efficient Algorithms for Parallel Computers, 1st Ed. (McGraw-Hill Book Company, New York).
[32] Aneja, Y.P. (1980) "An integer linear programming approach to Steiner problem in graphs", *Networks* **10**, 167–178.
[33] Tanenbaum, A.S. (1981) Computer Networks (Prentice-Hall, New York).
[34] Beasley, J.E. (1990) "OR-library: distributing test problems by electronic mail", *Journal of the Operational Research Society* **41**(11), 1069–1072.
[35] Beasley, J.E. (1989) "An SST-based algorithm for the Steiner problem in graphs", *Networks* **19**, 1–16.
[36] Duin, C.W. and Volgenant, A. (1989) "Reduction tests for the Steiner problem in graphs", *Networks* **19**, 549–567.
[37] Laursen, P.S. (1994) "Can parallel branch-and-bound without communication be effective?", *SIAM Journal on Optimization* **4**(2), 143–152.

Frederico Rodrigues Borges da Cruz received his Doctor degree in Computer Science from *Universidade Federal de Minas Gerais*, Belo Horizonte, Brazil. His research areas include computational statistics and operations research. His papers have appeared in *Location Science*, *Computer and Operations Research, European Journal of Operations Research*, and *Journal of Statistical Computation and Simulation*. Currently, he is an associate professor in the *Departamento de Estatística* at *Universidade Federal de Minas Gerais*, Belo Horizonte, Brazil. E-mail: fcruz@est.ufmg.br

Geraldo Robson Mateus received his Doctor degree from *Universidade Federal do Rio de Janeiro*, Rio de janeiro, Brazil. His research areas include network optimization and operations research. Some of his papers have appeared in *Annals of Operations Research*, *Journal of Heuristics*, *Journal of the Operational Research Society*, *The Annals of Regional Science*, *Telecommunication Systems*, among others. Currently, he is a Professor in the *Departamento de Ciência da Computação*, at *Universidade Federal de Minas Gerais*, Belo Horizonte, Brazil. E-mail: mateus@dcc.ufmg.br