TABU SEARCH WITH EXACT NEIGHBOR EVALUATION FOR MULTICOMMODITY LOCATION WITH BALANCING REQUIREMENTS[1]

BERNARD GENDRON AND JEAN-YVES POTVIN

Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7

and

Département d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7

PATRICK SORIANO

Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Québec, Canada H3C 3J7

and

Ecole des Hautes Etudes Commerciales, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7

ABSTRACT

In this paper, we present a tabu search heuristic for solving the multicommodity location problem with balancing requirements. This heuristic improves upon a previous implementation by performing an exact, rather than approximate, evaluation of neighboring solutions. It also includes a number of refinements, in particular, a new initialization procedure and enhanced neighborhood reduction techniques. The heuristic is shown to be very effective, as it identifies the optimal solution on every instance taken from a set of randomly generated problems. It also finds optimal or near-optimal solutions on a set of large-size instances derived from an actual application, with computation times that show little variance as compared with the currently best known exact method.

Key words : Tabu search, multicommodity location with balancing requirements.

RÉSUMÉ

Dans cet article, nous présentons une méthode de recherche avec tabous pour résoudre le problème de localisation multiproduit avec exigences d'équilibrage. Cette heuristique effectue une évaluation exacte des voisins, permettant ainsi d'améliorer les performances d'une méthode de recherche avec tabous précédemment développée pour le même problème, et basée sur une évaluation approximative. La méthode suggérée inclut également une nouvelle procédure d'initialisation, de même que des techniques sophistiquées de réduction des voisinages. Par des expérimentations numériques sur un ensemble de problèmes générés aléatoirement, nous montrons que la méthode est efficace, puisqu'elle permet d'identifier la solution optimale de chacun de ces problèmes. Sur des exemplaires de grande taille tirés d'une application réelle, l'heuristique produit des solutions optimales ou quasi-optimales, avec des temps de calcul relativement stables en comparaison de la meilleure méthode exacte connue à ce jour pour résoudre le problème.

Mots-clés : Recherche avec tabous, problème de localisation multiproduit avec exigences d'équilibrage.

1. INTRODUCTION

The multicommodity location problem with balancing requirements (MLB) was first introduced by Crainic, Dejax and Delorme [5]. The problem is motivated by an industrial application related to the management of a heterogeneous fleet of containers for an

[1]Recd. April 1998; Revd. Nov. 1998 INFOR vol. 37, no. 3, Aug. 1999

international maritime shipping company. Once a ship arrives at the port, the company has to deliver the loaded containers, which may come in several types and sizes, to designated in-land destinations. Following their unloading by the importing customer, empty containers are moved to a depot. Later on, they may be delivered from there to customers who request containers for subsequent shipping of their own products. In addition, empty containers often have to be repositioned to other depots. These interdepot movements are a consequence of regional unbalances in empty container availabilities and needs throughout the network: some areas lack containers of certain types, while others have a surplus. This requires balancing movements of empty containers among depots, which 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, namely, the costs of opening and operating the depots, and the costs generated by customer-depot and interdepot movements.

Among the procedures that have been proposed for solving the MLB [6, 7, 8, 9, 10, 11, 12, 13], the tabu search heuristic of Crainic, Gendreau, Soriano and Toulouse [8] (hereafter, denoted the CGST heuristic) proved to be particularly effective. This was illustrated through numerical comparisons with lower bounds obtained with a dual-ascent procedure [6]. Later on, Gendron and Crainic [12] observed that, for many instances, solutions obtained by the CGST heuristic fall short of optimal ones, computed by their branch-and-bound algorithm.

The CGST heuristic is based on an approximate evaluation of each neighbor of the current solution. This algorithmic choice was motivated by the observation that exact evaluations are computationally expensive, since each one involves the solution of a multicommodity uncapacitated minimum cost network flow problem (MCNF). However, with the advent of highly sophisticated linear programming (LP) solvers, including adaptations of the simplex method to network flow problems and advanced reoptimization capabilities, exact evaluations might now be a viable alternative. The objectives of this paper are to describe an improved tabu search heuristic based on exact neighbor evaluation and to show that the proposed method is more effective than the previous CGST implementation. To this end, computational results are reported on the set of randomly generated problems used for testing CGST [8]. In addition, results are reported on instances taken from a large-size actual application and comparisons with respect both to solution quality and computation times are provided with the best known exact method for solving the problem [12].

The organization of the paper is the following. In Section 2, we give a mathematical formulation of the MLB and describe the MCNF subproblem used to evaluate neighboring solutions. Section 3 presents the new tabu search heuristic, which shares with CGST a similar global search strategy, but also shows a number of important distinct features, including different initialization and neighborhood reduction procedures. Computational experiments are reported and analyzed in Section 4. The conclusion summarizes our work and proposes extensions. Throughout the paper, we assume familiarity with the principles of tabu search; for further details, the interested reader is referred to [14].

2. PROBLEM FORMULATION

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. There are several *commodities* (*types of containers*) moving through the network and 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 *transshipment* 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 (i.e., $O(j) \cup D(j) \neq \emptyset$). 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 in minimizing the costs incurred by moving flows of commodities through the network to satisfy the supplies at origins and the 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 non-negative and deterministic. A non-negative 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 non-negative 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 assumes value 1 if depot j is opened, and value 0 otherwise. The problem is then formulated as:

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

subject to

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

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

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

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

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

$$x_{ij}^p \geq 0, \quad \forall (i,j) \in A, p \in P, \quad (7)$$

$$y_j \in \{0,1\}, \quad \forall j \in T. \quad (8)$$

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 the interdepot costs satisfy the triangle inequality [5], an assumption that we follow throughout this text.

Upper bounds on the optimal value of this problem might be derived by fixing the vector of location variables y to some value $\overline{y}$. We then obtain the following MCNF:

$$Z(\overline{y}) = \sum_{j \in T} f_j \overline{y}_j + \min \left\{ \sum_{p \in P} \left(\sum_{(i,j) \in A_{OT}} c_{ij}^p x_{ij}^p + \sum_{(j,i) \in A_{TD}} c_{ji}^p x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right) \right\}, \quad (9)$$

subject to constraints (2) to (4), (7) and

$$x_{ij}^p \leq o_i^p \overline{y}_j, \qquad \forall j \in T, i \in O(j), p \in P, \tag{10}$$

$$x_{ji}^p \leq d_i^p \overline{y}_j, \qquad \forall j \in T, i \in D(j), p \in P. \tag{11}$$

This problem decomposes into $|P|$ single-commodity uncapacitated minimum cost network flow problems. As efficient solution procedures exist for this class of problems [1], heuristic methods that search the space of location variables appear computationally attractive. Both the new heuristic and the previous CGST heuristic exploit this observation.

3. TABU SEARCH

The CGST heuristic dates back to 1993 and was one of the first application of tabu search to mixed-integer problems [8]. The approach presented in this paper is based on a similar problem-solving scheme, but includes a number of improvements and refinements. In the following, we first describe the global search strategy, which is similar in both methods. Then, the most important specific features of the new heuristic are underlined and contrasted with those found in CGST.

3.1 Global Search Strategy

As mentioned in the previous section, the network flow structure of the problem is exploited by fixing the y variables to 0 or 1, thus determining the status of the corresponding depot (i.e., closed or open). For a given vector of values $\overline{y}$, a series of single-commodity minimum cost network flow problems are solved to obtain an allocation of customers to depots, as well as the interdepot balancing flows for that particular configuration of open and closed depots. A configuration of depots with its corresponding flows constitutes a solution to the problem, whose cost is the sum of fixed and flow costs over the open depots.

Since most customers are linked to only a subset of potential depot sites, some customers may not be connected to any open depot in a given solution, thus leading to infeasibility. To enable the search process to consider such infeasible solutions, an artificial depot is added to the network. This depot is always open and is connected to all customer nodes through high cost arcs (which ensure that no flow will pass through them, unless there is no feasible alternative). Obviously, the procedure keeps track of feasibility while the search progresses to make sure that the best proposed solution is indeed feasible.

The neighborhood structure is based on two different neighborhood types. The first one consists in moves where a single y_j variable is modified by opening a closed depot (*add* move) or by closing an open depot (*drop* move). In addition to this add/drop neighborhood, the procedure also considers *swap* moves in which one simultaneously opens a depot and closes another. Typically, only a fraction of the add/drop and swap neighborhood is considered at each move (see Section 3.4).

The motivation for this double neighborhood scheme stems from the following observations. First, in most problems, good solutions tend to have the same number of open depots. Second, the objective function value tends to vary significantly from one solution to the next when add/drop moves are performed. The add/drop neighborhood is therefore a good medium to get close to high quality solutions but, due to the large jumps in the objective, it can hardly perform an efficient search once a certain quality level has been reached. The swap neighborhood is thus used to attain the best possible configuration, once the "right" number of open depots has been determined.

During the search, a global iteration proceeds as follows. First, a sequence of add/drop moves is followed by a sequence of swap moves. If the best solution found after these two sequences, called *local_best*, is feasible, a strictly improving swap sequence is applied on a (possibly) larger fraction of the swap neighborhood, where only improving moves are accepted. One could view the latter as a kind of intensification phase. If *local_best* is not feasible, the costs on the arcs connected to the artificial depot are adjusted, as well as some control parameters, so that infeasible solutions look less attractive, before a new sequence of add/drop and swap moves is performed. Note that each sequence of moves is not of fixed length but rather ends when a maximum number of consecutive iterations have been performed without improving *local_best*.

After J consecutive global iterations without improving *local_best* (rather than after a fixed number of global iterations, as in CGST [8]), diversification is performed to encourage a wider exploration of the solution space. Diversification is applied to the best global solution found by the search up to that point and consists in reversing the status of the depots whose status - open or closed - has changed the least often over the course of the search (see [8] for more details).

To prevent cycling, two tabu lists are defined. List T_1, which is used for add/drop sequences, records the $|T_1|$ last depots added or dropped from the solution. It prevents the reversal of their status for as long as the depots stay on the list. The second list T_2 records pairs of depots involved in the last $|T_2|$ swaps performed by the search. It prevents both reversals and repetitions of these moves. In our implementation, the length of the tabu tenure is a random value chosen within a given interval, rather than a deterministic (but problem dependent) value, as in CGST [8].

The general problem-solving procedure may then be sketched as follows:

- **Initialization**

  Find an initial solution $\overline{y}_0$ and set the current solution to $\overline{y}_0$; $Z^* := Z(\overline{y}_0)$.

- **Main procedure**

  1. Perform a global iteration:
     - 1.1 Set *local_best* to the current solution.
     - 1.2 Perform an add/drop sequence.
     - 1.3 Perform a swap sequence.
     - 1.4 If *local_best* is not feasible, update the search parameters and go to Step 1.2.
     - 1.5 Perform a strictly improving swap sequence.
     - 1.6 If a new best solution is found, update Z^*.
  2. If at least I moves have been performed, stop the procedure.
  3. If J consecutive global iterations without improving *local_best* have been performed since the last diversification, perform diversification to identify a new current solution.
  4. Go to Step 1.

Given this general algorithmic scheme, the most important new features of the improved heuristic, which are described in the next subsections, may be summarized as follows:

- new initialization procedure;
- exact evaluation of neighbors;
- enhanced sampling procedure for neighborhood reduction.

3.2 Initialization

A starting solution for the tabu search is quickly produced through a local descent based on an approximation of the original objective function. This approximation is derived from the observation that a standard location-allocation problem (i.e., the simple plant location problem [3, 15]) is obtained by ignoring the interdepot balancing arcs A_{TT}. A simple heuristic for solving the latter problem starts by opening half of the depots with lowest fixed costs. Then, the following iterative procedure is applied:

1. Set the current solution to the initial solution.
2. Evaluate the neighborhood of the current solution as follows: for each depot, consider the cost of the new solution produced by changing the status of this depot (i.e., from closed to open or from open to closed).
3. If the minimum cost solution in the neighborhood is better than the current solution, set the current solution to this new solution and go to Step 2, otherwise stop.

Since the interdepot arcs are ignored, each neighboring solution is easily evaluated. Indeed, in this context, we can define a "customer" (i, p) as a combination of a customer $i \in O \cup D$ and a commodity $p \in P$. Then, if a depot j is to be opened, customer (i, p) closer to depot j than its currently assigned depot $j(i, p)$ is reallocated to j. The difference in cost between the new and the current solution is

$$\Delta_j = f_j - \sum_{p \in P} \left(\sum_{i \in O(j)} \left(c^p_{ij(i,p)} - c^p_{ij} \right)^+ o^p_i + \sum_{i \in D(j)} \left(c^p_{j(i,p)i} - c^p_{ji} \right)^+ d^p_i \right)$$

(where $a^+ = \max\{0, a\}$). Similarly, if a depot j is to be closed, customer (i, p) currently assigned to it is reallocated to the closest alternative open depot $j(i, p)$. The difference in cost between the new and the current solution is $-\Delta_j$.

This approach is to be opposed to the one found in the CGST heuristic, where the number of open depots in the initial solution is determined a priori, based on a rough estimate of the number of open depots associated with "good" solutions. The corresponding number of depots with lowest fixed costs are then opened to determine the initial configuration [8].

3.3 Neighborhood Evaluation

The tabu search calls CPLEX [4] to solve exactly the associated minimum cost network flow problems. Since the status of only one or two depots changes after an add, drop or swap move, the reoptimization features of CPLEX are fully exploited to limit the number of simplex iterations. In particular, the optimal basis associated with the current solution is provided as the initial basis for evaluating each neighbor. This is possible since the problem specification remains invariant throughout the simplex procedure: when a depot is closed, each incident arc is forbidden by simply setting upper bounds on the allowed flow to zero (c.f. constraints (10) and (11)). The latter bounding constraints can be implicitly handled by CPLEX and do not need to be part of the main problem formulation.

By contrast, CGST evaluates neighboring solutions approximately. First, the interdepot balancing flows are ignored and the impact of a move (add, drop or swap) is evaluated in the same way as in a standard location-allocation problem. Then, the modification to the interdepot balancing costs is approximated using an estimate of the balancing flow costs per commodity unit, which is derived from previous solutions encountered during the search and updated through exponential smoothing (details can be found in [8]).

3.4 Neighborhood Reduction

Exact evaluations are still quite computationally expensive and sophisticated procedures must be developed to reduce the neighborhood size without compromising solution quality. In the following, we explain how moves leading to neighboring solutions are ranked and how these ranks are exploited to bias the selection of moves that are then evaluated exactly.

3.4.1. Ranking. Two different rankings of neighboring solutions are developed for the add, drop and swap moves. The rankings use two types of approximations based either on interdepot balancing costs, or on fixed and allocation costs. In the following, T^1 and T^0 denote the sets of open and closed depots, respectively. Given these, the rankings may be described as follows:

- *Add (drop)*
  - Ranking 1: Rank depots $j \in T^0(T^1)$ in increasing (decreasing) order of

$$\omega_j = \sum_{p \in P} \left(\sum_{k \in T^1 \cap T^+(j)} c^p_{jk} + \sum_{k \in T^1 \cap T^-(j)} c^p_{kj} \right),$$

    i.e., the closed (open) depot with minimum (maximum) ω_j is ranked first.
  - Ranking 2: Rank depots $j \in T^0(T^1)$ in increasing (decreasing) order of

$$\Delta_j = f_j - \sum_{p \in P} \left(\sum_{i \in O(j)} \left(c^p_{ij(i,p)} - c^p_{ij} \right)^+ o^p_i + \sum_{i \in D(j)} \left(c^p_{j(i,p)i} - c^p_{ji} \right)^+ d^p_i \right),$$

    where, if $j \in T^0$, $j(i,p)$ is the open depot currently assigned to customer (i,p) and, if $j \in T^1$, $j(i,p)$ is the open depot other than j closest to customer (i,p). Thus, the closed (open) depot with minimum (maximum) Δ_j is ranked first.

- *Swap*
  - Ranking 1: For each depot $k \in T^0$, rank depots $j \in T^1$ in increasing order of

$$\omega_{kj} = \sum_{p \in P} \left(c^p_{jk} + c^p_{kj} \right)$$

    (if $j \notin T^+(k)$ or $j \notin T^-(k)$, then ω_{kj} assumes a large value). Thus, the open depot with minimum ω_{kj} is ranked first.
  - Ranking 2: For each depot $k \in T^0$, rank depots $j \in T^1$ in increasing order of

$$\Delta_{kj} = f_k - f_j + \sum_{p \in P} \left(\sum_{i \in O(j)} \left(c^p_{ij(i,p)} - c^p_{ij} \right)^+ o^p_i + \sum_{i \in D(j)} \left(c^p_{j(i,p)i} - c^p_{ji} \right)^+ d^p_i \right),$$

    where $j(i,p)$ is the open depot, excluding depot j but including depot k which is to be opened, closest to customer (i,p). Thus, the open depot with minimum Δ_{kj} is ranked first.

For swap moves, the open depots are thus ranked with respect to each closed depot. When a fraction of the neighborhood is desired, this fraction is taken from the sorted list of open depots at each closed depot k. This approach extends the swap neighborhood of the CGST heuristic, where only the "closest" open depot j (i.e., with minimum ω_{kj}) is candidate for a swap [8].

3.4.2. Ranking selection. At each move, a particular ranking is chosen as follows:

- *Add/drop*
  - When the add/drop neighborhood is considered, a choice must first be made between add and drop. To this end, a random number is generated between 0 and $|T| - 1$. If this number is less than $|T^1|$, drop is selected, otherwise add is selected. Hence, drop moves are more likely to be selected when $|T^1|$ is large, and conversely.
  - A random selection is then made between *Ranking 1* and *Ranking 2* for the selected type of move.
- *Swap*
  - When a swap move is considered, a random selection is made between *Ranking 1* and *Ranking 2*.

3.4.3. Sampling. Assuming that a particular type of move and a particular ranking are chosen, a sampling method is applied to determine the subset of moves to be evaluated exactly. The sampling is biased, as a move is more likely to be selected if it is better ranked. Assuming d possible moves, the best move (with rank 1) is associated with some *Max* value, while the worst move (with rank d) is associated with some *Min* value. The values of the other moves are then equally spaced between *Min* and *Max* using the formula:

$$v_i = Max - (Max - Min) \times \frac{i-1}{d-1}, \quad 1 \leq i \leq d,$$

where v_i is the value associated with the move of rank i. The selection probability p_i of each move is then:

$$p_i = \frac{v_i}{\sum_{j=1}^{d} v_j}, \quad 1 \leq i \leq d.$$

Assuming that $Min + Max = 2$, the selection bias in favor of the best moves can be increased by setting the *Max* value closer to 2, or reduced by setting its value closer to 1 [2, 16]. The reader is referred to Section 4 for the exact size of the samples used in the computational experiments. Note that a uniform random law is used in [8] to sample the add/drop and swap (with "closest" open depot only) neighborhoods for the CGST heuristic.

4. COMPUTATIONAL EXPERIMENTS

To evaluate the improved tabu search heuristic, computational experiments were performed on two classes of problems: small to medium-size randomly generated test problems and large-scale instances derived from an actual application. Problems in the first class are the same as those used in [8] to evaluate the CGST heuristic. The second class of problems is introduced here to measure the impact of using our method on realistic large-size instances. The currently best known exact method for solving the problem, the branch-and-bound algorithm of Gendron and Crainic [12], was applied on both test sets. We observed that the new heuristic can find the optimal solution on every problem in the first class, but requires more computation time than the exact algorithm. Since the latter is likely to generate very large search trees when the problem size increases, we conjectured that for realistic large-size instances, our heuristic might prove more efficient, while retaining solution quality. Our computational experiments on the second class of problems suggest that it is indeed the case.

Problem	$\|P\|$	$\|O\|$	$\|D\|$	$\|T\|$	$\|A_{OT}\|$	$\|A_{TD}\|$	$\|A_{TT}\|$
RND_1	1	125	125	25	875	875	600
RND_2	1	125	125	25	875	875	600
RND_3	4	125	125	25	879	879	600
RND_4	4	125	125	25	879	879	600
RND_5	3	124	124	26	871	871	650
RND_6	3	124	124	26	871	871	650
RND_7	6	125	125	25	875	875	600
RND_8	6	125	125	25	875	875	600
RND_9	1	124	124	26	868	868	650
RND_{10}	1	124	124	26	868	868	650
RND_{11}	4	124	124	26	869	869	650
RND_{12}	4	124	124	26	869	869	650
RND_{13}	1	219	219	44	2630	2630	1892
RND_{14}	1	219	219	44	2630	2630	1892
RND_{15}	2	219	219	44	2630	2630	1892
RND_{16}	2	219	219	44	2630	2630	1892
RND_{17}	1	220	220	43	2641	2641	1806
RND_{18}	1	220	220	43	2641	2641	1806
RND_{19}	2	219	219	44	2629	2629	1892
RND_{20}	2	219	219	44	2629	2629	1892
APP_1	12	289	289	130	1914	1914	890
APP_2	12	289	289	130	1914	1914	890
APP_3	12	289	289	130	1914	1914	890
APP_4	12	289	289	130	1914	1914	890
APP_5	12	289	289	130	1914	1914	890

Table 1: Dimensions of Test Problems

In this section, we summarize the results of these experiments. We first describe the characteristics of the test problems and then present how promising parameter settings were selected for the different types of problems. Finally, we compare the results of our new heuristic with those obtained by other methods.

4.1 Test Problems

Table 1 displays the dimensions of the test problems, where problems RND_1 to RND_{12} and RND_{13} to RND_{20} are, respectively, small and medium-size randomly generated instances, while problems APP_1 to APP_5 are based on the actual application. Note that each randomly generated problem identified with an even number is obtained from the previous one by multiplying the fixed costs by 10. For example, problems RND_1 and RND_2 are the same except for this modification. Problem APP_1 is from an actual large-scale application. The other problems in this class are obtained by perturbing all costs by a random factor chosen in the interval $[0.8, 1.8]$.

4.2 Parameter Settings

Preliminary experiments have been conducted to determine appropriate parameter values for our tabu search heuristic. The parameters fall into two classes: those that are critical with respect to both solution quality and computation time (typically, the setting of these parameters is problem dependent), and those that are less critical, that is,

%diversf	%add/drop	%swap	%stswap	optimal (/60)
0	60	0	0	40
0	60	10	0	43
0	60	0	30	51
0	60	10	30	52
0	100	0	0	32
0	100	10	0	46
0	100	0	30	51
0	100	10	30	52
10	60	0	0	40
10	60	10	0	53
10	60	0	30	54
10	60	10	30	52
10	100	0	0	47
10	100	10	0	50
10	100	0	30	58
10	100	10	30	56
20	60	0	0	43
20	60	10	0	49
20	60	0	30	56
20	60	10	30	53
20	100	0	0	46
20	100	10	0	50
20	100	0	30	56
20	100	10	30	60

Table 2: Parameter Settings for Random Problems

the performance of the algorithm remains similar for values "around" those that were finally selected. Here are the parameters in this last category with the values chosen in our tests:

- stopping criterion for each sequence of moves (add/drop and swap): 5 consecutive moves without improving *local_best*;
- diversification: performed after $J = 1$ consecutive global iterations without improving *local_best*;
- tabu tenures: randomly chosen in the interval $[2, 5]$;
- ranking selection: *Ranking 1* and *Ranking 2* are selected with equal probability for both add/drop and swap moves;
- selection bias in sampling: $Min = 0.5$ and $Max = 1.5$.

The remaining more "critical" parameters are:

- I: minimum number of moves performed during the whole search procedure;
- %diversf: percentage of depots to be complemented when a diversification step is performed;
- %add/drop: percentage of add/drop neighborhood evaluated at each move;
- %swap: percentage of swap neighborhood evaluated at each move;
- %stswap: percentage of strictly improving swap neighborhood evaluated at each move.

These parameters were adjusted for the two types of problems, that is, the randomly generated (20 instances) or the realistic large-size problems (5 instances). After some preliminary tests, the parameter I was fixed to 150 for the randomly generated problems, and to 50 for the large-size instances. Our tests revealed that we could have reduced this number to 100 for the random problems, without compromising solution quality. However, $I = 150$ was finally selected as a protection against a bad probabilistic sequence. For the large-size problems, increasing the number of moves provided no improvement in solution quality.

Tables 2 and 3 summarize the results obtained with different values of the critical parameters, on the randomly generated and the large-size instances, respectively. For each configuration of the parameters, we show the number of optimal solutions obtained with the tabu search (these optimal solutions were previously computed by the branch-and-bound algorithm [12]). Since our tabu search contains stochastic features, each problem was solved three times. Hence, the total number of runs for each configuration is 20x3=60, for the randomly generated problems, and 5x3=15, for the large-size instances.

Three main conclusions emerge from the first table:

- Swap moves are essential to identify effective solutions. Among the two types of swap moves, strictly improving swap sequences appear to be more effective. Since this might be attributed to the larger percentage of candidates evaluated at each strictly improving swap move, we have tried to increase %swap up to 30% and found no noticable improvement in solution quality. Since the size of the swap neighborhood increases quadratically with the number of depots, allowing more evaluations would be inefficient.
- Exploring the entire add/drop neighborhood is generally more effective than selecting only a portion of it. Notice that the size of the add/drop neighborhood grows linearly with the number of depots, so a complete exploration is realistic.
- Diversification is necessary to obtain optimal solutions for some difficult problems. The best observed value for %diversf was 20 (we have tried to increase %diversf and found no improvement).

Based on this last observation, we have decided to fix the parameter %diversf to 20 for the tests on the large-size instances. For these problems, preliminary tests have shown that using the same values for the other parameters would be computationally impractical. The results of these tests also allowed us to derive promising values, which are shown in Table 3. The figures in this table support our previous conclusions: (1) it is essential to perform swap moves and (2) a high value of %add/drop is indicated (note that we have tried to explore the entire add/drop neighborhood for these problems and found no improvement).

Based on these results, the values selected for the "critical" parameters, in order to compare our method with the existing ones, are:

- number of moves I: 150 for random problems, 50 for large-size problems;
- %diversf: 20;
- %add/drop: 100 for random problems, 60 for large-size problems;
- %swap: 10 for random problems, 5 for large-size problems;
- %stswap: 30 for random problems, 10 for large-size problems.

%diversf	%add/drop	%swap	%stswap	optimal (/15)
20	30	0	0	3
20	30	5	0	3
20	30	0	10	6
20	30	5	10	9
20	60	0	0	9
20	60	5	0	11
20	60	0	10	12
20	60	5	10	12

Table 3: Parameter Settings for Large-Size Problems

Prob	CGST				GPS			
	Z^*	iter Z^*	CPU Z^*	CPU	Z^*	iter Z^*	CPU Z^*	CPU
RND_1	42506	166	102	186	42499	42	38	140
RND_2	62249	13	6	169	61565	77	78	153
RND_3	109577	266	704	790	108914	59	306	818
RND_4	142063	10	19	690	142063	14	89	1260
RND_5	69194	166	296	550	69144	17	67	670
RND_6	88518	252	395	478	88518	1	4	813
RND_7	249366	108	441	1149	248178	2	17	1368
RND_8	291647	163	543	1007	290271	23	257	2163
RND_9	21134	33	22	186	21134	47	45	164
RND_{10}	46014	13	7	175	46014	3	3	185
RND_{11}	94270	237	627	767	93874	3	14	808
RND_{12}	137205	214	482	685	136642	46	300	1196
RND_{13}	23286	32	37	310	23286	0	1	808
RND_{14}	42130	203	211	302	42062	28	149	938
RND_{15}	66465	92	236	762	65230	49	598	2280
RND_{16}	89827	286	643	670	89068	90	1680	2798
RND_{17}	33271	175	213	356	33267	54	253	828
RND_{18}	56832	241	270	335	56832	101	637	1004
RND_{19}	61905	297	711	715	60959	16	213	2529
RND_{20}	101278	229	486	620	100851	72	1240	2674

Table 4: Comparison with the CGST Heuristic

4.3 Comparison with Other Methods

We first compare our method to the CGST heuristic. We have obtained from the authors the detailed results of their experiments on the randomly generated test problems. Extensive comparisons of the two heuristics are shown in Table 4. For each method ("CGST" and our procedure, called "GPS"), we indicate: the value of the best solution found, rounded in thousands of units ("Z^*"); the iteration (move) where this best solution was identified ("iterZ^*"); the CPU time, in seconds, up to iteration "iter Z^*"("CPUZ^*"); the total CPU time in seconds ("CPU"). Several important remarks

clarify the interpretation of these results. First, as our heuristic contains stochastic features, it was run three times and the results of the *worst* run are shown. Given that GPS always identifies an optimal solution, the worst run is the one that finds the optimal solution the latest. Second, our heuristic was run on a Sun Ultra workstation, while the CGST heuristic was run on a Sun Sparc 2, which is roughly 10 times slower than the Ultra. Third, the parameter I in the CGST heuristic was fixed to 300. Fourth, for the CGST heuristic, the results correspond to the "pure" tabu search and do not incorporate a descent procedure based on add/drop moves, which is applied at the end to the best solution found (as suggested in [8]). This descent procedure could improve the solution found by tabu search for 10 problems out of 20. However, we did not want to incorporate it into our analysis, since our aim was to compare exclusively the behavior of the two tabu search procedures, in particular, how fast they find their best solution.

We observe that the CGST heuristic is generally an order of magnitude faster than our method. However, the latter found the optimal solution on every instance, while the CGST heuristic could identify the optimal solution only on 6 out of the 20 instances; this figure can be raised by using the descent procedure at the end, but still, only 11 instances are then solved to optimality. Note that the CGST procedure is not expected to provide further improvements when a number of moves larger than 300 is performed. Indeed, Crainic, Toulouse and Gendreau [10] report that a parallel implementation of CGST, when a total of at least 600 moves are performed on 4 processors, did not improve solution quality when compared to the same implementation with at least 300 moves executed on 8 processors.

To isolate the impact of neighbor evaluation (since other features differentiate GPS from CGST), we have implemented a simple tabu search heuristic that resembles CGST, but with the approximate neighbor evaluation replaced by an exact one. This heuristic uses the same initialization as CGST and a uniform sampling, instead of our more sophisticated initialization and sampling procedures. On the random problems, when using the same parameter setting as GPS, the resulting method found an optimal solution 55 times over 60 runs. These results confirm the superiority of the exact neighbor evaluation over the approximate one used in CGST, and show that a large part of the improvement provided by the new heuristic can be explained by the exact neighbor evaluation. We also observed that the initialization procedure allows the method to identify the best solution significantly more rapidly than the initialization used in CGST, thus improving the robustness of the tabu search.

Although the exact neighbor evaluation is crucial in obtaining high quality solutions, the impact of using a tabu search framework is not negligeable. Indeed, to verify this conjecture, we have implemented a simple multi-start descent heuristic which proceeds as follows: each time we start a new descent, we randomly select half of the depots to be opened (instead of choosing those with the lowest fixed costs), and then proceed with the same initialization and search procedures as before, but with all tabu mechanisms and diversification disabled. When used with the same parameter setting as for GPS, this method found an optimal solution 42 times over 60 runs. Therefore, a simple multi-start descent heuristic that does not incorporate tabu search mechanisms, even if it is based on exact neighbor evaluation, is not powerful enough to find optimal solutions to all problems.

Table 5 compares GPS with the branch-and-bound approach of Gendron and Crainic [12] on the large-size instances. For our heuristic, we give the same statistics as in the previous table. For the branch-and-bound method ("BBAL"), we only give the optimal value ("Z^*"), rounded in thousands of units, and the total CPU time in seconds

Prob	BBAL		GPS			
	Z^*	CPU	Z^*	iterZ^*	CPUZ^*	CPU
APP_1	49182	3348	49183	22	2709	11468
APP_2	61156	3357	61156	46	5587	6932
APP_3	59914	14215	59914	19	2384	6852
APP_4	60049	9425	60049	30	3672	6537
APP_5	51924	29446	51924	30	3643	6388

Table 5: Comparison with the Branch-and-Bound Algorithm

("CPU"), since we did not have access to other statistics. Both algorithms were run on a Sun Ultra workstation.

These results show that our heuristic found the optimal solution on 4 out of the 5 test problems, and it came very close on the remaining one. Moreover, the CPU times are relatively stable, while those of the branch-and-bound algorithm vary widely, even if the problems are quite similar (in particular, they all have the same dimension). For 3 of these instances, the heuristic is computationally less expensive, while retaining the same solution quality. This result is certainly encouraging, as it demonstrates that our heuristic is well-adapted to realistic large-scale instances.

5. CONCLUSION

In this paper, we have presented a tabu search heuristic for solving the multicommodity location problem with balancing requirements. Our heuristic improves a previously developed tabu search procedure (the CGST heuristic) reported in [8], by performing an exact neighbor evaluation rather than an approximate one. The new implementation also includes a number of refinements, in particular, a new initialization procedure and enhanced neighborhood reduction techniques. Our heuristic found the optimal solution on every instance taken from a set of randomly generated test problems, previously designed for testing the CGST method. The new heuristic also found optimal or near-optimal solutions on large-size instances derived from an actual application, with computation times that showed little variance as compared to the branch-and-bound algorithm of Gendron and Crainic [12].

One possible way to reduce the computation time of the tabu search without affecting solution quality too much would be to apply exact evaluations only once in a while, and use approximations otherwise. Investigations along these lines could allow the heuristic to tackle instances that are out of the range of the branch-and-bound algorithm. Another interesting research avenue would be to adapt the tabu search procedure to similar problems having additional constraints, like capacity constraints on the depots. This can be done quite easily within the current tabu search framework (which is not the case for the exact branch-and-bound algorithm).

ACKNOWLEDGMENTS

Financial support for this project was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and the Québec Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR). We want to acknowledge the efforts of Christelle Rusque and Serge Bisaillon who helped us in implementing and testing our tabu search heuristic. We also want to thank three anonymous referees whose valuable comments have helped us write a better paper.

BIBLIOGRAPHY

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

[2] Baker J. (1985), "Adaptive Selection Methods for Genetic Algorithms", in *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, J.J. Grefenstette ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 101-111.

[3] Cornuéjols G., Nemhauser G.L. and Wolsey L.A. (1990), "The Uncapacitated Facility Location Problem", in *Discrete Location Theory*, R.L. Francis and P.B. Mirchandani eds., Wiley-Interscience, 119-168.

[4] CPLEX Optimization, Inc. (1995), *Using the CPLEX Callable Library (Version 4.0)*.

[5] Crainic T.G., Dejax P.J. and Delorme L. (1989), "Models for Multimode Multicommodity Location Problems with Interdepot Balancing Requirements", *Annals of Operations Research*, 18, 279-302.

[6] Crainic T.G. and Delorme L. (1993), "Dual-Ascent Procedures for Multicommodity Location-Allocation Problems with Balancing Requirements," *Transportation Science*, 27 (2), 90-101.

[7] Crainic T.G., Delorme L. and Dejax P.J. (1993), "A Branch-and-Bound Method for Multicommodity Location with Balancing Requirements", *European Journal of Operational Research*, 65 (3), 368-382.

[8] Crainic T.G., Gendreau M., Soriano P. and Toulouse M. (1993), "A Tabu Search Procedure for Multicommodity Location/Allocation with Balancing Requirements", *Annals of Operations Research*, 41, 359-383.

[9] Crainic T.G., Toulouse M. and Gendreau M. (1995), "Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements", *OR Spektrum*, 17 (2-3), 113-123.

[10] Crainic T.G., Toulouse M. and Gendreau M. (1996), "Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements", *Annals of Operations Research*, 63, 277-299.

[11] Gendron B. and Crainic T.G. (1993), "Parallel Implementations of a Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements", *INFOR*, 31 (3), 151-165.

[12] Gendron B. and Crainic T.G. (1995), "A Branch-and-Bound Algorithm for Depot Location and Container Fleet Management", *Location Science*, 3 (1), 39-53.

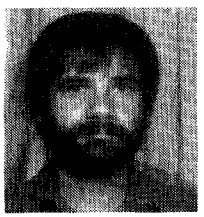
[13] Gendron B. and Crainic T.G. (1997), "A Parallel Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements", *Computers & Operations Research*, 24 (9), 829-847.

[14] Glover F. and M. Laguna (1997), *Tabu Search*, Kluwer.

[15] Krarup J. and Pruzan P.M. (1983), "The Simple Plant Location Problem: Survey and Synthesis", *European Journal of Operational Research*, 12, 36-81.

[16] Whitley D. (1989), "The GENITOR Algorithm: Why Rank-Based Allocation of Reproductive Trials is Best", in *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer ed., Morgan Kaufmann, San Mateo, CA, 116-121.

Bernard Gendron is an Assistant Professor in the Department of Computer Science and Operations Research at the University of Montréal. His research interests include integer and combinatorial optimization, large-scale optimization and parallel computing, and location and network design problems applied to transportation logistics and telecommunications. He is affiliated to the Centre for Research on Transportation at the University of Montréal and he has been a Visiting Scientist in the Operations Research Center at the Massachusetts Institute of Technology during the year 1995-96.



Jean-Yves Potvin is an Associate Professor in the Department of Computer Science and Operations Research at the University of Montréal. He is also an active member of the Centre for Research on Transportation (CRT). His research interests include network flows, combinatorial optimization and application of metaheuristics in transportation. He has published numerous articles in these fields. He is also Associate Editor of INFORMS Journal on Computing.

Patrick Soriano is an Assistant Professor in the Quantitative Methods Department of Ecole des Hautes Etudes Commerciales of Montréal. He is affiliated to the Centre for Research on Transportation (CRT) and to the Groupe d'Études et de Recherche en Analyse des Décisions (GERAD). His research interests include heuristic search techniques, combinatorial optimization, telecommunication and transportation planning, network flows and graphs. He is also Associate Editor of INFOR.