Tabu Search with Ejection Chains for the Vehicle Routing Problem with Private Fleet and Common Carrier

Jean-Yves Potvin
Marc-André Naud

November 2009

CIRRELT-2009-50

Tabu Search with Ejection Chains for the Vehicle Routing Problem with Private Fleet and Common Carrier

Jean-Yves Potvin[1,2,*], **Marc-André Naud**[2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7

Abstract. The problem reported in this paper is a variant of the classical vehicle routing problem, where customer requests for a transportation company can be served either by its private fleet of vehicles or assigned to an external common carrier. The latter case occurs if the demand exceeds the total capacity of the private fleet or if it is more economical to do so. Accordingly, the objective is to minimize the variable and fixed costs of the private fleet plus the costs charged by the common carrier. A tabu search heuristic with a neighborhood structure based on ejection chains is proposed to solve this problem. It is empirically demonstrated that this algorithm outperforms the best approaches reported in the literature on a set of benchmark instances with both homogeneous and heterogeneous fleets.

Keywords. Vehicle routing, private fleet, common carrier, tabu search, ejection chains.

Acknowledgements. Financial support for this work was provided by the Natural Sciences and Engineering Council of Canada (NSERC). This support is gratefully acknowledged.

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2009

1 Introduction

In the Vehicle Routing Problem with Private fleet and Common carrier (VRPPC), customer requests for a transportation company can be served either by its private fleet of vehicles or assigned to an external common carrier. The latter case occurs if the total capacity of the private fleet is exceeded or if it is more economical to do so. When a customer is assigned to the common carrier, a penalty is incurred which represents all costs associated with this assignment. The objective is to minimize the sum of fixed vehicle costs, travel costs and common carrier costs.

A single-vehicle version of this problem is first mentioned in [16] and later solved to optimality with a branch-and-bound algorithm for instances with up to 200 customers [8]. This problem is introduced as a special case of the Prize Collecting Traveling Salesman Problem in [2], since the private fleet does not need to serve all customers and a common carrier cost in incurred for unserved customers. The multi-vehicle version is first reported and solved in [6] with a savings-based construction heuristic, followed by intra-route and inter-route customer exchanges. The authors in [3] then propose an approach where two different initial solutions are improved with sophisticated customer exchanges. A more powerful perturbation metaheuristic called RIP, for Randomized construction, Improvement, Perturbation, is reported in [4]. This metaheuristic combines a local descent based on different neighborhood structures with two diversification strategies: a randomized construction procedure and a perturbation mechanism where pairs of customers are swapped. Results are reported on a set of benchmark instances with vehicle fleets that are either homogeneous or heterogeneous. Finally, in [7], a tabu search heuristic is shown to outperform these last results, but only on homogeneous instances.

Here, an improved tabu search heuristic is proposed, which exploits a powerful neighborhood structure based on ejection chains. This tabu search is shown to produce the best known results on the benchmark instances in [4], including those with an heterogeneous fleet of vehicles. The rest of the paper is organized as follow. In Section 2, the problem is formally introduced. Then, our tabu search is described in Section 3. Computational results are reported in Section 4. Finally, a conclusion follows.

2 Problem Formulation

The problem can be stated as follow. Let $G = (V, A)$ be a directed graph, with $V = \{1, 2, ..., n\}$ the vertex set and A the arc set. Vertex 1 is the depot and the other vertices are customers to be served. A travel cost c_{ij} is associated with each arc $(i, j) \in A$. Each customer $i \in V\backslash\{1\}$ is characterized by a demand q_i and a cost e_i when it is assigned to the common carrier. A private fleet of m vehicles, initially located at the depot, is available to serve the customers, where each vehicle k has capacity Q_k. Also, a fixed cost f_k is incurred when vehicle k is used. The goal of the VRPPC is to find a set of at most m routes for the private fleet such that:

- a vehicle from the private fleet serves a single route that starts and ends at the depot;
- every customer is visited exactly once either by a vehicle of the private fleet or by the common carrier;
- the total demand on each route served by vehicle k should not exceed its capacity Q_k;
- the sum of travel costs, fixed vehicle costs and common carrier costs is minimized.

The problem can be mathematically formulated as follow [4]. Let:

- x_{ij}^k be 1 if vehicle k visits vertex j immediately after vertex i, 0 otherwise, $i, j = 1, ..., n$, $i \neq j$; $k = 1, ..., m$;
- y_i^k be 1 if vehicle k visits vertex i, 0 otherwise, $i = 1, ..., n$; $k = 1, ..., m$;
- z_i be 1 if customer i is assigned to the common carrier, 0 otherwise, $i = 2, ..., n$; $k = 1, ..., m$;
- $u_i^k \geq 0$ be an upper bound on the load of vehicle k upon leaving customer i, $i = 2, ...n$; $k = 1, ...m$.

We have:

$$\min \sum_{k=1}^{m} f_k y_1^k + \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ij} x_{ij}^k + \sum_{i=2}^{n} e_i z_i \tag{1}$$

subject to

$$\sum_{k=1}^{m}\sum_{j=2}^{n} x_{1j}^{k} = \sum_{k=1}^{m}\sum_{i=2}^{n} x_{i1}^{k} \leq m, \tag{2}$$

$$\sum_{\substack{j=1\\ j\neq h}}^{n} x_{hj}^{k} = \sum_{\substack{i=1\\ i\neq h}}^{n} x_{ih}^{k}, \quad h = 1, ...n;\; k = 1, ...m, \tag{3}$$

$$\sum_{k=1}^{m} y_i^k + z_i = 1, \quad i = 2, ..., n, \tag{4}$$

$$\sum_{i=2}^{n} q_i y_i^k \leq Q_k, \quad k = 1, ..., m, \tag{5}$$

$$u_i^k - u_j^k + Q_k x_{ij}^k \leq Q_k - q_j, \quad i, j = 2, ...n,\; i \neq j;\; k = 1, ..., m. \tag{6}$$

In this formulation, the objective function (1) minimizes the sum of vehicle fixed costs, travel costs and common carrier costs (note that y_1^k is 1 when vehicle k is used, 0 otherwise). Constraints (2) state that at most m vehicles from the private fleet can be used. Constraints (3) specify that the same vehicle must enter and leave a given vertex. Constraints (4) assign each customer either to a vehicle from the private fleet or to the common carrier. Constraints (5) ensure that the vehicle capacity is not exceeded. Finally, constraints (6) eliminate subtours that do not include the depot.

With regard to this model, it is worth noting that our tabu search relaxes constraints (5), thus allowing infeasible solutions to be considered. A new objective $g(s) = f(s) + \alpha q(s)$ is defined, where $f(s)$ is the original objective value of solution s, as defined in equation (1), α is a positive parameter and $q(s)$ is a penalty term for exceeding vehicle capacity, that is

$$q(s) = \sum_{k=1}^{m} \left[\sum_{i=2}^{n} q_i y_i^k - Q_k \right]^+ ,$$

with $[x]^+ = \max\{0, x\}$. Clearly, when s is feasible, both $f(s)$ and $g(s)$ coincide, otherwise a penalty for overcapacity is incurred.

In the following section, our tabu search for solving the VRPPC is described.

3 Tabu search

The main components of the tabu search heuristic will be introduced in the next subsections. It is assumed that the reader is familiar with tabu search, otherwise a detailed presentation is found in [10]. In the following discussion, customers assigned to the common carrier are associated with a route served by vehicle $m + 1$. This vehicle has infinite capacity and the ordering of its customers is arbitrary (i.e., the assignment cost of customer i to this route is fixed and is always equal to e_i).

3.1 Assignment of customers to the common carrier

First, a number of customers are assigned to the common carrier, if the total demand exceeds the capacity of the private fleet. More precisely, the assignment procedure is as follow:

If $\sum_{i=2}^{n} q_i > \sum_{k=1}^{m} Q_k$ then

i. reindex the customers in increasing order of ratio $\frac{e_i}{q_i}$, $i = 2, ..., n$;

ii. assign the first h customers to the common carrier where

$$\sum_{i=2}^{h} q_i \geq \sum_{i=2}^{n} q_i - \sum_{k=1}^{m} Q_k \geq \sum_{i=2}^{h-1} q_i$$

Thus, as long as the total demand exceeds the capacity of the private fleet of vehicles, the customer with the lowest cost per unit of demand is assigned to the common carrier.

3.2 Initial solutions

Routes are created to serve customers that have not been assigned to the common carrier. To this end, two different randomized heuristics are applied.

Least-cost insertion

At each iteration, the next customer is selected at random and inserted into the current routes, by considering only locations that induce a small incremental cost. More precisely, the insertion place for a given customer is selected at random among the ϕ best feasible insertion places. All customers that cannot be feasibly inserted with this procedure are assigned to the common carrier.

Convex hull insertion

Customers are first assigned to vehicles of the private fleet, based on a dispersion value. Given some vehicle k and a subset of customers V_k assigned to it, the dispersion value of this vehicle is:

$$D_k = \begin{cases} \frac{\sum_{i,j \in V_k} d_{ij}}{|V_k|(|V_k|-1)} & \text{if } |V_k| > 1 \\ 0 & \text{if } |V_k| = 1, \end{cases}$$

where d_{ij} is an appropriate distance metric (e.g., euclidean distance). At each iteration, a customer i is selected at random and assigned to a feasible vehicle with the lowest dispersion value, based on its current set of assigned customers plus customer i. To create additional randomness, the dispersion value of each vehicle k is set to $D_k(1 + r_k)$, where r_k is chosen uniformly randomly in the interval $[0, \beta]$ and where β is a parameter. When the assignment is completed, a route is created for each vehicle by first identifying the convex hull of its assigned customers, using the procedure reported in [14]. Then, the remaining customers are inserted with the previously mentioned least-cost insertion procedure.

Once the initial solution is constructed with either the least-cost or convex hull insertion heuristic, a first-improvement local descent based on the neighborhoods described in subsection 3.3 is applied, followed by another descent based on 4-opt* exchanges [14]. These exchanges are a subset of 4-opt exchanges [12], where four arcs are removed from a route and four arcs are introduced to reconnect the subpaths and produce a new route. Here, a subset of possible exchanges is obtained by limiting the number of vertices in a subpath and by ensuring that at least one added arc is shorter than a removed arc.

3.3 Neighborhoods

In this subsection, the neighborhood structures used throughout the execution of the tabu search are first described. Then, a more complex structure based on ejection chains, which is applied only in particular circumstances, is presented. In both cases, the vertices served by the common carrier are considered like those served by the private fleet.

3.3.1 Basic neighborhoods

During the tabu search, the union of two different neighborhoods is considered. These neighborhoods are:

1. *Shift.* Remove customer i from its route and insert it at least cost in another route.

2. *Swap.* Exchange customers i in vehicle route k and i' in vehicle route k', with $i, i' = 2, ..., n$; $k,\ k' = 1, ..., m+1,\ k \neq k'$. That is, customers i and i' are first removed from routes k and k' and inserted at least cost in routes k' and k, respectively.

It is interesting to note that when the route associated with the common carrier is part of *Swap*, a customer request is moved from the common carrier to the private fleet, while another customer request takes its place. In the case of *Shift*, a customer is taken from the common carrier and added to the private fleet (or conversely). Thus, the assignment of customers between the common carrier and the private fleet is modified. The *Swap* neighborhood is also important for the success of this algorithm, because it is almost impossible to add a customer into a vehicle route with *Shift* (apart from the common carrier) when the capacity constraints are tight.

3.3.2 Ejection chains

Ejection chains have first been proposed in [9]. Basically, a customer is ejected from one route and inserted into another route, thus forcing a customer from that route to move to yet another route, and so on. The length of the chain is limited by the number of routes and can be cyclic or not. That is, it might end with the insertion of the last ejected customer in the route that started the process, thus producing a cycle, or in some other route not yet included in the chain.

In our tabu search implementation, a local descent using a neighborhood based on ejection chains is executed when (1) a local minimum with regard to the basic neighborhoods *Shift* and *Swap* is reached and (2) the value of this local minimum is within ϵ% of the best solution found thus far, where ϵ is a parameter. These restrictions are required due to the high computational requirements associated with ejection chains.

The implementation is based on the computation of a least-cost ejection path in a graph structure where the vertices correspond to customers and arcs to ejection and insertion moves. This is now explained.

Ejection and insertion moves

An ejection move can be sketched as follow, assuming that k and k' denote the current routes of customers i and i', respectively:

1. select customer i in route k and customer i' in route k', $k \neq k'$;

2. remove i from k;
3. insert i at the best feasible place in route k', assuming that customer i' has been ejected from that route.

In step 1, there are $O(n^2)$ ejection moves for a problem defined over n customers. Step 2 is evaluated in constant time, because the impact of removing each customer from its own route is computed only once at the start of the neighborhood evaluation and stored in a matrix. This is done in $O(n)$ and does not add to the complexity. The same is true for step 3, thus leading to an overall complexity of $O(n^2)$ for evaluating the ejection moves.

Insertion moves, where a customer is inserted in a route without forcing any ejection from that route, are evaluated in a similar way. Since there are $O(n)$ ways to select the customer to be moved and only $O(m)$ ways to select the target route, the complexity in this case is $O(nm)$. With $m \leq n$, the overall complexity for evaluating both insertion and ejection moves is thus $O(n^2)$.

Based on these moves and their associated cost, an ejection graph is constructed. This graph allows the identification of an ejection chain to be applied to the current local minimum, as it is explained below.

Ejection graph

The task of finding the best chain or cycle of ejection/insertion moves over the current set of routes is modeled as an elementary shortest path problem in a layered graph structure, where a layer is made of all vertices served in a given route. In a layer, there is also a dummy vertex that represents the route itself (see below). The inter-layer arcs in this ejection graph are of two different types, depending on the move:

- Ejection arcs (i, i') model the ejection of customer i' by customer i. The cost on these arcs corresponds to the insertion cost of customer i in route k' (without i') minus the savings obtained by removing i from its current route k.
- Insertion arcs $(i, dummy_{k'})$, connect customer i to the dummy associated with route k'. These arcs model the insertion of customer i in route k' (without ejection). The cost on these arcs corresponds to the insertion cost of customer i in k' minus the savings obtained by removing i from its current route k.

The shortest paths calculated over this graph are constrained as the ejection chains are not allowed to eject more than one customer per route. This restriction is necessary, otherwise the ejection costs would not be accurate for a second or any subsequent visit, due to previous ejections in the routes. This constraint also implies that negative cycles can be ignored since a layer cannot be visited twice (except to close a path in the case of an ejection cycle). The resulting constrained shortest path problem is NP-hard. In particular, the Generalized Traveling Salesman Problem (GTSP) is a special case of this problem as it corresponds to finding a shortest cycle that visits every route exactly once [13]. The problem is thus solved in a heuristic way using an adaptation of the all-pairs Floyd-Warshall shortest path algorithm [1].

This algorithm works a follows. For a given pair of vertices i, i' at iteration l, the algorithm considers paths obtained by merging the best current subpaths from i to i'' and from i'' to i', where the intermediary vertices in these subpaths cover at most route 1 to route $l-1$ and vertex i'' is not in one of these routes. The candidate vertices i'' are considered sequentially and the one that provides the first improvement is selected. If no improvement is found, the path from i to i' remains the same.

Since every ejection path should start at a vertex in a given layer and end at a dummy via an insertion move, we are only interested in the shortest paths for (vertex, dummy) pairs, among which the best one is selected at the end.

3.4 Self-adaptation

At each iteration, the value of parameter α for weighting infeasibility (overcapacity) in the objective function is modified by a factor $1+\delta > 1$. When the current solution is feasible, α is divided by $1+\delta$, to put less emphasis on feasibility and drive the search toward the infeasible domain. Conversely, when the current solution is infeasible, α is multiplied by $1+\delta$ to put more emphasis on feasibility and drive the search toward the feasible domain. In the computational experiments, δ is set to 1.25.

3.5 Tabu status

If customer i is removed from vehicle route k, it is forbidden to perform the inverse move (i.e., reinsert customer i in route k) for the next θ iterations, where θ is a uniform random value in the interval [7, 14].

3.6 Aspiration criterion

The tabu status of a move can be revoked if it leads to a feasible solution that is better than the best feasible solution found thus far.

3.7 Diversification

The value of a neighbor solution $\overline{s}$ is penalized by looking at the components of the move that leads from s to $\overline{s}$, and by considering the frequency of their use in the past [15]. Let ρ_i^k be the number of times customer i has been inserted in route k during the search process. The penalty is then proportional to the product of $f(\overline{s})$, $\sqrt{nm}$ and ρ^{min}, the minimum of the ρ_i^k's over all customers i and routes k that are part of the *Shift* or *Swap* move. If λ is the iteration number, the biased evaluation is then:

$$g(\overline{s}) + \gamma\sqrt{nm}f(\overline{s})\ \frac{\rho^{min}}{\lambda}$$

Note that the ρ_i^k values tend to diminish with problem size and the factor $\sqrt{nm}$ compensates for it. The parameter γ, set to 0.01, is also useful to adjust the intensity of the diversification. The overall effect is to drive the search toward less explored regions of the search space.

3.8 Stopping criterion

The search is stopped after a fixed number of iterations.

The performance of this tabu search on different benchmark instances is reported in the next section.

4 Computational Results

Computational experiments have been performed on benchmark instances produced by Bolduc et al. [4], where the travel cost between two vertices corresponds to the euclidean distance. The homogeneous instances are derived from the 14 VRP instances of Christofides et al. [5] and the 20 VRP instances of Golden et al. [11], for a total of 34 instances. In all cases, the customer coordinates, demands and vehicle capacities are those found in the original instances. The number of vehicles is set to $\frac{0.8q}{Q}$, where q is the total customer demand and Q the vehicle capacity. The fixed vehicle cost f is set to the average route length in the best known solution for the original instance, rounded to the nearest multiple of 20. The common carrier cost for customer i is set to $\frac{f}{\overline{n}} + \mu_i c_{1i}$ where $\overline{n}$ is the average number of customers per route in the best known solution for the original instance and μ_i equals 1, 1.5 or 2, depending if the demand of customer i is in the lowest, medium or highest customer demands for the instance [4]. The 34 heterogeneous instances are derived from the homogeneous ones by considering three different types of vehicles A, B and C with a capacity and a fixed cost that corresponds to 80%, 100% and 120% of those of the corresponding homogeneous instance.

All tests were executed on a workstation with a 2.2 GHz DualCore Opteron 275 AMD processor. This is the same processor as the one used in [7] to run another tabu search, called *TS*. Furthermore, the specifications of this processor are similar to those of the 3.6 GHz Xeon used in [4] to run the *RIP* heuristic.

In the following subsections, the benchmark instances are presented. Then, various parameter sensitivity experiments are reported. Finally, a comparison is provided with *TS* [7] and *RIP* [4], which are the two best known problem-solving methods for the VRPPC.

4.1 Benchmark instances

The characteristics of the homogeneous and heterogeneous test instances are shown in Tables 1 and 2, respectively. In Table 1, $n-1$ is the number of customers, m is the number of vehicles, and Q and f are the capacity and fixed cost of each vehicle, respectively. The names of the instances begin with *CE* or *G* for Christofides et al. [5] and Golden et al. [11], respectively. The heterogeneous instances are shown in Table 2. In this table, $n-1$ is the number of customers, m_A, m_B, m_C are the number, Q_A, Q_B, Q_C the capacity, and f_a, f_b, f_c the fixed cost of vehicles of type A, B and C, respectively. Sometimes, a vehicle type is not available for a given instance and the corresponding entry is thus empty.

In the following parameter sensitivity experiments, 5 instances are randomly selected from the classes *CE* and CE-H, and 6 instances from the classes *G* and G-H, for a total of 22 test instances.

Instance	$n-1$	m	Q	f
CE-01	50	4	160	120
CE-02	75	9	140	100
CE-03	100	6	200	140
CE-04	150	9	200	120
CE-05	199	13	200	100
CE-06	50	4	160	140
CE-07	75	9	140	120
CE-08	100	6	200	160
CE-09	150	10	200	120
CE-10	199	13	200	120
CE-11	120	6	200	180
CE-12	100	8	200	120
CE-13	120	6	200	260
CE-14	100	7	200	140
G-01	240	7	550	820
G-02	320	8	700	1060
G-03	400	8	900	1380
G-04	480	8	1000	1720
G-05	200	4	900	1620
G-06	280	5	900	1700
G-07	360	7	900	1460
G-08	440	8	900	1480
G-09	255	11	1000	60
G-10	323	13	1000	60
G-11	399	14	1000	80
G-12	483	15	1000	80
G-13	252	21	1000	60
G-14	320	23	1000	60
G-15	396	26	1000	60
G-16	480	29	1000	60
G-17	240	18	200	40
G-18	300	22	200	60
G-19	360	26	200	60
G-20	420	31	2000	60

Table 1: Homegeneous instances

		A			B			C		
Instance	$n-1$	m_A	Q_A	f_A	m_B	Q_B	f_B	m_C	Q_C	f_C
CE-H-01	50	2	160	140	2	192	168			
CE-H-02	75	4	112	80	5	168	120			
CE-H-03	100	2	160	112	2	200	140	2	240	168
CE-H-04	150	2	160	96	4	200	120	3	240	144
CE-H-05	199	7	160	80	5	200	100	2	240	120
CE-H-06	50	1	128	112	2	160	140	1	192	168
CE-H-07	75	4	112	96	3	140	120	2	168	144
CE-H-08	100	1	160	128	1	200	160	4	240	192
CE-H-09	150	4	160	96	3	200	120	3	240	144
CE-H-10	199	2	160	96	5	200	120	6	240	144
CE-H-11	120	2	160	144	2	200	180	2	240	216
CE-H-12	100	2	160	96	3	200	120	3	240	144
CE-H-13	120	1	160	208	4	200	260	1	240	312
CE-H-14	100	1	160	96	1	200	120	5	240	144
G-H-01	240	3	440	656	1	550	820	3	660	984
G-H-02	320	2	560	848	2	700	1060	4	840	1272
G-H-03	400	3	720	1104	3	900	1380	2	1080	1656
G-H-04	480	2	800	1376	4	1000	1720	2	1200	2064
G-H-05	200	2	720	1296	2	900	1620			
G-H-06	280	3	720	1360	2	900	1700	1	1080	2040
G-H-07	360	3	720	1168	1	900	1460	3	1080	1752
G-H-08	440	1	720	1184	2	900	1480	5	1080	1776
G-H-09	255	6	800	48	3	1000	60	3	1200	72
G-H-10	323	3	800	48	3	1000	60	6	1200	72
G-H-11	399	6	800	64	8	1000	80	1	1200	96
G-H-12	483	6	800	64	6	1000	80	4	1200	96
G-H-13	252	6	800	48	4	1000	60	10	1200	72
G-H-14	320	11	800	48	2	1000	60	11	1200	72
G-H-15	396	7	800	48	9	1000	60	10	1200	72
G-H-16	480	12	800	48	6	1000	60	11	1200	72
G-H-17	240	4	160	32	7	200	40	6	240	48
G-H-18	300	7	160	48	9	200	60	6	240	72
G-H-19	360	9	160	48	7	200	60	10	240	72
G-H-20	420	16	160	48	6	200	60	10	240	72

Table 2: Heterogeneous instances

4.2 Initial solution

We first evaluate the impact of the initial solution on the performance of our tabu search. To this end, the least-cost insertion *LC* and the convex hull insertion *CH* are compared using different values for their randomization parameters α (no randomization when $\alpha = 1$) and β (no randomization when $\beta = 0$), respectively. The numbers are shown in Table 3, based on 10 different restarts of the tabu search with 25,000 iterations per restart, for a total of 250,000 iterations. These numbers are average percentages over the best known solutions, taken over the 22 test instances. The first column is the initialization method, the second column *Best* is the average of the best solutions obtained over the 10 restarts and the third column *Avg* is the average over all restarts.

Although the differences between the various approaches are rather slim, the results with *CH* are superior to those obtained with *LC*. These numbers thus indicate that our tabu search is sensitive, to a certain extent, to the quality of the initial solution. The convex hull insertion method with $\beta = 0.1$ turns out to be the best and is used in the remaining experiments.

Method	Best	Avg
CH, $\beta = 0.0$	0.95%	1.30%
CH, $\beta = 0.1$	**0.62%**	**1.27%**
CH, $\beta = 0.2$	0.71%	1.35%
CH, $\beta = 0.3$	0.78%	1.44%
CH, $\beta = 0.4$	0.85%	1.41%
CH, $\beta = 0.5$	0.81%	1.45%
LC, $\phi = 1$	1.11%	1.50%
LC, $\phi = 2$	1.18%	1.86%
LC, $\phi = 3$	0.97%	1.56%
LC, $\phi = 4$	1.04%	1.63%
LC, $\phi = 5$	1.09%	1.68%

Table 3: Initialization methods

4.3 Ejection chain threshold

The local descent based on ejection chains is activated when the current local minimum is within ϵ% of the best solution found thus far. Clearly, this threshold has an important impact on the search behavior. If the threshold value is high, the ejection chain neighborhood is frequently activated, thus leading to better solutions but at a computational cost. Conversely, when the threshold value is low, solution quality suffers but the computational cost is reduced.

Table 4 shows the results obtained with different values of ϵ on the 22 test problems, using the format of the previous tables, except for the last column *CPU* which is the computation time in seconds. Note also that the ejection chain neighborhood is not activated when ϵ is equal to 0%.

ϵ	Best	Avg	CPU(s)
0.0%	0.60%	1.22%	3 577
0.5%	0.54%	1.16%	5 141
1.0%	0.45%	1.05%	8 484
1.5%	0.44%	1.02%	12 856
2.0%	0.44%	0.98%	21 344

Table 4: Activation thresholds

These results show that the integration of the ejection chain neighborhood provides a significant improvement over the two basic neighborhoods *Shift* and *Swap*. As expected, the general trend is an improvement in solution quality with larger values of ϵ, at the expense of additional CPU time. In the following experiments, $\epsilon = 1.0\%$ is favored as it looks as a good compromise between solution quality and computation time. Larger values of ϵ provide only small improvements, particularly when the best solution over 10 restarts is considered.

4.4 Number of iterations

Table 5 shows the evolution of the solution value with the number of iterations based on a single restart. As expected, large improvements are observed in the first iterations. Later on, these improvements progressively diminish until a plateau is reached. Based on these results, the number of iterations per restart is set at 50,000, as substantial improvements are observed between 25,000 and 50,000 iterations. Beyond 50,000 iterations, however, the improvements are rather modest.

4.5 Number of restarts

Table 6 now shows the results obtained by varying the number of restarts with 50,000 iterations per restart. These numbers clearly indicate the benefits of multiple restarts, although a plateau is quickly reached after approximately 10 restarts.

# iterations	Avg
5 000	1.27%
10 000	1.05%
20 000	0.92%
25 000	0.88%
30 000	0.82%
40 000	0.74%
50 000	0.68%
75 000	0.65%
100 000	0.63%

Table 5: Number of iterations (single restart)

# restarts	Best
1	0.68%
5	0.48%
10	0.40%
15	0.38%
20	0.37%

Table 6: Number of restarts

4.6 Comparison with state-of-the-art methods

In this section, a comparison is provided between the tabu search *TS* in [7], *RIP* in [4] and our tabu search *TS+*, based on the best parameter values identified in the previous subsections, namely, a solution construction using the convex hull insertion method with $\beta = 0.1$, 10 restarts with 50,000 iterations per restart and an activation threshold for the ejection chain neighborhood set to $\epsilon = 1.0\%$. Tables 7 and 8 report the detailed results on the 34 homogeneous and heterogeneous benchmark instances, respectively, including the total computation time in seconds. The solution values *Val* correspond to the best solutions obtained over 10 restarts with 50,000 iterations per restart in the case of *TS+* and *TS*. For *RIP*, the results are those reported in [4]. The rows *Avg.* summarize the results for each type of instances. For the solution values, the averages are expressed as average percentages over the best known solutions.

With regard to solution quality, *TS+* clearly outperforms both *TS* and *RIP* on all problem classes. The improvement is noteworthy on the largest instances of type G, in particular the heterogeneous ones. This is due to the ejection chain mechanism that allows multiple displacements of customers on different types of vehicles. These improvements come at a substantial computational cost, though, but this is not really critical given the static nature of the problem. It is also possible to reduce the number of restarts or the number of iterations per restart, if necessary, and still obtain high quality solutions. For example, by considering again the numbers in Table 6, we can see that *TS+* is only at 0.68% over the optimum on the 22 test instances after the first 50,000 iterations. This result is achieved in one tenth of the CPU time required by the full method based on 10 restarts. Alternately, increasing the computation time of *TS* to let it run as long as the full *TS+* method, does not lead to any significant improvement.

In [4, 7], the authors also report the best solutions ever obtained over the course of their experiments with different parameter values. Although we did not put much effort in this regard, we were able to tie the best known solutions on 12 instances with our tabu search and to improve the previously best known solutions on 32 instances over the 68 benchmark instances (as compared to 10 ties, 21 best for TS and 9 ties, 2 best for RIP).

5 Conclusion

This paper has described a tabu search for solving a vehicle routing problem where a private fleet of vehicles and a common carrier are available. The results show that the tabu search performs well and can easily reach solutions within 1% of the best known solutions. It is particularly effective on large heterogeneous instances due to the ejection chain mechanism that allows multiple displacements of customers on vehicles of different types. With regard to future developments, we would like to consider the integration of the ejection chain neighborhood within the basic tabu search neighborhoods (instead of performing only a local descent). Clearly, a filtering mechanism would be required to focus the search only on the most promising ejection chains. It would also be interesting to consider more abrupt diversification schemes to force the exploration of distant regions of the solution space.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

Instance	*TS+*		*TS*		*RIP*	
	Val	CPU(s)	Val	CPU(s)	Val	CPU(s)
CE-01	1 119.47	249	1 119.47	243	1 132.91	25
CE-02	1 814.52	339	1 814.52	330	1 835.76	73
CE-03	1 930.66	810	1 921.10	786	1 959.65	107
CE-04	2 525.17	2 006	2 525.17	1 932	2 545.72	250
CE-05	3 117.10	3 532	3 113.58	3 099	3 172.22	474
CE-06	1 207.47	252	1 207.47	255	1 208.33	25
CE-07	2 006.52	340	2 006.52	327	2 006.52	71
CE-08	2 056.59	816	2 060.17	851	2 082.75	110
CE-09	2 435.97	1 882	2 438.43	1 853	2 443.94	260
CE-10	3 401.83	3 457	3 406.82	3 111	3 464.90	478
CE-11	2 332.36	1 310	2 353.39	1 263	2 333.03	195
CE-12	1 952.86	595	1 952.86	604	1 953.55	128
CE-13	2 860.89	1 321	2 882.70	1 300	2 864.21	188
CE-14	2 216.97	642	2 216.97	650	2 224.63	110
Avg	0.25%	1 254	0.35%	1 186	0.75%	178.1
G-01	14 190.01	11 838	14 218.83	6 383	14 388.58	651
G-02	19 208.52	52 206	19 729.96	12 152	19 505.00	1 178
G-03	24 592.18	15 9404	25 653.58	22 413	24 978.17	2 061
G-04	34 802.08	55 087	36 022.73	38 339	34 957.98	3 027
G-05	14 261.31	8 478	14 673.56	8 750	14 683.03	589
G-06	21 498.03	15 911	22 278.99	14 453	22 260.19	1 021
G-07	23 513.06	55 140	24 191.41	20 528	23 963.36	1 628
G-08	30 073.56	57 290	30 627.91	30 599	30 496.18	2 419
G-09	1 325.62	8 191	1 328.14	6 110	1 341.17	832
G-10	1 590.82	17 623	1 590.83	9 388	1 612.09	1 294
G-11	2 173.80	32 843	2 172.28	14 927	2 198.45	2 004
G-12	2 495.02	85 876	2 492.75	23 097	2 521.79	2 900
G-13	2 274.12	5 045	2 278.99	3 608	2 286.91	802
G-14	2 703.31	9 769	2 705.00	6 104	2 750.75	1 251
G-15	3 161.26	19 520	3 158.92	9 248	3 216.99	1 862
G-16	3 638.39	46 751	3 639.11	13 137	3 693.62	2 778
G-17	1 633.35	4 720	1 636.11	4 026	1 701.58	806
G-18	2 710.21	8 920	2 705.90	6 220	2 765.92	1 303
G-19	3 497.72	14 184	3 497.54	10 125	3 576.92	1 903
G-20	4 306.89	24 763	4 311.17	13 689	4 378.13	2 800
Avg	0.56%	34 678	0.92%	13 665	1.29%	1 655.5

Table 7: Homogeneous instances

Instance	*TS+*		*TS*		*RIP*	
	Val	CPU(s)	Val	CPU(s)	Val	CPU(s)
CE-H-01	1 191.70	260	1 191.70	257	1 192.72	26
CE-H-02	1 791.21	348	1 795.51	337	1 798.26	72
CE-H-03	1 917.96	808	1 926.33	790	1 934.85	105
CE-H-04	2 481.68	1 985	2 481.64	1 956	2 493.93	251
CE-H-05	3 143.01	3 424	3 143.92	2 959	3 195.66	490
CE-H-06	1 206.82	251	1 206.82	254	1 210.23	25
CE-H-07	2 031.85	320	2 035.90	325	2 042.79	74
CE-H-08	1 986.51	845	1 991.23	814	2 015.72	112
CE-H-09	2 447.58	1 930	2 445.49	1 889	2 445.88	267
CE-H-10	3 272.37	3 424	3 271.70	3 095	3 304.69	482
CE-H-11	2 336.51	1 339	2 325.74	1 270	2 308.76	188
CE-H-12	1 915.05	604	1 912.47	607	1 908.74	130
CE-H-13	2 868.13	1 365	2 872.14	1 250	2 842.18	195
CE-H-14	1 907.75	672	1 925.46	658	1 920.36	114
Avg	0.41%	1 255	0.53%	1 176	0.70%	180.8
G-H-01	14 194.13	11 937	14 174.27	6 425	14 408.31	647
G-H-02	18 537.70	49 553	19 056.69	12 849	18 663.15	12 54
G-H-03	25 177.92	119 962	25 299.93	22 587	25 561.55	2 053
G-H-04	34 991.21	40 791	35 388.06	40 957	35 495.66	2 904
G-H-05	15 411.82	7 542	15 895.94	7 996	16 138.50	512
G-H-06	19 859.30	19 541	20 381.35	13 569	20 329.04	1 005
G-H-07	23 481.28	91 673	23 915.77	22 625	24 184.83	1 608
G-H-08	27 334.84	186 252	27 521.28	34 082	27 710.66	2 584
G-H-09	1 329.27	18 293	1 331.11	5 927	1 346.03	814
G-H-10	1 555.59	15 643	1 554.96	10 871	1 575.82	1 332
G-H-11	2 195.83	32 079	2 191.23	14 455	2 218.91	2 140
G-H-12	2 482.92	42 240	2 535.00	21 083	2 510.07	2 970
G-H-13	2 237.38	18 013	2 231.88	4 058	2 253.45	733
G-H-14	2 684.70	10 429	2 685.51	6 303	2 711.81	1 246
G-H-15	3 127.33	21 112	3 123.60	9 766	3 156.93	1 895
G-H-16	3 621.85	62 174	3 853.21	15 712	3 649.09	2 785
G-H-17	1 664.08	5 226	1 674.91	3 969	1 705.48	762
G-H-18	2 708.73	11 325	2 722.32	6 177	2 759.99	1 299
G-H-19	3 443.59	16 271	3 445.85	8 418	3 517.48	1 892
G-H-20	4 314.16	29 698	4 306.53	13 922	4 413.82	2 733
Avg	0.30%	40 488	1.18%	14 088	1.25%	1 658.4

Table 8: Heterogeneous instances

References

[1] Ahuja, R.K., T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications. Prentice-Hall: Englewood Cliffs, New Jersey, 1993.

[2] Bienstock D., M.X. Goemans, D. Simchi-Levi, D. Williamson, "A Note on the Price Collecting Traveling Salesman Problem", *Mathematical Programming 59*, 413–420, 1993.

[3] Bolduc M.-C., J. Renaud, F. Boctor, "A Heuristic for the Routing and Carrier Selection Problem", Technical Report 2006-006, Faculté des Sciences de l'Administration, Université Laval, Québec, Canada, 2006.

[4] Bolduc M.-C., J. Renaud, F. Boctor, G. Laporte "A Perturbation Metaheuristic for the Vehicle Routing Problem with Private Fleet and Common Carriers", *Journal of the Operational Research Society 59*, 776–787, 2008.

[5] Christofides N., A. Mingozzi and P. Toth. "The vehicle routing problem". In *Combinatorial Optimization*, N. Christofides, A. Mingozzi, P. Toth and C. Sandi, eds., Wiley, Chichester, UK, 315–338, 1979.

[6] Chu C.-W., "A Heuristic Algorithm for the Truckload and Less-Than-Truckload Problem", *European Journal of Operational Research 165*, 657-667, 2005.

[7] Côté J.-F., J.-Y. Potvin, "A Tabu Search Heuristic for the Vehicle Routing Problem with Private Fleet and Common Carrier", forthcoming in *European Journal of Operational Research*, 2009.

[8] Diabi M. and R. Ramesh, "The Distribution Problem with Carrier Service: A Dual Based Penalty Approach", *ORSA Journal on Computing 7*, 24-35, 1995.

[9] Glover, F., "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems", *Discrete Applied Mathematics 65*, 223–253, 1996.

[10] Glover F. and M. Laguna, Tabu Search, Kluwer, Boston, 1997.

[11] Golden B.L., E.A. Wasil, J.P. Kelly, I.-M. Chao, "The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Tests and Computational Results", in T.G. Crainic, G. Laporte (Eds.), *Fleet Management and Logistics*, Kluwer, Boston, 33-56, 1998.

[12] Lin S., "Computer Solutions of the Traveling Salesman Problem", *Bell System Technical Journal 44*, 2245–2269, 1965.

[13] Noon C.E., J.C. Bean, "A Lagrangian Based Approach for the Asymmetric Generalized Traveling Salesman Problem". *Operations Research 39*, 623–632, 1991.

[14] Renaud J., F.F. Boctor, G. Laporte, "A Fast Composite Heuristic for the Symmetric Traveling Salesman Problem", *ORSA Journal on Computing 8*, 134–143, 1996.

[15] Taillard É.D., "Parallel Iterative Search Methods for Vehicle Routing Problems", *Networks 23*, 661–673, 1993.

[16] Volgenant T., R. Jonker, "On Some Generalizations of the Travelling-Salesman Problem", *Journal of the Operational Research Society 38*, 1073–1079, 1987.