Pergamon

0966-8349(95)00005-4

Location Science, Vol. 3, No. 1, pp. 39–53, 1995
Elsevier Science Ltd
Printed in Great Britain.
0966-8349/95 $9.50 + 0.00

A BRANCH-AND-BOUND ALGORITHM FOR DEPOT LOCATION AND CONTAINER FLEET MANAGEMENT

BERNARD GENDRON and TEODOR GABRIEL CRAINIC

Centre de recherche sur les transports, Université de Montréal, C.P. 6128 Succursale centre-ville, Montréal, QC H3C 3J7, Canada

(*Received* 25 *August* 1992)

Abstract—The multicommodity location problem with balancing requirements is related to one of the major logistics issues faced by distribution and transportation firms: the management of a fleet of vehicles over a medium to long term planning horizon. To solve this problem, we present a branch-and-bound algorithm in which bounds are computed by a dual-ascent procedure. We particularly emphasize the design of efficient branching, fathoming and preprocessing rules. The algorithm was tested on a wide variety of randomly generated problems, and on a large-scale application to the planning of the land operations of a heterogenous container fleet. Results show that the algorithm is highly efficient, and outperforms other existing methods.

Keywords: Branch-and-bound, dual-ascent, multicommodity location with balancing requirements.

Résumé—Le problème de localisation multiproduit avec exigences d'équilibrage est relié à un des aspects importants de la planification logistique des entreprises de transport et de distribution, soit la gestion à moyen et long termes d'une flotte de véhicules. Nous présentons un algorithme de séparation et évaluation progressive (branch-and-bound) qui calcule des bornes au moyen d'une méthode d'ascension duale et qui met en évidence des règles efficaces de branchement, de cessation de fouille et de fixation de variables. L'algorithme a éré testé sur une grande variété de problèmes générés aléatoirement, ainsi que sur une application de grande taille au problème de la planification du transport terrestre de conteneurs. Les résultats montrent que l'algorithme est extrêmement efficace et qu'il l'emporte sur les autres méthodes proposées dans la litérature.

Mots-clés: Branch-and-bound, méthode d'ascention duale, problème de localisation multiproduit avec exigencess d'équilibrage.

1. INTRODUCTION

The multicommodity location problem with balancing requirements was first introduced by Crainic, Dejax and Delorme (1989). The problem is motivated by the following industrial application, related to the management of a heterogeneous fleet of containers by an international maritime shipping company. Once a ship arrives at port, the company has to deliver loaded containers, which may come in several types and sizes, to designated inland destinations. Following their unloading by the importing customer, empty containers are moved to a depot. From there, they may be delivered to customers who request containers for subsequent shipping of their own products. Furthermore, containers often have to be repositioned to other depots. These interdepot movements are a consequence of the regional imbalances in empty container availabilities and needs throughout the network: some areas lack containers of certain types, while others have surpluses of them. This requires balancing movements of empty containers among depots, and thus differentiates this problem from classical location–allocation applications. The general problem is therefore to locate depots in order to collect the supply of empty containers available at

customers' sites and to satisfy the customer requests for empty containers, while minimizing the total operating costs: the costs of opening and operating the depots, and the costs generated by customer–depot and interdepot movements.

Crainic, Delorme and Dejax (1993) show that standard location methods are not efficient for this problem, but their branch-and-bound algorithm cannot prove the optimality of the solution in a reasonable amount of time, except for simple data instances. Consequently, heuristics were proposed to solve the problem. Crainic and Delorme (1993) present a dual-ascent procedure which generally produces tight bounds, but still can exhibit large gaps on particular instances. An adaptation of the tabu search metaheuristic (Crainic, Gendreau, Soriano and Toulouse, 1992) generally obtains better solutions, but requires rather large computing times.

In this paper, we present an exact branch-and-bound algorithm which combines an improved dual-ascent bounding procedure, with efficient branching, fathoming and preprocessing rules. In Section 2, we formulate the model and derive lower bounds that define a dual-ascent procedure similar to the one proposed by Crainic and Delorme (1993). Section 3 gives an overview of the branch-and-bound algorithm and discusses issues related to the design of branching, fathoming and preprocessing rules. Extensive experiments conducted both on a wide variety of randomly generated problems and on data from a large-scale application are used to identify the most efficient branch-and-bound design and to compare it to other algorithmic approaches. The computational results of these experiments are reported and analyzed in Section 4.

2. PROBLEM FORMULATION AND BOUNDS

To formulate the problem, we consider a directed network $G = (N, A)$, where N is the set of nodes and A is the set of arcs. Moving through the network, there are several *commodities* (*types of containers*), represented by set P. The set of nodes may be partitioned into three subsets: O, the set of *origin* nodes (*supply customers*); D, the set of *destination* nodes (*demand customers*); and T, the set of *transhipment* nodes (*depots*). For each depot $j \in T$, we define $O(j) = \{i \in O : (i, j) \in A\}$ and $D(j) = \{i \in D : (j, i) \in A\}$, the sets of customers adjacent to this depot, and we assume that there exists at least one origin or destination adjacent to each depot j ($O(j) \cup D(j) \neq \emptyset$). For each node $i \in N$, we define the sets of depots adjacent to this node in both directions: $T^+(i) = \{j \in T : (i, j) \in A\}$, and $T^-(i) = \{j \in T : (j, i) \in A\}$. Since it is assumed that there are no arcs between customers, the set of arcs may be partitioned into three subsets: customer-to-depot arcs, $A_{OT} = \{(i, j) \in A : i \in O, j \in T\}$; depot-to-customer arcs, $A_{TD} = \{(i, j) \in A : i \in T, j \in D\}$; and depot-to-depot arcs, $A_{TT} = \{(i, j) \in A : i \in T, j \in T\}$.

The problem consists of minimizing costs incurred by moving flows through the network in order to satisfy supplies at origins and demands at destinations. For each supply customer $i \in O$, the supply of commodity p is noted o_i^p, while for each demand customer $i \in D$, the demand for commodity p is noted d_i^p. All supplies and demands are assumed to be nonnegative and deterministic. A nonnegative cost c_{ij}^p is incurred for each unit of flow of commodity p moving on arc (i, j). In addition, for each depot $j \in T$, a nonnegative fixed cost f_j is incurred if the depot is opened.

Let x_{ij}^p represent the amount of flow of commodity p moving on arc (i, j), and y_j be the binary location variable that takes value 1 if depot j is opened, and value 0 otherwise. The problem is then formulated as:

$$Z = \min \sum_{j \in T} f_j y_j + \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} c_{ij}^p x_{ij}^p + \sum_{(j,i) \in A_{TD}} c_{ji}^p x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right\} \tag{1}$$

$$\sum_{j \in T^+(i)} x_{ij}^p = o_i^p \qquad \forall i \in O,\ p \in P \tag{2}$$

$$\sum_{j \in T^-(i)} x_{ji}^p = d_i^p \qquad \forall i \in D,\ p \in P \tag{3}$$

$$\sum_{i \in D(j)} x_{ji}^p + \sum_{k \in T^+(j)} x_{jk}^p - \sum_{i \in O(j)} x_{ij}^p - \sum_{k \in T^-(j)} x_{kj}^p = 0 \qquad \forall j \in T,\ p \in P \tag{4}$$

$$x_{ij}^p \leqslant o_i^p y_j \qquad \forall j \in T,\ i \in O(j),\ p \in P \tag{5}$$

$$x_{ji}^p \leqslant d_i^p y_j \qquad \forall j \in T,\ i \in D(j),\ p \in P \tag{6}$$

$$x_{ij}^p \geqslant 0 \qquad \forall (i,j) \in A,\ p \in P \tag{7}$$

$$y_j \in \{0, 1\} \qquad \forall j \in T. \tag{8}$$

Constraints (2) and (3) ensure that supply and demand requirements are met, relations (4) correspond to flow conservation constraints at depot sites, while equations (5) and (6) forbid customer-related movements through closed depots. Note that analogous constraints for the interdepot flows are redundant if interdepot costs satisfy the triangle inequality (Crainic, Dejax and Delorme, 1989), an assumption that we follow throughout this text.

Lower bounds on the optimal value of this problem may be derived by considering the *strong relaxation*, obtained by replacing the integrality constraints (8) by $y_j \geqslant 0$, $\forall j \in T$. The dual of the resulting linear program, noted $\mathscr{D}$, may be formulated as:

$$Z_{\mathscr{D}} = \max \sum_{p \in P} \left\{ \sum_{i \in O} o_i^p \mu_i^p + \sum_{i \in D} d_i^p v_i^p \right\} \tag{9}$$

$$\mu_i^p - \lambda_j^p - \gamma_{ij}^p \leqslant c_{ij}^p \qquad \forall (i,j) \in A_{OT},\ p \in P \tag{10}$$

$$v_i^p + \lambda_j^p - \gamma_{ji}^p \leqslant c_{ji}^p \qquad \forall (j,i) \in A_{TD},\ p \in P \tag{11}$$

$$\lambda_j^p - \lambda_k^p \leqslant c_{jk}^p \qquad \forall (j,k) \in A_{TT},\ p \in P \tag{12}$$

$$\sum_{p \in P} \left\{ \sum_{i \in O(j)} o_i^p \gamma_{ij}^p + \sum_{i \in D(j)} d_i^p \gamma_{ji}^p \right\} \leqslant f_j \qquad \forall j \in T \tag{13}$$

$$\gamma_{ij}^p \geqslant 0 \qquad \forall (i,j) \in A_{OT},\ p \in P \tag{14}$$

$$\gamma_{ji}^p \geqslant 0 \qquad \forall (j,i) \in A_{TD},\ p \in P. \tag{15}$$

Here, μ_i^p and v_i^p are the dual variables associated with, respectively, the supply and demand constraints (2) and (3), λ_j^p are linked to the balancing constraints (4), while γ_{ij}^p and γ_{ji}^p correspond to constraints (5) and (6), respectively.

Our approach to computing bounds is inspired by Crainic and Delorme (1993). They derive two subproblems of $\mathscr{D}$: one obtained by fixing nonnegative γ variables to values satisfying constraints (13), and the other by fixing λ variables to values satisfying relations (12). Here, we derive the two subproblems by relating them to Lagrangean relaxations of the original problem.

The first subproblem may be obtained by dualizing constraints (5) and (6), using nonnegative γ multipliers:

$$Z(\gamma) = \min \sum_{j \in T} \left\{ f_j - \sum_{p \in P} \left(\sum_{i \in O(j)} o_i^p \gamma_{ij}^p + \sum_{i \in D(j)} d_i^p \gamma_{ji}^p \right) \right\} y_j \tag{16}$$

$$\sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} (c_{ij}^p + \gamma_{ij}^p) x_{ij}^p + \sum_{(j,i) \in A_{TD}} (c_{ji}^p + \gamma_{ji}^p) x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right\}$$

subject to constraints (2)–(4), (7) and (8). This subproblem decomposes into two parts: one that depends only on the flow variables and the other only on the location variables. This last part of the subproblem may be solved easily, without considering the integrality constraints (8):

$$\sum_{j \in T} \min \left\{ 0, f_j - \sum_{p \in P} \left(\sum_{i \in O(j)} o_i^p \gamma_{ij}^p + \sum_{i \in D(j)} d_i^p \gamma_{ji}^p \right) \right\}. \tag{17}$$

Thus, for fixed values of the γ multipliers, the Lagrangean subproblem has the integrality property (Geoffrion, 1974), since its optimal value can be obtained even with the integrality conditions relaxed. Consequently, the best lower bound on the optimal value of the original problem one can hope for when using this Lagrangean relaxation can be obtained by restricting the γ multipliers to values that satisfy constraints (13). For such values, the Lagrangean subproblem is equivalent to the following subproblem, called *FLIP relaxation*:

$$Z(\gamma) = \min \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} (c_{ij}^p + \gamma_{ij}^p) x_{ij}^p + \sum_{(j,i) \in A_{TD}} (c_{ji}^p + \gamma_{ji}^p) x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right\} \tag{18}$$

subject to constraints (2)–(4) and (7).

This problem is a *multicommodity uncapacitated minimum cost network flow problem* (MCNF), and thus decomposes into $|P|$ single-commodity uncapacitated minimum cost network flow problems (for a recent survey of methods for solving this type of problem, see Ahuja, Magnanti and Orlin, 1993). From an optimal solution to this subproblem, an upper bound on the optimal value of the original problem may be easily computed by setting y_j to 1 whenever there is flow moving through depot j, and to 0 otherwise.

The second subproblem may be derived by relaxing the balancing constraints (4), and by introducing them into the objective with λ multipliers:

$$Z(\lambda) = \min \sum_{j \in T} f_j y_j + \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} (c_{ij}^p + \lambda_j^p) x_{ij}^p \right. \tag{19}$$

$$\left. + \sum_{(j,i) \in A_{TD}} (c_{ji}^p - \lambda_j^p) x_{ji}^p + \sum_{(j,k) \in A_{TT}} (c_{jk}^p - (\lambda_j^p - \lambda_k^p)) x_{jk}^p \right\}$$

subject to constraints (2), (3) and (5)–(8). Note that the x_{jk}^p variables are nonnegative and appear only in the relaxed balancing constraints (4). Thus, by restricting the multipliers to values satisfying constraints (12) of $\mathscr{D}$, we obtain the following subproblem, called *FLOP relaxation*:

$$Z(\lambda) = \min \sum_{j \in T} f_j y_j + \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} (c_{ij}^p + \lambda_j^p) x_{ij}^p + \sum_{(j,i) \in A_{TD}} (c_{ji}^p - \lambda_j^p) x_{ji}^p \right\} \tag{20}$$

subject to constraints (2), (3), and (5)–(8).

This problem is an *uncapacitated location problem* (ULP), also called simple plant location problem (Krarup and Pruzan, 1983), and uncapacitated facility location problem (Cornuejols, Nemhauser and Wolsey, 1990). One of the most efficient methods for solving the ULP is the DUALOC algorithm proposed by Erlenkotter (1978). The algorithm works on a condensed formulation of the dual of the linear relaxation, in which the γ variables do not have to be explicitly computed. From any dual solution, the algorithm derives a primal solution satisfying the integrality constraints. The couple of primal–dual solutions thus obtained may violate some of the complementary slackness conditions. Through a dual-ascent procedure, which starts with some initial dual solution, the algorithm incrementally adjusts the dual variables to reduce complementary slackness violations. The algorithm also incorporates an adjustment procedure that tries to increase the dual objective value through decreasing some dual variables while increasing others. The dual solution thus obtained is not necessarily optimal for the dual of the linear relaxation of the ULP, but it provides a lower bound of good quality with a rather limited computational effort. For complete details of the method, the reader is referred to Erlenkotter (1978) and Van Roy and Erlenkotter (1982). For our purposes, it is sufficient to recall that the initial dual solution must satisfy the following conditions:

$$\mu_i^p \geqslant \min_{j \in T^+(i)} \{c_{ij}^p + \lambda_j^p\} \quad \forall i \in O,\ p \in P(o_i^p > 0) \tag{21}$$

$$v_i^p \geqslant \min_{j \in T^-(i)} \{c_{ji}^p - \lambda_j^p\} \quad \forall i \in D,\ p \in P(d_i^p > 0). \tag{22}$$

An upper bound on the optimal value of the original problem may be easily derived from a primal solution $\bar{y}$ obtained by DUALOC, by solving a related MCNF:

$$Z(\bar{y}) = \min \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} c_{ij}^p x_{ij}^p + \sum_{(j,i) \in A_{TD}} c_{ji}^p x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right\} \tag{23}$$

subject to (2)–(4), (7), and

$$x_{ij}^p \leqslant o_i^p \bar{y}_j \quad \forall j \in T,\ i \in O(j),\ p \in P \tag{24}$$

$$x_{ji}^p \leqslant d_i^p \bar{y}_j \quad \forall j \in T,\ i \in D(j),\ p \in P. \tag{25}$$

The FLIP and FLOP relaxations may be solved iteratively, by using as input to one the multipliers generated by the other. One then obtains an increasing sequence of lower bounds. To justify this assertion, first note that any feasible dual solutions to problems FLIP and FLOP are also feasible to problem $\mathcal{D}$, since the multipliers are fixed at values that satisfy the dual constraints. Furthermore, recall that problem FLIP is solved to optimality, while a dual-ascent procedure (DUALOC) is used to compute a lower bound to problem FLOP. Hence, if an optimal dual solution to problem FLIP can be used directly as input to DUALOC, the dual objective is guaranteed to increase. Indeed, this is the case since any optimal dual solution (μ, v, λ) to problem FLIP satisfies conditions (21) and (22). To prove this, we may proceed by contradiction as follows: suppose, without loss of generality, that there exists $i \in O, p \in P\,(o_i^p > 0)$, such that $\mu_i^p < \min_{j \in T^+(i)}\{c_{ij}^p + \lambda_j^p\} = c_{ij^*}^p + \lambda_{j^*}^p$. Since the γ multipliers are nonnegative, $\mu_i^p < c_{ij^*}^p + \lambda_{j^*}^p + \gamma_{ij^*}^p$. Then, there exists $\xi_i^p > 0$ such that $\mu_i^p + \xi_i^p \leqslant c_{ij^*}^p + \lambda_{j^*}^p + \gamma_{ij^*}^p$. Consequently, by setting $\bar{\mu}_i^p = \mu_i^p + \xi_i^p$, we obtain a feasible solution to the dual of problem FLIP for which the objective increases strictly by a quantity

$o_i^p \xi_i^p > 0$. But this contradicts the fact that (μ, ν, λ) is an optimal solution to problem FLIP.

Note that, after solving the FLOP relaxation, the γ multipliers can be increased by removing some proportion of the slack that may appear in the dual constraints (13). The resulting *gamma adjustment rule* (Crainic and Delorme, 1993) consists in adding the term $\theta s_j/\Delta_j$ to each γ multiplier related to depot j, where $0 \leqslant \theta \leqslant 1$, and s_j is the *slack variable*: $s_j = f_j - \Sigma_{p \in P}(\Sigma_{i \in O(j)} o_i^p \gamma_{ij}^p + \Sigma_{i \in D(j)} d_i^p \gamma_{ji}^p)$, and $\Delta_j = \Sigma_{p \in P}(\Sigma_{i \in O(j)} o_i^p + \Sigma_{i \in D(j)} d_i^p)$.

3. BRANCH-AND-BOUND ALGORITHM

The general idea of a branch-and-bound (BB) algorithm is to try to solve the original problem, and, in case of an unsuccessful attempt, to decompose it into easier subproblems, by using a *branching rule.* These subproblems are further divided, unless their optimal solution is found or it is determined that they cannot lead to an optimal solution to the original problem (*fathoming rules*). For each generated subproblem, *preprocessing rules* may also be applied in order to delete redundant constraints, or to fix some variables.

We adapt these ideas to design an exact algorithm to solve the multicommodity location problem with balancing requirements. For each generated subproblem, we apply a procedure to obtain lower and upper bounds based on the developments presented in Section 2. These bounds are used to define efficient fathoming and preprocessing rules. To represent location variables that are fixed through branching and preprocessing rules, we define the sets $T_{01} = \{j \in T: y_j \in \{0, 1\}\}$, $T_0 = \{j \in T: y_j = 0\}$, and $T_1 = \{j \in T: y_j = 1\}$ of *free, closed,* and *open* depots, respectively. To generate subproblems from a given subproblem S, we use a dichotomic branching rule: a depot $j^* \in T_{01}$ is chosen according to some criterion, and $S_0^{j^*}$ is obtained by transferring j^* to T_0, while $S_1^{j^*}$ results from transferring j^* to T_1. According to the terminology of trees, $S_0^{j^*}$ and $S_1^{j^*}$ are the *0-son node* and the *1-son node*, respectively, of the *father node* S, and the original problem, where all depots are free, is the *root node.* To decide which generated subproblem should be examined in priority, we use the *depth-first rule*: choose one of the subproblems that has been generated most recently. This rule minimizes computer storage requirements (Ibaraki, 1987), although it may generate a large number of subproblems. However, when a good heuristic is used to compute efficient upper bounds, as is the case here, this disadvantage may be reduced.

Formally then, the *BB algorithm* keeps a stack Λ of generated subproblems, as well as the value Z^u of the best solution identified thus far, and proceeds as follows:

(1) (*initialization*) S is the original problem: $T_{01} \leftarrow T$, $T_0 \leftarrow \varnothing$, $T_1 \leftarrow \varnothing$. $\Lambda \leftarrow \varnothing$. $Z^u \leftarrow +\infty$.
(2) (*preprocessing rule*) Attempt to fix some variables (T_{01}, T_0 and T_1 may be modified).
(3) (*bounding procedure*) Perform the bounding procedure on S (Z^u may be updated); if S may be fathomed, goto 5.
(4) (*branching rule*) Choose $j^* \in T_{01}$ and generate $S_0^{j^*}$ and $S_1^{j^*}$; select one of them to examine next, as subproblem S, and add the other to Λ. Goto 2.
(5) (*stopping test*) If $\Lambda = \varnothing$, STOP; Z^u is the optimal value of the original problem.
(6) (*backtracking*) Select the subproblem S on top of Λ. If it may be fathomed goto 5, otherwise, goto 2.

The performance of the algorithm is mainly influenced by three factors: the tightness of the bounds, the ability to avoid unnecessary computations through fathoming and preprocessing rules, and the way subproblems are generated and selected. We examine these issues in the remainder of the section.

3.1. *Bounding procedure*

The following dual-ascent bounding procedure is executed at Step 3 of the BB algorithm. Lower bounds are computed on the optimal value $Z_{\tilde{\mathscr{D}}}$ of the modified dual $\tilde{\mathscr{D}}$, which is obtained from $\mathscr{D}$ by adding the constant term $\Sigma_{j \in T_1} f_j$ to the objective function, and by replacing the fixed costs f_j ($j \in T$) in constraints (13) by the modified fixed costs $\tilde{f}_j$ ($j \in T$), defined as follows: $\tilde{f}_j = f_j$, if $j \in T_{01}$; $\tilde{f}_j = +\infty$, if $j \in T_0$; $\tilde{f}_j = 0$, if $j \in T_1$. The *FLIP–FLOP procedure* may then be formally stated as follows:

(1) (*initialization*) Initialize γ, Z_0^l, a lower bound on $Z_{\tilde{\mathscr{D}}}$, and Z^u, an upper bound on Z. Set the iteration counter t to 1.
(2) (*integrality test*) If $T_{01} = \varnothing$, compute an upper bound Z_t^u on Z by solving an MCNF; if $Z_t^u < Z^u$, $Z^u \leftarrow Z_t^u$; STOP.
(3) (*lower bound*) Compute a lower bound Z_t^l on $Z_{\tilde{\mathscr{D}}}$ either by solving the FLIP relaxation (if $t = 1 \bmod 2$), or by applying DUALOC to the FLOP relaxation (if $t = 0 \bmod 2$).
(4) (*lower bound test*) If $Z_t^l \geqslant Z^u$, STOP.
(5) (*upper bound*) Compute an upper bound Z_t^u on Z either from the optimal solution of the FLIP relaxation (if $t = 1 \bmod 2$), or by solving an MCNF derived from the best primal solution to the FLOP relaxation identified by DUALOC (if $t = 0 \bmod 2$).
(6) (*upper bound update*) If $Z_t^u < Z^u$, $Z^u \leftarrow Z_t^u$.
(7) (*stopping test*) If $Z^u - Z_t^l < \varepsilon_1 Z_t^l$ or $Z_t^l - Z_{t-1}^l < \varepsilon_2 Z_{t-1}^l$ or $t = t_{max}$, STOP.
(8) (*preprocessing rule*) Attempt to fix somes variables (T_{01}, T_0 and T_1 may be modified).
(9) $t \leftarrow t + 1$. Goto 2.

To initialize the procedure, two extremal strategies are possible. In *restarting* mode, which is the default at the root node, γ and Z_0^l are initialized to 0. In *recuperation* mode, we use the values generated at the father node to initialize γ and Z_0^l. In all cases, Z^u is initialized to the value of the best solution identified thus far by the BB algorithm.

The procedure starts with a FLIP, a choice experimentally proven to be superior (Crainic and Delorme, 1993). Indeed, if a FLOP is first solved, one does not take into account the influence of the balancing flows. In particular, some depots may be given very large values for their associated γ multipliers, and consequently become "unattractive", although they might subsequently be required in order to satisfy the balancing constraints.

Note that the lower bound test performed at Step 4 includes the usual feasibility test that stops computations when the relaxation is determined to be infeasible. Indeed, we assume in our description that any infeasible subproblem takes an infinite optimal value.

The stopping test uses three parameters ε_1, ε_2 and t_{max} that can be adjusted by the user. The first stops the procedure when the relative gap between the lower and upper bounds is sufficiently small, the second comes into play when the lower bound has not sufficiently increased from one iteration to the next, while the third limits the number of iterations.

3.2. *Fathoming and preprocessing*

A first obvious fathoming criterion, performed at Step 4 of the FLIP–FLOP procedure, eliminates a subproblem with a lower bound higher than Z^u. Two properties can be used, however, to implement stronger fathoming and preprocessing rules to either eliminate subproblems from further examination or to reduce the number of variables considered when computing bounds.

The first property, based on dual information, makes use of the slack variables associated with constraints (13):

Slack Property: Let S be a subproblem, Z^l be a lower bound on $Z_{\mathcal{D}}$ corresponding to a feasible solution $(\mu, \nu, \lambda, \gamma)$ of the dual $\tilde{\mathcal{D}}$, Z^u be an upper bound on Z, and $j \in T_{01}$. If $(Z^l + s_j) \geqslant Z^u$, then $y_j = 0$ in any optimal solution to S.

To see that this property holds, it suffices to show that $(Z^l + s_j)$ is a lower bound on $Z(S_1^j)$, the optimal value of the 1-son of S obtained by transferring j to T_1. Since $\tilde{f}_j = 0$ in the formulation of S_1^j, the γ multipliers associated to j are all equal to 0. Then, a dual feasible solution to S_1^j can be obtained from $(\mu, \nu, \lambda, \gamma)$ by setting $\bar{u}_i^p = \mu_i^p - \gamma_{ij}^p$, $\forall i \in O(j)$, $p \in P$, and $\bar{v}_i^p = v_i^p - \gamma_{ji}^p$, $\forall i \in D(j)$, $p \in P$, while all other variables are kept at their current values. But the value of this solution is precisely $Z^l + f_j - \Sigma_{p \in P}(\Sigma_{i \in O(j)} o_i^p \gamma_{ij}^p + \Sigma_{i \in D(j)} d_i^p \gamma_{ji}^p) = Z^l + s_j$.

A second property that can be used as a preprocessing rule determines when a depot must be opened in order to satisfy supply and demand requirements:

OD Property: Let S be a subproblem. If, for a given commodity p, there exists an origin (destination) i with $o_i^p > 0$ ($d_i^p > 0$) such that only one depot j is adjacent to i, then $y_j = 1$ in any feasible solution to S.

Among the possibilities to exploit these properties, we selected two for subsequent experiments. The *slack fathoming rule* consists of using the Slack Property as a fathoming rule when a 1-son node is generated at Step 4 of the BB algorithm. The *slack preprocessing rule* applies the Slack Property to fix variables at Step 8 of the FLIP–FLOP procedure, and when some depots are being closed, it also makes use of the OD Property. In this case, the OD Property is only applied once a FLOP relaxation has been solved, since to use the test after a FLIP would require dual values to be modified in case depots are opened, which would be rather inefficient. The Slack Property is also used each time a 1-son node is selected at Step 6 of the BB algorithm, assuming the relevant information from its father has been saved. Finally, the OD Property is used at Step 2 of the BB algorithm, when the root node or a 0-son is considered.

Another technique that may be used to avoid unnecessary computations is the *bound elimination test*. Used in conjunction with the recuperation initialization strategy, this test eliminates, for some subproblems, the need to apply the bounding procedure at Step 3 of the BB algorithm. To use the test, four assumptions must be satisfied: a 0-son node, $S_0^{j^*}$, is considered; when solving its father node, the last iteration of the FLIP–FLOP procedure was a FLIP; in the optimal solution that was found to this FLIP problem, there was no flow circulating through depot j^*; and, finally, when considering $S_0^{j^*}$, no depot is opened by using the OD Property at Step 2 of the BB algorithm. Then, since $S_0^{j^*}$ is obtained from its father by closing depot j^*, and no depot is being opened due to the OD Property, the lower bound that would be computed by solving a FLIP on $S_0^{j^*}$ must be equal to the lower bound generated when solving the father. Thus, the FLIP–FLOP procedure would be stopped immediately by the application of the stopping test, assuming, of course, that Z_0^l is initialized to the value of the lower bound at the father node.

3.3. *Branching rules*

Branching rules are used to choose the next depot variable to fix (j^* in Step 4 of the BB algorithm), and to determine which subproblem should be examined first: $S_0^{j^*}$ or $S_1^{j^*}$.

We distinguish two categories of branching rules: primal rules make use of the information obtained when computing the last upper bound, while dual rules exploit the dual variables generated while computing the last lower bound.

From prior experiments, it was found that the most promising primal rules compare the "positive" activity at depot j, as measured by customer–depot flows circulating through j, to the fixed cost required to open depot j. Crainic, Delorme and Dejax (1993) use similar rules in their BB algorithm, and succeed in rapidly finding good solutions. Formally, given a solution x to an upper bound subproblem, one computes

$$X_j = \left\{ \sum_{p \in P} \left(\sum_{(i,j) \in A_{OT}} x_{ij}^p + \sum_{(j,i) \in A_{TD}} x_{ji}^p \right) \Big/ f_j \right\} \quad \forall j \in T_{01} \tag{26}$$

to define the two rules:

X_0 *rule*: Choose $j^* = \arg\min_{j \in T_{01}} \{X_j\}$ and select first subproblem $S_0^{j^*}$.
X_1 *rule*: Choose $j^* = \arg\max_{j \in T_{01}} \{X_j\}$ and select first subproblem $S_1^{j^*}$.

A first class of dual rules is based on the observation that relatively large negative values of λ_j^p imply that depot j tends to attract more flow of commodity p than it sends to customers. A similar argument applies when the λ_j^p have large positive values. Thus, a large value of $|\lambda_j^p|$ indicates that depot j has significant shortage or surplus of commodity p, resulting in interdepot balancing flows. Hence, using the same rationale as for the primal rules defined above, one measures

$$\Lambda_j = \left(\sum_{p \in P} |\lambda_j^p| \right) \Big/ f_j \quad \forall j \in T_{01} \tag{27}$$

to define the two rules:

Λ_0 *rule*: Choose $j^* = \arg\min_{j \in T_{01}} \{\Lambda_j\}$ and select first subproblem $S_0^{j^*}$.
Λ_1 *rule*: Choose $j^* = \arg\max_{j \in T_{01}} \{\Lambda_j\}$ and select first subproblem $S_1^{j^*}$.

A second class of dual rules compares the fixed cost to open depot j to the values of its related γ multipliers. One makes use of the slack variables s_j associated to constraints (13) of $\tilde{\mathscr{D}}$, to define the two rules:

Γ_0 *rule*: Choose $j^* = \arg\max_{j \in T_{01}} \{s_j\}$ and select first subproblem $S_0^{j^*}$.
Γ_1 *rule*: Choose $j^* = \arg\min_{j \in T_{01}} \{s_j\}$ and select first subproblem $S_1^{j^*}$.

4. EXPERIMENTAL RESULTS

The experimentation aims at identifying the most efficient branch-and-bound design, and is centered around five main issues:

- *Gamma adjustment*: What is the influence of the gamma adjustment rule on the efficiency of the algorithm? In particular, for which values of θ do we obtain the best performance?
- *Restarting versus recuperation*: Which initialization strategy is the most efficient?
- *Stopping parameters*: For how long should the FLIP–FLOP procedure be executed? What is the "ideal" tradeoff between the number of generated subproblems and the time spent solving each of these subproblems?
- *Slack fathoming versus slack preprocessing*: What is the most efficient way to use the Slack Property, through fathoming or preprocessing?
- *Branching rule*: Which branching rule is the most efficient?

To answer these questions, we implemented the algorithm in FORTRAN/77, using the primal simplex code RNET (Grigoriadis and Hsu, 1979) to solve uncapacitated minimum cost network flow problems. In order to solve uncapacitated location problems, we coded our own multicommodity version of DUALOC (Erlenkotter, 1978). The dual–ascent phase follows Erlenkotter, while the adjustment step is the primal–dual procedure of Van Roy and Erlenkotter (1982). This adjustment phase is repeated as long as the dual objective continues to increase. The code was compiled with the *f77* compiler using the $-O$ option, and all experiments were performed on SUN Sparc2 workstations.

The tests are conducted on randomly generated problems (the generator is described in Crainic, Delorme and Dejax, 1993) and on data based on an actual large-scale application. Table 1 displays the dimensions of the test problems, where problems P_1 to P_{12} and P_{13} to P_{24} are, respectively, medium and large-size randomly generated instances, while problems P_{25} to P_{28} are based on the actual application. The parameter F, the last column of the table, indicates the relative importance of the fixed costs. For randomly generated problems, two levels, 1 and 2, are used, while levels 1–4 define the instances based on the actual data. In fact, a problem at level i ($i > 1$) is obtained from a problem at level $i - 1$ by multiplying the fixed costs by 10. Thus, for example, problems P_1 and P_2 are the same except for this modification.

A first experiment is dedicated to determining the relative efficiency of the branching rules. For each data instance, the six branching rules are tested, executing the BB algorithm

Table 1. Dimensions of the test problems

Prob	$\|P\|$	$\|O\|$	$\|D\|$	$\|T\|$	$\|A_{OT}\|$	$\|A_{TD}\|$	$\|A_{TT}\|$	F
P_1	1	125	125	25	875	875	600	1
P_2	1	125	125	25	875	875	600	2
P_3	4	125	125	25	879	879	600	1
P_4	4	125	125	25	879	879	600	2
P_5	3	124	124	26	871	871	650	1
P_6	3	124	124	26	871	871	650	2
P_7	6	125	125	25	875	875	600	1
P_8	6	125	125	25	875	875	600	2
P_9	1	124	124	26	868	868	650	1
P_{10}	1	124	124	26	868	868	650	2
P_{11}	4	124	124	26	869	869	650	1
P_{12}	4	124	124	26	869	869	650	2
P_{13}	1	219	219	44	2630	2630	1892	1
P_{14}	1	219	219	44	2630	2630	1892	2
P_{15}	2	219	219	44	2630	2630	1892	1
P_{16}	2	219	219	44	2630	2630	1892	2
P_{17}	1	220	220	43	2641	2641	1806	1
P_{18}	1	220	220	43	2641	2641	1806	2
P_{19}	2	219	219	44	2629	2629	1892	1
P_{20}	2	219	219	44	2629	2629	1892	2
P_{21}	1	219	219	44	2629	2629	1892	1
P_{22}	1	219	219	44	2629	2629	1892	2
P_{23}	2	220	220	43	2647	2647	1806	1
P_{24}	2	220	220	43	2647	2647	1806	2
P_{25}	12	289	289	87	1810	1810	746	1
P_{26}	12	289	289	87	1810	1810	746	2
P_{27}	12	289	289	87	1810	1810	746	3
P_{28}	12	289	289	87	1810	1810	746	4

up to 1000 iterations. No gamma adjustment is used, and the recuperation initialization strategy and slack fathoming rule are implemented. The stopping parameters are given the following values: $\varepsilon_1 = \varepsilon_2 = 0.01$, and $t_{max} = 10$. Other algorithmic options display the same general behavior with respect to the relative performance of the branching rules. Table 2 shows, for each problem and each branching rule, the number of iterations performed ("generated nodes"). When the algorithm is stopped before proving optimality of the best solution found, a "+" indicates that an optimal solution was obtained, while a "−" denotes the contrary.

Table 2. Tests on branching rules (generated nodes)

Prob	X_0	X_1	Λ_0	Λ_1	Γ_0	Γ_1
P_1	263	389	267	205	326	1000^+
P_2	161	816	157	936	84	1000^+
P_3	63	733	61	289	70	877
P_4	390	1000^+	465	503	116	1000^-
P_5	599	1000^-	592	1000^-	695	1000^+
P_6	45	110	44	64	31	153
P_7	316	682	326	705	565	1000^+
P_8	354	1000^-	354	1000^-	628	1000^+
P_9	318	270	328	219	112	383
P_{10}	120	330	119	271	188	742
P_{11}	282	257	268	273	195	460
P_{12}	707	1000^-	684	504	787	1000^+
P_{13}	146	1000^+	169	712	114	1000^+
P_{14}	513	1000^-	556	1000^-	1000^+	1000^-
P_{15}	1000^-	1000^-	1000^-	1000^-	1000^-	1000^-
P_{16}	100	332	108	178	60	1000^+
P_{17}	370	1000^-	393	1000^-	431	1000^-
P_{18}	226	282	227	453	64	1000^-
P_{19}	1000^-	1000^-	1000^-	1000^-	1000^+	1000^-
P_{20}	1000^-	1000^-	1000^-	1000^-	1000^+	1000^-
P_{21}	1000^+	1000^-	1000^+	1000^-	1000^+	1000^-
P_{22}	498	1000^-	488	1000^-	200	1000^-
P_{23}	1000^-	1000^-	1000^-	1000^-	1000^+	1000^-
P_{24}	1000^+	1000^+	1000^+	1000^+	1000^+	1000^+
P_{25}	201	114	201	149	51	165
P_{26}	585	361	585	509	253	626
P_{27}	1000^+	837	1000^+	1000^-	260	499
P_{28}	228	163	228	159	89	128

A few conclusions emerge from the experiment. In particular, the rules that select first a 1-son node are generally outperformed by the corresponding rules that explore first a 0-son node. Also, the X_0 and Λ_0 rules behave almost identically since they both measure the "importance" of a depot in terms of the flow component of the problem. However, the Λ_0 rule should be preferred since it is computationally less expensive. These rules are, however, clearly outperformed by the Γ_0 rule, especially when applied to solve large-size problems. The superiority of this rule may be easily explained since it combines powerfully with the Slack Property to reduce the number of generated subproblems. Therefore, our general conclusion is that the Γ_0 rule is preferable and should be used in all cases.

The way to exploit the Slack Property, through either fathoming or preprocessing, and the gamma adjustment rule are the objects of a second experiment. We only report the results of the extremal adjustments: no adjustment ($\theta = 0$) and maximum adjustment ($\theta = 1$), since prior experiments have shown their superiority over intermediate adjustments

($0 < \theta < 1$). This behavior may be easily explained. On the one hand, using no adjustment maintains the slack variables at their highest values, thus preserving the strength of the Slack Property. On the other hand, the maximum adjustment maximizes the number of 0-son nodes for which bounds are not computed at Step 3 of the BB algorithm, following the application of the bound elimination test. Indeed, since the maximum adjustment increases significantly the routing costs in the formulation of the FLIP problem, it yields a better chance of identifying a network flow solution with inactive depots. In this experiment, the Γ_0 branching rule and the recuperation initialization strategy are used. The stopping parameters are given the same values as previously. Table 3 displays, for each problem and each combination of adjustment ($\theta = 0$ or $\theta = 1$) and either fathoming or preprocessing rules, the total number of generated subproblems ("nodes") and the elapsed CPU time in seconds ("time"), required to find an optimal solution.

Table 3. Fathoming versus preprocessing

	Fathoming				Preprocessing			
	$\theta = 0$		$\theta = 1$		$\theta = 0$		$\theta = 1$	
Prob	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
P_1	326	56	358	56	245	45	355	54
P_2	84	13	98	20	67	15	67	16
P_3	70	51	78	54	79	72	77	53
P_4	116	74	122	88	131	114	149	111
P_5	695	298	707	285	507	239	617	245
P_6	31	24	48	27	11	24	21	20
P_7	565	570	581	575	523	574	579	569
P_8	628	564	388	331	431	538	511	465
P_9	112	17	114	18	111	16	111	17
P_{10}	188	37	67	19	149	42	103	31
P_{11}	195	131	199	139	205	154	235	159
P_{12}	787	432	401	258	441	294	423	300
P_{13}	114	52	148	61	79	35	107	44
P_{14}	1638	848	1387	872	993	911	619	567
P_{15}	29551	36606	15297	18431	6751	9707	8473	10217
P_{16}	60	89	94	101	25	58	37	58
P_{17}	431	199	338	156	361	172	397	174
P_{18}	64	42	98	52	47	39	51	36
P_{19}	4041	4371	4135	4233	6391	7022	6935	7197
P_{20}	5237	4962	3028	3015	4803	5429	4201	4215
P_{21}	4914	2270	5378	2451	6577	3226	8079	3389
P_{22}	200	82	193	75	143	77	193	69
P_{23}	45629	40246	29009	25306	10263	10046	10355	9440
P_{24}	3373	2796	1662	1412	2031	3765	1179	1108
P_{25}	51	46	92	109	69	123	71	123
P_{26}	253	394	303	487	195	343	303	438
P_{27}	260	384	234	311	295	1305	393	646
P_{28}	89	164	162	195	15	50	15	50

The results shown in the table demonstrate the superiority of the preprocessing option when used to solve very difficult problems, such as P_{15} and P_{23}. For all other problems, both options are equally effective, but still a slight advantage belongs to the preprocessing option. The effects of the gamma adjustment rule on the performance of the algorithm is more difficult to assess. In particular, when the preprocessing option is used, the two extremal adjustments perform equally well. However, subsequent experiments with different

stopping parameters ($\varepsilon_1 = \varepsilon_2 = 0.0001$) have shown that using no adjustment is generally more efficient than performing maximum adjustment.

A third experiment explores both the initialization strategies and the adjustment of the stopping parameters. The recuperation approach is compared to a hybrid initialization strategy, which consists of applying the restarting mode when a subproblem is obtained from backtracking, and recuperation otherwise. Prior experiments have demonstrated the superiority of both methods over a pure restarting strategy. Two adjustments of the stopping parameters are tested: $\varepsilon_1 = \varepsilon_2 = \varepsilon = 0.01$, and $\varepsilon = 0.0001$, along with $t_{max} = 10$ (note that this maximum is never achieved). The Γ_0 branching rule and the slack preprocessing option are used, and no adjustment of the γ multipliers is performed. Table 4 shows the results of this experiment, for each problem and each combination of stopping parameters ($\varepsilon = 0.01$ or 0.0001) and initialization strategies (recuperation or hybrid).

Table 4. Recuperation versus hybrid initialization

	Recuperation				Hybrid			
	$\varepsilon = 0.01$		$\varepsilon = 0.0001$		$\varepsilon = 0.01$		$\varepsilon = 0.0001$	
Prob	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
P_1	245	45	113	36	43	18	39	19
P_2	67	15	45	17	53	15	37	16
P_3	79	72	41	65	39	65	13	33
P_4	131	114	47	107	49	130	41	213
P_5	507	239	159	143	167	167	57	94
P_6	11	24	7	29	7	26	5	31
P_7	523	574	263	560	221	462	77	354
P_8	431	538	59	514	63	358	41	438
P_9	111	16	77	16	51	13	41	14
P_{10}	149	42	45	27	57	31	39	21
P_{11}	205	154	75	92	49	85	23	61
P_{12}	441	294	105	349	63	220	41	230
P_{13}	79	35	39	36	35	39	35	45
P_{14}	993	911	235	759	175	616	145	583
P_{15}	6751	9707	1363	5604	727	3192	355	2581
P_{16}	25	58	9	69	19	206	9	119
P_{17}	361	172	257	173	91	84	97	96
P_{18}	47	39	7	44	41	69	7	44
P_{19}	6391	7022	1189	2645	741	2679	711	3783
P_{20}	4803	5429	307	3028	419	3626	267	3966
P_{21}	6577	3226	877	801	359	497	275	489
P_{22}	143	77	77	87	65	122	65	127
P_{23}	10263	10046	3061	6376	635	1790	467	1918
P_{24}	2031	3765	361	1709	249	1700	141	1233
P_{25}	69	123	69	123	75	160	49	293
P_{26}	195	344	193	371	69	332	37	181
P_{27}	295	1305	63	330	169	1987	53	691
P_{28}	15	50	3	92	7	61	3	103

The results support the following general conclusion: it is preferable to spend more time at each node computing tighter bounds than to stop the bounding procedure early in the hope of avoiding useless work. Indeed, the hybrid strategy, which performs more work on each subproblem treated following backtracking, is clearly superior to the recuperation rule. Regarding the adjustment of the stopping parameters, a smaller ε generally performs better, particularly when the recuperation strategy is used.

As a general conclusion, we recommend the following algorithmic design: use the Γ_0 branching rule, the slack preprocessing option, the hybrid initialization strategy, with no adjustment of the γ multipliers, and set the stopping parameters to the following values: $\varepsilon_1 = 0.0001, \varepsilon_2 = 0.0001$ and $t_{max} = 10$. To assess the competitiveness of this implementation, we compare it to other methods reported in the literature. Table 5 shows the results of two heuristics: "TABU", the tabu search method of Crainic, Gendreau, Soriano and Toulouse (1992), and "DUAL", the dual-ascent heuristic of Crainic and Delorme (1993). For the tabu search heuristic, the results of problems P_{25} to P_{28} are not available. The table also displays the results of an implementation of the branch-and-bound approach ("BBWR") of Crainic, Delorme and Dejax (1993), where a maximum of 10000 subproblems are explored. Note that, by using this limit on the number of subproblems, optimality could be proved for only six of the 28 problems. The last two columns of the table display the results obtained with our method ("BBFF"). For all methods, the value of the best solution found (Z^u), rounded in thousands of units, and the elapsed CPU time in seconds on SUN Sparc2 workstations ("time") are indicated.

The results show that the BB algorithm outperforms the other methods, either in solution quality, in computing times, or in both. Furthermore, it is important to note that even if the dual-ascent heuristic generally identifies good quality solutions, it only finds an optimal solution to 25% of the problems in our data set, while the BB algorithm improves by as much as 1%–8% the solution of another 25% of the problems. Hence, the extra computational effort required by the BB algorithm is well paid off by the improvement in solution quality.

Table 5. Comparison with other methods

	TABU		DUAL		BBWR		BBFF	
Prob	Z^u	Time	Z^u	Time	Z^u	Time	Z^u	Time
P_1	42499	194	42499	1	42499	1993	42499	20
P_2	61565	176	65320	5	61565	69	61565	18
P_3	109177	889	108914	3	108914	6994	108914	37
P_4	142063	718	143036	17	142063	3488	142063	218
P_5	69144	609	69487	6	69611	4034	69144	98
P_6	88518	498	88518	16	88518	332	88518	34
P_7	248308	1333	248329	3	249142	9804	248178	361
P_8	291199	1050	292080	45	290271	9929	290271	445
P_9	21134	195	21406	1	21134	1926	21134	15
P_{10}	46014	183	50011	6	46014	127	46014	23
P_{11}	93874	802	94216	9	93874	5929	93874	66
P_{12}	136642	727	137668	22	136642	5890	136642	234
P_{13}	23286	340	23349	5	23286	5030	23286	49
P_{14}	42310	330	43909	52	42062	372	42062	587
P_{15}	65495	1103	65308	14	66975	10298	65230	2591
P_{16}	89827	734	89068	25	89068	11254	89068	126
P_{17}	33271	396	33601	7	33464	4960	33267	100
P_{18}	56832	366	57553	26	56832	5017	56832	47
P_{19}	61174	922	61052	20	63937	9552	60959	3794
P_{20}	101278	678	101420	59	102159	8658	100851	3975
P_{21}	49026	423	48917	7	49348	4148	48860	493
P_{22}	76949	198	77589	12	77386	4701	75903	131
P_{23}	96163	811	96312	16	98311	4148	96114	1928
P_{24}	138908	990	133454	18	133454	7946	133454	1241
P_{25}	–	–	53176	5	53229	19587	53176	313
P_{26}	–	–	56526	9	56829	19130	56515	201
P_{27}	–	–	72385	41	72339	22919	72339	712
P_{28}	–	–	166506	32	166506	5556	166506	122

5. CONCLUSION

The multicommodity location problem with balancing requirements is related to one of the major logistics issues faced by distribution and transportation firms: the management of a fleet of vehicles over a medium to long term planning horizon. In the particular context of the land transportation management of a heterogeneous fleet of containers by an international maritime shipping company, savings of up to 40% of the total transportation cost of empty containers have been identified (Dejax *et al.*, 1987) by finding approximate solutions to the model. Furthermore, in the same context, both algorithmic and solution efficiencies are of prime importance, since this logistics problem has to be solved regularly due to variations in patterns of demands, transportation costs, space availability and costs for container warehousing.

To solve the problem, we presented a branch-and-bound algorithm in which bounds are computed by a dual-ascent procedure. Several branching, fathoming and preprocessing rules were introduced and experimental results allowed us to identify an efficient algorithmic design. They have also shown that the method performs well on a wide variety of problems, including an actual large-scale application. Furthermore, for these problems, the algorithm outperforms other methods proposed in the literature.

Acknowledgements—Financial support for this project was provided by the NSERC of Canada and the Fonds FCAR of the Province of Québec.

REFERENCES

Ahuja, R. K., Magnanti, T. L. & Orlin, J. B. (1993) *Network flows: theory, algorithms, and applications.* Englewood Cliffs, NJ: Prentice-Hall.

Cornuejols, G., Nemhauser, G. L. & Wolsey, L. A. (1990) The uncapacitated facility location problem. In R. L. Francis & P. B. Mirchandani (Eds), *Discrete location theory* (pp. 119–168). New York: Wiley-Interscience.

Crainic, T. G., Dejax, P. J. & Delorme, L. (1989) Models for multimode multicommodity location problems with interdepot balancing requirements. *Annals of Operations Research*, **18**, 279–302.

Crainic, T. G. & Delorme, L. (1993) Dual-ascent procedures for multicommodity location–allocation problems with balancing requirements. *Transportation Science*, **27**, 90–101.

Crainic, T. G., Delorme, L. & Dejax, P. J. (1993) A branch-and-bound method for multicommodity location with balancing requirements. *European Journal of Operational Research*, **65**, 368–382.

Crainic, T. G., Gendreau, M., Soriano, P. & Toulouse, M. (1992) A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations Research*, **41**, 359–383.

Dejax, P., Crainic, T. G., Delorme, L., Blise, M., de Tocqueville, E. & Hodgson, J. (1987) Planification tactique du transport terrestre des conteneurs vides—Rapport de synthèse. Ecole Centrale, Paris: Rapport LEIS.

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

Geoffrion, A. M. (1974) Langrangean relaxation for integer programming. *Mathematical Programming Study*, **2**, 82–114.

Grigoriadis, M. D. & Hsu, T. (1979) RNET—The Rutgers minimum cost network flow subroutines, Rutgers University, New Brunswick, New Jersey.

Ibaraki, T. (1987) Enumerative approaches to combinatorial optimization. *Annals of Operations Research*, **10–11**.

Krarup, J. & Pruzan, P. M. (1983) The simple plant location problem: Survey and synthesis. *European Journal of Operational Research*, **12**, 36–81.

Van Roy, T. J. & Erlenkotter, T. (1982) A dual-based procedure for dynamic facility location. *Management Science*, **28**, 1091–1105.