POPMUSIC FOR THE WORLD LOCATION ROUTING PROBLEM

ADRIANA C.F. ALVIM AND ÉRIC D. TAILLARD

ABSTRACT. POPMUSIC — Partial optimization metaheuristic under special intensification conditions — is a template for tackling large problem instances. This template has been shown to be very efficient for various combinatorial problems like p-median, sum of squares clustering, vehicle routing and map labeling. In terms of algorithmic complexity, one of the most complex part of POPMUSIC template is to find an initial solution. This article presents a method for generating an appropriate initial solution to the location routing problem by producing in $O(n^{3/2})$ an approximate solution to the capacitated *p*-Median problem. The method is tested on *LRP* instances with millions of entities.

1. INTRODUCTION

Location Routing Problems are well known optimization problems. The reader is referred to [8] for a survey. This reference reports that practical location-routing problems may involve thousands of entities that have to be serviced from a set of hundreds of depots. However, high-quality heuristics like GRASP, GRASP with Path Relinking [12], Memetic Algorithms with Population Management [11], Granular Taboo Search [13] treats small instances including few hundreds of entities and a dozen of potential depot locations. Even clustering approaches like [2] are not considering larger instances.

The goal of this paper is to show that much larger instances can be treated by embedding these high-quality heuristics in POPMUSIC template.

Several versions of location routing problems exist, and we have considered the following one in order to generate large and freely available instances based on the World *TSP* [19]. Let N be a set of n customers (or entities). Each customer has a non-negative demand q_i. It is supposed that a function $d(i, j)$ measuring the distance between customers i and j is available. Set N is also the set of n possible depot locations with unlimited capacity each and where is based an unlimited vehicle fleet, each vehicle having a given capacity Q. The opening cost of a depot is D. The cost of a route is the length of traversed edges. The objective is to find which depots should be opened and which routes should be constructed to minimize the total cost (depots opening cost plus routes cost) such that:

- Each demand q_i must be served by a single vehicle.
- Each route starts and ends at the same depot.
- The total demand of a route does not exceed Q.

The location routing problem (*LRP*) considered in this article is a simplified version of a real problem that was submitted to us several years ago. A company producing potato chips has the policy of verifying the freshness of its products and refurnishing itself the shelfs in the selling points. The number of selling points in Switzerland is more than 32'000. The stores are refurnished with small vehicles that carry the goods from local depots. Any room that can be rented can be used as a depot. The depots are refurnished directly from the production places with

Key words and phrases. Location routing, Meta-heuristics, POPMUSIC, *TSP*, *VRP*, *p*-Median, Large-Scale Optimization, Taboo Search.
Date: March 8, 2013.

large trucks, but this is done independently from the *LRP*. To simplify the problem, we consider that each selling point can be used as depot (with a given renting cost). In practice, it is observed that a vehicle can service few dozen of selling points before turning back to the depot. These simplifications allow to work on problem instances generated on public data whose distribution is not too far from real life. The aim of the present work is to show the pertinence of POPMUSIC approach for dealing with large instances (more than 10^6 entities). The approach presented in this paper can also be used for solving large problem with additional complications, provided that an optimization method for solving small instances is available.

The paper is organized as follows: Section 2 recalls the POPMUSIC template and show how it can be used in the context of the *LRP*. The main advantage of POPMUSIC approach is that the algorithmic complexity for optimizing a given initial solution is typically linear with the problem size. Such a complexity is lower than that of other approaches specifically designed for limiting the algorithmic complexity of high-quality heuristics. For instance, the approach of [20] for the Vehicle Routing Problem, which is a sub-problem of the *LRP*, includes a step with quadratic algorithmic complexity. The memory requirement of this approach is also quadratic, meaning that problem instances with, let us say, 10^5 entities cannot be solved.

The main difficulty in implementing a POPMUSIC approach for large instances is to get an initial solution of adequate quality with a complexity that is lower than $O(n^2)$. For instance, the extended Clarke & Wright constructive algorithm used in [12] has a complexity in $O(n^3 m)$, where m is the number of potential depot site ($m = n$ for the instances treated in this article). Section 3 presents a technique based on solving approximatively a p-median problem with capacity for getting an initial solution to the *LRP* in $O(n^{3/2})$.

Computational results are presented in Section 4. Concluding remarks and future research avenues are presented in the last section.

2. POPMUSIC TEMPLATE

Our first works on POPMUSIC method start in the beginning of the 90's [14] with an application to the vehicle routing problem with capacity (*VRP*). In this reference, we used the fact that a subset of *VRP* tours is also a *VRP*. By decomposing a given *VRP* solution into subsets of independent tours, and solving these subproblems in parallel, it is possible to find excellent solutions to *VRP* instances with few hundreds of customers. [15] shows that a similar principle can be applied to centroid clustering problems (sum of squares clustering, *p*-Median, multi-source Weber problem). The size of the instances considered in this reference is few order of magnitude larger: several thousands of entities and thousands of centers. The implementation discussed in [15] has a complexity growing quadratically with the number of centers and quasi-linearly with the number of entities. This complexity remains however quadratic with the number of entities for general *p*-Median instances with data given by a $n \times n$ matrix. Later, in [16], the POPMUSIC template was formalized. Algorithm 1 presents the POPMUSIC template.

The basic idea of POPMUSIC is to locally optimize sub-parts of a solution, once a solution of the problem is available. These local optimizations are repeated until no improvements are found. Let us suppose that a solution S can be represented as a set of disjoint *parts* $s_1, \ldots, s_q$. Let us also suppose that a distance measure can be defined between two parts. The central idea of POPMUSIC is to select a part s_s, called *seed part*, and a number $r < q$ of the closest parts from the seed part s_s to form a subproblem called R. If parts and subproblems are defined in

Algorithm 1: `POPMUSIC` template

Data: Initial solution S composed of q disjoint parts $s_1, \ldots, s_q$
Result: Improved solution S

```
1  U = s_1, ..., s_q;
2  while U ≠ ∅ do
3      Select s_s ∈ U // s_s: seed part;
4      Build a subproblem R composed of the r parts of S which are the closest
       to s_s;
5      Try to optimize R;
6      if R has been improved then
7          Update solution S;
8          Remove from U the parts not belonging to S anymore;
9          Include in U the parts of optimized subproblem R;
10     else // R not improved
11         Remove s_s from U;
```

an appropriate way, an improvement in the solution of the whole problem can be found for every improvement in the subproblem.

To avoid generating twice the same subproblem, a set U of parts is stored. U contains the seed parts that can be used to define a subproblem that can potentially improve the solution. Once U is empty, all subproblems have been examined without success and the process stops. If subproblem R has been successfully improved, a number of parts from $s_1, \ldots, s_q$ have been changed and further improvements may be found in the neighborhood of these parts. In this case, all the parts used for building R are inserted in U before continuing the process.

Potentially, the complexity of this template can be very high since set U is not reduced at each iteration. However, several implementations [1, 15, 18] shown empirically that the number of iterations of an algorithm based on this template grows quasi-linearly with q.

To translate this template into a (pseudo-) code for a given problem, several choices must be made:

- The procedure for obtaining the initial solution
- The definition of a part
- How a seed-part is selected
- The definition of the distance between parts for creating a subproblem
- The improvement procedure for optimizing subproblems

For the *LRP*, various definitions for a part can be imagined:

- An entity
- A tour (with all entities constituting the tour)
- A depot (with all tours and entities assigned to the depot)

In the present work, we choose to decompose a solution into tours. So, a part is a tour. The distance between two parts is the minimal distance between two entities belonging to different tours. In order to filter the number of candidate parts, only the tours connected to the r depots that are the closest from the seed part are considered.

The set U is managed as a stack, this means that the last part that has been introduced in U will be the next to be chosen as seed-part. In the present work, we have also tried to select the seed-part randomly in U. The improvement procedure for the subproblems, which are therefore multi-depot *VRP* (*MDVRP*), is based on

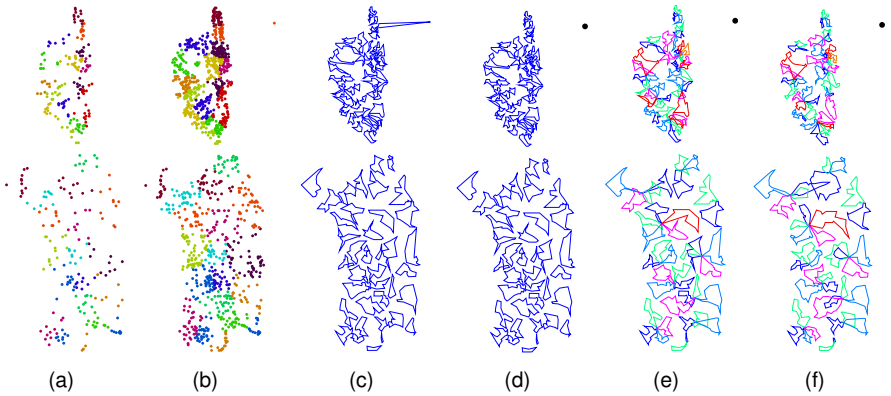


FIGURE 1. Main phases of the proposed method illustrated on a small instance including entities on Corsica and Sardinia Islands. 1(a) Finding superclusters on a sample of entities. 1(b) Finding superclusters on all entities. 1(c) Decomposition of supercluster into clusters and building *TSP* on clusters. 1(d) Splitting large *TSP* tours. 1(e) Merging tours and creating a feasible *MDVRP* solution. 1(f) Opimizing *MDVRP* solution with POPMUSIC.

taboo search. The last performs t iterations for a *MDVRP* containing t entities. The moves considered in this taboo search are the swap of two entities belonging to different tours or the move of an entity from one tour to another. In some situations, this kind of move allows to reduce the number of tours (consequently the number of vehicles used). The implementation of this taboo search follows the lines of those used in [14]. The complexity of this implementation is $O(t^3)$.

The only parameter of POPMUSIC is r, the number of parts put together to create a subproblem. The value $r = 6$ has been chosen for all experiments in this paper. This value is a good trade-off between the ability of taboo search to find good solutions in a reduced computational effort. Indeed, if subproblems are too small, improvements cannot be found and if the subproblems are too large, their improvements can take a prohibitive computational effort.

Since r is chosen independently from the problem size and since a quasi-linear number of POPMUSIC iterations is observed, the body of POPMUSIC is also quasi-linear. So, the most difficult portion of an adaptation of POPMUSIC to the *LRP* is to get a feasible initial solution of decent quality. In terms of algorithmic complexity, the initialization is the most complex part. The next section shows how to generate a feasible initial solution of decent quality in $O(n^{3/2})$.

3. EFFICIENT GENERATION OF A SOLUTION TO THE *LRP*

Since the instances considered in this article are geometrical ones (coordinates on the earth ellipsoid), it would have been possible to use computational geometry algorithms for building a feasible *LRP* solution, for instance with the help of a Delaunay triangulation. The last can be obtained in $O(n\ logn)$. We chose a more general method, where we make the hypothesis that we only have a function $d(i, j)$ providing the distance between entities i and j. In this work, we use the Euclidean distance $d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$.

Algorithm 2 has been designed for finding a feasible *LRP* solution that is suitable for initial POPMUSIC solution. The main phases of this algorithm are illustrated on Figure 1.

The most difficult point of this algorithm is the implementation of a heuristic procedure for finding good solutions to the capacitated p-median problem (*CPMD*).

Algorithm 2: Generating a feasible *LRP* solution for large instances

Data: n entities with quantity q_i and distance measure $d(i,j)$, $i,j = 1,\ldots,n$
Result: Feasible *LRP* solution

`// Sampling`

Take a sample E of $20\sqrt{n}$ entities (Figure 1(a));

`// Center location for super-clusters`

Solve a relaxation of a *p*-Median with capacity (*CPMD*) on E with $p = \sqrt{n}$;

`// Super-cluster building` (Figure 1(b))

Assign all n entities to the closest among p centers found at previous step ;

`// Clusters building`

for $k = 1,\ldots,\sqrt{n}$ **do**

  Decompose super-cluster SC_k into $p_k = \lceil \sum_{i \in SC_k} q_i / T \rceil$ clusters by solving a relaxation of a *CPMD*

`//` *`TSP`*`building` (Figure 1(c))

for *all clusters obtained at previous step* **do**

  Find a traveling salesman tour on the cluster entities

`// Splitting large` *`TSP`* (Figure 1(d))

for *all tours found at previous step with length larger than* D **do**

  Decompose the entities of the tour into $1, 2, \ldots$ clusters by solving *CPMD*s.

  Find a *TSP* tour for each cluster and for each decomposition.

  Retain the decomposition for which the sum of *TSP* lengths $+ d \cdot D$ is the lowest, where d is the number of clusters in the decomposition

`// Locating depots`

Open a depot for each tour (at the position of *CPMD* center);

repeat

  for *each depot* **do**

    Try to merge the depot with a depot at distance less than D; the tours attached to the removed depot are attached to the new depot position with a cheapest insertion rule. The merging of 2 depots is done in a greedy manner, if the resulting cost is diminished.

until *No merge with cost decrease exists*;

`// Feasible` *`MDVRP`* (Figure 1(e))

Cut each tour for which the sum of quantities is larger than Q (vehicle capacity) into 2 tours;

The *CPMD* can be formulated as follows, where x_{ij} are variables indicating if entity i is assigned to center j ($x_{ij} = 1$) or not ($x_{ij} = 0$):

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d(i,j)\cdot x_{ij}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in \{1,\ldots,n\} \tag{1}$$

$$x_{ij} \leq x_{jj} \quad \forall i,j \in \{1,\ldots,n\} \tag{2}$$

$$\sum_{j=1}^{n} x_{jj} = p \tag{3}$$

$$\sum_{i=1}^{n} q_i \cdot x_{ij} \leq Q \quad \forall j \in \{1,\ldots,n\} \tag{4}$$

$$x_{ij} \in \{0,1\} \quad \forall i,j \in \{1,\ldots,n\} \tag{5}$$

The objective is to minimize the star distances (sum of the distance of all entities to their allocated center). Set of constraints (1) ensures that an entity is allocated to a single center. Set of constraints (2) ensures that an entity cannot be allocated to a center that is not opened. Note that variable x_{jj} indicates if entity j is considered as a center. Constraint (3) ensures that exactly p centers are opened. Set of constraints (4) ensures that a center does not accept a demand larger than its capacity. Note here that all centers have the same capacity Q.

Finding a feasible solution to the *CPMD* is NP-complete, since the bipartition problem can be polynomially transformed into *CPMD*. In a first phase, solutions violating slightly constraint (4) can be admitted since the vehicle capacity limitation is satisfied by splitting overloaded tours in step 17 of Algorithm 2. To avoid splitting too many tours and a degradation of *LRP* solution, we suggest to replace vehicle capacity Q in constraint (4) by a target value $T \leq Q$. This value is a parameter of the method and allows to take into consideration the variance of the demands. To find a solution slightly violating constraints (4), we consider a Lagrangean relaxation of *CPMD*:

$$\text{maximize}_{\lambda\geq 0}\text{minimize}_{\,x} \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d(i,j)\cdot x_{ij} + \sum_{j=1}^{n} \lambda_j \cdot \Big(\sum_{i=1}^{n} q_i \cdot x_{ij} - T\Big) \tag{6}$$

$$\text{subject to} \quad (1)\ (2)\ (3)\ (5)$$

A gradient method can be used to find good λ's values, by adjusting the value of λ_j as follows: $\lambda_j \leftarrow \max(0, \lambda_j + \ell(\sum_{i=1}^{n} q_i \cdot x_{ij} - T))$, where ℓ is the step length. In other words, penalty λ_j is increased if constraint (4) is violated for center j and it is decreased (without taking negative values) if center j has still additional capacity. Step length ℓ must be proportional to distance unit and inversely proportional to capacity unit.

To find good solutions to (6), we propose Algorithm 3. This algorithm alternates the allocation of entities to centers, the centers repositioning and the λ's update.

Line 2 of Algorithm 3 introduces variable f which is used for setting gradient step length. Initially, the step length ℓ is given by the average distance of entities to centers divided by target cluster capacity. This length is diminished at each iteration by multiplying f by 0.99.

Preliminary computational experiments showed that the solution is stabilized after few hundreds of iterations: All entities are allocated to their nearest center (distance + penalty), the centers are positioned at best among the entities and the λ's are not changing enough for modifying the entities allocations. So, we decided

Algorithm 3: Generating an almost feasible solution to the *CPMD*

Data: n entities with quantity q_i, $(i = 1, \ldots, n)$, p, T
Result: Allocation variables $\mathbf{x}^*$: p clusters with quantities close to T

1 Randomly choose p centers among the n entities
2 $f = 1.0$; $\lambda_j = 0; j = 1 \ldots, n$;
3 **for** *150 iterations* **do**
4 Allocate entities to the closest center (with penalty; update x_{ij} variables);
5 **for** *Each of the* p *clusters* **do**
6 Find the best position of the center among cluster entities (update x_{jj} variables)
7 **if** *Current solution* $\mathbf{x}$ *is better than* $\mathbf{x}^*$ **then**
8 $\mathbf{x}^* \leftarrow \mathbf{x}$
9 $f \leftarrow 0.99 \cdot f$;
10 **for** $j = 1, \ldots, p$ **do**
11 $\lambda_j \leftarrow \max\left(0, \lambda_j + f \cdot \frac{1}{n}(\sum_{i=1}^{n}\sum_{j=1}^{n} d(i,j) \cdot x_{ij}) \cdot \frac{1}{T}(\sum_{i=1}^{n} q_i \cdot x_{ij} - T)\right)$

to perform a total of 150 iterations. At Line 7, a solution is considered to be better if it diminished the number of clusters for which the sum of quantities is larger than Q, or, for the same number of cluster capacity violations, if the sum of distances is diminished.

As the results of Algorithm 3 depends on a random positioning of the centers, the last is executed 5 times and the best of 5 returned solutions is retained. Finally, for diminishing the number of clusters for which the allocated quantity is larger than Q, a local search is applied. This local search transfers repeatedly an entity from an overloaded cluster to its second nearest center, if the last has enough capacity for accepting this entity. The local search stops when no improving move is found.

3.1. **Complexity analysis of Algorithm 3.** Applied to a *CPMD* instance with n entities and p centers, the complexity of Algorithm 3 is $O(n \cdot p + n^2/p)$. Indeed, Step 4 can be trivially implemented in $O(n \cdot p)$ and Step 6 in $O(n^2/p^2)$ since each cluster has n/p entities on average. The complexity of all the other steps is lower.

3.2. **Complexity analysis of Algorithm 2.** Once a *CPMD* procedure is available, finding an initial *LRP* solution of adequate quality is relatively easy, but the size of subproblems to solve must be carefully chosen in order to maintain the complexity as low as possible. The sampling of entities in Step 1 is not there for diminishing the global complexity of Algorithm 2, but for speeding it up. Finding the positions of supercluster centers consists in calling Algorithm 3 with $O(\sqrt{n})$ entities and $\sqrt{n}$ centers. The complexity of Step 2 is therefore $O(n)$. Without sampling, this phase is in $O(n^{3/2})$ and would take most of the computational time.

The assignment of all n entities to the nearest of $\sqrt{n}$ (super-)center in Step 3 can be trivially implemented in $O(n^{3/2})$.

The decomposition of super-clusters into clusters in Step 4 consists in calling $O(\sqrt{n})$ times Algorithm 3 with $O(\sqrt{n})$ entities and $\sqrt{n}$ centers. Therefore, the complexity of finding $O(n)$ clusters respecting approximatively the capacity Q of the vehicles can be done in $O(n^{3/2})$.

For a given value of Q, the maximal number of entities per tour is also fixed. This means that finding all *TSP* tours and eventually splitting long tours (Steps 6 and 8) can be done in $O(n)$.

TABLE 1. Characteristics of the problem instances considered in this paper. Entities are chosen among those of the World TSP data.

Instance number	Number of entities	Minimal longitude	Maximal longitude	Minimal latitude	Maximal latitude
1	17,237	-5	15	20	40
2	46,750	-5	15	30	45
3	113,193	15	30	40	50
4	115,858	-5	15	40	50
5	260,374	-5	30	30	50
6	1,904,711	-180	180	-90	90

The complexity of Step 16 (merging depots) depends on depot opening cost D. It is useless to merge two depots at a distance larger than D. Making the hypothesis that, for all depots, there is a constant number of other depots at distance lower than D, Step 16 can be implemented with a complexity lower than $O(n^2)$. However, if the opening cost is very high, this step could degenerate in $O(n^2)$. If the number of depots is low (in $O(\sqrt{n})$), another depot positioning strategy should be adopted, for instance by solving a *p*-Median problem. Such an approach has been suggested in [13]. However, the instances proposed in this work have a value of D relatively limited and we observed a computational effort for this step that is less than 1.2% of global computational effort.

Finally making the initial solution feasible in Step 17 can be trivially implemented in $O(n)$. So, the complexity of finding an initial solution to the *LRP* is in $O(n^{3/2})$. This complexity is empirically verified by numerical experiments in the next section.

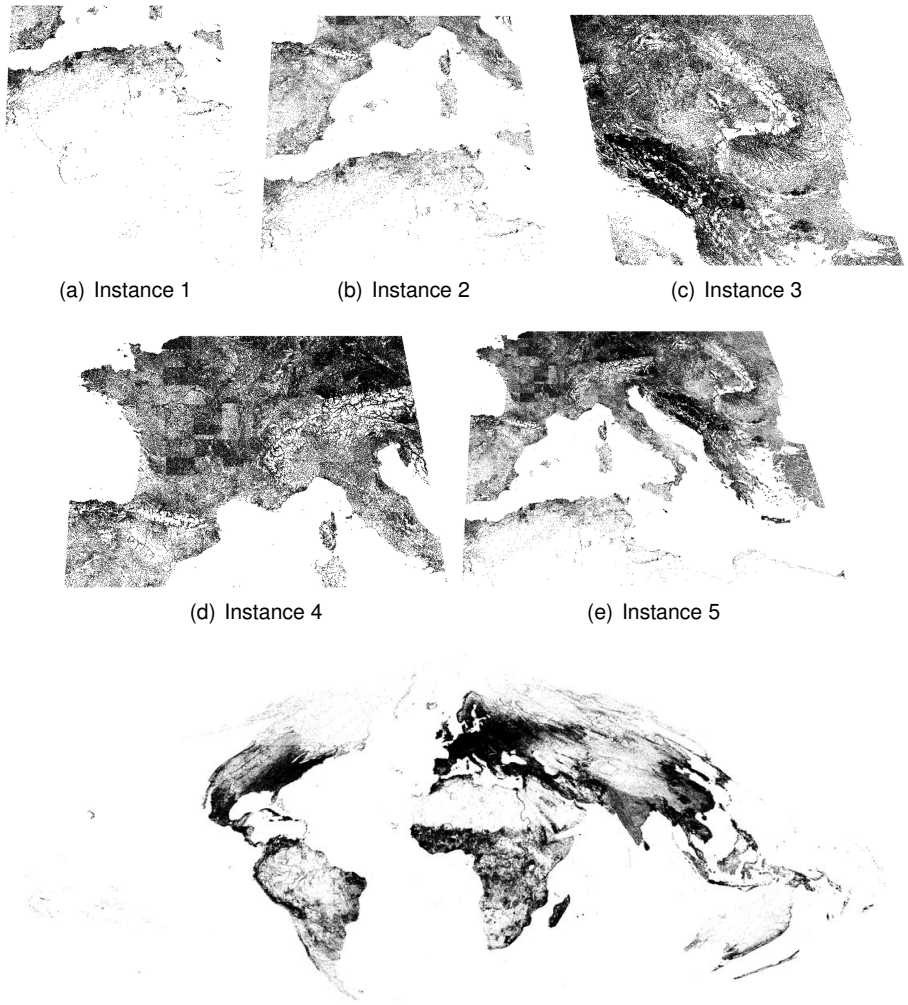
4. NUMERICAL RESULTS

Since there are no large size *LRP* instances publicly available, we have generated new benchmarks on the base of the World TSP data [19]. 6 different entities sets have been considered, by taking various windows (longitude, latitude) among the entities of the World TSP. Table 1 gives the main characteristics of these 6 instances which are illustrated in Figure 2.

Each city i of the TSP World instance is specified by its longitude λ_i and its latitude θ_i, given in degrees and decimal form. Our world *LRP* instance is specified by its Euclidean x_i, y_i and z_i positions (expressed in meters) computed with the Geodetic Reference System (GRS80 [4]):
$x_i = r_i * cos(\lambda_i) * cos(\theta_i)$
$y_i = r_i * sin(\lambda_i) * cos(\theta_i)$
$z_i = r_i * (1.0 - e^2) * sin(\theta_i)$
where $r_i = 6378137.0/\sqrt{1.0 - e^2 * sin^2(\theta_i)}$, and $e^2 = 0.00669438$.

For testing the sensitivity of our method with respect to the type of demands for the entities, two types of problem instances were considered. The first type of instances (unit demands) have a uniform demand of 1 for all entities. The second type of instances (variable demands) have demand q_i for each entity that is generated using a pseudo-random generator $q_i = (107 \cdot \lfloor |x_i| \rfloor + 97 \cdot \lfloor |y_i| \rfloor + 163 \cdot \lfloor |z_i| \rfloor) \mod 29 + 1$. For the second type, we can consider that the quantities are randomly, uniformly generated between 1 and 29 with an average of 15.

For testing the sensitivity of our method with respect to the vehicle capacity, we have considered $Q = 10; 20; 40$ for unit demand instances and $Q = 150; 300; 600$ for variable demand instances (so, a vehicle tour visits 10, 20 or 40 entities on average).



(a) Instance 1 (b) Instance 2 (c) Instance 3

(d) Instance 4 (e) Instance 5

(f) Instance 6

FIGURE 2. The 6 regions of the world considered in this paper for creating new *LRP* benchmarks

Finally, for testing the sensitivity of our method with respect to the depot opening cost, we have considered $D = 50,000; 100,000; 200,000$.

4.1. **Parameter settings.** Since a method based on `POPMUSIC` embeds several algorithms, we provide here the various choices we made for the options and parameters used.

4.1.1. *`POPMUSIC` body.*

- A part: a vehicle tour
- Distance between parts: minimal distance between 2 entities belonging to different tours
- Size of *MDVRP* subproblems: $r = 6$
- *MDVRP* improvement procedure: Taboo search
- U set is managed as a stack

4.1.2. *Taboo search for* MDVRP*.* The taboo search used for optimizing *MDVRP* instances follows the lines of [14].

- Moves: Transfer an entity from one tour to another or exchange 2 elements belonging to different tours
- Number of iterations: t for an instance with t entities
- Taboo status: moving again an entity
- Taboo duration: $0.5 \cdot t \cdot u^3 + 0.1t$ where u is a random number uniformly distributed between 0 and 1
- Aspiration criterion: a taboo move improves the best solution or a transfer move empties a tour

4.1.3. CPMD*.*

- Number of iterations (allocation — λ modification — relocation): 150
- Gradient step size decrease: 0.99
- Target capacity T for the clusters: $T = Q$ for unit demand instances and $T = Q - 10$ for variable demand instances.
- The procedure is repeated 5 times and the best solution is taken

4.2. **Analysis of results.** We show the sensitivity of our method with respect to the instances, parameter settings and options by considering a standard problem: The demands are variable, $Q = 300$ and $D = 100,000$. Additional computational results on different problem instances, obtained by varying the values of Q, D and demands are provided in Section 4.7. The method was implemented in C, compiled under Windows 7 with `gcc` compiler and `-O2` option. Only one core of Intel i7 (930, 2.83GHz) was used. The memory used by our implementation never exceed 1.5Gb for the largest instance. Table 2 provides for all 6 instances of different size the following informations and numerical values observed.

- The number of entities
- A lower bound on the number of vehicles ($\lceil \sum q_i/Q \rceil$)
- The number of clusters created
- The total length of *TSP* tours, before splitting
- The number of *TSP* obtained after splitting long tours
- The number of depots remaining after merging
- The total length of tours of the initial feasible *MDVRP* solution
- The total length of tours after improvement with `POPMUSIC`
- The number of vehicles used in *MDVRP* improved solution
- Total cost of *MDVRP* improved solution (which is equal to total length plus number of depots times depot cost)
- A lower bound of optimal total cost
- The CPU time for solving the *CPMD* (superclusters + clusters)
- The CPU time for `POPMUSIC` optimization
- The total CPU time

	Instance					
	1	2	3	4	5	6
Problem size n	17,237	46,750	113,193	115,858	260,374	1,904,711
Vehicles lower bound	862	2,332	5,670	5,790	13,024	95,238
Number of clusters	961	2,520	6,034	6,157	13,728	99,220
TSP length	100,382,746	196,926,915	295,442,154	334,179,645	782,337,809	9,713,446,886
Number of *TSP* tours	1,005	2,539	6,036	6,162	13,760	100,581
Number of depots	291	621	1,143	1,226	2,774	25,080
Length *MDVRP*	98,183,526	217,103,691	344,836,963	386,390,737	895,446,441	10,388,058,905
Length `POPMUSIC`	91,506,754	200,080,586	315,839,461	353,454,191	818,078,899	9,463,512,294
Number of vehicles	996	2,543	6,047	6,176	13,829	102,504
Total cost	120,606,754	262,180,586	430,139,461	476,054,191	1,095,478,899	11,971,512,294
Lower bound	67,098,741	142,180,289	217,665,536	248,768,006	568,212,071	6,613,228,345
Time clusters	7	21	63	61	172	2,497
CPU `POPMUSIC`	362	1,093	3,034	3,108	6,712	44,553
Total time	369	1,116	3,102	3,174	6,897	47,598

TABLE 2. Results for standard problem instances (Demand uniformly distributed between 1 and 29; $Q = 300$; $T = 290$; $D = 100,000$). CPU times in seconds on one Intel i7 core.

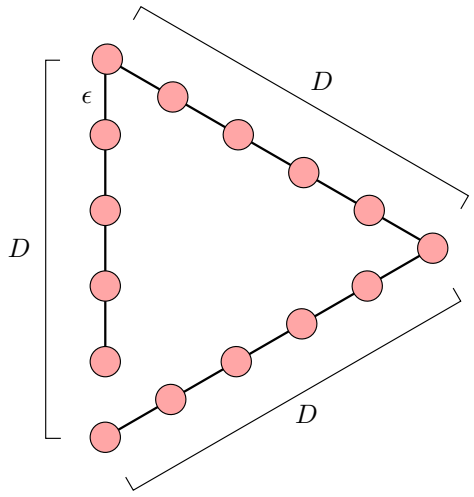


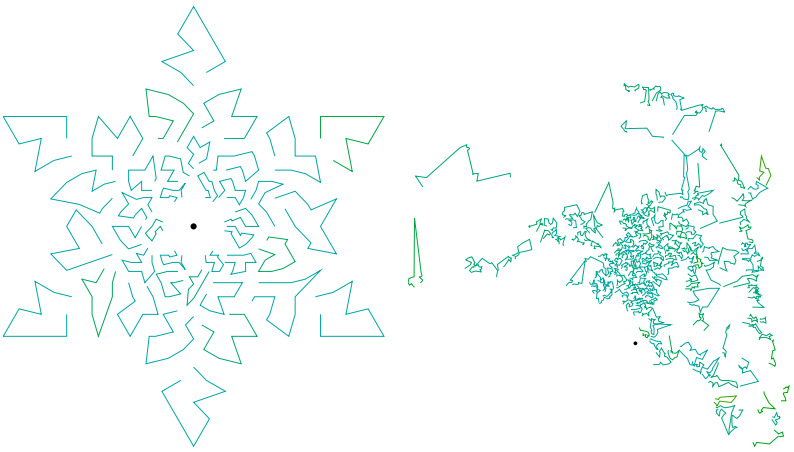
FIGURE 3. A *LRP* instance for which a minimum spanning tree is, asymptotically, a lower bound to the optimal solution. The weight of the spanning tree is $3D - \epsilon$ while the length of an optimal *LRP* solution is $3D$. At least one depot must be opened, for a cost of D. So, the lower bound is $4D - \epsilon$ while optimum *LRP* cost is $4D$.

Since these large instances are new, it is difficult to have an idea about the quality of solutions obtained. Indeed, good lower bound are hard to find for theses large instances, given that classical mathematical models embeds 3-index variables. The purpose of this article is not to study lower bounds for large *LRP*s. A simple lower bound can be obtained by computing a spanning tree MST of minimum weight. Then, all the edges of MST that are larger than depot opening cost D are removed to obtain a forest F. A depot is opened for each connected component of F. The lower bound is given by the cost of the edges of F plus depot opening costs. Such a lower bound is tight for special instances, as shown in Figure 3, where the lower bound is $4D - \epsilon$ and the optimal *LRP* solution has a value of $4D$. The lower bound is also tight for instances with all entities separated by a distance larger than D. In this case, F has no edges. Without making assumptions on the distance measure between entities, the computation of a lower bound is at least in $\Omega(n^2)$. For the largest instance, the time for computing the lower bound given in Table 2 is higher than the CPU time for `POPMUSIC` optimization. The quality of the lower bounds is certainly very bad.

Let us mention however, that the length (line "Length POPMUSIC") of the solution for Instance 6 (corresponding to the world *TSP* data) is about 26% above the length of optimal *TSP* tour. This length is neither an upper bound to the *LRP* optimum length (since there is a higher number of travel in a *LRP* solution due to the capacity constraints implying additional travels to the depots) nor a lower bound (since the *TSP* tour may have travels larger than D, the opening cost of a depot). For instance, for $D = 15,000$ (about 4 times the average travel length between 2 *TSP* cities), the total length is lower than the optimum *TSP* length.

In this table, we see that the initial *MDVRP* solution is improved by about 10% by `POPMUSIC`. The number of vehicles in the final solution is 6.1% to 15.5% above the minimum number of vehicles. Knowing that several depots only service 1 or 2 entities (in sparse regions), an average vehicle load above 90% means that the vehicle tours are very constrained by their capacity. Even if the upper bounds are near 2 times the value of the lower bounds, we are confident that these results are

not too bad, after having made the following experiment: We took few of the largest academic *VRP* instances publicly available, i.e. those proposed by [6, 20] and we observed the solutions quality obtained with the following method: The entities are first clustered with our *CPMD* approach (trying different T capacity targets for having sets of entities not larger than vehicle capacity Q) and a *TSP* is solved for each cluster (+ the unique depot). For these instances with 420 and 3,000 entities, the value of the feasible *VRP* solutions observed were typically 8% to 10% above the best solutions known, reported in [7, 20]. The computational time was less than 0.4 seconds for the instance with 420 entities and about 10s for the instances with 3,000 entities. For these instances, the best solutions known are 117% and 1010% above the lower bound proposed above. Figure 4 illustrates the solutions obtained with this experiment.



(a) Largest instance of [6] (b) First instance of [20]

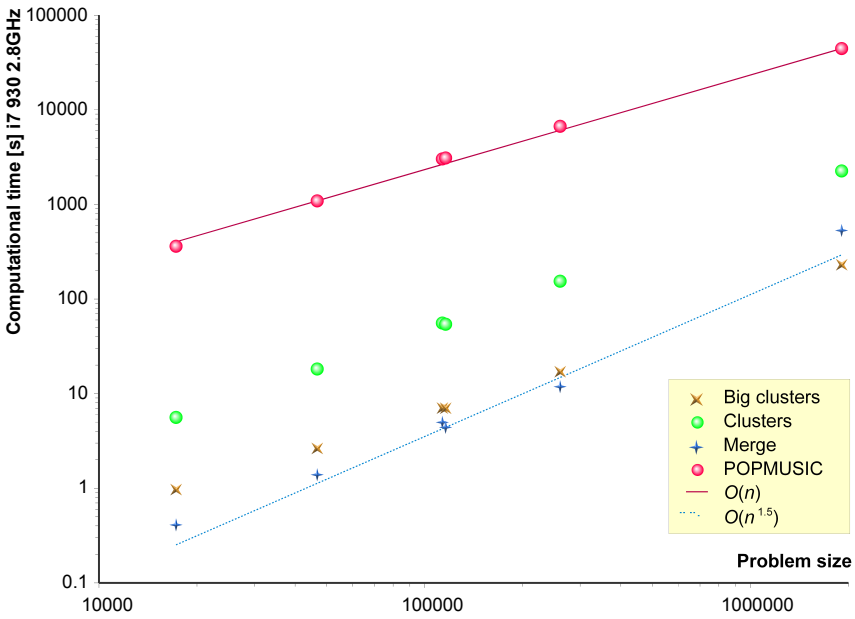
FIGURE 4. Solutions of 2 of the largest *VRP* instances publicly available, obtained just by solving a *TSP* on the clusters produced by our *CPMD* procedure. No further improvement of the tours with a local search. The computational time is 0.37 second for 4(a) and 10.4s for 4(b). The tour length is about 8% above best solutions known for both instances.

4.3. **Sensitivity with respect to problem size.** Figure 5 provides the computing times of the most important steps of our algorithm as a function of the problem size. In this figure, we remark that:

- The building of superclusters and decomposing them into clusters requires a computational effort growing slightly less than the $O(n^{3/2})$ expected.
- The placement of depots by merging *TSP* tours requires a computational effort in $O(n^{3/2})$.
- The improvement of all subproblems grows linearly with n. This step requires the largest computational effort, even for the largest instance. As observed for centroid clustering [15] and map labeling [1], the number of subproblems optimized in `POPMUSIC` seems to grows linearly with problem size.

4.4. **Sensitivity with respect to problem types.** Figure 6 provides the average vehicle load for various instances. The vehicle load is the ratio between the sum of quantities on a tour and the vehicle capacity. We see in this figure that the

FIGURE 5. Evolution of computational effort as a function of problem size, for various parts of the algorithm.

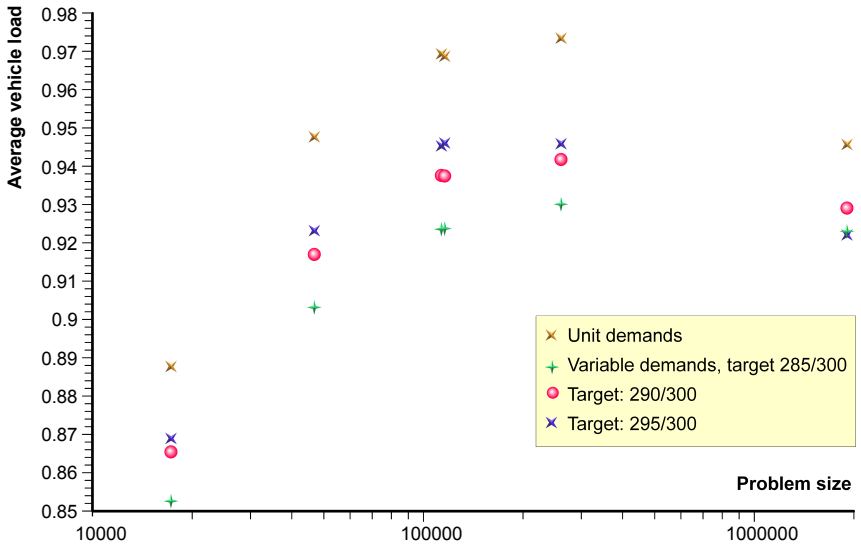


load is systematically above 0.85. For VRP instances, such a load is relatively high. For the *LRP* the load mainly depends on the sparsity of the entities. The smallest and the largest instances are characterized by large zones with very few entities, encouraging the opening of depots for single entities. For unit demand instances, the vehicle load is very high, reaching 0.97 for dense instances. For variable demand instances, the vehicle load depends on the target quantity used for creating clusters. Again, for dense instances, vehicle loads are almost reaching the target.

Figure 7 analyses the influence of solution length and cost with respect to target volume. For variable demand instances, this figure plots the ratio of the total cost over the best *LRP* length found for various values of target volume. If target volume $T \leq 0.95Q$, there is a lost of volume in the vehicles and their number is higher, implying a higher number of trips to the depots and also a higher number of depots. This tends to increase the final cost even if POPMUSIC starts with a *LRP* solution with a shorter length.

If target $T \geq Q$, there are more clusters with volume above vehicle capacity, implying a higher number of *TSP* that are split. The clusters are also less compact. However, we see that `POPMUSIC` is able to improve bad solutions with a large number of *TSP* tours that are split and the final cost is relatively insensitive to T value. This suggests that T parameter could be removed (setting $T = Q$) for the problem treated in this article. However, T parameter could be important for other types of problems, for instance if there are fixed costs for using vehicles. For $D = 100,000$, we see that the cost for opening the depots corresponds to 30% to 40% of the

FIGURE 6. Sensitivity of final vehicle load with respect to target cluster volume



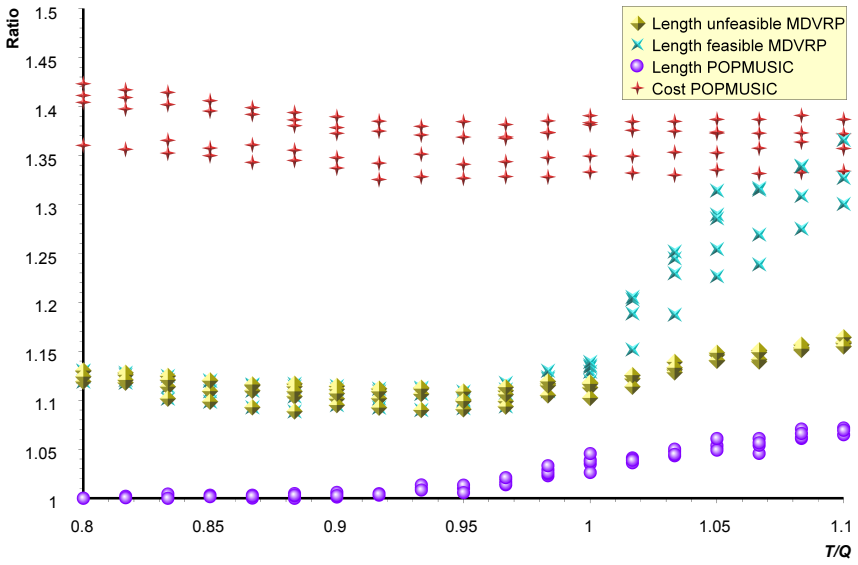
length of the tours. For the final total cost, we see that a target $0.9Q \leq T \leq Q$ seems to be a good compromise.

Figure 8 plots the computational time as a function of vehicle capacity Q. In this figure, we see that the increase of computational effort is proportional to Q^3, as expected. Indeed, the average number of entities per tour linearly grows with Q. Since *MDVRP* subproblems solved in `POPMUSIC` have a fixed number r of tours and since the taboo search has a complexity growing with the cube of the number of entities, the observation are coherent with complexity analysis.

4.5. **Sensitivity with respect to depot opening cost.** Since the basis of subproblem building are tours (parts in `POPMUSIC` terminology), our method seems to be not sensitive to depot opening cost. Indeed, the size of subproblems so defined is independent on D. The only dependency we can observe concerns the computational time needed for merging depots. The larger D is, the higher the number of depots are merged. Figure 9 plots the evolution of computational time of Step 16 of Algorithm 2 as a function of problem size for various D values. We see that the merge procedure takes a computational time growing proportionally to $n^{3/2}$ and is almost independent of D. The picture would certainly have been different for another definition of parts, including all entities attached to a given depot.

4.6. **Sensitivity with respect to initial solution and improvement procedure.** Figure 10 analyses the influence of using a better initial solution or using a better improvement procedure. For getting a better initial solution, we applied a `POPMUSIC` algorithm after having decomposed the problem into clusters, along the lines of [18]. With such a procedure, *CPMD* solutions are improved by about 5%.

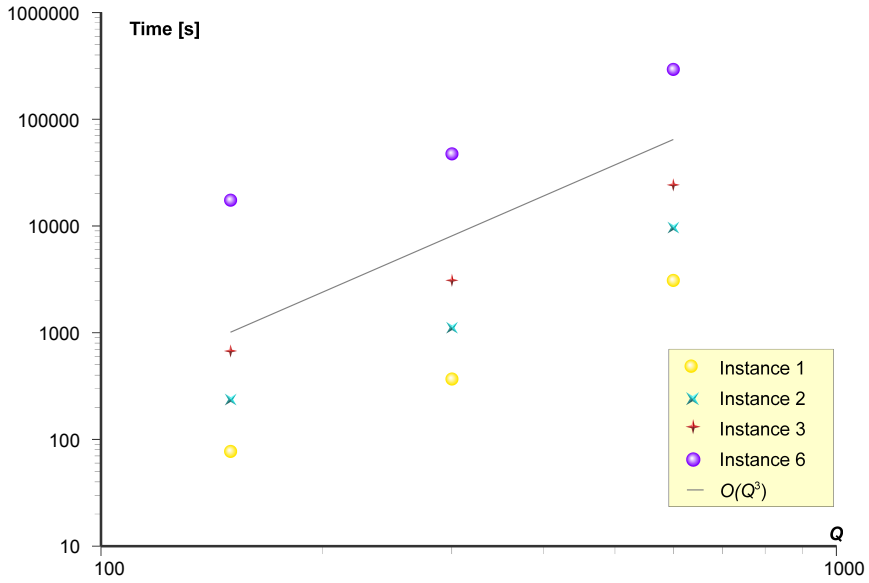
FIGURE 7. Influence of target clusters volume on the length and cost of final solution. All problem instances with variable demands and $Q = 300$. Lengths and costs have been divided by the best *LRP* length observed for each instance.



For getting better solutions to sub-problems we multiplied by 2 or by 4 the number of taboo iterations in the improvement procedure (implying computational times multiplied by the same ratio). Figure 10 provides the ratio of the solution cost of the improved versions with the cost of our algorithm with standard parameters. We see that the improvements are very low, generally below 0.5%, with the exception of the smallest instance for which the improvement reaches 1.2%. Such an improvement is mainly due to a better *CPMD* solution that was improved by 10%. So, it seems that our method is relatively insensitive to initial solution, as soon as the last is of decent quality. The clustering approach is very important for reaching an appropriate quality. Also, by allowing more iterations to the improvement procedure, the global improvements are moderate. However, we think that better solutions might be obtained by using a taboo search that relocates the depot, i.e. by using an improvement procedure that solves *LRP* instances rather than *MDVRP* as we did.

4.7. **Detailed computational results.** As mentioned above, by taking various windows (longitude, latitude) among the entities of the World TSP, six different entities sets have been considered. The standard instances have variable customer demand ($1 \leq q_i \leq 29$), vehicle capacity $Q = 300$ and depot opening cost $D = 100,000$. The standard parameter setting has a target value $T = Q - 10$ for variable demand and $T = Q$ for fixed demand. The number of taboo iterations in standard runs is $t = n$ and the set U of unoptimized part is managed as a stack.

FIGURE 8. Evolution of computational effort as a function of vehicle capacity.



For each entities sets, we also used other values for vehicle capacity ($Q = 10; 20; 40$ for unit demand instances and $Q = 150; 600$ for variable demand instances) and depot opening cost ($D = 50,000; 200,000$), creating a total of 48 test instances.

In addition, we provide computational results obtained for the standard instances by varying the target capacity parameter ($T = 285; 295$), the number of taboo iterations ($t = 2n$ and $t = 4n$), the improvement of the decomposition into clusters with POPMUSIC technique (*CPMD**) and set U management (random choice).

Table 3 provides the computational times for generating the initial solution. Table 4 provides the computational times for improving the initial solution with POPMUSIC. Table 5 provides the improvement of the cost of the solution, expressed in percent, that has been obtained by applying POPMUSIC. Table 6 provides the final cost obtained after POPMUSIC run. This last table shows that POPMUSIC is relatively insensitive to the method parameters: even working much more for getting a better initial solution (*CPMD**) has little impact on the final cost. Managing U set randomly tends to speed-up the method a little bit and to produce slightly worse solution, but this is not systematic.

5. CONCLUSIONS

We have shown in this article that POPMUSIC template can be applied to *LRP* instances of several order of magnitude larger than those commonly treated in the literature. POPMUSIC strategy is able to heuristically solve large instances (more than 10^6 customers) of a location routing problem. The algorithmic complexity of POPMUSIC depends more on the generation of a decent initial solution than on the

FIGURE 9. Influence of computational effort for merging depots for various values of D (depot opening cost).

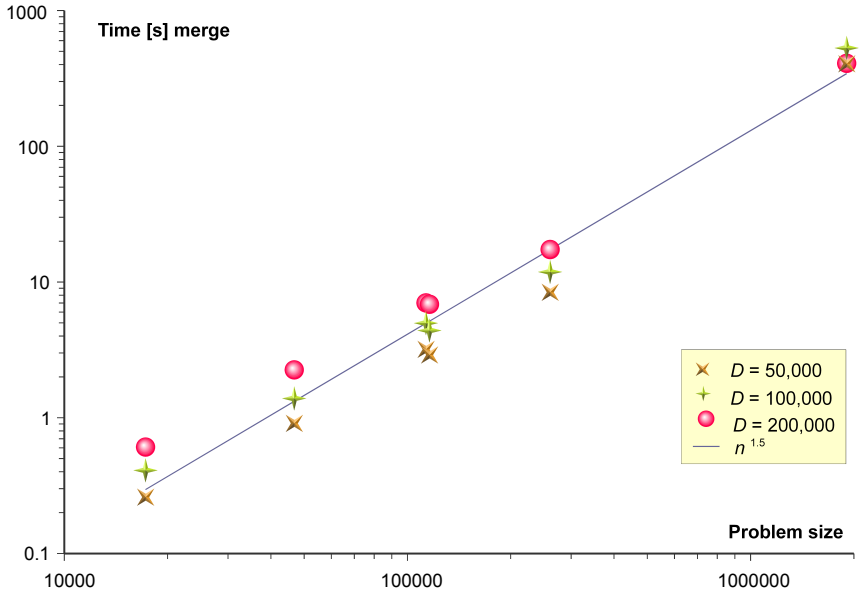


TABLE 3. Computational time to get a feasible initial solution, in seconds on one Intel i7 core.

	Problem size					
Variant	17,237	46,750	113,193	115,858	260,374	1,904,711
Standard	7	23	68	66	185	3,045
$D = 50,000$	7	22	62	64	182	2,984
$D = 200,000$	7	24	66	68	184	2,896
$Q = 150$	8	26	82	80	251	5,435
$Q = 600$	10	31	73	76	189	2,320
$Q = 10$	7	23	72	74	233	4,960
$Q = 20$	7	23	63	66	175	2,811
$Q = 40$	10	28	74	75	188	2,156
$t = 2n$	7	22	63	66	178	2,966
$t = 4n$	7	22	63	66	180	2,962
*CPMD**	355	1,052	2,705	2,963	5,757	36,635
$T = 285$	7	22	64	69	179	2,895
$T = 295$	8	22	64	66	183	2,931
U: random	7	22	63	64	178	2,820

optimization of this solution with a local search. This article presents a way to generate such an initial solution by solving approximatively a *CPMD*. A solution to this problem can be obtained in $O(n^{3/2})$ without using geographical informations

FIGURE 10. Influence of initial solution quality and improvement procedure

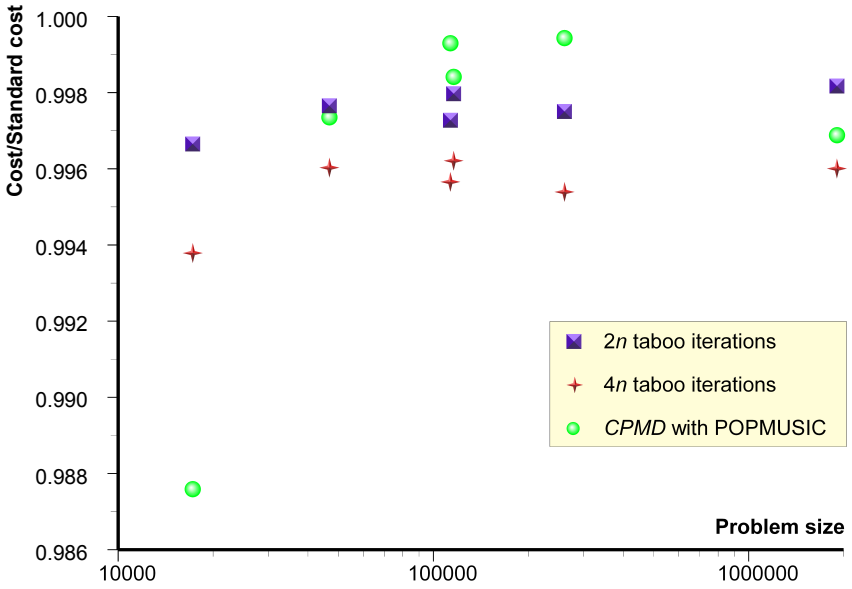


TABLE 4. Computational time to improve the initial solution with POPMUSIC, in seconds on one Intel i7 core.

	Problem size					
Variant	17,237	46,750	113,193	115,858	260,374	1,904,711
Standard	362	1,093	3,034	3,108	6,712	44,553
$D = 50,000$	337	1,000	2,657	2,702	5,977	41,169
$D = 200,000$	415	1,257	3,371	3,376	7,289	49,584
$Q = 150$	70	211	593	576	1,356	12,051
$Q = 600$	3,090	9,646	24,212	27,417	54,824	291,605
$Q = 10$	86	250	682	700	1,563	12,514
$Q = 20$	594	1,925	5,432	5,607	11,724	74,712
$Q = 40$	4005	14,896	45,897	46,231	101,841	600,830
$t = 2n$	761	2,226	6,037	6,056	13,253	87,959
$t = 4n$	1548	4,508	11,913	12,110	27,099	176,877
*CPMD**	88	331	1,102	1,076	2,799	43,211
$T = 285$	392	1,163	3,117	3,297	6,743	46,824
$T = 295$	423	1,096	2,859	3,013	6,362	45,155
U: random	281	788	1,936	2,056	4,489	32,340

other than for computing a distance between 2 entities. This allows to deal with problem instances including millions of entities.

5.1. **Research perspectives.** There are potential improvements for our method, for instance by using a less basic improvement procedure for solving subproblems, like the iterative searches proposed in [13]. Then, the influence of the definition

TABLE 5. Improvements obtained with POPMUSIC over initial solution, expressed in percent.

	Problem size					
Variant	17,237	46,750	113,193	115,858	260,374	1,904,711
Standard	7.3	8.5	9.2	9.3	9.5	9.8
$D = 50,000$	6.8	7.3	7.7	7.9	8.3	8.5
$D = 200,000$	9.2	9.7	10.3	10.3	10.6	10.7
$Q = 150$	7.4	7.7	8.9	8.6	9.4	9.4
$Q = 600$	9.3	10.1	10.3	10.7	9.8	9.4
$Q = 10$	6	6.1	6.1	6.4	6.2	6.3
$Q = 20$	8.2	9.3	9.3	9.5	8.8	7.6
$Q = 40$	8.8	9.9	10.8	11	10.2	8.1
$t = 2n$	7.8	8.8	9.6	9.6	9.8	10
$t = 4n$	8.2	9.1	9.8	9.9	10.1	10.3
*CPMD**	4.9	5.3	5.6	5.8	5.1	4.2
$T = 285$	7.7	9	9.3	9.7	9.3	10.2
$T = 295$	7.8	8.5	9.3	9.6	9.4	9.3
U: random	7.4	8.5	8.9	9	9.3	9.6

TABLE 6. Absolute value of final cost.

	Problem size					
Variant	17,237	46,750	113,193	115,858	260,374	1,904,711
Standard	120,606,754	262,180,587	430,139,462	476,054,191	1,095,478,899	11,971,512,294
$D = 50,000$	105,527,279	233,571,406	380,297,149	422,298,538	970,139,834	10,751,806,821
$D = 200,000$	139,253,415	300,496,000	496,943,248	548,456,575	1,261,973,725	13,622,702,787
$Q = 150$	149,257,348	336,631,365	568,728,130	627,504,037	1,435,686,930	15,111,422,493
$Q = 600$	104,311,652	221,590,472	353,452,289	392,244,201	905,311,798	10,226,935,742
$Q = 10$	146,590,678	329,349,802	555,585,993	612,466,586	1,399,924,584	14,768,168,944
$Q = 20$	118,581,395	257,852,000	424,244,255	470,534,133	1,079,931,492	11,824,034,476
$Q = 40$	104,962,416	221,425,429	350,645,458	390,351,803	898,740,118	10,137,017,763
$t = 2n$	120,203,176	261,565,818	428,967,967	475,088,562	1,092,747,557	11,949,671,929
$t = 4n$	119,857,545	261,140,192	428,271,733	474,250,824	1,090,435,064	11,923,729,093
*CPMD**	119,110,324	261,487,420	429,838,045	475,300,850	1,094,862,284	11,934,236,071
$T = 285$	120,374,633	261,835,995	431,088,578	476,021,788	1,094,107,559	11,921,201,625
$T = 295$	120,982,470	262,103,904	431,304,976	477,655,689	1,099,990,473	12,061,445,385
U: random	120,529,721	262,226,401	430,977,509	476,931,265	1,095,878,122	11,985,158,109

of proximity between 2 tours has to be studied, as well as the management policy of the seed-parts. A more elaborated management could allow to parallelize the method. It seems to be possible to solve problems in $O(\sqrt{n})$ using $O(n)$ processors. Then, the efficiency of POPMUSIC template should be studied for problem instances of higher dimension (e.g. adding time windows). For such instances, the definition of distance between 2 parts of a solution must be revisited. Finally, the approach should be tried on other problems like capacitated or two-echelon *LRP*s, truck and trailer routing problems with satellite depots or periodic variants of these problems [3, 5, 9, 10, 12, 17].

ACKNOWLEDGEMENTS

First author was partially supported by FAPERJ (Project E-26/110.552/2010) and CAPES (Process 1715-09-7), Brazil. Both authors were partially supported by the Institute for embedded systems, HEIG-VD, project AILSI HES-SO-31022.

Second author was partially supported by RCSO-TIC project MetaNetGpu HES-SO-29143.

REFERENCES

[1] Adriana C.F. Alvim and Éric D. Taillard. POPMUSIC for the point feature label placement problem. *European Journal of Operational Research*, 192(2):396–413, January 2009.

[2] Sérgio Barreto, Carlos Ferreira, José Paixão, and Beatriz Sousa Santos. Using clustering analysis in a capacitated location-routing problem. *European Journal of Operational Research*, 179(3):968 – 977, 2007.

[3] Maurizio Boccia, Teodor Crainic, Antonio Sforza, and Claudio Sterle. A metaheuristic for a two echelon location-routing problem. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 288–301. Springer Berlin / Heidelberg, 2010.

[4] GRS 80. `http://en.wikipedia.org/wiki/grs_80`. last visited on november 04, 2011.

[5] S.K. Jacobsen and O.B.G. Madsen. A comparative study of heuristics for a two-level routing-location problem. *European Journal of Operational Research*, 5(6):378 – 387, 1980.

[6] Feiyue Li, Bruce L. Golden, and Edward A. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & OR*, 32:1165–1179, 2005.

[7] Yuichi Nagata and Olli Bräysy. Efficient local search limitation strategies for vehicle routing problems. In Jano I. van Hemert and Carlos Cotta, editors, *EvoCOP*, volume 4972 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2008.

[8] Gábor Nagy and Saïd Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.

[9] V.-P. Nguyen, Christian Prins, and Caroline Prodhon. A multi-start iterative local search with tabu list and path relinking for the two-echelon location routing problem. *Engineering Applications of Artificial Intelligence*, 25(1):56–71, 2012.

[10] V.-P. Nguyen, Christian Prins, and Caroline Prodhon. Solving the two-echelon location routing problem by a hybrid GRASP x path relinking complemented by a learning process. *European Journal of Operational Research*, 216(1):113–126, 2012.

[11] Christian Prins, Caroline Prodhon, and Roberto Wolfler Calvo. A memetic algorithm with population management (MA|PM) for the capacitated location-routing problem. In Jens Gottlieb and Günther R. Raidl, editors, *EvoCOP*, volume 3906 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2006.

[12] Christian Prins, Caroline Prodhon, and Roberto Wolfler Calvo. Solving the capacitated location-routing problem by a GRASP complemented by a learning process and a path relinking. *4OR*, 4(3):221–238, 2006.

[13] Christian Prins, Caroline Prodhon, Angel B. Ruiz, Patrick Soriano, and Roberto Wolfler Calvo. Solving the capacitated location-routing problem by a cooperative lagrangean relaxation-granular tabu search heuristic. *Transportation Science*, 41(4):470–483, 2007.

[14] É. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[15] É. D. Taillard. Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9:51–73, 2003.

[16] É. D. Taillard and S. Voss. POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 613–629. Kluwer Academic Publishers, 2001.

[17] Juan G. Villegas, Christian Prins, Caroline Prodhon, Andrés L. Medaglia, and Nubia Velasco. GRASP/VND and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots. *Engineering Applications of Artificial Intelligence*, 23(5):780 – 794, 2010. Advances in metaheuristics for hard optimization: new trends and case studies.

[18] P. Waelti, T. Mautor, and É. D. Taillard. Application de méta-heuristiques au problème de la p-médiane. In *ROADEF conference*, Avignon, 2003.

[19] World TSP. `http://www.tsp.gatech.edu/world/index.html`. last visited on november 04, 2011.

[20] Emmanouil E. Zachariadis and Chris T. Kiranoudis. A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & OR*, 37(12):2089–2105, 2010.

APPENDIX: DETAILED COMPUTATIONAL RESULTS

As mentioned above, by taking various windows (longitude, latitude) among the entities of the World TSP, six different entities sets have been considered. For each set, we also used different values for vehicle capacity ($Q = 10; 20; 40$ for unit demand instances and $Q = 150; 300; 600$ for variable demand instances) and depot opening cost ($D = 50,000; 100,000; 200,000$), creating a total of 48 test instances. In addition, we provide computational results obtained by varying target capacity parameter ($Q_t = 285; 295$ for the standard instances with $Q = 300$, $D = 100,000$ and variable demands).

Tables 8 and 9 report detailed computational results for one independent run of `POPMUSIC` for the World Location Routing Problem applied to first set of 24 test instances. Tables 10 and 11 report computational results for the remaining 24 test instances. Tables 12 and 13 report results for standard test instances but different values of target capacity parameter. Each table is divided in 2 or 4 sections. For each section, the first line specifies values for vehicle capacity Q, target capacity Q_t, depot opening cost D and the type of demand. The second line in the section is the caption of each column. The meaning of the caption of each column is described in Table 7.

TABLE 7. Table is divided in two parts. First part shows column legends in order they appear in Tables 8, 10 and 12. Second part shows column legends in order they appear in Tables 9, 11and 13.

Caption	**Short description**
I.	Instance number.
sum d.	Sum of all demands: $\sum_{i=1}^{n} q_i$.
lb v.	Vehicle lower bound: $\lceil \frac{\sum_{i=1}^{n} q_i}{Q} \rceil$.
#s.c.	Number of super clusters (Line 2 of Algorithm 2).
obj. s.c.	Objective function value of super clusters solution (Line 3 of Algorithm 2).
cpu	Time in seconds needed to create super clusters solution.
#c.	Number of clusters (Line 4 of Algorithm 2).
obj. c.	Objective function value of clusters solution (Line 4 of Algorithm 2).
#v.c.	Effective number of vehicles needed for cluster solution.
lb v.s.c.	Lower bound on the number of vehicles considering #s.c. clusters.
cpu	Time in seconds needed to create all clusters solution (Line 4 of Algorithm 2).
length tsp	Length of all *TSP*s.
cpu	Time in seconds needed to create all *TSP*s (Line 6 of Algorithm 2).
#v.s.	Number of vehicles after split.
cpu	Time in seconds needed to split all tours (Line 8 of Algorithm 2).
I.	Instance number.
#d.	Number of depots (Line 12 of Algorithm 2).
l. tsp	Length of all merged *TSP*s.
cpu	Time in seconds needed to merge *TSP*s.
l. mdvrp	Length of initial *MDVRP* (Line 8 of Algorithm 2).
#v.	Number of vehicles used in *MDVRP* initial solution.
cpu	Time in seconds needed to make feasible *MDVRP* solution (Line 8 of Algorithm 2).
l. pop	Length after `POPMUSIC`.
#v. pop	Number of vehicles after `POPMUSIC`.
cpu pop	Time in seconds needed to execute `POPMUSIC`.
cost	Total cost: length plus #d $\times D$.
l./v.	Ratio between tour length and number of used vehicles (l. pop/#v. pop).
v./d.	Ratio between number of used vehicles and number of depots (#v. pop/#d.)
total cpu	Total time in seconds to execute proposed method for a *LRP* solution.

TABLE 8. Detailed computational results for first set of 24 test instances.

Experiments with $Q = 10;\ Q_t = 10;\ D = 100,000$ and unit demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	17237	1724	131	520703968.3	0.95	1785	112409434.5	1802	1785	5.43	103294565.2	0.02	1832	0.02
2	46750	4675	216	1331814723	2.54	4771	226190489.8	4837	4771	19.19	201363320.4	0.05	4792	0.02
3	113193	11320	336	2400440641	6.55	11466	343998327.4	11623	11466	60.87	300794014.7	0.11	11467	0
4	115858	11586	340	2646709739	6.79	11747	386664358.2	11851	11747	62.87	340646159.1	0.11	11754	0
5	260374	26038	510	7911822751	17.07	26267	895412023.4	27017	26267	199.28	784492738.1	0.27	26298	0.02
6	1904711	190472	1380	1.72036E+11	230.99	191117	10800817525	208354	191117	3509.5	9495236871	1.92	192650	1.36

Experiments with $Q = 20;\ Q_t = 20;\ D = 100,000$ and unit demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	17237	862	131	520703968.3	0.98	926	167718745.2	927	926	5.51	100607525.1	0.03	975	0.09
2	46750	2338	216	1331814723	2.93	2439	338246353.1	2449	2439	18.61	197552883.4	0.09	2461	0.11
3	113193	5660	336	2400440641	6.61	5819	509761028.1	5842	5819	50.95	294162645.2	0.21	5822	0.05
4	115858	5793	340	2646709739	7.07	5964	569825979.6	5974	5964	53.98	333174744	0.22	5974	0.05
5	260374	13019	510	7911822751	16.23	13270	1327870080	13408	13270	145.87	769025947.5	0.49	13291	0.2
6	1904711	95236	1380	1.72036E+11	226.09	95918	16344668637	102169	95918	2171.11	9372615526	3.72	97239	9.02

Experiments with $Q = 40;\ Q_t = 40;\ D = 100,000$ and unit demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	17237	431	131	520703968.3	0.97	498	246099247.2	499	498	7.42	98066428.23	0.07	544	0.48
2	46750	1169	216	1331814723	2.6	1275	490735397	1278	1275	21.93	194306397.3	0.17	1294	1.04
3	113193	2830	336	2400440641	6.55	2997	737526843	2999	2997	58.6	289823905.8	0.43	3001	1.17
4	115858	2897	340	2646709739	6.81	3070	818490246	3070	3070	60.19	327998952.6	0.44	3077	1.39
5	260374	6510	510	7911822751	16.25	6758	1931345387	6784	6758	152.94	758872361	1	6783	3.54
6	1904711	47618	1380	1.72036E+11	220.04	48316	24071207688	49853	48316	1709.17	9220678431	7.57	49517	53.22

Experiments with $Q = 150;\ Q_t = 140;\ D = 100,000$ and variable demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	1.06	1917	108927951.2	1944	1785	6.07	102017600.8	0.01	1967	0.02
2	699386	2332	216	1327704558	2.9	5101	223005728.8	5217	4772	21.69	204287220.5	0.05	5123	0.02
3	1700791	5670	336	2421822600	6.89	12298	340335141.6	12775	11511	69.19	310329469.6	0.11	12300	0.02
4	1736919	5790	340	2666412176	6.77	12573	383085372.4	12971	11748	68.28	349146222.1	0.11	12578	0
5	3906988	13024	510	7948835766	17.25	28156	890361416.7	29675	26306	215.44	813794328.2	0.24	28186	0.02
6	28571199	95238	1380	1.72732E+11	223.75	204777	10773601615	225793	191164	3815.1	9898158297	1.86	206419	1.51

TABLE 9. Detailed computational results for first set of 24 test instances (continuation).

Experiments with $Q = 10;\ Q_t = 10;\ D = 100,000$ and unit demands.													
I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	451	107184559.1	0.31	107606095.2	1846	0	101490678.1	1840	86.44	146590678.1	55157.98	4.08	93.19
2	1017	240346298	1.11	241601090	4847	0	227649802.3	4828	250.46	329349802.3	47151.99	4.75	273.47
3	1894	386890696.6	4.65	388696005.4	11609	0	366185993	11575	681.77	555585993	31635.94	6.11	754.18
4	2035	433695392.1	4.43	435008069.4	11853	0	408966585.7	11827	699.6	612466585.7	34579.06	5.81	774.03
5	4607	988460104.1	16.32	997526652.5	26940	0	939224584.5	26800	1562.61	1399924584	35045.69	5.82	1796.08
6	42738	10881257508	1212.33	11156709080	205209	0.02	10494368944	202260	12514.41	14768168944	51885.54	4.73	17474.63
Experiments with $Q = 20;\ Q_t = 20;\ D = 100,000$ and unit demands.													
I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	289	97047067.12	0.42	97075201.21	976	0	89681395.01	971	593.58	118581395	92359.83	3.36	600.66
2	586	217349211.9	1.36	217713256.8	2471	0	199252000.2	2467	1924.57	257852000.2	80766.92	4.21	1947.77
3	1119	341077019.4	4.7	341434693.8	5841	0	312344255.4	5839	5431.93	424244255.4	53492.77	5.22	5494.69
4	1214	382199941.4	4.65	382290015.8	5984	0	349134133.2	5980	5607.38	470534133.2	58383.63	4.93	5673.58
5	2699	879191939.8	11.93	881244924.1	13406	0	810031492.4	13374	11724.26	1079931492	60567.63	4.96	11899.51
6	24932	9917528399	396.67	10039583430	101951	0.01	9330834476	100704	74712.1	11824034476	92656.05	4.04	77522.62
Experiments with $Q = 40;\ Q_t = 40;\ D = 100,000$ and unit demands.													
I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	193	93135807.8	0.55	93188343.21	545	0	85662415.59	541	4005.38	104962415.6	158340.88	2.8	4014.91
2	357	204005249.4	1.83	204143487.3	1297	0	185725428.9	1294	14896.17	221425428.9	143528.15	3.62	14923.86
3	660	315300899.1	6.51	315338820.4	3003	0	284645457.7	3001	45897.36	350645457.7	94850.2	4.55	45970.87
4	715	353953082.1	5.92	353953082.1	3077	0	318851802.9	3077	46230.81	390351802.9	103624.25	4.3	46305.79
5	1587	814619581.3	14.19	815405212.6	6806	0	740040118	6800	101841.32	898740118	108829.43	4.28	102029.77
6	14551	9346179075	118.45	9385762985	50716	0	8681917763	50305	600829.93	10137017763	172585.58	3.46	602986.25
Experiments with $Q = 150;\ Q_t = 140;\ D = 100,000$ and variable demands.													
I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	479	108244821.8	0.36	108820505.8	1993	0	101357348.4	1965	69.84	149257348.4	51581.35	4.1	77.41
2	1077	244636467.6	1.19	246519693.8	5231	0	228931365.2	5159	211.12	336631365.2	44375.14	4.79	237.06
3	1963	400178533.6	5.13	405464467.5	12718	0	372428130.2	12405	592.64	568728130.2	30022.42	6.32	674.22
4	2125	445950957.3	4.85	450704999.6	12942	0	415004037.3	12649	576.45	627504037.3	32809.24	5.95	656.7
5	4819	1025788857	17.47	1043504857	29509	0	953786929.6	28526	1356.29	1435686930	33435.71	5.92	1607.3
6	44418	11338347550	1389.15	11673114211	222659	0.01	10669622493	211699	12051.24	15111422493	50399.97	4.77	17486.44

TABLE 10. Detailed computational results for second set of 24 test instances.

Experiments with $Q = 300;\ Q_t = 290;\ D = 50,000$ and variable demands.														
I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.96	961	163837548.8	962	926	5.53	100382746.2	0.03	1078	0.26
2	699386	2332	216	1327704558	2.55	2520	332657117.3	2530	2440	18.03	196926915.4	0.09	2572	0.58
3	1700791	5670	336	2421822600	6.63	6034	502315904.7	6077	5844	51.46	295442154.3	0.21	6042	0.6
4	1736919	5790	340	2666412176	6.84	6157	563267898.7	6195	5958	52.83	334179645.7	0.21	6169	0.73
5	3906988	13024	510	7948835766	16.25	13728	1323715459	13996	13277	154.92	782337810	0.48	13819	1.8
6	28571199	95238	1380	1.72732E+11	226.94	99220	16404788202	106283	95926	2314.93	9713446887	3.8	104448	28.42
Experiments with $Q = 300;\ Q_t = 290;\ D = 100,000$ and variable demands.														
I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.97	961	163837548.8	962	926	5.63	100382746.2	0.03	1005	0.11
2	699386	2332	216	1327704558	2.64	2520	332657117.3	2530	2440	18.25	196926915.4	0.09	2539	0.11
3	1700791	5670	336	2421822600	7.04	6034	502315904.7	6077	5844	55.85	295442154.3	0.22	6036	0.05
4	1736919	5790	340	2666412176	7.02	6157	563267898.7	6195	5958	54.23	334179645.7	0.22	6162	0.05
5	3906988	13024	510	7948835766	17.08	13728	1323715459	13996	13277	154.98	782337810	0.53	13760	0.24
6	28571199	95238	1380	1.72732E+11	230.28	99220	16404788202	106283	95926	2266.54	9713446887	3.6	100581	9.42
Experiments with $Q = 300;\ Q_t = 290;\ D = 200,000$ and variable demands.														
I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.98	961	163837548.8	962	926	5.55	100382746.2	0.03	981	0.03
2	699386	2332	216	1327704558	2.87	2520	332657117.3	2530	2440	18.43	196926915.4	0.09	2526	0.02
3	1700791	5670	336	2421822600	6.58	6034	502315904.7	6077	5844	51.68	295442154.3	0.2	6035	0
4	1736919	5790	340	2666412176	6.86	6157	563267898.7	6195	5958	53.7	334179645.7	0.22	6159	0
5	3906988	13024	510	7948835766	16.3	13728	1323715459	13996	13277	149.06	782337810	0.5	13737	0.02
6	28571199	95238	1380	1.72732E+11	223.28	99220	16404788202	106283	95926	2252.81	9713446887	3.57	99627	2.41
Experiments with $Q = 600;\ Q_t = 590;\ D = 100,000$ and variable demands.														
I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.97	502	250860630.4	503	497	7.49	98265660.71	0.06	553	0.47
2	699386	2332	216	1327704558	2.84	1298	486025890.4	1298	1271	24.86	194729185.8	0.17	1319	1.05
3	1700791	5670	336	2421822600	6.55	3054	728302965.7	3058	3009	58.2	289632957.9	0.42	3055	1.12
4	1736919	5790	340	2666412176	7.07	3109	813373201.6	3109	3056	60.56	327848705.2	0.45	3113	1.37
5	3906988	13024	510	7948835766	16.19	6885	1921833602	6901	6768	153.04	760971571.1	0.98	6913	3.45
6	28571199	95238	1380	1.72732E+11	230.32	49129	24146292654	50870	48314	1854.3	9428796446	7.85	50258	56.57

TABLE 11. Detailed computational results for second set of 24 test instances (continuation).

Experiments with $Q = 300;\ Q_t = 290;\ D = 50,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	438	89293843.29	0.26	89315199.33	1079	0	83627279.48	1066	337.31	105527279.5	78449.61	2.43	344.39
2	846	204972448.1	0.91	205185187.9	2582	0	191271406.1	2573	999.72	233571406.1	74337.9	3.04	1021.99
3	1580	324157438.4	3.19	324483729.3	6085	0	301297148.7	6056	2657.12	380297148.7	49751.84	3.83	2719.46
4	1681	364387075.1	2.9	364813731.9	6206	0	338248538.2	6182	2701.82	422298538.2	54715.07	3.68	2765.57
5	3807	841768926.2	8.41	844452384.1	14072	0	779789834.3	13884	5976.98	970139834.3	56164.64	3.65	6159.38
6	36154	9604347773	405.96	9707293995	110401	0.01	8944106821	106256	41168.5	10751806821	84175.08	2.94	44152.48

Experiments with $Q = 300;\ Q_t = 290;\ D = 100,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	291	98162170.61	0.41	98183526.64	1006	0	91506754.31	996	362.17	120606754.3	91874.25	3.42	369.35
2	621	216894231.2	1.39	217103691.4	2549	0	200080586.6	2543	1093.08	262180586.6	78678.96	4.1	1115.66
3	1143	344325334.1	4.98	344836963.1	6079	0	315839461.6	6047	3033.66	430139461.6	52230.77	5.29	3102.06
4	1226	385958529.8	4.4	386390737.2	6199	0	353454191.4	6176	3107.64	476054191.4	57230.28	5.04	3173.79
5	2774	891959324.8	11.87	895446441.4	14008	0	818078899	13829	6711.7	1095478899	59156.76	4.99	6896.93
6	25080	10256168391	531.29	10388058905	106264	0.02	9463512294	102504	44552.7	11971512294	92323.35	4.09	47597.85

Experiments with $Q = 300;\ Q_t = 290;\ D = 200,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	197	109014662.6	0.61	109036018.7	982	0	99853414.78	976	414.63	139253414.8	102308.83	4.95	421.89
2	437	233432435	2.26	233703403.2	2536	0	213096000.2	2529	1257.15	300496000.2	84260.97	5.79	1280.91
3	798	371409606.7	7.05	372129555	6077	0	337343247.7	6046	3370.68	496943247.7	55796.1	7.58	3436.45
4	856	415526822.1	6.88	416185811.3	6195	0	377256574.6	6171	3376.35	548456574.6	61133.78	7.21	3444.26
5	1937	962145355.9	17.41	967163914.9	13978	0	874573724.5	13814	7288.93	1261973725	63310.68	7.13	7472.76
6	17820	10966863806	409.97	11136917681	104884	0.01	10058702787	101547	49584.01	13622702787	99054.65	5.7	52479.87

Experiments with $Q = 600;\ Q_t = 590;\ D = 100,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	192	92981940.7	0.59	92985283.4	554	0	85111651.66	548	3089.97	104311651.7	155313.23	2.85	3099.6
2	365	203840395.3	1.98	203840395.3	1319	0	185090471.8	1319	9645.93	221590471.8	140326.36	3.61	9676.92
3	670	315897832.4	6.36	315993360	3058	0	286452288.5	3057	24211.84	353452288.5	93703.73	4.56	24284.73
4	711	355498451.4	6.04	355498451.4	3113	0	321144200.7	3113	27416.96	392244200.7	103162.29	4.38	27492.68
5	1608	816847717	14.38	817420705.6	6927	0	744511797.9	6918	54824.3	905311797.9	107619.51	4.3	55012.87
6	14532	9553216775	166.76	9599460948	51656	0	8773735742	50803	291604.62	10226935742	172701.13	3.5	293924.52

TABLE 12. Detailed computational results for standard test instances with alternative values for Q_t parameter.

Experiments with $Q = 300;\ Q_t = 285;\ D = 100,000$ and variable demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.97	973	160676188.1	976	926	5.58	99368947.37	0.03	1018	0.09
2	699386	2332	216	1327704558	2.76	2568	325530641.2	2571	2440	17.83	195694857.2	0.08	2581	0.11
3	1700791	5670	336	2421822600	6.6	6139	496674809.6	6144	5844	52.08	293237518.3	0.21	6140	0.05
4	1736919	5790	340	2666412176	7.49	6263	554883371.1	6264	5958	56.42	332313581.8	0.36	6269	0.03
5	3906988	13024	510	7948835766	16.32	13962	1302950974	14014	13277	149.93	773537495	0.48	13996	0.2
6	28571199	95238	1380	1.72732E+11	223.34	100966	16215017516	104798	95926	2275.8	9691174397	3.52	102387	8.87

Experiments with $Q = 300;\ Q_t = 295;\ D = 100,000$ and variable demands.

I.	sum d.	lb v.	#s.c.	obj. s.c.	cpu	#c.	obj. c.	#v.c.	lb v.s.c.	cpu	length tsp	cpu	#v.s.	cpu
1	258347	862	131	532947433	0.97	944	168330771.2	960	926	6.03	101369800	0.03	990	0.12
2	699386	2332	216	1327704558	2.55	2486	336675670.6	2530	2440	17.48	197691716.6	0.08	2504	0.12
3	1700791	5670	336	2421822600	6.74	5932	513511263	6103	5844	51.88	299820942.3	0.2	5935	0.08
4	1736919	5790	340	2666412176	6.74	6058	572955485.8	6216	5958	54.34	338501222.5	0.21	6062	0.05
5	3906988	13024	510	7948835766	16.43	13504	1344910943	14142	13277	153.59	790058821.4	0.48	13542	0.27
6	28571199	95238	1380	1.72732E+11	226.65	97547	16559703971	109755	95926	2289.33	9699713976	3.71	98907	10.11

TABLE 13. Detailed computational results for standard test instances with alternative values for Q_t parameter (continuation).

Experiments with $Q = 300;\ Q_t = 285;\ D = 100,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	294	97916973.95	0.41	98021213.94	1021	0	90974633.32	1011	391.99	120374633.3	89984.8	3.44	399.11
2	625	217161417.1	1.29	217278136.8	2584	0	199335995.3	2582	1162.92	261835995.3	77202.17	4.13	1185.1
3	1179	342240256.5	4.63	342387080	6145	0	313188577.9	6139	3116.63	431088577.9	51016.22	5.21	3180.43
4	1246	385608938.7	4.29	385639521.4	6270	0	351421787.9	6268	3296.68	476021787.9	56066.02	5.03	3365.53
5	2820	886743636	11.69	887309553.7	14045	0	812107558.5	14003	6743.27	1094107559	57995.26	4.97	6922.44
6	25523	10255980029	379.46	10328438563	105616	0.01	9368901625	103181	46823.66	11921201625	90800.65	4.04	49718.44

Experiments with $Q = 300;\ Q_t = 295;\ D = 100,000$ and variable demands.

I.	#d.	l. tsp	cpu	l. mdvrp	#v.	cpu	l. pop	#v. pop	cpu pop	cost	l./v.	v./d.	total cpu
1	286	99185934.46	0.43	99575701.71	1003	0	92382469.61	992	423.12	120982469.6	93127.49	3.47	430.76
2	602	218108582.7	1.29	219102582.9	2547	0	201903903.8	2526	1095.88	262103903.8	79930.29	4.2	1117.5
3	1118	347212353.4	4.7	349300438.3	6089	0	319504976.2	5998	2858.74	431304976.2	53268.59	5.36	2922.59
4	1212	388407038.5	4.2	390752998.5	6209	0	356455689.2	6120	3012.94	477655689.2	58244.39	5.05	3078.73
5	2722	896661045.9	12.08	905414928.3	14089	0	827790472.5	13769	6361.56	1099990473	60119.87	5.06	6544.97
6	24815	10239075573	397.6	10467891520	107737	0.01	9579945385	103286	45154.99	12061445385	92751.64	4.16	48086.27

(A.C.F. Alvim) DEPARTMENT OF APPLIED INFORMATICS, FEDERAL UNIVERSITY OF THE STATE OF RIO DE JANEIRO, AVENIDA PASTEUR 458, RIO DE JANEIRO, RJ, 22.290-240, BRAZIL.
E-mail address, A.C.F. Alvim: adriana(at)unriotec.br

(É.D. Taillard) DEPARTMENT OF INDUSTRIAL SYSTEMS, UNIVERSITY OF APPLIED SCIENCES OF WESTERN SWITZERLAND, ROUTE DE CHESEAUX 1, CASE POSTALE, CH-1401 YVERDON, SWITZERLAND.
E-mail address, É.D. Taillard: eric.taillard(at)heig-vd.ch