Towards a Grid Enabled Branch-and-Bound Algorithm

José Viterbo Filho [1], Lúcia M. A. Drummond [1],
Eduardo Uchoa [2], Maria Clícia S. Castro [3]
[1] Department of Computer Science – Fluminense Federal University
lucia@dcc.ic.uff.br (corresponding author)
[2] Department of Production Engineering – Fluminense Federal University
uchoa@producao.uff.br
[3] Department of Informatics – State University of Rio de Janeiro
clicia@ime.uerj.br

September 18th, 2003

Abstract

This work introduces a distributed branch-and-bound algorithm to be run on a wide-area-network system provided with an infra-structure that allows to execute applications that require much computational power, the so-called Grid. Although branch-and-bound algorithms are considered to be well suited for parallel implementations, its search trees can be highly unbalanced. Moreover, usually it is not possible to predict in advance the size of each subtree. Thus we propose a distributed branch-and-bound algorithm featuring procedures of load balance. The distributed algorithm presented was applied on an already existing sequential branch-and-bound algorithm for the Steiner Problem in Graphs. Good speedups were obtained, allowing the resolution of a number of formerly open instances from the SteinLib library in reasonable times. This fact indicates the great potential of the procedures developed.

Keywords: Branch-and-bound, distributed algorithms, load balance, Steiner Problem in Graphs

1 Introduction

Some of the most important real world optimization problems are classified as NP-hard. Branch-and-bound is by far the most widely used technique for finding the optimal solution of those problems. Branch-and-bound algorithms searches the space of solutions through implicit enumeration in subtrees, since the complete enumeration would be impossible due to the exponentially increasing number of potential solutions. The calculation of good upper and lower bounds for the optimal solutions in the subtrees is used to limit the growing of the enumeration tree.

Branch-and-bound algorithms are considered to be well suited for parallel implementations because the computation of subtrees can be accomplished independently. Essentially the only global information in the algorithm is the value of the current best solution.

This works aims to develop a distributed branch-and-bound algorithm to be run on a wide-area-network system provided with an infra-structure that allows to execute applications that require much computational power, the so-called Grid. Large scale computational grids are gaining popularity. Examples of such grids exist in the academic world (SETI@Home[2], Globus[10], and Nile[17]) as well in the commercial sector (Entropia and Parabon among others).

We propose a distributed branch-and-bound algorithm featuring procedures of load balance. Initially a subtree is associated to each process. Upon finishing the execution, it asks to a neighbor

process for another part of its local subtree. The algorithm does not use a master process to control the balance because it could become a bottleneck. It favors the load balance among processors concentrated geographically, since, typically, grids are usually composed of clusters whose processors are connected via high-speed links and the clusters, geographically distant, are connected through low-speed links, in a hierarchical fashion. The termination procedure proposed also takes advantage of this usual hierarchical structure of grids.

The distributed algorithm was applied on an already existing branch-and-bound algorithm for solving the Steiner Problem in Graphs (SPG), a classical NP-hard problems. This sequential algorithm is, to the best of our knowledge, the only capable of solving almost all "incidence instances", a set of 400 hard SPG instances proposed by Duin [9] in 1993 as a benchmark and a challenge for future algorithms. Today, as can be checked in the SteinLib website [13], only 20 incidence instances could not be solved yet. Our initial goal in parallelizing that branch-and-bound was to obtain the solution of those 20 remaining instances. As can be seen in our preliminary computational results, this goal was partially fulfilled, since 5 instances could be solved for the first time using 16 processors. Solving the 15 remaining instances, at least with that branch-and-bound algorithm, will probably require the computing power from hundreds of processors.

The computational experiments are been run on a grid that gathers clusters from some universities from Rio de Janeiro. The programming language used was C++ and Message Passage Interface (MPI), version MPICH-G2, the Globus MPI version for parallelism [12].

The remainder of this paper is organized as follows. In the next section we present the sequential branch-and-bound algorithm. Section 3 presents the related work. Section 4 introduces the proposed distributed algorithm. In Section 5 we summarize our experimental environment and preliminary results are shown. Finally, in Section 6, we present our conclusions.

2 Sequential Branch-and-Bound for the SPG

The SPG is defined as follows: given a graph $G = (V, E)$, positive edge costs c and a set $T \subseteq V$ of *terminal vertices*, find a connected subgraph (V', E') of G with $T \subseteq V'$ minimizing $\sum_{e \in E'} c_e$. The directed cut formulation (DCF) of SPG works over the directed graph $G_D = (V, A)$ obtained by replacing each edge in E by two opposite arcs and choosing any terminal r to be the *root*. Let W be the collection of all vertex-sets w containing some terminal but not the root and let $\delta^-(w)$ be the directed cut made up by the arcs entering w. Let x_a indicate whether arc a belongs to the solution (an arborescence rooted at r) or not. The linear relaxation of DCF, (P), and its dual, (D), are:

$$
\begin{array}{cll}
 & \text{Min} & c(x) = \sum(c_a x_a \,:\, a \in A) \\
(P) & \text{s.t.} & x(\delta^-(w)) \geq 1,\ \forall w \in W \\
 & & x_a \geq 0,\ \forall a \in A
\end{array}
$$

$$
\begin{array}{cll}
 & \text{Max} & v(y) = \sum(y_w \,:\, w \subset W) \\
(D) & \text{s.t.} & \sum(y_w \,:\, a \in \delta^-(w)) \leq c_a,\ \forall a \in A \\
 & & y_w \geq 0,\ \forall w \in W
\end{array}
$$

Strong formulations for the SPG, like DCF, usually give lower bounds quite close to the optimum integer value. However, the solution of those formulations can be very expensive. Wong [23] proposed a *dual ascent*, a fast algorithm for the approximate solution of (D). Such approximate solutions also give valid lower bounds. Let y be a feasible solution of (D). Denote the reduced cost w.r.t. y by $\bar{c}_a = c_a - \sum(y_w : a \in \delta^-(w))$. Initialize $y = 0$ and at each iteration increase y_w as much as possible, where w corresponds to a maximal set of vertices that can reach a terminal, but not

r, by arcs of zero reduced cost. Good primal solutions may be obtained searching the final set of zero reduced cost arcs. Poggi de Aragão, Uchoa and Werneck [18] proposed three additional dual heuristics to improve the lower bounds and reduce instance sizes.

- **Dual Scaling:** Given a feasible solution y of (D), let $y' = \alpha y$, where $\alpha \in [0, 1]$ (typically $\alpha = 0.875$). Then apply dual ascent starting from y'. This simple procedure may be enough to improve the lower bounds.

- **Dual Adjustment:** Let y be a feasible dual solution and $\bar{x}$ be the best known integral solution. Make y' equal to y, but with $y'_w = 0$ for every w such that $\delta^-(w)$ contains more than one arc in $\bar{x}$. Afterwards, apply dual ascent starting from y'. Apart from improving lower bounds, this procedure may help to eliminate arcs by reduced costs.

- **Active Fixation by Reduced Costs:** Let y be a dual solution with cost Z_d, and let Z_{INC} be the cost of the best known integral solution. Given an arc a such that $c_a > Z_{INC} - Z_d$, make y' equal to y, but with $y'_w = 0$ for every w such that $a \in \delta^-(w)$. Then apply a modified dual ascent forbidding the use of a, to get a new dual solution y''. After that, perform dual ascent again allowing the use of a, obtaining y'''. The reduced cost $\bar{c_a}$ w.r.t. y'' is equal to c_a. If the cost of y'' plus c_a exceeds Z_{INC}, then arc a can be fixed to 0. Other arcs can be fixed using solution y'''.

The resulting branch-and-bound algorithm can be described by the pseudo-code shown in Figure 1. The recursive function **ProcessNode** receives as parameters a directed Steiner instance represented as $\langle G_D, c, T\rangle$ and the the best dual solution y^* of its father ($\mathbf{0}$ in the root node). The functions DUALASCENT(y), DUALADJUST(y) and ACTIVEFIX(y) perform dual ascent, dual adjust and active fixation over a dual solution y. The branch rule chooses a vertex v incident to a maximum number of saturated arcs. In the first subproblem v and its adjacent arcs are removed from G_D. In the second subproblem v is considered as a terminal vertex.

Further details on this branch-and-bound algorithm [1] can be found in [18, 21, 22]. This algorithm represented a significant improvement in the solution of SPG instances with relatively few vertices (many edges are no problem). It could solve about 50 incidence instances for the first time, the larger one with 640 vertices and 200,000 edges. Only 20 instances from that set could not be solved in reasonable time (less than one day on a PC). The wish to solve those remaining instances was our first motivation to work on distributed branch-and-bound algorithms.

The sequential branch-and-bound for the Steiner has the following good characteristics for parallelization on grids: there are many nodes in the tree whose evaluations are fast, what favors load balance; and it does not use expensive commercial LP solvers, such as CPLEX or XPRESS, that can not be assumed to exist on machines in a grid.

3 Related Work

There are many papers about parallel branch-and-bound algorithms [1, 3, 4, 6, 8, 11, 14, 15].

Some of them present libraries dedicated to the development of these methods such as PPBB-Lib [12], ZRAM [7] and BOB [19].

Anomalies of speedups in parallel branch-and-bound algorithms have also been investigated. One would expect that attacking a problem with p processors instead of one, resulted in speedups

[1] The branch-and-bound code is included in a large computational package named *BOSSA* and is available upon request to its authors.

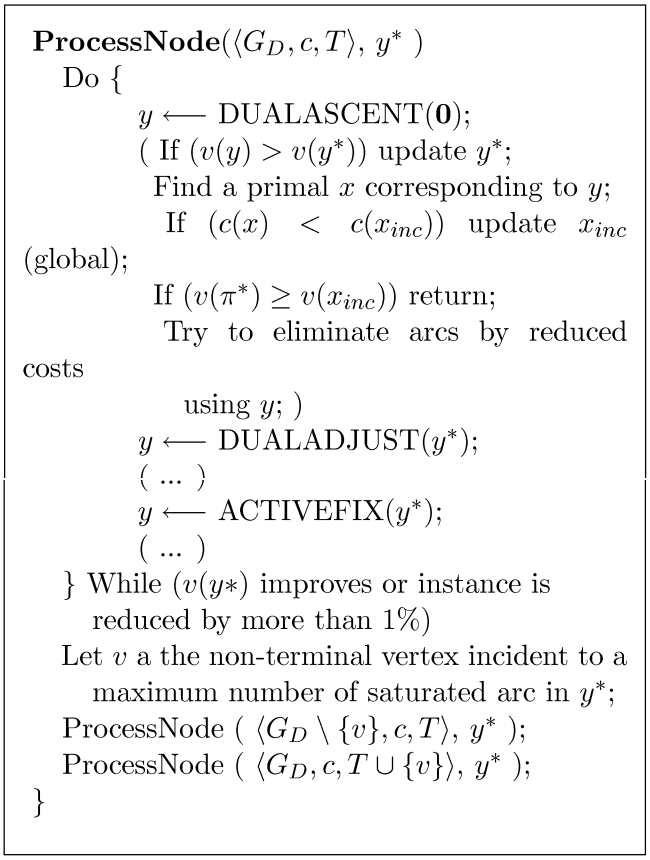

Figure 1: Sequential Branch-and-Bound Algorithm

close to p, but in examples of parallelizations of branch-and-bound algorithms some anomalies have been observed: detrimental anomalies (speedup less than 1), and acceleration anomalies (speedups greater than p) [4, 14].

The strategies of search in parallel have also been researched. The Best-First strategy and the Depth-First strategy are regarded as the main strategies for solving combinatorial optimization problems by parallel branch-and-bound algorithms. The Best-First because of efficiency with respect to the number of nodes explored and the Depth-First for reasons of space efficiency. Some results about the performance of these strategies in parallel algorithms can be found in [6, 15].

Implementations of parallel branch-and-bound in specific distributed systems have been conducted in [3] that presented parallel branch-and-bound algorithms to be run on NOWS and in [1] that presented a hierarchical master-worker algorithm for parallel branch-and-bound to be executed on grids.

We can cite other papers that consider the development of parallel branch-and-bound algorithms for specific optimization problems such as [5, 8, 16].

4 A Distributed Branch-and-Bound Algorithm

A branch-and-bound algorithm sets out to traverse the entire search space by recursively refining the search space into disjoint subspaces. A branch-and-bound tree is produced by the branching procedure. At any given point each node represents a subtree. Any of these containing feasible solutions within the current bounds are subject for further investigation which is done by recursively splitting into subtrees. If a feasible solution exists, we eventually find it as a leave node. The evaluation of nodes are independent tasks resulting in a lot of opportunities for parallelism.

The distributed algorithm proposed is composed by the following procedures: Initial distribution, Leader election, Load balance, Pruning and Termination detection, described in the next subsections.

4.1 Initial Distribution

The initial distribution phase corresponds to the initial assignment of subtrees to processes. At each branching a process sends a message, called *Branch* with the new node of the branch-and-bound algorithm to what we call next process (*nextproc*). A process of identification i selects *nextproc*, to send the new node as follows: $nextproc = i + 2^{level}$, where *level* represents the depth of i in the tree. When *nextproc* reaches a value greater than the number of processors available, process i stops sharing its local load at each branching.

Consider this procedure with 8 processors and consider that only one process is executed on each processor. We identify each processor as $P_0, P_1, P_2, P_3, P_4, P_5, P_6$ and P_7. Initially, P_0 shares its load with P_1 (level=0) and goes on processing the other half of the tree. In the second level (level=1) of the tree, P_0 shares its load with P_2, while P_1 shares its load with P_3. In the third level (level=2), P_0 shares its load with P_4, P_1 shares its load with P_5, P_2 with P_6, and P_3 with P_7. See Figure 2.

4.2 Leader Election

At each cluster a leader is chosen that will be helpful in the other procedures. Each cluster elects as leader the process with the smallest identification and P_0 is the application leader.

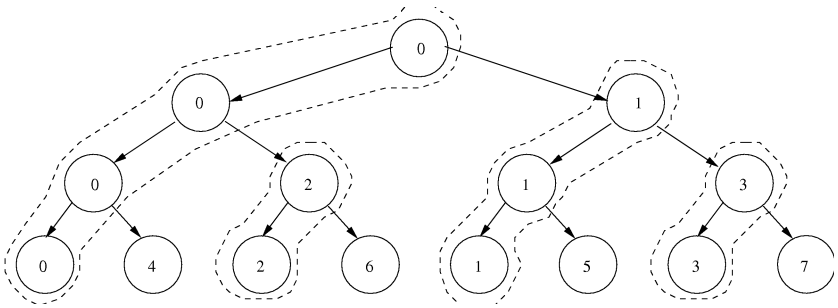

Figure 2: Load Distribution

4.3 Load Balance

Because the required time to process a subtree may vary a lot, a procedure to balance the load among the processors is frequently employed in this problem. Distributed dynamic load balance methods can be classified according to the acting component of the load transfer (sending process or receiving process) [6]. There are two types of load balance algorithms: sender initiated and receiver initiated that are described next.

The difference between them is that in the sender initiated algorithm, a processor with load to share looks for potential receivers while in the receiver initiated processors having low load send load requests to other processors.

We implemented the algorithm receiver initiated with a round-robin strategy. Thus, upon finishing the execution of a subtree, each process sends a message with the result (*Result*) to process 0 and asks to a neighbor for another part of its subtree. The algorithm proposed favors the load balance among processors concentrated geographically, since, typical grids are composed of clusters whose processors are connected via high-speed links and the clusters, that are geographically distant, are connected through low-speed links.

In this way, each process i sends a message, called *Loadrequest*, to a process j whose rank is calculated as follows: $j = (i+1)\ mod\ (cs) + cl$ where cs represents the number of processors in the cluster to which process i belongs and cl the cluster leader, in order to request load. If this process does not have load to share, it sends in turn a message to process i, called $Idle$. In this case process i tries to get load of another process $j = (i+2)\ mod\ (cs) + cl$. If it receives a message *Idle* again, process i still tries a process $j = (i+3)\ mod\ (cs) + cl$ and so on. Finally, if process i is not able to obtain a subtree from any of its neighbors it begins a termination procedure, described later. The load balance strategy can be adapted according to the cluster sizes. To implement this strategy a static schedule of processes to processors was defined so that processes of consecutive identifications are assigned to processors of the same cluster whose size is known *a priori*. The main objective of this procedure is to favor balance among processors concentrated within the same site.

4.4 Pruning

The calculation of good upper bounds for the optimal subtrees is used to limit the growing of the enumeration tree. The pruning procedure is responsible for broadcasting the upper bound among the processors across the grid.

Thus, upon finding a primal, each process sends it to the local leader and to processes within the same cluster. The leader then broadcasts it to other leaders which in turn forward it to processors in their respective clusters. Upon receiving a *Primal* message, if the received primal is smaller than the local one, the local value is replaced by the new one.

When an early pruning occurs in the initial distribution phase, it prevents the respective process from sending a branch message to another one that is waiting for it. In this case the process which has no load to share sends an *Idle* message to that waiting process. This one in turn is free to ask load for another process.

4.5 Termination Detection

In a distributed algorithm, where there is no central point of global control as well no previous knowledge of the number of messages to be exchanged by processes, each process requires a global knowledge of the application to determine when the overall computation is done and consequently to finish its local computation. In our algorithm, in order to terminate the algorithm a process sends the message *SuspectEnd* to the leader of its cluster. The leader upon receiving this message from all processes of its cluster forwards it to all other leaders. A leader upon receiving this message from all other leaders and also from all other processes of its cluster, broadcasts this message to the processes of its cluster and terminates. Other processes terminate their processing upon receiving this message from their leaders.

4.6 Types of Messages

We can summarize the set of types of messages exchanged as follows:

- *Branch*: it contains the nodes of the tree previously executed.
- *LoadRequest*: it contains a request for load.
- *Idle*: it contains the information that the process is sending that this message does not have load to share.
- *Primal*: it contains the new upper-bound (primal).
- *Result*: it contains the result found by the sender.
- *SuspectEnd*: it contains the information that the process sending this message finished its local execution.

5 Preliminary Results

Our parallel algorithm has been developed in the programming language C++ and MPICH-G2 [12], a Globus-enabled version of the MPI parallel programming toolkit.

Our experiments will be run on GridRio that is an initiative to create a research oriented computational grids across the state of Rio de Janeiro. Today, two universities take part of GridRio: UFF (Universidade Federal Fluminense) and PUC (Pontifícia Universidade Católica do Rio de Janeiro). The following resources are currently available in GridRio: 8 PC's Athlon with 1.3GHz of clock, and 2 PC's Pentium III with 800MHz of clock running the Globus Toolkit version 2.4 and 10 Red Hat Linux version 7.3 at UFF and 32 PC's Pentium IV with 1.4 GHz of clock, running the Globus Toolkit version 2.2 and Linux at PUC. It must be extended with the participation of two other institutions.

In our preliminary experiments we ran our tests on the PUC cluster using 16 processors. We executed each instance three times.

The best known results and the results obtained by our algorithm are presented in Table 1. The execution times, in seconds and hours, required to execute each instance sequentially, in parallel, and its corresponding speedups are also presented.

In order to test the algorithm we used some open instances of a benchmark that is a set of 400 hard SPG instances. Our initial goal in parallelizing that branch-and-bound was to obtain the solution of the 20 remaining open instances.

In our preliminary computational results, this goal was partially fulfilled, since 5 instances could be solved for the first time using 16 processors with good speedups.

Instance	Best Known	Optimum	Sequential Time (second/hour)	Parallel Time (second/hour)	Speedup
i640-211	12022	11984	524387.2 / 145.6	40877.99 / 11.3	12.82
i640-212	11795	11795	13794.5 / 3.8	3755.08 / 1.04	3.67
i640-213	11879	11879	37429.4 / 10.3	3034.06 / 0.84	12.33
i640-214	11898	11898	350482.4 / 97.3	38262.58 / 10.6	9.16
i640-215	12081	12081	105131.8 / 29.2	9391.87 / 2.6	11.19

Table 1: Results of the sequential and parallel algorithms

The algorithm proposed presented good speedups, although in the instance i640-212 the speedup was low. It occurs probably due to the branching and pruning behavior of this instance. If a good solution is, by chance, discovered early, the subtrees can be much better pruned.

Anomalies of parallel branch-and-bound algorithms speedups have been extensively studied, as can be seen in related literature.

6 Concluding Remarks

The distributed branch-and-bound algorithm that we proposed was able to obtain the solution of 5 open instances of a benchmark for the SPG problem with good speedups.

We developed load balance and termination algorithms adapted to the usual hierarchical structure of grids aiming to reduce the communication among the clusters, that are connected through low-speed links. We intend to run the other 15 instance of the problem on GridRio, using also the UFF cluster. Those instances probably require much more computational power and will benefit by our procedures.

Another important point to be tackled is the fault tolerance of parallel algorithms executed on grids. This is important mainly due to our long execution time applications. Thus, we also intend to include procedures of fault tolerance in our parallel algorithm allowing for them to continue the execution despite the presence of faults. The strategy being developed is based on independent checkpoints. Periodically, each process will ask to a neighbor process within the same cluster for saving its checkpoint without coordination with other processes. The algorithm is being designed to tolerate only one single permanent fault.

References

1| K. Aida, W. Natsume and Y. Futakata, *Distributed Computing with Hierarchical Master-Worker Paradigm for Parallel Branch-and-Bound Algorithm*, Proceedings of Third International Symposium on Cluster Computing and Grid, pp. 156-162, 2003.

2] D. Anderson and D. Werthimer, *The seti@home Project*, http://stiahome.ssl.berkeley.edu/.

3] R. Batoukov and T. Soverik, *A Generic Parallel Branch-and-Bound Environment on a Network of Workstations*, Proceedings of Hiper-99, pp.474-483, 1999.

4] A. Bruin, G.A.P. Kindervater and H.W.J.M. Trienekens, *Asynchronous Parallel Branch-and-Bound and Anomalies*, Erasmus University, Department of Computer Science, EUR-CS-95-05, Roterdam, Netherlands,1995.

5] J. Clausen and M. Perregaard, *Solving Large Quadratic Assignment Problems in Parallel*, Technical Report DIKU 1994/22, 1994.

6] J. Clausen, *Parallel Branch-and-Bound, Principles and Personal Experiments*, Kluwer Academic Publishers, pp. 239-267, 1997.

7] T. Crainic and C. Roucairol, editors, *Parallel Optimization*, Proceedings of POC'S99, Annals of OR, University of Versailles-Saint Quentin and Yvelines and University of Montreal, 1999.

8] Y. Denneulin, B. Le Cun, T. Mautor and J.-F. Mhaut, *Distributed Branch-and-Bound Algorithms for Large Quadratic Assignment Problems*, Proceedings of the Fifth Computer Science Technical Section on Computer Science and Operation Research, Dallas-USA, 1996.

9] C. Duin, *Steiner's Problem in Graphs*, Ph.D. thesis, University of Amsterdam, 1993.

[10] I. Foster and C. Kessekman, *The Globus Project: a Status Report*, Proceedings of the Seventh Heterogeneous Computing Workshop (HCW'98), pp. 4-18, 1998.

[11] B. Gendron and T.G. Cranic, *Parallel Branch-and-Bound Algorithms: Survey and Synthesis*, Operation Research 42, pp. 1042-1066, 1994.

[12] S. Tschöke and T. Polzer, *Portable Parallel Branch-and-Bound Library (ppbb-lib0)*, User Manual Version 2.0, University of Paderborn, 1999, http://www.globus.org/.

[13] T. Koch and A. Martin and S. Voss, *SteinLib: An Updated Library on Steiner Tree Problems in Graphs*, Konrad-Zuse-Zentrum für Informationstechnik Berlin, ZIB-Report 00-37, Berlin, 2000, http://elib.zib.de/steinlib.

[14] T.-H. Lai and S. Sahni, *Anomalies in Parallel Branch-and-Bound Algorithms*, Communications of the ACM 27, pp. 594-602, 1984.

[15] B. Mans and C. Roucairol, *Performance of Parallel Branch-and-Bound Algorithms with Best-first Search*, Discrete Applied Mathematics 66 (1), pp. 57-76, 1996.

[16] M. Perregaard and J. Clausen, *Solving Large Job Shop Scheduling Problems in Parallel*, Technical Report DIKU 1994/35, 1994.

[17] F. Previato, M. Ogg, and A. Ricciardi, *Experience with Distributed Replicated Objects: The Nile Project*, Theory and Practice of Object Systems 4(2), pp. 107-115, 1998.

[18] M. Poggi de Aragão and E. Uchoa and R. F. Werneck, *Dual Heuristics on the Exact Solution of Large Steiner Problems*, Proceedings of the Brazilian Symposium on Graphs, Algorithms and Combinatorics, Electronic Notes in Discrete Mathematics Vol. 7, 2001.

[19] Catherine Roucairol, Van-Dat Cung and Nassir Yahfoufi, *Design, Implementations and Robustness in Parallel Branch-and-Bound*, Technical Report PRISM 2000/19, l'Université de Versailles Saint-Quentin, 2000.

[20] M. Snir, S. M. Otto, S. Huss-Lederman and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, 1996.

[21] E. Uchoa, *Algoritmos para problemas de Steiner com aplicações em projeto de circuitos VLSI*, Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2001.

[22] *Problema de Steiner em Grafos: Algoritmos Primais, Duais e Exatos*, Master's Thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2001

[23] R. Wong, *A Dual Ascent Approach for Steiner Tree Problems on a Directed graph*, Mathematical Programming 28, pp.271-287, 1984.