PARALLEL IMPLEMENTATIONS OF BOUNDING PROCEDURES FOR MULTICOMMODITY CAPACITATED NETWORK DESIGN PROBLEMS

Bernard Gendron
Centre de recherche sur les transports
and Département d'informatique
et de recherche opérationnelle
Université de Montréal

Teodor Gabriel Crainic
Centre de recherche sur les transports
Université de Montréal
and Département des sciences administratives
Université du Québec à Montréal

September 1994

Abstract

This paper presents and analyzes parallel implementations of bounding procedures for multicommodity capacitated network design problems. Experiments on a wide variety of problems are conducted on three different environments. We assess the relative merits of these architectures when used to solve our problems and we show that the parallel implementations can significantly alleviate the task of solving large-size realistic instances, having millions of variables and thousands of commodities.

Key words : Multicommodity capacitated network design, relaxation methods, parallel computation.

Résumé

Dans cet article, nous présentons et analysons des implantations parallèles de procédures de calcul de bornes pour des problèmes de conception de réseaux multiproduit avec capacité. Nous décrivons les résultats d'essais numériques sur un grand éventail de problèmes en utilisant trois architectures différentes. Nous évaluons les mérites relatifs de ces architectures et nous montrons que les implantations parallèles peuvent grandement alléger la tâche de résoudre des exemplaires de grande taille, ayant plusieurs millions de variables et des milliers de produits.

Mots-clés : Conception de réseau multiproduit avec capacités, méthodes de relaxation, calcul parallèle.

1 Introduction

Network design models have been known for long as useful planning tools in areas such as transportation, telecommunications, manufacturing and logistics, among others. Comprehensive surveys on the applications of network design models and their resolution by mathematical programming techniques can be found in papers by Magnanti and Wong [13], Minoux [14], and Magnanti [11]. As pointed out by the last author, *multicommodity capacitated* versions of a general *fixed charge network design model* pose considerable algorithmic challenges to researchers. In particular, the complexity of solving these models tends to increase proportionally with the number of commodities [4]. An attractive way to tackle this problem, that we explore in this paper, is to develop solution methods that decompose by commodities and can be efficiently implemented in parallel computing environments.

In an earlier paper [4], we presented and compared several relaxations of a multicommodity capacitated fixed charge network design problem. Among these relaxations, the best results were obtained by a procedure that decomposes by commodities and exploits the network structure of the problem. We also proposed and implemented a heuristic based on the resource-decomposition principle, used in solving multicommodity network flow problems. In this paper, we present and analyze parallel implementations of the resulting bounding procedures on coarse-grained MIMD message-passing architectures.

The paper is organized as follows. In Section 2, we formulate the problem and describe the bounding procedures. Their implementation in a message-passing parallel computing environment is the subject of Section 3. Section 4 reports the results of experiments conducted on a network of Transputers, a distributed network of workstations and a shared-memory multiprocessor that simulates a message-passing environment. In Conclusion, we summarize our results and discuss extensions to this work.

2 Formulation and Bounding Procedures

We consider a *directed network* $G = (N, A)$, where N is the set of nodes and A is the set of arcs, on which several commodities, represented by set P, are moving. For each commodity p, the set of nodes may be partitioned into three subsets: $O(p)$, the set of nodes which send more flow of commodity p than the quantity they receive, called *origin nodes*; $D(p)$, the set of nodes which receive more flow of commodity p than the quantity they send, called *destination nodes*; $T(p)$, the set of nodes which receive and send the same quantity of flow of commodity p, called *transshipment nodes*. For each commodity p, we associate to each origin $i \in O(p)$ a positive *supply* o_i^p, and to each destination $i \in D(p)$ a positive *demand* d_i^p. We assume that $\sum_{i \in O(p)} o_i^p = \sum_{i \in D(p)} d_i^p = t_p$ for each commodity p. To each arc (i, j), we associate a positive *total capacity* u_{ij} on the amount of flow of all commodities that can move through the arc, and nonnegative *partial capacities* b_{ij}^p on the amount of flow of commodity p that can move through the arc. Without loss of generality, we assume that $b_{ij}^p \leq \min(u_{ij}, t_p)$, for each arc (i, j) and each commodity p, and that $u_{ij} \leq \sum_{p \in P} b_{ij}^p$, for each arc (i, j). A nonnegative *routing cost* c_{ij}^p is incurred for each unit of flow of commodity p that moves through arc (i, j). Moreover, a nonnegative *fixed cost* f_{ij} is added when any amount of flow moves through arc (i, j). The problem consists in minimizing the sum of all costs, while satisfying supply and demand requirements, flow conservation at transshipment nodes, and capacity constraints.

To formulate the problem, we define continuous nonnegative *flow variables* x_{ij}^p, which represent the amount of flow of commodity p moving on arc (i, j), and binary *design variables* y_{ij}, defined as follows:

$$y_{ij} = \begin{cases} 0, & if\ x_{ij}^p = 0,\ \forall\ p \in P \\ 1, & otherwise \end{cases} \qquad \forall\ (i, j) \in A \tag{1}$$

We also introduce the sets of outward and inward neighbors, respectively, of any node i: $N^+(i) = \{j \in N\,|(i, j) \in A\}$; $N^-(i) = \{j \in N\,|(j, i) \in A\}$. The problem may then be formulated as follows:

$$Z = \min \sum_{p \in P} \sum_{(i,j) \in A} c_{ij}^p x_{ij}^p + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{2}$$

$$\sum_{j \in N^+(i)} x_{ij}^p - \sum_{j \in N^-(i)} x_{ji}^p = \begin{cases} o_i^p & if\ i \in O(p) \\ -d_i^p & if\ i \in D(p) \\ 0 & if\ i \in T(p) \end{cases} \quad \forall\, i \in N,\ p \in P \tag{3}$$

$$x_{ij}^p \geq 0 \quad \forall\, (i,j) \in A,\ p \in P \tag{4}$$

$$\sum_{p \in P} x_{ij}^p \leq u_{ij} y_{ij} \quad \forall\, (i,j) \in A \tag{5}$$

$$x_{ij}^p \leq b_{ij}^p y_{ij} \quad \forall\, (i,j) \in A,\ p \in P \tag{6}$$

$$y_{ij} \leq 1 \quad \forall\, (i,j) \in A \tag{7}$$

$$y_{ij} \geq 0 \quad \forall\, (i,j) \in A \tag{8}$$

$$y_{ij}\ integer \quad \forall\, (i,j) \in A \tag{9}$$

2.1 Lower Bounding Procedure

To compute lower bounds, we propose an approach based on relaxing constraints 5 and 6 [4]. The resulting Lagrangean subproblem takes the following form:

$$Z(\alpha,\beta) = \min \sum_{p \in P} \sum_{(i,j) \in A} \left(c_{ij}^p + \alpha_{ij} + \beta_{ij}^p\right) x_{ij}^p + \sum_{(i,j) \in A} \left(f_{ij} - \alpha_{ij} u_{ij} - \sum_{p \in P} \beta_{ij}^p b_{ij}^p\right) y_{ij} \tag{10}$$

subject to 3, 4, 7, 8 and 9, where α and β are nonnegative Lagrangean multipliers associated to constraints 5 and 6, respectively. This problem decomposes into two parts. The first part, which depends only on the flow variables, is a multicommodity uncapacitated minimum cost network flow problem, and thus decomposes itself into $|P|$ single-commodity minimum cost network flow problems (for a recent survey of methods for solving this type of problems, see Ahuja, Magnanti and Orlin [1]). The second part, which depends only on the design variables, may be solved easily, without considering the integrality constraints 9:

$$\sum_{(i,j) \in A} \min\left(0, f_{ij} - \alpha_{ij} u_{ij} - \sum_{p \in P} \beta_{ij}^p b_{ij}^p\right) \tag{11}$$

Since the Lagrangean subproblem can be solved without the integrality constraints, it has the Integrality property of Geoffrion [5]. Thus, the best lower bound one can hope for when using this Lagrangean relaxation is equal to the optimal value of the continuous

relaxation, obtained by dropping the integrality constraints 9. Gendron and Crainic [4], however, report computational experiments suggesting that the Lagrangean relaxation approach outperforms simplex methods, used in solving the continuous relaxation.

As is well–known (see, for example, Nemhauser and Wolsey [15]), the Lagrangean dual, considered as a a function of α and β, is concave, but nondifferentiable. To maximise it, and consequently derive the best Lagrangean lower bound, we use an adaptation of the subgradient method, based on the early work of Held, Wolfe and Crowder [7] (for a recent survey of nondifferentiable optimization techniques, see Lemaréchal [9]). Given an optimal solution $(\overline{x}, \overline{y})$ to the Lagrangean subproblem, a subgradient $(s(\alpha), s(\beta))$ of the Lagrangean dual at (α, β) is given by:

$$s_{ij}(\alpha) = \sum_{p \in P} \overline{x}_{ij}^p - u_{ij}\overline{y}_{ij} \quad \forall\, (i,j) \in A \tag{12}$$

$$s_{ij}^p(\beta) = \overline{x}_{ij}^p - b_{ij}^p \overline{y}_{ij} \quad \forall\, (i,j) \in A,\ p \in P \tag{13}$$

At each step of the lower bounding procedure, the Lagrangean subproblem is solved and the multipliers are adjusted by moving $\theta(\alpha, \beta) = \lambda(Z^u - Z(\alpha, \beta))/||(s(\alpha), s(\beta))||^2$ along the direction of $(s(\alpha), s(\beta))$, where $0 < \lambda \leq 2$, Z^u is an upper bound on the network design problem, and $||.||$ denotes the Euclidean norm. We describe in the next section an upper bounding procedure to compute Z^u. To approximate as closely as possible the optimal value of the Lagrangean dual, we allow the scalar λ to vary. To determine an appropriate adjustment of λ, we count the number of consecutive steps without improvement in the lower bound. When this number exceeds a given value, we halve the current value of λ (unless the result is smaller than a threshold, fixed at 0.00001; we then set λ to this value to avoid numerical problems).

The lower bounding procedure may be performed in alternation with the upper bounding procedure. Given known values (α, β) for the multipliers, $\widetilde{Z}^l$, the best known lower bound, l^*, the number of consecutive steps (in previous executions of the procedure) without improvement in the lower bound, and the value of the scalar λ, the lower bounding procedure may be stated as follows:

1. Initialize iteration counter: $l \leftarrow 1$.

2. Solve the x-problem ($|P|$ minimum cost network flow problems) to obtain $\overline{x}$;
   $Z(\alpha,\beta) \leftarrow \sum_{p\in P}\sum_{(i,j)\in A}(c_{ij}^p + \alpha_{ij} + \beta_{ij}^p)\overline{x}_{ij}^p$;
   $s_{ij}(\alpha) \leftarrow \sum_{p\in P}\overline{x}_{ij}^p,\ \forall (i,j)\in A$;
   $s_{ij}^p(\beta) \leftarrow \overline{x}_{ij}^p,\ \forall (i,j)\in A, p\in P$.

3. Solve the y-problem to obtain $\overline{y}$;
   $Z(\alpha,\beta) \leftarrow Z(\alpha,\beta) + \sum_{(i,j)\in A}\min(0, f_{ij} - \alpha_{ij}u_{ij} - \sum_{p\in P}\beta_{ij}^p b_{ij}^p)$;
   $s_{ij}(\alpha) \leftarrow s_{ij}(\alpha) - u_{ij}\overline{y}_{ij},\ \forall (i,j)\in A$;
   $s_{ij}^p(\beta) \leftarrow s_{ij}^p(\beta) - b_{ij}^p\overline{y}_{ij},\ \forall (i,j)\in A, p\in P$.

4. If $Z(\alpha,\beta) > \widetilde{Z}^l$, $\widetilde{Z}^l \leftarrow Z(\alpha,\beta)$ and $l^* \leftarrow 0$; go to 6.

5. Increment l^*; if $l^* \geq l^*_{\max}$, update the parameter λ and reinitialize l^*: $\lambda \leftarrow \max\left(0.00001, \frac{\lambda}{2}\right)$ and $l^* \leftarrow 0$.

6. Update (α,β) by moving along the direction of $(s(\alpha), s(\beta))$: $\alpha \leftarrow \alpha + \theta(\alpha,\beta)s(\alpha)$, $\beta \leftarrow \beta + \theta(\alpha,\beta)s(\beta)$.

7. If $l \geq l_{\max}$, stop; otherwise, increment l and go to 2.

2.2 Upper Bounding Procedure

To compute upper bounds, we consider the following multicommodity capacitated network flow problem, which is defined relative to a subset $\overline{A}$ of A:

$$Z(M) = \min \sum_{p\in P}\sum_{(i,j)\in\overline{A}}\left(c_{ij}^p + f_{ij}/u_{ij}\right)x_{ij}^p \tag{14}$$

$$\sum_{j\in\overline{N}^+(i)} x_{ij}^p - \sum_{j\in\overline{N}^-(i)} x_{ji}^p = \begin{cases} o_i^p & if\ i\in O(p) \\ -d_i^p & if\ i\in D(p) \\ 0 & if\ i\in T(p) \end{cases} \quad \forall\ i\in N,\ p\in P \tag{15}$$

$$\sum_{p\in P} x_{ij}^p \leq u_{ij} \quad \forall\ (i,j)\in\overline{A} \tag{16}$$

$$x_{ij}^p \leq b_{ij}^p \quad \forall\ (i,j)\in\overline{A},\ p\in P \tag{17}$$

$$x_{ij}^p \geq 0 \quad \forall\ (i,j)\in\overline{A},\ p\in P \tag{18}$$

In this formulation, for every node i, the sets of outward and inward neighbors, with respect to $\overline{A}$, are denoted $\overline{N}^{+}(i)$ and $\overline{N}^{-}(i)$, respectively.

Given a feasible solution $\overline{x}$ to this multicommodity network flow problem, an upper bound on the optimal value of the network design problem is given by:

$$Z^u = \sum_{p \in P} \sum_{(i,j) \in \overline{A}} c_{ij}^p \overline{x}_{ij}^p + \sum_{(i,j) \in \overline{A}} f_{ij} \overline{y}_{ij} \tag{19}$$

where

$$\overline{y}_{ij} = \begin{cases} 0, & if\ \overline{x}_{ij}^p = 0,\ \forall\ p \in P \\ 1, & otherwise \end{cases} \qquad \forall\ (i,j) \in \overline{A} \tag{20}$$

To compute such a feasible solution, we reformulate the problem as follows:

$$Z(M) = \min\ Z^u(v) \tag{21}$$

$$\sum_{p \in P} v_{ij}^p = u_{ij} \quad \forall\ (i,j) \in \overline{A} \tag{22}$$

$$v_{ij}^p \leq b_{ij}^p \quad \forall\ (i,j) \in \overline{A},\ p \in P \tag{23}$$

$$v_{ij}^p \geq 0 \quad \forall\ (i,j) \in \overline{A},\ p \in P \tag{24}$$

where $Z^u(v)$ is the optimal value of the following *resource-decomposition subproblem*, which decomposes into $|P|$ single-commodity minimum cost network flow problems:

$$Z^u(v) = \min \sum_{p \in P} \sum_{(i,j) \in \overline{A}} \left(c_{ij}^p + f_{ij}/u_{ij} \right) x_{ij}^p \tag{25}$$

subject to constraints 15, 18 and

$$x_{ij}^p \leq v_{ij}^p \quad \forall\ (i,j) \in \overline{A},\ p \in P \tag{26}$$

Thus, given a point $\overline{v}$, the computational effort of finding a feasible solution involves projecting this point on the set defined by relations 22 to 24, and then solving $|P|$ minimum cost network flow problems. The projection can be accomplished by solving $\left|\overline{A}\right|$ singly-constrained quadratic programming problems (see Kennington and Helgason [8] for an efficient resolution procedure).

We determine $\overline{v}$ by finding an optimal solution to the following *price-decomposition subproblem*, obtained from Lagrangean relaxation with respect to the total capacity constraints 16:

$$Z^l(\gamma) = \min \sum_{p \in P} \sum_{(i,j) \in \overline{A}} \left(c_{ij}^p + f_{ij}/u_{ij} + \gamma_{ij} \right) x_{ij}^p - \sum_{(i,j) \in \overline{A}} \gamma_{ij} u_{ij} \tag{27}$$

subject to constraints 15, 17 and 18. The nonnegative Lagrangean vector γ is initially set to 0 and subsequently adjusted by moving $\theta(\gamma) = 2 \times (Z^u(M) - Z^l(\gamma))/||s(\gamma)||^2$ along the direction of a subgradient $s(\gamma)$ of $Z^l(\gamma)$. Given an optimal solution $\overline{v}$ to the price-decomposition subproblem, such a subgradient is given by:

$$s_{ij}(\gamma) = \sum_{p \in P} \overline{v}_{ij}^p - u_{ij} \quad \forall\ (i,j) \in \overline{A} \tag{28}$$

Initially, when no upper bound on $Z(M)$ is known, we set $Z^u(M) = 2 \times Z^l(0)$. Once a first upper bound is found, we set $Z^u(M)$ to this value.

The upper bounding procedure stops when a maximum number of iterations has been performed, or when one of the bound, either $Z^l(\gamma)$ or $Z^u(v)$, is the same as the corresponding bound computed at the previous iteration. Given a subset $\overline{A}$ of A and a current best known upper bound $\widetilde{Z}^u$ on the optimal value of the network design problem, the upper bounding procedure may be stated as follows:

1. Initialize iteration counter: $i \leftarrow 1$; Lagrangean vector: $\gamma \leftarrow 0$; and previous bounds: $Z_{i-1}^l \leftarrow 0$ and $Z_{i-1}^u \leftarrow +\infty$.

2. Solve price-decomposition subproblem ($|P|$ minimum cost network flow problems); if there is no feasible solution, stop; otherwise, let $\overline{v}$ be an optimal solution;
   $Z^l(\gamma) \leftarrow \sum_{p \in P} \sum_{(i,j) \in \overline{A}} (c_{ij}^p + f_{ij}/u_{ij} + \gamma_{ij}) \overline{v}_{ij}^p - \sum_{(i,j) \in \overline{A}} \gamma_{ij} u_{ij}$;
   $s_{ij}(\gamma) \leftarrow \sum_{p \in P} \overline{v}_{ij}^p - u_{ij},\ \forall (i,j) \in \overline{A}$.

3. If $Z^l(\gamma) = Z_{i-1}^l$, stop; otherwise, $Z_{i-1}^l \leftarrow Z^l(\gamma)$.

4. Project $\overline{v}$ on the set defined by relations 22 to 24 ($|\overline{A}|$ singly-constrained quadratic problems); let v be the solution.

5. Solve resource-decomposition subproblem ($|P|$ minimum cost network flow problems); if there is no feasible solution, go to 8; otherwise, let $\overline{x}$ be an optimal solution; $Z^u(v) \leftarrow \sum_{p \in P} \sum_{(i,j) \in \overline{A}} (c_{ij}^p + f_{ij}/u_{ij}) \overline{x}_{ij}^p$.

6. If $Z^u(v) = Z_{i-1}^u$, stop; otherwise, $Z_{i-1}^u \leftarrow Z^u(v)$.

7. Compute an upper bound Z^u on the network design problem by using relations 19 and 20. Update the best known upper bound: $\tilde{Z}^u \leftarrow \min\left(\tilde{Z}^u, Z^u\right)$.

8. If $i \geq i_{\max}$, stop; otherwise, increment i and update γ by moving along the direction of $s(\gamma)$: $\gamma \leftarrow \gamma + \theta(\gamma) s(\gamma)$; go to 2.

2.3 Composite Procedure

The composite procedure combines the two bounding procedures and implements a *dichotomic rule* to define the sets $\overline{A}$ used to compute upper bounds. This rule makes use of the *reduced fixed costs*, defined for each arc (i,j) as: $r_{ij} = f_{ij} - \alpha_{ij} u_{ij} - \sum_{p \in P} \beta_{ij}^p b_{ij}^p$. The rule proceeds as follows: the arcs are first sorted in increasing order of their reduced fixed costs, and the first half define $\overline{A}$. Following the application of the upper bounding procedure, if an improvement is found, the second half of the arcs in $\overline{A}$ (those with the largest reduced fixed costs) are dropped. Otherwise, if no improvement is obtained (this is the case, in particular, if no feasible solution is found), the first half of the arcs not in $\overline{A}$ (those with the smallest reduced fixed costs) are added to it. The process continues until it is no longer possible to add or remove an arc without obtaining a set already considered.

The composite procedure may be stated as follows:

1. Initialize iteration counter: $k \leftarrow 1$; variables of the upper bounding procedure: $\tilde{Z}^u \leftarrow +\infty$; and variables of the lower bounding procedure: $\alpha \leftarrow 0$, $\beta \leftarrow 0$, $\tilde{Z}^l \leftarrow 0$, $l^* \leftarrow 0$ and $\lambda \leftarrow 1$.

2. Initialize the variables of the dichotomic rule: $p^l \leftarrow 0$, $p^u \leftarrow 1$; sort the arcs in increasing order of their reduced fixed costs.

3. $p^m \leftarrow (p^u - p^l)/2$; if $(p^m \times |A|) < 1$, go to 5; otherwise, define $\overline{A}$ as the first $(p^l + p^m)$ arcs.

4. Apply the upper bounding procedure; if no improvement is found, $p^l \leftarrow p^l + p^m$, otherwise $p^u \leftarrow p^l + p^m$; go to 3.

5. If $k \geq k_{\max}$, stop; otherwise, increment k.

6. Apply the lower bounding procedure; go to 2.

3 Parallel Implementations

We propose a parallel synchronous algorithm that exploits the inherent decomposition properties of the bounding procedures. The algorithm is designed for coarse-grained MIMD message-passing architectures. To obtain efficient implementations on such parallel computing platforms, a key objective is to minimize communication time. This can be achieved by minimizing both the number and the size of the messages sent across the network of processors. With this objective in mind, we propose a *static load balancing* method to allocate tasks to processors. Assuming we have S working processes distributed across the multiprocessor system, and N tasks to perform in parallel, the static load balancing strategy will allocate $O(N/S)$ tasks to each process. This load balancing method is performed only once and does not require any communication between processes. When N is very large compared to S, and the number of tasks assigned to each process differs by at most one, this method divides computing time by a factor proportional to S, assuming that each task takes about the same amount of time and exactly one process is mapped onto every processor.

We exploit two types of decomposition: by commodities and by arcs (this one only at the projection step of the upper bounding procedure). The load balancing method consists for each process t to allocate itself a subset $P(t)$ of the commodities and a subset $A(t)$ of the arcs. Subsequently, for the portions of the algorithm that are performed in parallel and decomposed either by commodities or by arcs, each process t performs operations only on $P(t)$ or on $A(t)$.

3.1 Parallel Lower Bounding Procedure

The lower bounding procedure can be entirely decomposed by commodities, in the sense that every process t works only on $P(t)$. Two communication steps are required: one after the x-problem is solved, and the other before updating the multipliers. The first one computes the lower bound $Z(\alpha, \beta)$, the subgradient $s(\alpha)$, and also, for each arc (i, j), the sum $\sum_{p \in P} \beta_{ij}^p b_{ij}^p$, used in solving the y-problem. The second communication step is required to compute the norm $||s(\alpha), s(\beta)||^2$, used in the stepsize $\theta(\alpha, \beta)$. It can be easily seen that each of these two communication steps consists of a *single node accumulation* and a *single node broadcast* [2]: one process first receives from all other processes their partial information, and then sends to all other processes the result of combining the partial informations.

The parallel lower bounding procedure may be stated as follows:

1. Initialize iteration counter: $l \leftarrow 1$.

2. Solve the x-problem in parallel: each process t solves $|P(t)|$ minimum cost network flow problems to obtain $\overline{x}(t)$;
   $Z(\alpha, \beta; t) \leftarrow \sum_{p \in P(t)} \sum_{(i,j) \in A} (c_{ij}^p + \alpha_{ij} + \beta_{ij}^p(t)) \overline{x}_{ij}^p(t)$;
   $s_{ij}(\alpha; t) \leftarrow \sum_{p \in P(t)} \overline{x}_{ij}^p(t), \ \forall (i, j) \in A$;
   $s_{ij}^p(\beta; t) \leftarrow \overline{x}_{ij}^p(t), \ \forall (i, j) \in A, p \in P(t)$;
   $B_{ij}(t) \leftarrow \sum_{p \in P(t)} \beta_{ij}^p(t) b_{ij}^p, \ \forall (i, j) \in A$.

3. Communication (single node accumulation and single node broadcast):
   $Z(\alpha, \beta) \leftarrow \sum_t Z(\alpha, \beta; t)$;
   $s_{ij}(\alpha) \leftarrow \sum_t s_{ij}(\alpha; t), \ \forall (i, j) \in A$;
   $B_{ij} \leftarrow \sum_t B_{ij}(t), \ \forall (i, j) \in A$.

4. Each process t solves the y-problem to obtain $\overline{y}$;
   $Z(\alpha, \beta) \leftarrow Z(\alpha, \beta) + \sum_{(i,j) \in A} \min(0, f_{ij} - \alpha_{ij} u_{ij} - B_{ij})$;
   $s_{ij}(\alpha) \leftarrow s_{ij}(\alpha) - u_{ij} \overline{y}_{ij}, \ \forall (i, j) \in A$;
   $s_{ij}^p(\beta; t) \leftarrow s_{ij}^p(\beta; t) - b_{ij}^p \overline{y}_{ij}, \ \forall (i, j) \in A, p \in P(t)$.

5. If $Z(\alpha,\beta) > \widetilde{Z}^l$, $\widetilde{Z}^l \leftarrow Z(\alpha,\beta)$ and $l^* \leftarrow 0$; go to 7.

6. Increment l^*; if $l^* \geq l^*_{\max}$, update the parameter λ and reinitialize l^*: $\lambda \leftarrow \max\left(0.00001, \frac{\lambda}{2}\right)$ and $l^* \leftarrow 0$.

7. Each process t computes the partial norm $D(t) \leftarrow \sum_{p\in P(t)} \sum_{(i,j)\in A} (s^p_{ij}(\beta;t))^2$.

8. Communication (single node accumulation and single node broadcast):
   $D \leftarrow \sum_t D(t)$.

9. Compute the norm $D \leftarrow D + \sum_{(i,j)\in A} (s_{ij}(\alpha))^2$.

10. Each process t updates $(\alpha, \beta(t))$ by moving along the direction of $(s(\alpha), s(\beta;t))$:
    $\alpha \leftarrow \alpha + (\lambda(Z^u - Z(\alpha,\beta))/D)s(\alpha)$;
    $\beta(t) \leftarrow \beta(t) + (\lambda(Z^u - Z(\alpha,\beta))/D)s(\beta;t)$.

11. If $l \geq l_{\max}$, stop; otherwise, increment l and go to 2.

Given S working processes, if we denote by $b(S)$ the unit time for a single node accumulation and a single node broadcast, the overall communication complexity is $O(|A| \times b(S))$. Clearly, $b(S)$ depends on the topology of the architecture, the mapping of processes onto processors and the communication protocol used. For a distributed network of workstations and for a ring topology, $b(S)$ increases, at best, linearly with S, while for tree and hypercube topologies, it increases, at best, logarithmically with S.

3.2 Parallel Upper Bounding Procedure

The upper bounding procedure can be decomposed by commodities, except for the projection step, which can be decomposed by arcs. As a result, the communication steps are significantly more complex than in the lower bounding procedure. In particular, to obtain its subset $\overline{v}(t)$ of elements of $\overline{v}$, required to perform the projection, each process t must obtain these elements from all other processes. This communication step results in a *total exchange* [2]: each process t sends a block of elements $\overline{v}(t,r)$ to every other process r. Thus, assuming there are S working processes, the size of each block sent

across the network, by using this communication protocol, is $O(|A| \times |P|/S^2)$. This can be reduced significantly by sending only non-zero elements, with pointers to the elements to recover their original position in $\overline{v}$. This trick is performed only when the number of zero elements is sufficiently large, say more than half of the total number of elements. An alternative to this communication protocol consists in performing a *single node gather* and a *single node scatter* [2]: one process first receives from all other processes their blocks of elements of $\overline{v}$, decomposed by commodities, and then sends to all other processes their blocks of elements of $\overline{v}$, decomposed by arcs. In this case, the size of each block is $O(|A| \times |P|/S)$, which can be reduced significantly by sending only the non-zero elements.

Another complex communication issue is to stop the computation rapidly when subproblems are determined to be infeasible. This situation can happen when solving the price-decomposition subproblem at the first iteration of the procedure, and the resource-decomposition subproblems at any iteration of the procedure. We then implement an *asynchronous protocol*: when solving a minimum cost network flow problem, if a process finds no feasible solution, it sends asynchronously a signal to all other processes. Before solving a minimum cost network flow problem, each process verifies if a signal was sent by another process. If a process solves all its assigned minimum cost network flow problems, it waits for a signal to determine either if another process has found no feasible solution to one of its assigned problem, or if all processes have solved their assigned problems. The asynchronous protocol, when implemented carefully, has the same communication complexity as a single node accumulation and broadcast.

In addition, three other synchronous communication steps are required, each consisting of a single accumulation and a single broadcast. One computes $Z^l(\gamma)$ after solving the price-decomposition subproblem, the other computes $\overline{y}$, $Z^u(v)$ and Z^u after solving the resource-decomposition subproblem, while the third one computes the subgradient $s(\gamma)$ before updating γ.

The parallel upper bounding procedure may thus be stated as follows:

1. Initialize iteration counter: $i \leftarrow 1$; Lagrangean vector: $\gamma \leftarrow 0$; and previous bounds:

$Z^l_{i-1} \leftarrow 0$ and $Z^u_{i-1} \leftarrow +\infty$.

2. Solve price-decomposition subproblem in parallel: each process t solves $|P(t)|$ minimum cost network flow problems; asynchronous protocol (only if $i = 1$): if there is no feasible solution, stop; otherwise, let $\overline{v}$ be an optimal solution;
   $Z^l(\gamma; t) \leftarrow \sum_{p \in P(t)} \sum_{(i,j) \in \overline{A}} (c^p_{ij} + f_{ij}/u_{ij} + \gamma_{ij}) \overline{v}^p_{ij}(t)$;
   $s_{ij}(\gamma; t) \leftarrow \sum_{p \in P(t)} \overline{v}^p_{ij}(t), \ \forall (i,j) \in \overline{A}$.

3. Communication (single node accumulation and single node broadcast):
   $Z^l(\gamma) \leftarrow \sum_t Z^l(\gamma; t) - \sum_{(i,j) \in \overline{A}} \gamma_{ij} u_{ij}$.

4. If $Z^l(\gamma) = Z^l_{i-1}$, stop; otherwise, $Z^l_{i-1} \leftarrow Z^l(\gamma)$.

5. Communication (total exchange, or single node gather and single node scatter): each process t sends its block $\overline{v}(t)$ decomposed by commodities, and receives its block $\overline{v}(t)$ decomposed by arcs.

6. Parallel projection of $\overline{v}$: each process t solves $|\overline{A}(t)|$ singly-constrained quadratic problems; let $v(t)$ be the solution.

7. Communication (total exchange, or single node gather and single node scatter): each process t sends its block $v(t)$ decomposed by arcs, and receives its block $v(t)$ decomposed by commodities.

8. Solve resource-decomposition subproblem in parallel: each process t solves $|P(t)|$ minimum cost network flow problems; asynchronous protocol: if there is no feasible solution, go to 12; otherwise, let $\overline{x}(t)$ be an optimal solution and $\overline{y}(t)$ be derived from relation 20 restricted to $P(t)$;
   $Z^u(v; t) \leftarrow \sum_{p \in P(t)} \sum_{(i,j) \in \overline{A}} (c^p_{ij} + f_{ij}/u_{ij}) \overline{x}^p_{ij}(t)$;
   $Z^u(t) \leftarrow \sum_{p \in P(t)} \sum_{(i,j) \in \overline{A}} c^p_{ij} \overline{x}^p_{ij}(t)$.

9. Communication (single node accumulation and single node broadcast):
   $\overline{y}_{ij} \leftarrow \bigcup_t \overline{y}_{ij}(t), \ \forall (i,j) \in A$;

$Z^u(v) \leftarrow \sum_t Z^u(v;t)$;
$Z^u \leftarrow \sum_t Z^u(t) + \sum_{(i,j)\in\overline{A}} f_{ij}\overline{y}_{ij}$.

10. If $Z^u(v) = Z^u_{i-1}$, stop; otherwise, $Z^u_{i-1} \leftarrow Z^u(v)$.

11. Update the best known upper bound: $\tilde{Z}^u \leftarrow \min\left(\tilde{Z}^u, Z^u\right)$.

12. If $i \geq i_{\max}$, stop; otherwise, increment i.

13. Communication (single node accumulation and single node broadcast):
$s_{ij}(\gamma) \leftarrow \sum_t s_{ij}(\gamma;t) - u_{ij},\ \forall (i,j) \in \overline{A}$.

14. Update γ by moving along the direction of $s(\gamma)$: $\gamma \leftarrow \gamma + \theta(\gamma)s(\gamma)$; go to 2.

If a total exchange protocol is implemented at steps 5 and 7, the overall communication complexity is $O(\max(|A| \times b(S), |A| \times |P| \times e(S)/S^2))$, where $e(S)$ is the unit time for a total exchange on S working processes. If a single node gather and a single node scatter are used at steps 5 and 7, the communication complexity is $O(\max(|A| \times b(S), |A| \times |P| \times g(S)/S))$, where $g(S)$ is the unit time for a single node gather and a single node scatter using S working processes. Again, $e(S)$ and $g(S)$ depend on the topology of the architecture. For a distributed network of workstations and for ring and tree topologies, $e(S)$ is $O(S^2)$ and $g(S)$ is $O(S)$, while for a hypercube topology, $e(S)$ is $O(S \log S)$ and $g(S)$ is $O(S)$.

4 Experimental Results

We report the results obtained with two *master-slave* implementations of the parallel bounding procedures, where a master process takes care of all communications among a set of slave processes. In the first implementation, called *master-not-working*, the slaves are the only processes to take part in the decomposition and the master is used only for communication purposes. In the *master-working* implementation, the master also takes part in the decomposition. In both implementations, a star virtual topology is implied by the communication scheme. Consequently, even if the actual topology is a tree or a

hypercube, the single node broadcast, for example, would take a linear time. Note also that, by using this approach, we implement steps 5 and 7 of the parallel upper bounding procedure as a single node gather and a single node scatter.

To perform our experiments, we use three different environments. The first environment, called *Meiko environment*, consists of a network of 16 T800 Transputers connected with a SUN SPARC operating as host. The Transputers, each having four communication ports and a memory of 4 MB, are linked by a 3-ary tree topology, with the master process residing at the root of this tree. The peak communication speed is 20 Mbits/s. Within this environment, the code is programmed in FORTRAN/77, using the Meiko Communicating Sequential (CS) Tools library of functions, and compiled with *mf77*, the Meiko FORTRAN/77 compiler. The second environment, called *SUNSPARC5 environment*, is a network of eight SPARC5 workstations, each having a memory of 32 MB and connected via an Ethernet cable providing a peak communication speed of 10 Mbits/s. The third environment, called *SUNServer1000 environment*, consists of a Server1000 equipped of eight processors communicating through a shared-memory of 256 MB. Within these two environments, the code is programmed in FORTRAN/77, using the Parallel Virtual Machine (PVM) library of functions, and compiled with *f77*.

To solve minimum cost network flow problems, we use a modified version of the primal simplex code RNET (version 3.61), due to Grigoriadis and Hsu [6]. The modifications are required in order to handle real costs and capacities, and to allow fast reoptimizations by storing the solutions to all $|P|$ single-commodity flow problems. The parameters of the procedures are given the following values: $k_{\max} = 2$, $l_{\max} = 500$, $l^*_{\max} = 20$, and $i_{\max} = 5$.

The experiments are conducted on a wide variety of problems. We report the results obtained with five of these problems, which are representative of a larger set of 17 test problems. Table 1 summarizes the characteristics and dimensions of these five problems. They are divided into two categories: bipartite problems, where the underlying network structure is a bipartite graph, and single OD problems, where the network has no particular structure, but a commodity is defined as an origin-destination pair.

We compare the parallel implementations to the sequential code by using the *speedup*

Problem	$\|N\|$	$\|A\|$	$\|P\|$	$\|O(p)\|$	$\|D(p)\|$	structure
P1	50	625	100	25	25	bipartite
P2	100	2500	100	50	50	bipartite
P3	30	700	100	1	1	single OD
P4	30	700	400	1	1	single OD
P5	50	1000	1000	1	1	single OD

Table 1: Characteristics of Test Problems

measure, defined for a particular problem and a given number of processors S, as the ratio of the time to solve the problem with the sequential code to the time to solve it with the parallel code using S processors. To assess performances adequately, we use the *real (wall clock) time* and we synchronize processors between two time measures. Apart from the total time, including input/output, we also measure separately the times to compute lower and upper bounds.

We first illustrate the respective behavior of the two implementations by solving problem P1 within the Meiko environment. Figures 1 and 2 show the speedup curves obtained by, respectively, the master-not-working and master-working implementations, where one process is assigned to each processor. For this problem, the total sequential time is 12621 seconds, with 3499 and 9045 seconds dedicated to the upper and lower bounding procedures, respectively. The figures clearly show that, when the number of processors reaches a certain level, the master-working implementation becomes inefficient, particularly with respect to the upper bounding procedure. On the contrary, the master-not-working implementation shows constant improvement and achieves a higher maximum speedup. However, for few processors, say less than 10, the master-working implementation is preferable.

We next illustrate the effect of various dimensions with experiments on the SUN-Server1000 environment. We only show the results obtained with the master-working implementation, since it outperforms the master-not-working implementation within this environment. Figures 3 and 4 show the speedup curves for problems P1 and P2, respectively. For problem P1, the total sequential time is 835 seconds, with 234 and 594 seconds

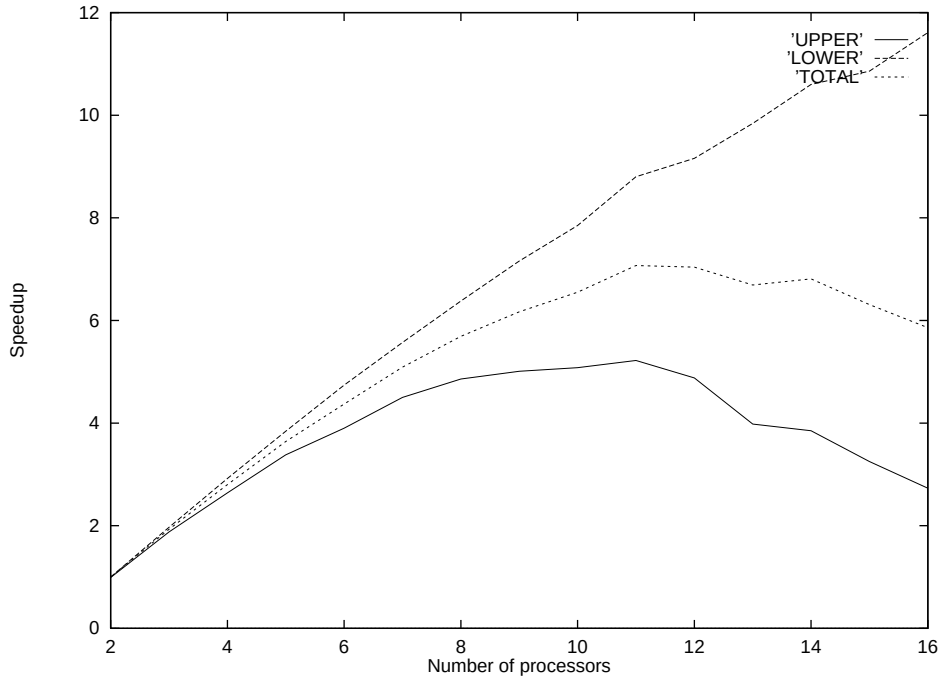

Figure 1: Speedup curve for problem P1 within the Meiko environment (master-not-working implementation)

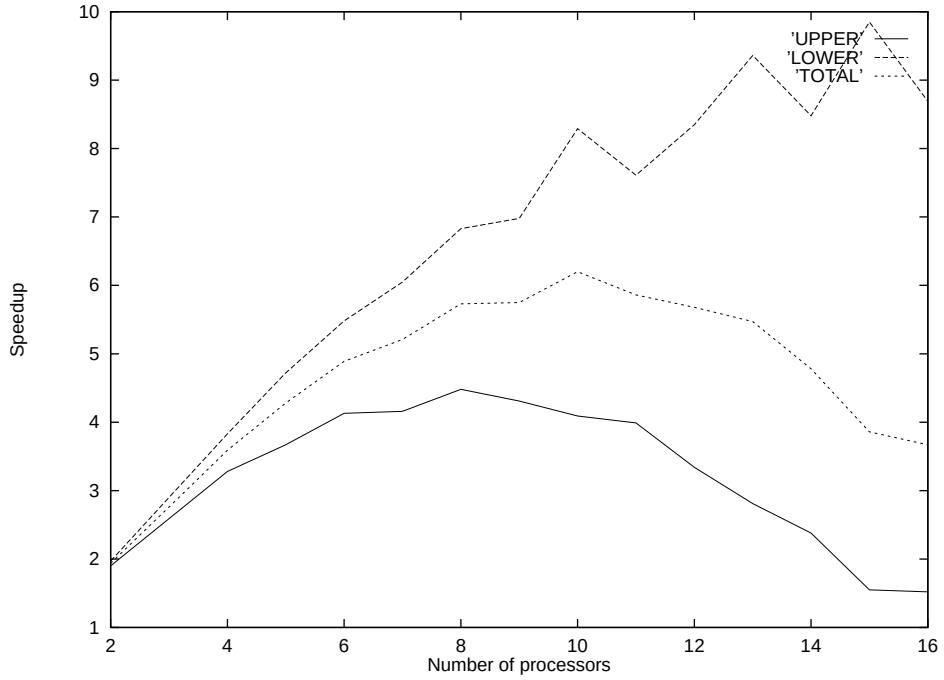

Figure 2: Speedup curve for problem P1 within the Meiko environment (master-working implementation)

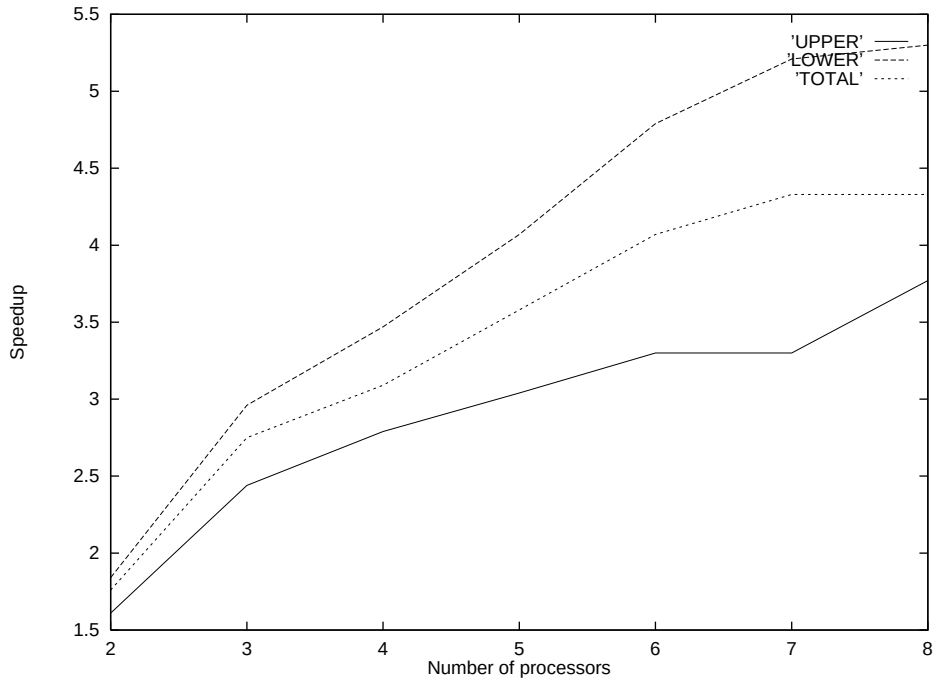

Figure 3: Speedup curve for problem P1 within the SUNServer1000 environment (master-working implementation)

dedicated to the upper and lower bounding procedures, respectively, while for problem P2, the total sequential time is 3897 seconds, divided into 1250 and 2620 seconds for the upper and lower bounding procedures, respectively. Results indicate only a slight improvement in performance, from P1 to P2, as the number of arcs is multiplied by four. This is in contrast with the major improvement obtained when the number of commodities is increased. We illustrate this fact with the results of problems P3 and P4, given in Figures 5 and 6, respectively. For problem P3, the total sequential time is 346 seconds, with 93 and 246 seconds dedicated to the upper and lower bounding procedures, respectively, while for problem P4, the total sequential time is 1824 seconds, divided into 589 and 1204 seconds for the upper and lower bounding procedures, respectively.

To assess the relative merits of the three environments, we use problem P1 as a basis of comparison. Figure 7 shows the speedup curve obtained with the SUNSPARC5 environment, by using the master-working implementation, where the total sequential time is 894 seconds, with 242 and 641 seconds dedicated to the upper and lower bounds,

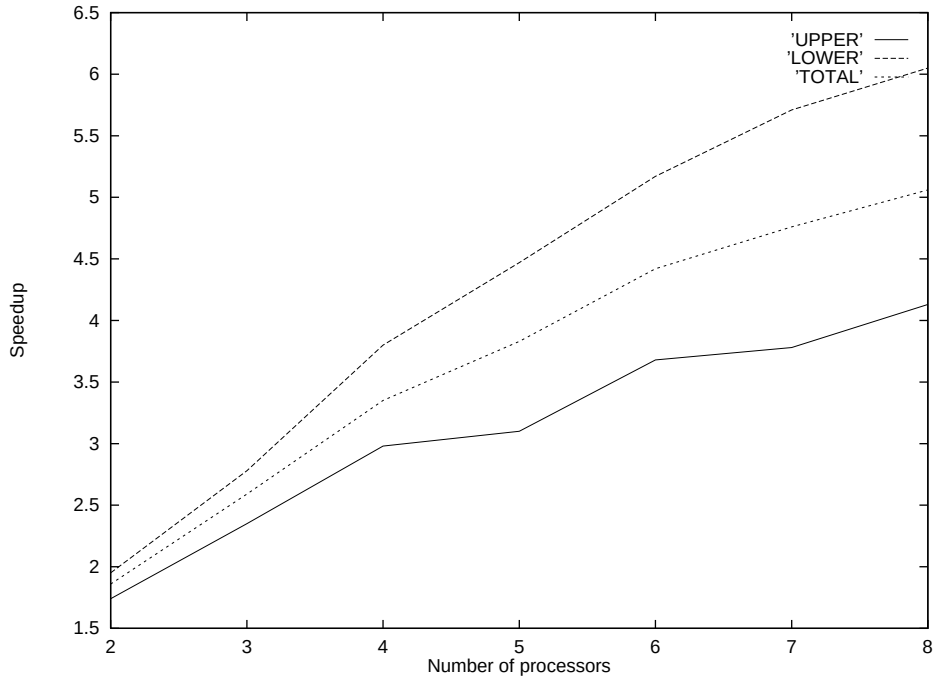

Figure 4: Speedup curve for problem P2 within the SUNServer1000 environment (master-working implementation)

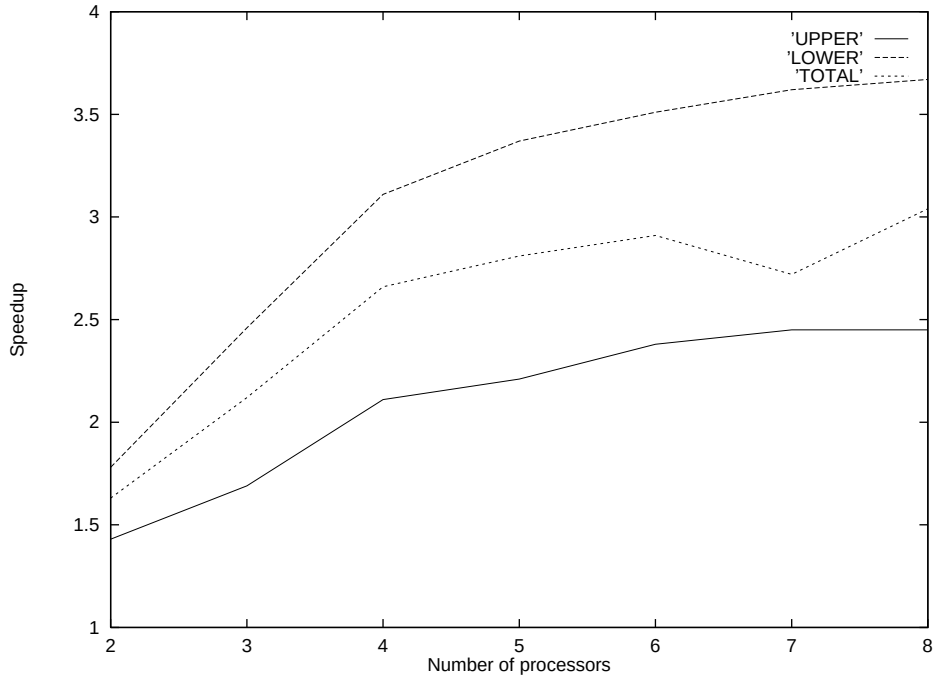

Figure 5: Speedup curve for problem P3 within the SUNServer1000 environment (master-working implementation)

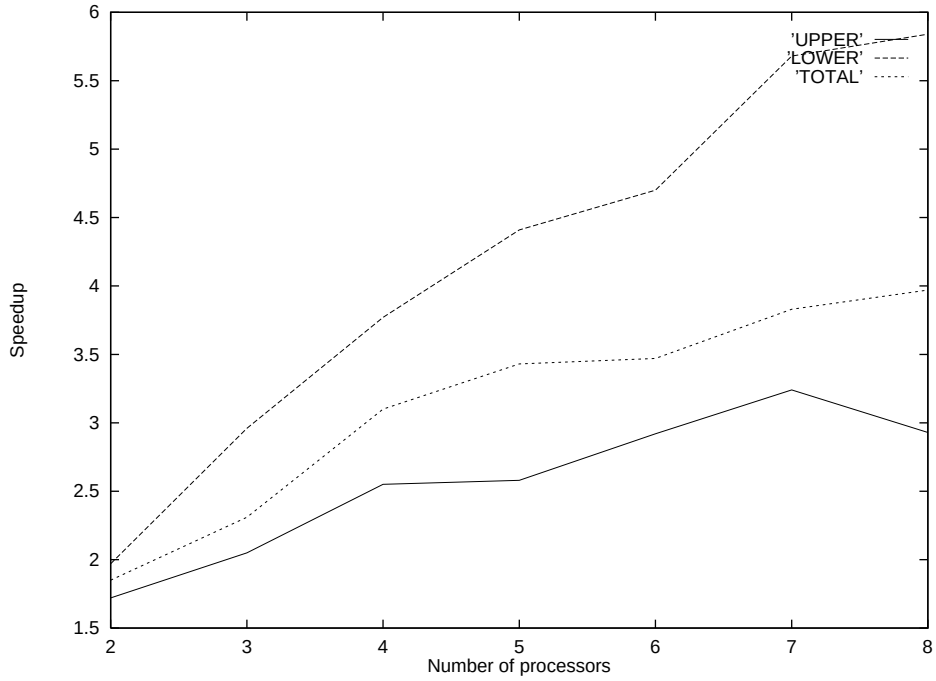

Figure 6: Speedup curve for problem P4 within the SUNServer1000 environment (master-working implementation)

respectively. In comparison to the results with the other environments (see Figures 2 and 3), the speedups are relatively modest, even though a single processor of the Server1000 and a SPARC5 have about the same processing capability. Thus, the slow communication speed of the Ethernet cable and the serial access to it are serious obstacles to efficient parallel implementations.

We now illustrate the difficulty of obtaining reliable measures on undedicated systems, such as the SUNSPARC5 and SUNServer1000 environments. This last architecture is particularly interesting, since in addition to being a multitask-multiuser system, it provides the user no control on the allocation of processes onto processors. Figure 8 shows two frequent anomalies that occur when using the SUNServer1000 environment. First, a superlinear speedup can be observed (for example, see the time of the lower bounding procedure on two processors), and second, a significant slowdown occurs when six processors (more precisely, six processes running on a maximum of six processors) are used. These anomalies can be attributed to the presence of background processes and

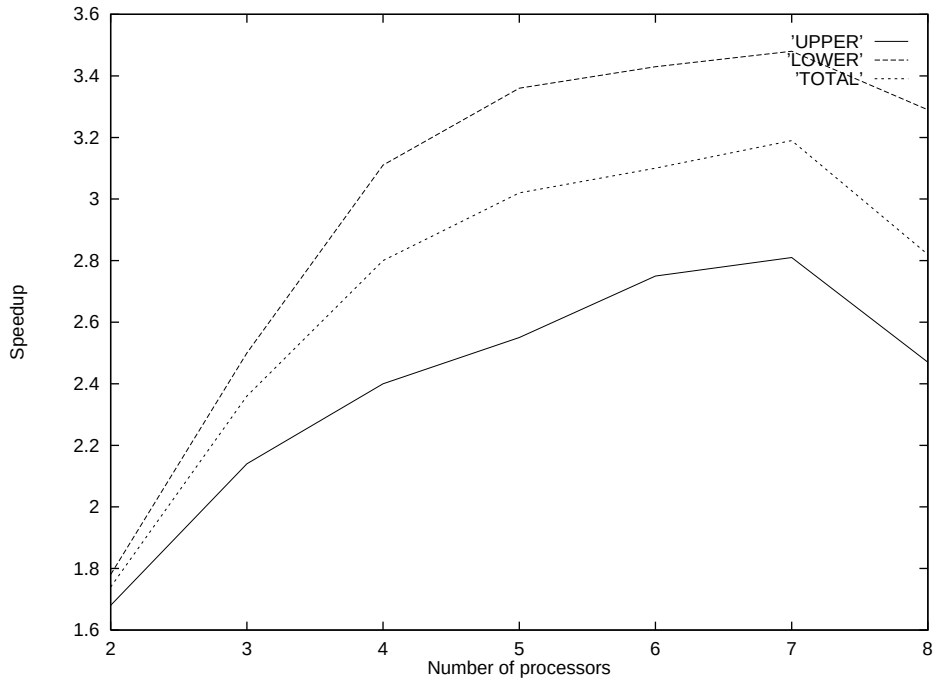

Figure 7: Speedup curve for problem P1 within the SUNSPARC5 environment (master-working implementation)

to memory management characteristics, such as paging effects. The results of this figure should be compared with those of Figure 4, where the anomalies disappeared after other runs were made.

To conclude this section, we report the results of solving problem P5, which has more than 1 million variables and 1000 commodities, on the SUNServer1000 environment, by using the master-working implementation. Figure 9 shows the speedup curve for this problem, where the total sequential time is 6631 seconds, divided into 2707 and 3897 seconds for the upper and lower bounding procedures, respectively. The results indicate that, with respect to the lower bounding procedure, near-linear speedups can be obtained for such realistically-sized problems, while the projection step and its communication requirements represent serious limitations on the performance of the upper bounding procedure.

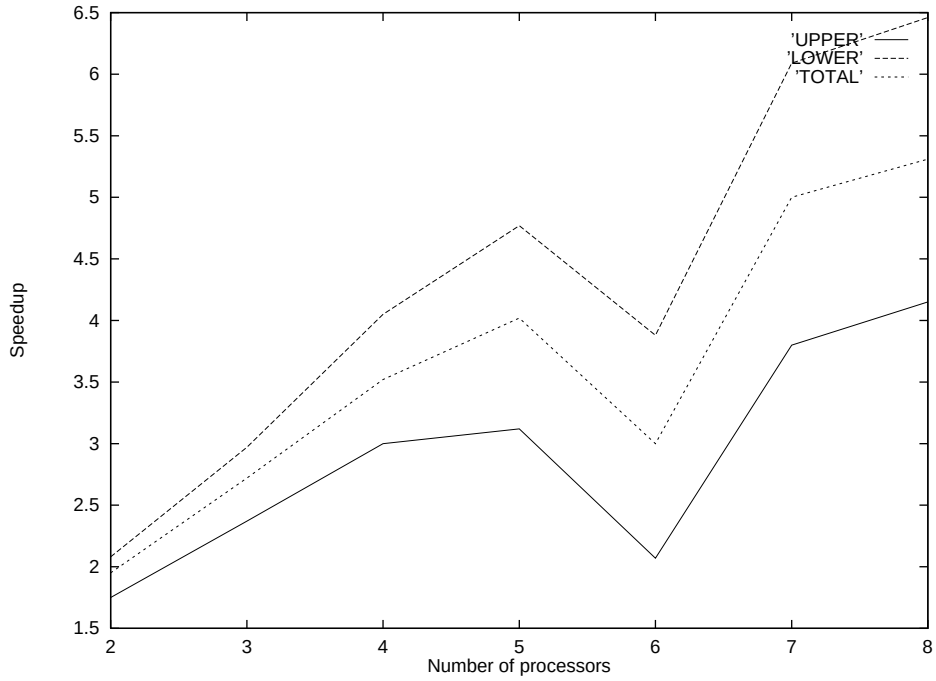

Figure 8: Anomalous speedup curve for problem P2 within the SUNServer1000 environment (master-working implementation)

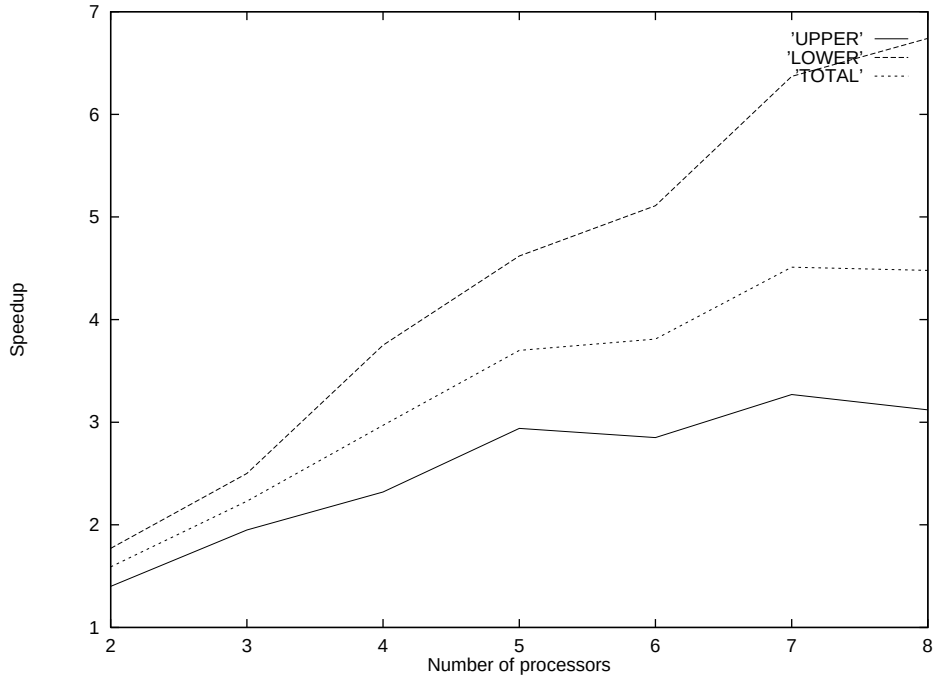

Figure 9: Speedup curve for problem P5 within the SUNServer1000 environment (master-working implementation)

5 Conclusion

We presented and analyzed parallel implementations of bounding procedures for multicommodity capacitated network design problems. Our experiments were conducted on three different environments. We assessed the relative merits of these architectures when used to solve our problems. More importantly, we showed that the parallel implementations can significantly alleviate the task of solving large-size realistic instances, with millions of variables and thousands of commodities.

In spite of these successes, we should mention that the instances that we considered are far from being solved exactly. In particular, their continuous relaxations provide poor approximations to their optimal values. To deal with this problem, many researchers [16, 18, 19, 20, 21, 17, 12] studied the polyhedral structure of several related fixed charge network design models. A natural extension to the present work would consist in relaxing, in a Lagrangean sense, valid inequalities added to the formulation (for examples of such approaches for solving difficult combinatorial problems, see Fischer [3] and Lucena [10]). Such relaxations would preserve the same network structure, and consequently lead to a similar parallel lower bounding procedure. Ultimately, the combination of polyhedral knowledge and parallel computation holds the promise of solving to near-optimality large-size realistic instances of these difficult problems.

Acknowledgments

This research has been supported by grants from the Fonds F.C.A.R. of the Province of Québec, and the Natural Sciences and Engineering Research Council of Canada. We also want to acknowledge the efforts of Mr. Benoit Bourbeau who helped us with the experimentations.

References

[1] Ahuja R.K., Magnanti T.L. and Orlin J.B. (1993), *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall.

[2] Bertsekas D.P. and Tsitsiklis J.N. (1989), *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall.

[3] Fischer M.L. (1990), *Optimal Solution of Vehicle Routing Problems Using Minimum k-Trees*, Technical Report, Department of Decision Sciences, The Wharton School, University of Pennsylvania.

[4] Gendron B. and Crainic T.G. (1994), *Relaxations for Multicommodity Network Design Problems*, Publication CRT-965, Centre de recherche sur les transports, Université de Montréal.

[5] Geoffrion A.M. (1974), *Lagrangean Relaxation for Integer Programming*, Mathematical Programming Study, 2, 82-114.

[6] Grigoriadis M.D. and Hsu T. (1979), *RNET 3.61 Documentation*, Rutgers University.

[7] Held M., Wolfe P. and Crowder H.P. (1974), *Validation of Subgradient Optimization*, Mathematical Programming, 6, 62-88.

[8] Kennington J.L. and Helgason R.V. (1980), *Algorithms for Network Programming*, Wiley.

[9] Lemaréchal C. (1989), *Nondifferentiable Optimization*, in *Handbooks in Operations Research and Management Science*, Volume 1, *Optimization*, G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd editors, 529-572, North-Holland.

[10] Lucena A. (1993), *Steiner Problem in Graphs: Lagrangean Relaxation and Cutting-Planes*, presented at NETFLOW93, San Miniato, Italy, October 3-7 (see Technical Report TR-21/93, Dipartimento di Informatica, Università degli Studi di Pisa, 147-154).

[11] Magnanti T.L. (1993), *Modeling and Solving Network Design Problems*, presented at NETFLOW93, San Miniato, Italy, October 3-7 (see Technical Report TR-21/93, Dipartimento di Informatica, Università degli Studi di Pisa, 155-159).

[12] Magnanti T.L., Mirchandani P. and Vachani R. (1993), *The Convex Hull of Two Core Capacitated Network Design Problems*, Mathematical Programming, 60, 233-250.

[13] Magnanti T.L. and Wong R.T. (1984), *Network Design and Transportation Planning: Models and Algorithms*, Transportation Science, 18(1), 1-55.

[14] Minoux M. (1989), *Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications*, Networks, 19, 313-360.

[15] Nemhauser G.L. and Wolsey L.A. (1988), *Integer and Combinatorial Optimization*, Wiley-Interscience.

[16] Padberg M.W., Van Roy T.J. and Wolsey (1985), *Valid Linear Ineqalities for Fixed Charge Problems*, Operations Research, 33(4), 842-861.

[17] Rardin R.L. and Wolsey L.A. (1991), *Valid Inequalities and Projecting the Multicommodity Extended Formulation for Uncapacitated Fixed Charge Network Flow Problems*, Working Paper, School of Industrial Engineering, Purdue University.

[18] Van Roy T.J. and Wolsey L.A. (1985), *Valid Inequalities and Separation for Uncapacitated Fixed Charge Networks*, Operations Research Letters, 4(3), 105-112.

[19] Van Roy T.J. and Wolsey L.A. (1986), *Valid Inequalities for Mixed 0-1 Programs*, Discrete Applied Mathematics, 4, 199-213.

[20] Van Roy T.J. and Wolsey L.A. (1987), *Solving Mixed Integer Programming Problems using Automatic Reformulation*, Operations Research, 35(1), 45-57.

[21] Wolsey L.A. (1989), *Submodularity and Valid Inequalities in Capacitated Fixed Charge Networks*, Operations Research Letters, 8, 119-124.