Engineering Applications of Artificial Intelligence 24 (2011) 555–566

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Leveraging saving-based algorithms by master–slave genetic algorithms

Maria Battarra [a], Stefano Benedettini [b,*], Andrea Roli [b]

[a] Industrial Engineering Department, Kadir Has University, İstanbul, Turkey
[b] DEIS–Cesena, Alma Mater Studiorum, Università di Bologna, Italy

ARTICLE INFO

Article history:
Received 8 April 2010
Received in revised form
19 January 2011
Accepted 19 January 2011
Available online 4 March 2011

Keywords:
Saving-based algorithms
Genetic algorithms
Clarke and Wright algorithm
Esau-Williams algorithm
Heuristic algorithms

ABSTRACT

Saving-based algorithms are commonly used as inner mechanisms of efficient heuristic construction procedures. We present a general mechanism for enhancing the effectiveness of such heuristics based on a two-level genetic algorithm. The higher-level algorithm searches in the space of possible merge lists which are then used by the lower-level saving-based algorithm to build the solution. We describe the general framework and we illustrate its application to three hard combinatorial problems. Experimental results on three hard combinatorial optimization problems show that the approach is very effective and it enables considerable enhancement of the performance of saving-based algorithms.
© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Saving-based algorithms are well-known constructive heuristics. In the literature, saving algorithms have been proposed to solve a wide variety of combinatorial problems. The distinctive feature of these algorithms is the idea of starting with a "disaggregated" initial solution, i.e., a set of partial solutions which are iteratively merged until a complete solution is built. More precisely, a *saving criterion* is established and all merges are ranked with respect to their *saving value*. The most promising feasible merge operation is implemented and the procedure is iterated as long as all merges in the ranked list have been evaluated. This simple approach can be adopted for many combinatorial optimization problems and leads to easy to implement heuristics. Another appealing characteristic of these heuristics is their flexibility: complex constraints can be imposed by considering a suitable merge feasibility control, thus simplifying the definition of the problem model. Moreover, even sophisticated objective functions can be evaluated by adapting the saving expression. Given these characteristics, saving algorithms are frequently used for solving complex real world problems.

Examples of problems from the literature solved through saving algorithms include the capacitated vehicle routing problem (CVRP) (Clarke and Wright, 1964), the CVRP with time windows (Van Landeghem, 1988), the multiple trip CVRP (Fleischmann, 1990), the fleet size and mix vehicle routing problem (FSMF) (Golden et al., 1984) and the capacitated minimum spanning tree problem (CMSTP) (Esau and Williams, 1966). Examples of algorithms to solve real world problems in which a saving algorithm is involved are Golden et al. (1977), Begur et al. (1997), Erkut et al. (2000), Gronalt et al. (2003), Chan and Baker (2005), Lee and Atiquzzaman (2005), Faulin et al. (2005), Kant et al. (2008) and Cetinkaya et al. (2009).

Despite these advantages, saving-based heuristics are not as effective as the state-of-the-art metaheuristics. In fact, there is generally a considerably large gap between the solution quality achieved by saving heuristics and the best known solutions. For this reason, in the last decades, saving algorithms are not frequently considered as stand-alone methods: improved versions are used for solving large-sized problems or they provide initial solutions for metaheuristic algorithms. Indeed, metaheuristics fed with good quality initial solutions are usually much more effective.

At first sight, saving algorithms might be considered as out-of-date methods because of their early origin, however, they are still an active research topic. The following are some examples of papers which appeared recently about saving algorithms for routing problems: Corominas et al. (2010), Juan et al. (2010), Doyuran and Catay (2011), and Gajpal and Abad (2010). Many more are the recent articles in which saving algorithms are used to find good quality initial solutions. The success of these algorithms is probably due to their effectiveness with respect to other classical constructive heuristics. For a comparative analysis of the performance of a saving heuristic for the CVRP, we refer to Toth and Vigo (2001). The saving heuristic for the CVRP beats well-known constructive heuristics from the literature in a thorough experimental comparison. To the best of our knowledge, this kind of analysis has not been performed for any other saving heuristic, but we expect similar results.

The attempts done to improve the performance of these heuristic methods are mainly focused on adding diversification to greedy

* Corresponding author.
E-mail addresses: maria.battarra@khas.edu.tr (M. Battarra), s.benedettini@unibo.it (S. Benedettini), andrea.roli@unibo.it (A. Roli).

0952-1976/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2011.01.007

search. For example parametric saving expressions have been considered (i.e., Öncan and Altınel, 2005). Diversified saving expressions lead to a different sequence of merge operations and corresponding different solutions. The diversifying idea is the same even considering other algorithms to perturb the merge sequence. Examples of such algorithms include GRASP techniques (i.e., de Souza et al., 2004), randomized algorithms (i.e., Daskin, 2002) or Monte Carlo based techniques (i.e., Faulin et al., 2008). However, these enhancements require more computational effort, the tuning of some parameters or "problem-dependent" adaptations (i.e., the definition of a suitable neighborhood or a suitable saving expression). Consequently, they cannot be applied to other saving heuristics without code modifications and some time spent in tuning the parameters.

In what follows, we present a "Master–Slave Genetic" (MSG) algorithm able to improve the performance of saving algorithms in a more general setting. The MSG can be seen as a "two-level genetic algorithm" (TLG), where TLGs represent a broad category of metaheuristics sharing a common design principle. In a MSG algorithm, the problem resolution is decomposed into two interacting stages, in which the first stage consists of a genetic algorithm.

TLG algorithms are widely adopted and their paradigm is interpreted in different ways. For example, Chau (2004) proposes a TLG to solve a site allocation problem, in which a genetic algorithm defines first the location decisions, then an LP solver provides the corresponding flow assignment. A similar idea is employed for an allocation problem by El-Araby et al. (2003), and in a flow interception problem by Yang et al. (2009). TLGs are used in a different spirit by Georgopoulou and Giannakoglou (2009) for the unit commitment problem. A GA generates a solution (chromosome) for a relaxed problem, which is then repaired by a second algorithm. Khajavirad et al. (2009) present also a TLG for the joint product family platform selection and design problem.

However, Shestak et al. (2008), solving a task allocation problem on distributed computing systems, Santiago-Mozos et al. (2005), solving a course timetabling problem, and Lacomme et al. (2005), solving the capacitated arc routing problem, interpret the TLG paradigm in the same way as in our contribution. The solution construction is decomposed into two stages: a high-level GA produces first a permutation of elements in a set (chromosome), then, given the chromosome information, a lower-level solver computes and evaluates the solution value. This value is then returned to the GA, so as to perform selection.

Even if our method is similar to the ones presented in the previous cited works, there is one major innovation: instead of devising an *ad-hoc* lower-level heuristic, we use and enhance an existing constructive procedure. Our TLG design is based on the structural characteristics of saving algorithms: the merge operations are ranked with respect to their saving value, producing an ordered list. The list's permutations describe all feasible solutions for the original problem and provide a solution representation form of a chromosome. Such a chromosome contains all the information required by a saving heuristic (i.e., the *lower-level* algorithm) to build a feasible solution.

In our framework, the genetic algorithm is a *master* algorithm that, exploiting the experience collected during the previous iterations, permutes saving lists and produces promising offspring, such as new lists/chromosomes. The fitness evaluation of each new chromosome is then performed by the slave algorithm. During iterations, the genetic algorithm learns which are the most suitable permutations of the merge list and it is able to converge to good quality solutions in a reasonable computing time. We would like to point out that the epithet "master–slave" has been adopted because of the similarity to the parallel algorithm architecture that bears the same name: the *master* (upper-level) algorithm sends tasks to the *slave* (lower-level) algorithm. In this context, a task is the evaluation of the chromosomes' fitness and, at the same time, the construction of a new solution. The slave algorithm is *deterministically* executed: it constructs a solution on the basis of the input provided by the master and returns the fitness value to it. The master learns from the information provided by the slave. We emphasize that this master–slave framework is general, because it can be implemented once and then used for any problem by replacing the slave algorithm.

This generality is, in our opinion, the most innovative contribution of the MSG algorithm. All saving heuristic enhancements proposed so far are dedicated to a specific algorithm when applied to a particular problem. The tests and the results reported are dependent on sets of parameters of both the problem and the algorithm and cannot be applied to another saving heuristic without modifying the algorithm and performing an extensive tuning.

MSG is a framework able to produce high quality solutions for any saving algorithm. Designing and implementing a saving heuristic for a combinatorial problem is a relatively easy task. A software able to plug the slave algorithm into a general framework, and improve its performance is, in our opinion, a remarkable contribution. In fact, this general framework could be employed to improve the performance of state-of-the-art software codes and provide fast and high quality answers to a large class of industrial problems.

For completeness, we also mention the fact that MSG algorithms share some similarities with cooperative coevolutionary algorithms (CCEAs), in which a number of populations are evolved in parallel with Husband and Mill (1991) and Potter and De Jong (2000). In CCEAs, the solution to the problem is decomposed into subcomponents and the populations are defined in such a way that the solution can be built by picking one individual per population and combining them. In this way, the individuals are evaluated as a function of the quality of the complete solution, i.e., their fitness also depends on the individuals of the other populations.

As a final remark, we observed that a parallel implementation of the algorithm is possible. The slave algorithm could be executed on different processors, one for each offspring to be evaluated. This would save up most of the computing time, because the evaluation of the fitness is the time-consuming task in MSG. However, in this contribution, we evaluated the fitness of each offspring in a non-distributed fashion, in order to make a fair comparison with previous algorithms that have not been implemented in a parallel fashion.

The paper is structured as follows. In Section 2, the MSG algorithm is presented in detail, and in Sections 3–5 the performance of our approach is reported for three different problem variants and compared against the best known results. The first problem is the CVRP. Then, the MSG is tested on a more challenging CVRP variant, the FSMF. Finally, our method is tested in solving a different combinatorial problem, the CMSTP. For each problem, we briefly review the related literature and we report the results of our experiments. In Section 6, conclusions are drawn.

2. The master–slave genetic algorithm

Algorithm 1. Master–slave high-level framework

1: $\mathcal{P} \leftarrow$ buildInitialPopulation(n) {set of n individuals}
2: evaluate($\mathcal{P}$)
3: **while** terminating conditions not me **do**
4: $\quad\mathcal{P}' \leftarrow$ applyGeneticOperators($\mathcal{P}$) {operators depend on individual representation}
5: $\quad$evaluate($\mathcal{P}'$) {use slave algorithm}
6: $\quad\mathcal{P} \leftarrow$ bestOf(n, $\mathcal{P}$, $\mathcal{P}'$) {take best n individuals as per steady-state}
7: **end while**
8: **return** min($\mathcal{P}$)

The core idea of our MSG is based on the possibility of splitting the solution construction into two nested phases. In the first

phase, the parameters of a solution construction procedure are set by a *master* solver and in the second phase the solution is actually built by a *slave* solver. For example, in the first phase, merge operations are ordered and, in the second phase, this sequence is used for constructing a solution. In a sense, the problem is decomposed into two, interdependent, sub-problems. The solution to the first problem is an input for the second, which actually constructs a solution. In our TLG framework, the first algorithm is a genetic algorithm, while the second is a saving-based heuristic algorithm. The quality of the merge list provided by the master is then evaluated on the basis of the objective function value of the solution built by the slave. In fact, the master explores a search space of 'parameters': the objective value of the points in this space is the value of the solution returned by the slave.

To fully define a genetic algorithm, and in particular our master algorithm, one has to specify:

- a solution coding for defining the individual structure;
- a selection procedure;
- a population update procedure;
- genetic operators and how they are applied to the population;
- an evaluation criterion for assigning fitness to the individuals;
- an initialization step to generate an initial population.

In the following, we detail each of these algorithmic components.

Solution coding: An individual is a permutation that represents a list of merge operations. The merge ordering is crucial for the slave procedure to compute a good solution. An individual maps merge operations from a reference merge list to a new merge list fed into the slave procedure. In order to obtain such a reference list we adopted a deterministic procedure that depends solely on the instance and a saving formula: first we construct every possible merge of two client nodes and we compute the corresponding saving value, then we sort the merges in a non-increasing way according to their saving value. The saving formula used in this preliminary phase is problem dependent and well-known from the literature (see Sections 3–5 for an overview on saving formulas).

Selection and update: The selection procedure is a simple roulette-wheel, in which individuals are taken for mating with a probability proportional to their respective fitness values. As for population update, we chose to implement a steady-state genetic algorithm.

Recombination operators: We adopted the usual definition of crossover and mutation for permutations of length n. The mutation of an individual encoded as a permutation simply involves a random swap of two elements. The average number of swaps that can be performed on each chromosome is an algorithm parameter, called *mutation rate* or m_r for short. The recombination operator is a two-parent one-point crossover adapted for permutations: first a random point-cut is chosen and the permutations p and q are split into two sub-sequences (p_1, p_2) *and* (q_1, q_2), respectively. The new permutations p' and q' are constructed in this way[1]: at first $p' = p_1$, then all the elements of q_2 which are not in p_1 are orderly appended to p'; if the length of p' is still less than n, the procedure keeps appending elements to p' taken from subsequence q_1 if those elements are not already present in p_1. The crossover operator is always applied.

Fitness evaluation: The evaluation criterion in our master–slave architecture is actually provided by the slave algorithm that builds a solution and computes its value according to the objective function. The implementation of the master–slave solver is such that the only problem-dependent part is the one concerning the saving-based heuristic.

[1] We take into account only the construction of p', since the other one is symmetrical.

2.1. Genetic master–slave implementations

This section presents first a basic implementation of the master algorithm, named Simple, then introduces the three master algorithm variants that have been developed. For each variant, we highlight the differences with Simple and the advantages over it.

The Simple initial population consists of 10% identical chromosomes made of the *original saving list* and 90% random permutations. The original saving list is the permutation suggested by a well-known and effective saving algorithm (i.e., the Clarke and Wright algorithm for the CVRP), which is therefore a promising permutation. The random permutations are obtained by randomly ordering the list of savings. These permutations increase diversification in the initial population. The fraction of identical permutations has been determined by trial and error, on a subset of the test set instances. In Simple, crossover and mutation, as described in the section above, are performed over the whole chromosome.

Based on our preliminary experiments with Simple, we decided to explore possible improvements of the master algorithms by modifying, in turn, the composition of the initial population, the chromosome structure and the genetic operators.

The Params master algorithm considers an initial population composed of higher quality chromosomes, generated by means of well-known parametric saving algorithms from the literature. These algorithms consider a parametric saving expression and they are able to produce better quality results than their non-parametric counterparts. Their saving lists are, therefore, likely to be higher quality chromosomes.

The parameter values in the saving expressions have been generated by drawing uniform random samples in the interval [0, p], where p is an algorithm parameter. The performance of Params has been tested by considering $p=3$ and 4. The p values have been chosen to exploit the experience built in Öncan and Altınel (2005), Golden et al. (1984), and Öncan and Altınel (2009), where the authors considered similar ranges for the parameters.

The last two master algorithms incorporate improved variants of the crossover and mutation operators. The Simple crossover and mutation operators allow the swap of genes from the head of the list to the tail, resulting in a drastic diversification. On the other hand, the original saving list produces good quality solutions, so it should not be completely distorted during the search. Moreover, the first few elements in the saving list are those which determine a larger cost reduction and their order in the list is crucial.

The Reduced master algorithm performs crossover and mutation only on the first $d=15\%$, 20% and 25% (where d is an additional parameter) of the whole chromosome, so as to intensify the search in these genes and not mix them with the ones at the tail of the chromosome. The initial population is generated in a similar manner as in Simple, but keeps the final part of the chromosomes constant. Let l be the length of a chromosome, the first $\lambda = \lfloor d \cdot l \rfloor$ positions are either the identical permutation $(1,\ldots,\lambda)$ (10% probability) or a random permutation of integers in $[1,\lambda]$ (90% probability); the remaining $l-\lambda$ are always filled with the sequence $(\lambda+1,\ldots,l)$.

The Split master algorithm is a specialization of Reduced, in which crossover and mutation are performed separately on the head of the chromosome (i.e., 15%, 20% and 25% of the whole) and on the tail (i.e., 85%, 80% and 75% of the whole). This algorithm shares the advantages of Reduced, providing in addition possible refinement in the chromosome tail. Split has the same configuration parameter d as Reduced and a very similar scheme for generating the initial population: let $\lambda = \lfloor d \cdot l \rfloor$, the "head" of each chromosome is a permutation (either identical or random) of integers in $[1,\lambda]$, while the "tail" is another permutation of integers in $[\lambda+1,l]$.

Table 1
MSG variants.

Master	P_{size}	m_r	Relevant parameter
SIMPLE	100, 200	1, 9	–
REDUCED	100, 200	1, 9	15%, 20%, 25%
SPLIT	100, 200	1, 9	15%, 20%, 25%
PARAMS	100, 200	1, 9	3, 4

An extensive statistical analysis has been performed to define the best configuration of parameters for MSG when applied to each problem variant. The detailed results are reported in Sections 3–5. We also defined common ranges for the parameters. The population sizes, P_{size}, considered are 100 and 200 individuals, whereas the mutation rates are 1 and 9 (means 1 and 9 random swaps on average per chromosome are applied). Note that the number of offspring at each iteration is equal to the population size, that is 100 and 200 individuals. If we consider all the possible combinations of master algorithm variants with their respective parameters, 36 different algorithm instantiations are evaluated.

Last, we introduce a compact nomenclature to identify an algorithm instantiation. We write, separated by dots: the name of the master variant, population size, mutation rate and, if required, an additional parameter. SIMPLE does not require any further parameter besides population size and mutation rate, while PARAMS requires a value for p and SPLIT and REDUCED both require a value for the d parameter. So, for example, an indicator like PARAMS.200.1.3 stands for PARAMS master variant with a population size of 200, and a mutation rate of 1 and $p=3$. The MSG variants are summarized in Table 1. Our experiments were performed on an Intel Xeon 3 GHz, with 8 GB of RAM, for all problem variants and algorithms. The detailed solution values for each test set and for each problem studied can be find at this URL http://apice.unibo.it/xwiki/bin/view/StefanoBenedettini/Papers.

In the following sections, we illustrate the application of the master–slave algorithm on three hard combinatorial problems, for which efficient saving-based heuristics are available.

3. A master–slave Clarke and Wright algorithm

The most famous example of saving heuristics is probably the Clarke and Wright algorithm (Clarke and Wright, 1964). This heuristic was one of the first attempts to solve the *capacitated vehicle routing problem* (CVRP) and it is still frequently employed in the Operations Research Community (see, for example, Toth and Vigo, 2001; Laporte et al., 2000). In what follows, we introduce the CVRP and the required notation. The Clarke and Wright algorithm and the parametric enhancements are then described in detail.

A graph $G(V,E)$ is given, where V is the vertex set and E is the edge set. The vertex 0 is the depot, such as the special vertex in which an unlimited number of vehicles of capacity Q are located. The vertices $V\backslash\{0\}$ are the customers and each of them demands $q_i, i \in V\backslash\{0\}$. Each customer has to be visited exactly once. Each edge $(i,j) \in E$ is associated with a routing cost $c_{ij} \geq 0$. We define a *route* as the tour a vehicle performs starting from the depot and servicing a set of customers. A route is feasible if the sum of the customer demands does not exceed the vehicle capacity Q. The CVRP consists of determining a set of feasible routes with minimum total routing cost, in which the demand of each customer is satisfied.

The disaggregated solution considered in the Clarke and Wright algorithm is composed of single-customer routes. Given this initial solution, the Clarke and Wright algorithm evaluates the saving obtained by merging the routes in which customers i and j are external customers (i.e., connected to the depot), as:

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}. \tag{1}$$

If $s_{ij} > 0$, the merge operation is convenient and if the sum of the demands of the two routes does not exceed Q, the merge operation is feasible. The Clarke and Wright algorithm evaluates the saving values for each pair of customers *a priori*, through (1). Then the saving values are ordered in non-decreasing fashion, resulting in a saving list. Merge operations are evaluated considering the order of the saving list and implemented if the new solution is feasible and more convenient. This greedy algorithm has $O(n^2 \log n)$ complexity (see Jothi and Raghavachari, 2004).

Two versions of the Clarke and Wright algorithm have been presented in the literature, namely the *parallel* and the *sequential*, but we presented the parallel implementation which performs better than the sequential one (Toth and Vigo, 2001). The Clarke and Wright algorithm is fast and simple, but the gap with respect to the best known solutions is most of the times large. In order to improve its performance, enhancements have been proposed in the literature.

An important class of enhancements consists of more accurate saving expressions. To this end, Gaskell (1967) and Yellow (1970) introduced the *route-shape parameter* λ, Yellow (1970) added an additional term scaled by a parameter μ and finally Öncan and Altınel (2005) included a third term in the saving expression, weighted by ν. The saving expression, including all three additional terms, results in:

$$s_{ij} = c_{i0} + c_{j0} - \lambda c_{ij} + \mu |c_{0i} - c_{j0}| + \nu (q_i + q_j)/\overline{q}, \tag{2}$$

where $\overline{q}$ is the average demand, such as $\overline{q} = \sum_{i \in V_c} q_i / |V_c|$. This enriched saving expression works well, as shown by Öncan and Altınel (2005), but the three parameters have to be tuned and this activity can be time consuming. Battarra et al. (2008b) proposed a tuning technique producing high quality results (i.e., the gap with respect to the best known solutions is half of the one obtained with the non-parametric Clarke and Wright algorithm), in a limited computing time.

3.1. Experimental results

In order to test the performance of our MSG algorithm, we considered the same test instances as in Öncan and Altınel (2005), which consist of a well-known benchmark set for the CVRP. In greater detail, our benchmark set is composed of seven instances from Christofides et al. (1979) (*CMT*), eight instances from Christofides and Eilon (1969) (*E*), and 72 instances from Augerat et al. (1995) (*A*, *B* and *P*). All these instances may be downloaded at the site http://www.branchandcut.org, as well as the best known solution values for the Augerat et al. (1995) and the Christofides and Eilon (1969) instances. Note that the Christofides et al. (1979) instances have been solved by Öncan and Altınel (2005), considering double-precision distances. The best known solution values for such instances are available at the web site http://neo.lcc.uma.es/radi-aeb/WebVRP/.

The Augerat et al. (1995) instances have been solved to optimality, whereas the solution values for the instances *E-n76-k8* and *E-n101-k14* have not been solved to optimality. Moreover, the instance *E-n30-k4* is not mentioned in the web site http://www.branchandcut.org; the same solution value as in Öncan and Altınel (2005) is reported.

We implemented 36 versions of the master–slave algorithm, according to the variants and parameter settings described in Section 2. We selected the best variant among these candidates according to the following procedure. Each candidate was run five times for 5 min on each instance and the average solution $\hat{c}$ value

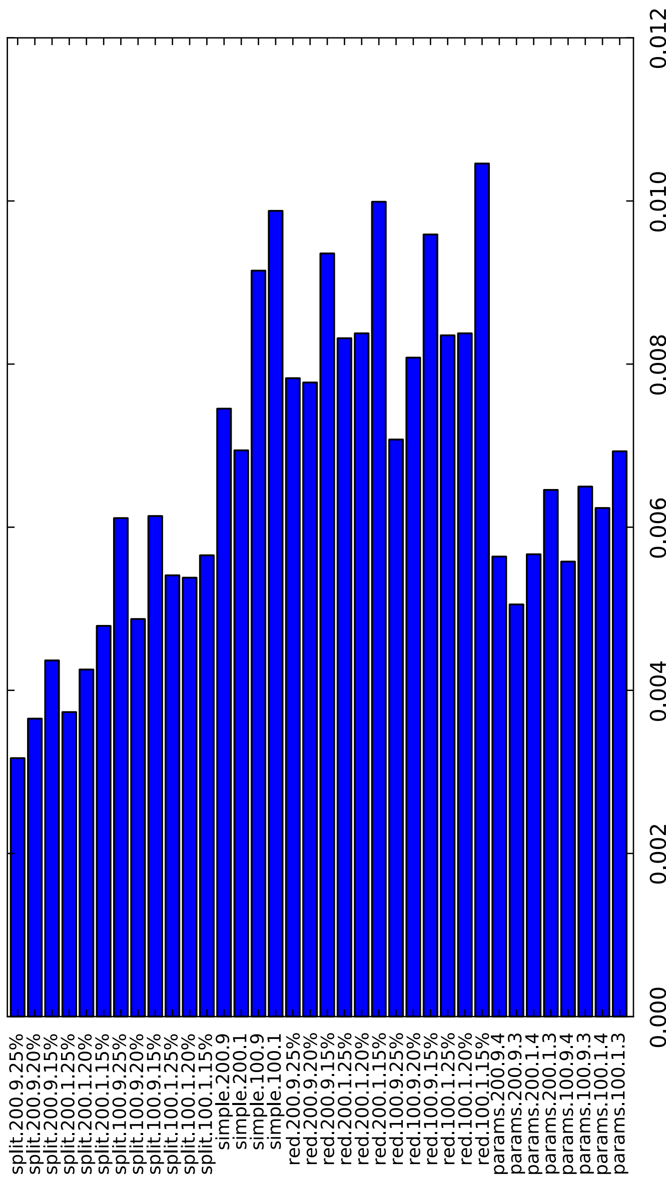

Fig. 1. Comparison among the CVRP candidates. Bars denote the median of the relative deviation between the solution returned by the candidate and the optimal (or best known) solution.

was taken as the representative result per instance. In order to compare the algorithm candidates, we computed the relative deviation $\overline{c} = (\hat{c} - c^*)/c^*$ from the optimal/best known solution c^*. The median of the relative deviations is also considered as a global quality indicator of the candidates. Fig. 1 shows the comparison among the candidates.

To assess statistical evidence of the differences observed, we used the paired Wilcoxon test (Conover, 1999), applied with a one-sided alternative hypothesis (i.e., we want to have statistical evidence for one algorithm being better than the other) and *p-value* equal to 0.05. The test indicates the group of Split.200 as best candidates and we decided to choose the one with the lowest median, namely, Split.200.9.25%, as our best variant to be compared against the state-of-the-art.

Table 2 reports the computational results obtained on each test set (namely *A*, *B*, *P*, *E* and *CMT*), by the best MSG (Split.200.9.25%) and a randomized multistart algorithm (Multistart-cvrp). The randomized multistart algorithm consists of a multistart implementation of the Öncan and Altınel (2005) algorithm, in which the

Table 2
The CVRP experimental results (solutions with a different number of vehicles included). Entries are the average percentage deviations from the best known solutions.

	5 min		25 min	
	Split.200.9.25%	Multistart-cvrp	Split.200.9.25%	Multistart-cvrp
A	0.279 (0.441)	1.940 (1.022)	0.271 (0.425)	1.925 (1.031)
B	0.150 (0.782)	1.554 (1.181)	0.147 (0.782)	1.542 (1.168)
P	−0.067 (1.311)	1.634 (1.684)	−0.067 (1.311)	1.622 (1.673)
E	0.375 (0.421)	1.783 (1.473)	0.286 (0.381)	1.728 (1.415)
CMT	1.321 (1.385)	3.550 (1.834)	1.006 (1.194)	3.405 (1.876)
Avg.	0.412(0.868)	2.092 (1.439)	0.329 (0.819)	2.044 (1.433)

Table 3
The CVRP experimental results (solutions with a different number of vehicles excluded). Entries are the average percentage deviations from the best known solutions.

	5 min		25 min	
	Split.200.9.25%	Multistart-cvrp	Split.200.9.25%	Multistart-cvrp
A	0.286 (0.448)	2.048 (1.150)	0.278 (0.431)	2.032 (1.160)
B	0.292 (0.659)	1.746 (1.051)	0.288 (0.660)	1.733 (1.037)
P	0.411 (0.421)	2.910 (2.290)	0.411 (0.421)	2.896 (2.288)
E	0.521 (0.625)	2.113 (1.837)	0.432 (0.620)	1.777 (1.445)
CMT	1.641 (1.498)	3.541 (2.138)	1.362 (1.246)	3.392 (2.172)
Avg.	0.630 (0.730)	2.472 (1.693)	0.554 (0.676)	2.366 (1.620)

parameters of the saving expression (2) are randomly selected within the interval [0, 5]. The randomized multistart algorithm is iterated as long as the time limit is reached. Two time limits are considered, the same as for the MSG algorithm: 5 and 25 min. Although the multistart algorithm does not represent the state-of-the-art, this comparison is useful to empirically measure the effectiveness of the master algorithm for the search. This comparison assesses the effectiveness of the genetic learning mechanism with respect to a randomized search.[2] Results show that MSG effectively enhances a saving heuristic better than a randomized search and that the solutions returned are close to the best known results.

For each algorithm and computing time, the first column in the table (denoted by the name of the adopted algorithm) reports the average percentage deviation with respect to the best known solution values in the literature. The second column reports in parenthesis the standard deviation. The last line of the table reports the column averages.

The CW algorithm and its enhancements do not impose any constraint on the number of vehicles used. Sometimes the MSG algorithm or the randomized multistart algorithms obtain even better quality results than the best known solutions, because the number of employed vehicles is different. Table 3 reports the same results as Table 2, but the results refer to the minimum cost solutions in which the number of vehicles employed is the same as in the best known solutions. Note that sometimes Split.200.9.25% and Multistart-cvrp cannot find any solution with the prescribed number of vehicles. In this situation, the instance is not included in the averaged results.

The average percentage deviations from the best known solutions obtained by the MSG algorithm Split.200.9.25% are small: the average on all test instances is roughly about the

[2] This is also confirmed by the statistical test on all the three problems we considered.

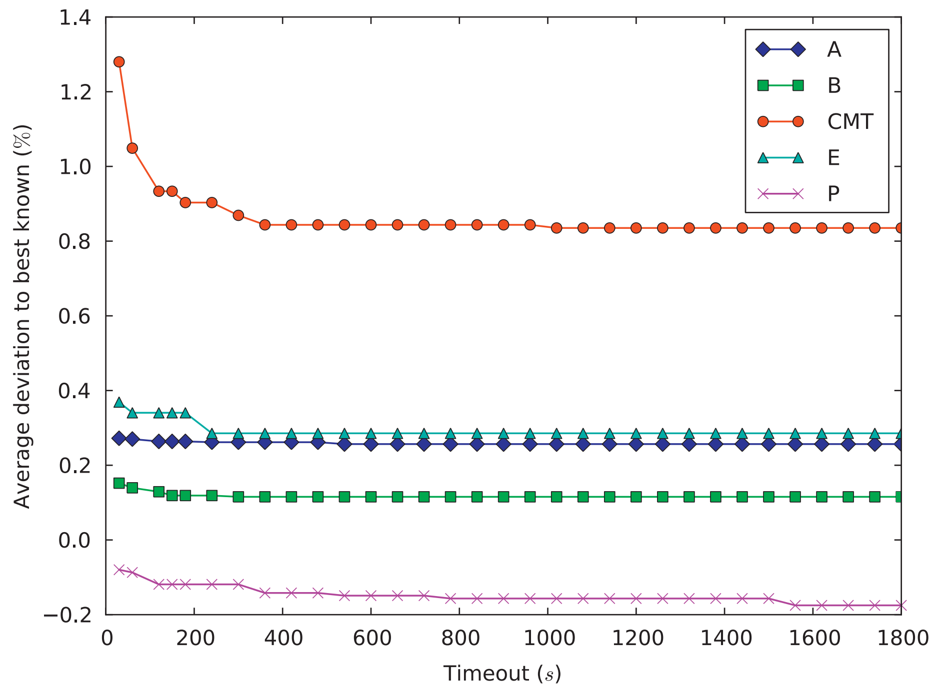

Fig. 2. Solution quality over time for Split.200.9.25% (CVRP). One data-point for each timeout value. Lines are mere guides to the eyes.

0.4%, whereas the largest average deviation is 1.3% on the *CMT* instances that are the most difficult in the benchmark set. Note that the average deviations are about five times smaller than the corresponding deviations for the randomized multistart algorithm. Moreover, the MSG algorithm attains smaller average standard deviations than the randomized multistart algorithm, given the intrinsic convergency properties of genetic algorithms.

Five executions of 1 min each are sufficient for the MSG algorithms to find high quality solutions. A larger computing time, such as five iterations of 5 min each, can be slightly useful for the most difficult instances, like the ones in the *CMT* test set.

To complete our experimental evaluation, we present a runtime analysis of our best MSG for the CVRP. We decided not to perform a similar analysis for the multistart because even with a five-fold increase in computation time the improvements are negligible. To carry out such an analysis we proceed as follows: we run Split.200.9.25% 10 times on each instance and we record the best solution yielded (without any restriction on the number of vehicles employed). We run these experiments with a timeout of 30 min. The results from this study are summarized in Fig. 2. The *x*-axis reports the timeout, in seconds, for the experiments. The *y*-axis reports the average percentage deviation with respect to the best known solutions on each test set, namely *A*, *B*, *P*, *E* and *CMT*.

This study provides an estimation of how much Split.200.9.25% is able to exploit longer timeouts. Fig. 2 shows that running our MSG longer than 400 s does not gain significant improvements.

4. A master–slave saving heuristic for the fleet size and mix vehicle routing problem

The FSM-F is a rich CVRP variant, in which a heterogeneous fleet of vehicles is available. *M* different types of vehicles can be used: each type has associated a capacity, $q_k, k \in 1,\dots,M$, and a fixed cost, $F_k, k \in 1,\dots,M$. The number of vehicles for each type is unlimited. An FSM-F solution consists of a CVRP solution, a set of feasible routes servicing the customers, and of a fleet assignment, such as the vehicles performing the routes. The problem objective is to minimize both the routing costs and the fixed costs associated with the fleet. For a comprehensive survey on heterogeneous vehicle routing problems, the reader can refer to Baldacci et al. (2008a). In our opinion, the FSM-F is an interesting problem to test the performance of the MSG algorithm. In fact, this problem is more general and difficult to solve than the CVRP. The decisions are not only about routing, but also about the fleet of vehicles.

In the literature, Golden et al. (1984) proposed a saving-based algorithm for the FSM-F, in which they considered enhanced saving expressions. In fact, the Clarke and Wright saving formula (1) does not consider the cost associated with the fleet of vehicles and, in some instances, it produces very low quality solutions. The enhanced saving expression, performing more consistently in Golden et al. (1984), is

$$s_{ij} = c_{0i} + c_{oj} - \lambda c_{ij} + \delta(w) F'(P(z_i + z_j) - z_i - z_j), \quad (3)$$

where z_i is the load of the route *i*, $P(z)$ is the capacity of the smaller vehicle that can serve *z*, and $F'(z)$ the fixed cost of the vehicle that has a capacity less than of equal to *z*. The Boolean parameter $\delta(w)$ equals 1 if $w > 0$, 0 otherwise. The parameter λ is responsible for diversifying the search, as the corresponding parameter in (2). It varies in range [0, 3] and its value is increased 0.1 at each iteration (i.e., 31 executions of the parametric algorithm are performed). Clearly, the saving list resulting from this saving expression has to be updated whenever a merge is performed. In fact, when the route *i* is merged with the route *j*, the less expensive vehicle that can serve the merged route is chosen. Moreover, the quantity z_k has to be updated for the resulting new route *k*, the savings recalculated for the customers previously in the routes *i* or *j* and the list possibly reordered. This algorithm results in a larger computing time, both due to these list rearrangements and for the 31 runs required to tune the parameter λ.

In our experimentation, we do not seek an efficient saving expression, but an effective master algorithm. The slave algorithm

has to remain as simple and fast as possible, whereas decisions have to be taken by the genetic learning mechanism. This kind of experiment provides a clear idea of our approach's performance: our chromosomes have to carry rich information (i.e., the routing and the fleet decisions), whereas no optimization is performed at the slave algorithm level. If the method works, the success is due to the learning mechanism of the genetic algorithm.

With this approach, our slave algorithm does not consider any reordering of the saving list and no fixed costs are included in the saving expression. The slave algorithm evaluates the merge operations as suggested by the list proposed by the genetic algorithm and it performs the merge if it is feasible. Whenever a larger vehicle is required and it is available, it is chosen, disregarding its fixed cost. In this way, the slave algorithm is very fast and easy to implement, but the master algorithm is also responsible for a suitable fleet choice.

The FSM-F slave algorithm terminates its execution when the merge list is completely analyzed. As a result, as many merges as possible are implemented and the resulting routes contain a large number of customers. The final solution consists very likely of few vehicles with the highest capacity.

Note that a merge operation can decrease the solution quality (i.e., the merge of two routes may force the use of a larger vehicle and the associated fixed cost has to be paid, so it may not be beneficial). As a result, an intermediate solution can be of higher quality than the final one: for this reason, our slave algorithm returns the best solution found during the execution.

4.1. Experimental results

The benchmark set considered is composed of 12 instances from the literature, originally proposed by Golden et al. (1984). The instances are available at the web site http://apice54.ingce.unibo.it/hvrp/testinstances_g.php. We considered the same instances as in Baldacci et al. (2008a) and Baldacci and Mingozzi (2009) (i.e., the ones in which the customer coordinates are provided) and we refer to the second paper for the updated values of the best known solutions. Note that the Euclidean distances between customers are double precision numbers.

The FSM-F represents a challenging problems variant, given the problem's intrinsic difficulty. Moreover, the saving algorithm implemented is the simplest of the ones proposed in Golden et al. (1984) and does not include in the saving expression any aspect related to the heterogeneous fleet. We expect a poor performance from such a slave algorithm. On the other hand, the master genetic algorithm is expected to detect a better quality fleet, by permuting the chromosome.

We tested 36 candidates of the master–slave algorithm, according to the variants described in Section 2. We selected the best variant among these candidates according to the procedure illustrated in Section 3.1. Fig. 3 shows the comparison among the candidates.

We decided to choose Params.100.9.3 as the best candidate, as the one with the significantly lowest median, although the statistical test could not prove that it was better than Params.100.9.4.

We compared the best MSG algorithm, namely Params.100.9.3, with a randomized multistart approach (Multistart-hcvrp). The randomized multistart algorithm consists of repeated iterations of the most consistent saving algorithm in Golden et al. (1984), such as the algorithm considering the saving expression (3). At each iteration, the parameter λ is randomly generated in the interval [0, 5]. The randomized multistart algorithm is iteratively executed, as long as the same time limits for the MSG algorithms are reached (5 and 25 min).

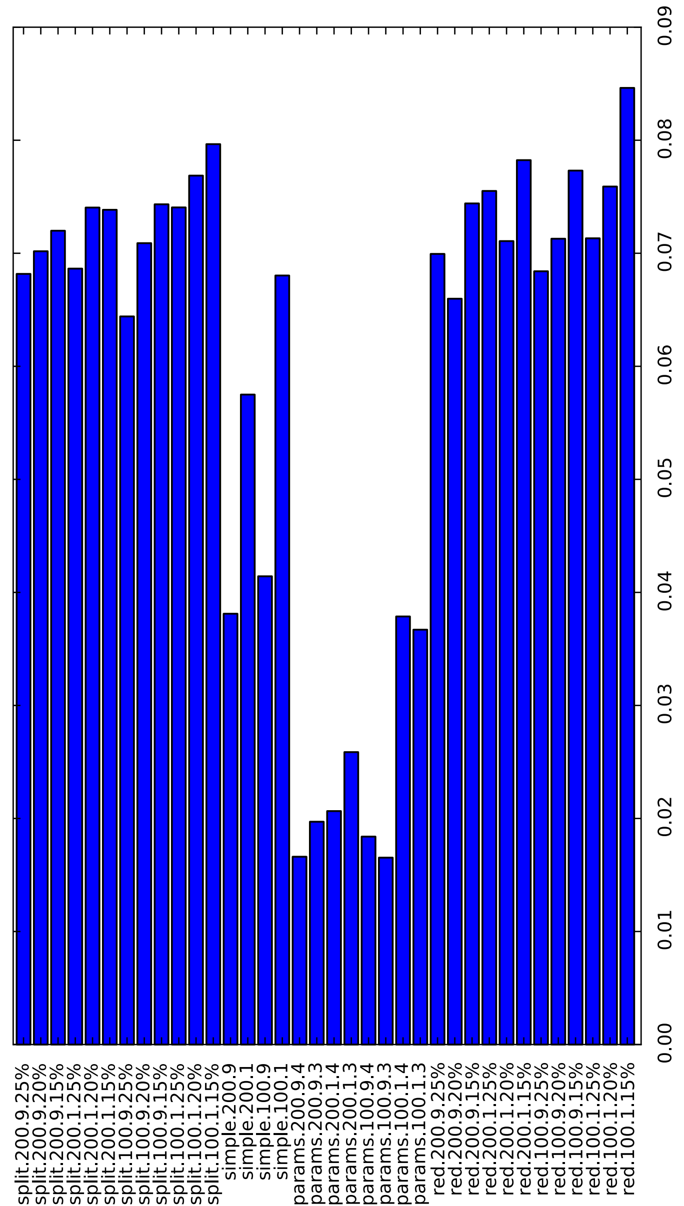

Fig. 3. Comparison among the FSM-F candidates. Bars denote the median of the relative deviation between the solution returned by the candidate and the optimal (or best known) solution.

Table 4
The FSMF experimental results. Entries are the average percentage deviations from the best known solutions.

	5 min		25 min	
	Params.100.9.3	Multistart-hcvrp	Params.100.9.3	Multistart-hcvrp
G3	0.000	0.213	0.000	0.213
G4	0.196	7.557	0.196	7.557
G5	0.698	4.081	0.698	4.081
G6	0.001	0.079	0.001	0.079
G13	2.709	4.503	2.417	4.503
G14	0.409	0.650	0.409	0.650
G15	2.056	6.336	1.886	6.336
G16	2.622	5.778	2.368	5.778
G17	8.286	7.646	7.332	7.646
G18	5.124	5.638	4.639	5.638
G19	0.686	0.915	0.686	0.915
G20	5.226	6.268	5.226	6.268
Avg.	2.334	4.139	2.155	4.139

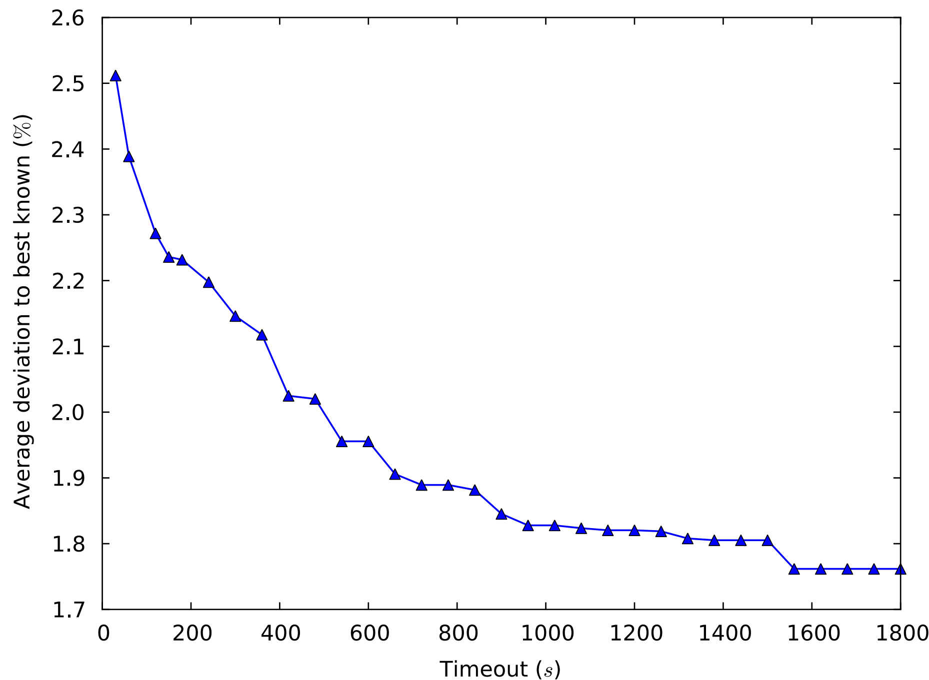

Fig. 4. Solution quality over time for PARAMS.100.9.3 (FSM-F). One data-point for each timeout value. Lines are mere guides to the eyes.

Results are reported in Table 4. For each instance and for each algorithm adopted, the average percentage deviation with respect to the optimal solution is provided. The last line of the table displays the column averages.

The MSG algorithms produce solutions 2–2.5% away from the optimal ones, given that this problem is more difficult than the CVRP. It is interesting to note that the MULTISTART-HCVRP approach attains solutions that are on average 4.1% away from the optimal, so the genetic learning mechanism is able to reduce the gap by almost 2% with respect to a method in which learning mechanisms are not adopted. Note that a longer algorithm execution (i.e., 25 min of computing time) provides a marginal benefit on the average solution quality.

We conclude this section by presenting the results of the runtime analysis on PARAMS.100.9.3. We did not perform such analysis on the multistart because Table 4 shows no improvement over a five-fold increase in execution time. This study was performed in the same manner as for the CVRP: we recorded the best solution returned by PARAMS.100.9.3 in 10 runs on each instance and we set a timeout of 30 min. Fig. 4 summarizes these results. The x-axis reports the timeout, in seconds, for the experiments. The y-axis reports percentage deviation with respect to the best known solution averaged over our 12 benchmark instances.

The chart show that, unlike SPLIT.200.9.25%, PARAMS.100.9.3 is able to take advantage of longer run times.

5. A master–slave Esau–Williams algorithm

The *capacitated minimum spanning tree problem* (CMSTP) is a well-known NP-hard combinatorial optimization problem (see Papadimitriou, 1978), which may be defined as follows. Consider an undirected graph $G=(V, E)$, where $V=\{0, \ldots, n\}$ is the vertex set and $E=\{(i,j) : i,j \in V\}$ is the edge set. Vertex 0 is the *root*, whereas the remaining vertices $i \in V \setminus \{0\}$ have a non-negative demand d_i. Each edge $(i,j) \in E$ is associated with a non-negative cost c_{ij} and the edges incident with the root are called *gates*. The CMSTP calls for the determination of a minimum cost spanning tree such that the demand served on each subtree linked to the root through a gate does not exceed a given maximum capacity Q. The CMSTP arises in real world telecommunication network design problems and in the last decades it has attracted the attention of many researchers, both for its theoretical and practical relevance. Several exact and heuristic algorithms (including approximation algorithms with performance guarantees) have been proposed to solve it.

The Esau–Williams algorithm (EW) is the first method proposed in the literature to solve the CMSTP and it is a constructive saving-based heuristic similar to the well-known Clarke and Wright algorithm (Clarke and Wright, 1964). This algorithm is flexible, fast and easy to implement; these characteristics make the algorithm an excellent candidate to solve real world complex instances as well as a *building block* for metaheuristic approaches (see Amberg et al., 1996; Patterson et al., 1999; Patterson and Pirkul, 2000 and Ahuja et al., 2003). Given the interest in this algorithm, several enhancements were presented in the literature to improve its performance. A recent and effective enhancement is the parametric algorithm presented in Öncan and Altınel (2009), where the improvement of the solution quality comes at a cost of extra computational time requirement, since a large number of parameter values is used within a brute force enumerative approach.

In the EW algorithm, an initial solution in which each vertex is directly connected to the root is fed to the algorithm. At each iteration two subtrees are selected, the merging of which results in the maximum saving and the capacity constraint is not violated by the resulting tree. A merge operation on two subtrees consists of removing the gate with the higher cost and introducing the least-cost edge connecting the subtrees. More precisely, the saving obtained by merging subtrees A and B can be computed as follows:

$$s_{AB} = \max\{c_{0A}, c_{0B}\} - \overline{c}_{AB}, \tag{4}$$

where $c_{0A} = \min\{c_{0i}, i \in A\}$ and $c_{0B} = \min\{c_{0j}, j \in B\}$ are the costs of the gates of subtrees A and B, and $\overline{c}_{ij} = \min\{c_{ij}, i \in A, j \in B\}$ is the cost

of the edge connecting the subtrees. If $s_{AB} > 0$, the merge operation is convenient and if $\sum_{i \in A \cup B} d_i \le Q$ the merge is feasible. Note that after a merge operation, the saving values have to be updated and the most promising couple of subtrees is considered as the merge candidate for the next iteration. This algorithm is simple, easy to implement and its computational complexity is $O(n^2 \log n)$ (see Jothi and Raghavachari, 2004).

In order to improve the performance of the Esau–Williams algorithm, different enhancements have been proposed in the literature. Jothi and Raghavachari (2004) proposed a parametric Esau–Williams enhancement, in which the saving expression includes a new term considering the *bin packing* nature of the CMSTP. The new saving expression is the following:

$$s_{AB} = \max\{c_{0A}, c_{0B}\} - \overline{c}_{AB} \left(\sum_{i \in A} d_i \right)^{\delta}, \tag{5}$$

where δ is a parameter in the range 0–1. The modified saving formula favors the merge operations in which the first subtree is servicing the largest possible demand.

Following a similar reasoning, Öncan and Altınel (2009) proposed a more sophisticated saving formula, in which three parameters are involved:

$$s_{AB} = \max\{c_{0A}, c_{0B}\} - \alpha \overline{c}_{AB} + \beta |c_{0A} - c_{0B}| + \gamma \left(\sum_{i \in A \cup B} d_i \right) \Big/ \overline{d}, \tag{6}$$

where $\overline{d}$ is the average demand. Note that this saving formula mimics the one considered by Öncan and Altınel (2005) in enhancing the performance of the Clarke and Wright algorithm. The performance of this saving expression has been tested by tuning the parameters through an enumerative procedure. The parameter α variation range is [0.1, 2], the parameters β,γ variation range is [0, 2] and the increasing step for each parameter is 0.1. The resulting number of parameter vectors (α,β,γ) considered is 8820, thus the computational time required is increased by almost four orders of magnitude. However, the improvement obtained with respect to the Esau–Williams algorithm is remarkable.

5.1. Experimental results

We tested our approach using the same test instances as in Öncan and Altınel (2009), which represent a classical benchmark set for the CMSTP. Namely, the instances we solved are from the *tc40*, *te40*, *tc80*, *te80*, *cm50*, *cm100* and *cm200* test sets, where the first part of the set names illustrate the instance characteristics. The instances denoted as *tc* have the root located in a central position with respect to the other vertices and each vertex requires unit demand. The *te* instances have the root in an eccentric position with respect to the other vertices (i.e., in a corner) and each vertex requires unit demand. The *cm* set has customers with non-unit demand. The second part of the instance set name denotes the number of vertices involved. Each set consists of 15 instances: for each capacity value, 5 instances are given. The instances from the same set, with the same capacity and same number of customers are identified by their relative number, denoted by *Id.*, in the following.

All the instances are included in the ORLIB library, at the web address http://people.brunel.ac.uk/~mastjjb/jeb/info.html. The best known solutions are the ones reported by Öncan and Altınel (2009), except for the results for the sets *cm50*, *100* and *200* with $Q=200$ and 400. These instances have been recently considered by Uchoa et al. (2008) and some of the solution values were updated.

We implemented 36 versions of the master–slave algorithm, according to the variants described in Section 2. We selected the best variant among these candidates according to the procedure illustrated in Section 3.1. Fig. 5 shows the comparison among the candidates.

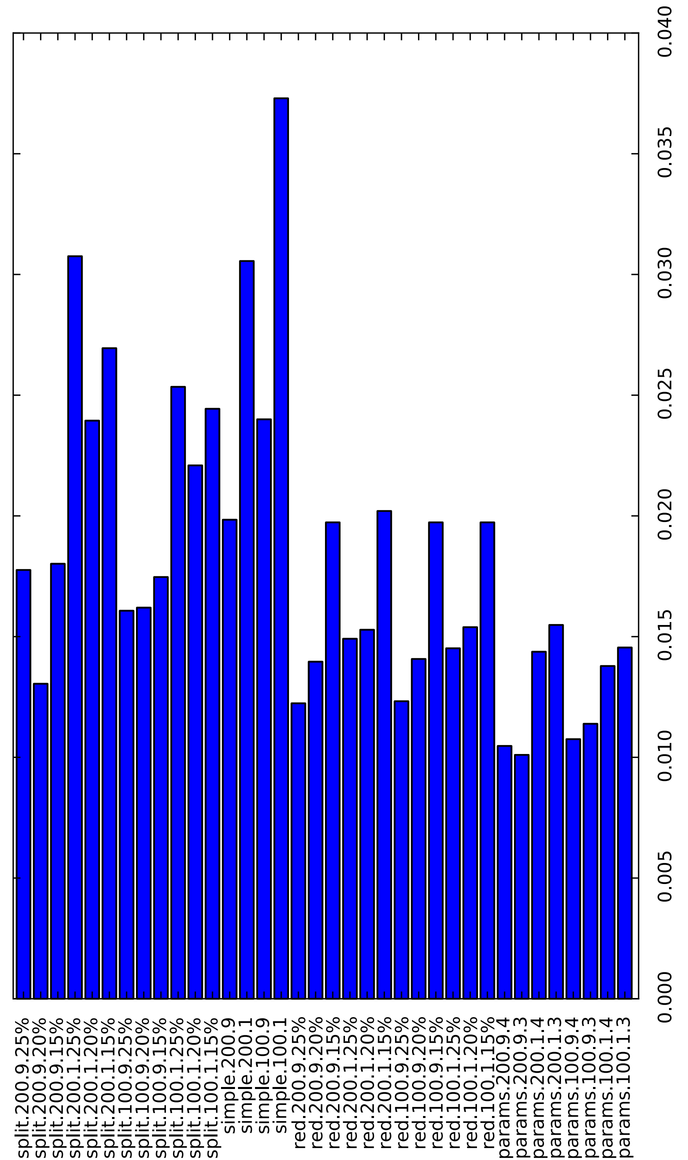

Fig. 5. Comparison among the CMSTP candidates. Bars denote the median of the relative deviation between the solution returned by the candidate and the optimal (or best known) solution.

The comparison shows that the group of variants Params.100 is the one with the significantly best performance. The statistical test could not distinguish between Params.200.9.3 and Params.200.9.4; however, we chose the first one because it has the lowest median relative cost. Params.200.9.3 generates the initial population according to Eq. (6).

The best MSG algorithm, namely Params.200.9.3, is compared with a randomized multistart approach, namely Multistart-cmst. The Multistart-cmst algorithm is based on the iterative execution of the Öncan and Altınel (2009) parametric Esau and Williams algorithm. The three parameters of the saving expression (6) are randomly generated at each iteration, by considering a uniform distribution in the interval [0, 5]. The randomized multistart algorithm is executed for the same overall time as for the MSG algorithm, i.e., 5 and 25 min.

Results are summarized in Table 5. The average percentage deviations from the best known solutions and the average

Table 5
The CMST computational results. Entries are the average percentage deviations from the best known solutions.

	5 min		25 min	
	PARAMS.200.9.3	MULTISTART-CMST	PARAMS.200.9.3	MULTISTART-CMST
*tc*40Q3	0.105 (0.151)	1.384 (0.614)	0.028 (0.056)	1.384 (0.614)
*tc*40Q5	0.643 (0.464)	1.527 (0.271)	0.643 (0.464)	1.527 (0.271)
*tc*40Q10	0.000 (0.000)	0.320 (0.640)	0.000 (0.000)	0.320 (0.640)
*tc*80Q5	3.381 (0.808)	4.164 (0.855)	2.667 (0.852)	4.001 (0.610)
*tc*80Q10	1.963 (0.999)	2.959 (0.919)	1.387 (0.884)	2.890 (0.836)
*tc*80Q20	0.710 (0.620)	0.766 (0.969)	0.71 (0.620)	0.766 (0.969)
*te*40Q3	−0.036 (0.072)	0.780 (0.516)	−0.036 (0.072)	0.780 (0.516)
*te*40Q5	0.318 (0.399)	2.354 (0.755)	0.217 (0.434)	2.354 (0.755)
*te*40Q10	0.584 (0.657)	2.628 (1.709)	0.584 (0.657)	2.628 (1.709)
*te*80Q5	0.771 (0.400)	1.303 (0.564)	0.544 (0.371)	1.248 (0.544)
*te*80Q10	2.578 (0.501)	3.764 (0.825)	2.161 (0.229)	3.715 (0.864)
*te*80Q20	1.093 (1.224)	3.711 (2.418)	0.790 (1.116)	3.549 (2.356)
*cm*50Q200	0.567 (0.633)	1.819 (1.272)	0.546 (0.650)	1.819 (1.272)
*cm*50Q400	1.271 (0.835)	2.862 (0.888)	1.138 (0.778)	2.862 (0.888)
*cm*50Q800	1.883 (0.903)	2.997 (1.510)	1.605 (0.840)	2.997 (1.510)
*cm*100Q200	14.888 (1.814)	20.043 (2.424)	11.918 (1.764)	19.438 (3.068)
*cm*100Q400	6.017 (1.291)	8.908 (1.744)	5.297 (1.315)	7.649 (0.806)
*cm*100Q800	1.101 (0.480)	1.977 (1.170)	0.986 (0.522)	1.759 (0.868)
*cm*200Q200	27.005 (2.485)	25.441 (3.056)	22.36 (1.675)	24.863 (2.716)
*cm*200Q400	20.475 (4.415)	21.281 (4.272)	15.173 (3.454)	19.062 (3.803)
*cm*200Q800	8.837 (2.573)	8.453 (3.450)	4.756 (1.758)	7.274 (2.769)
Avg.	4.484 (1.034)	5.688 (1.469)	3.499 (0.881)	5.375 (1.352)

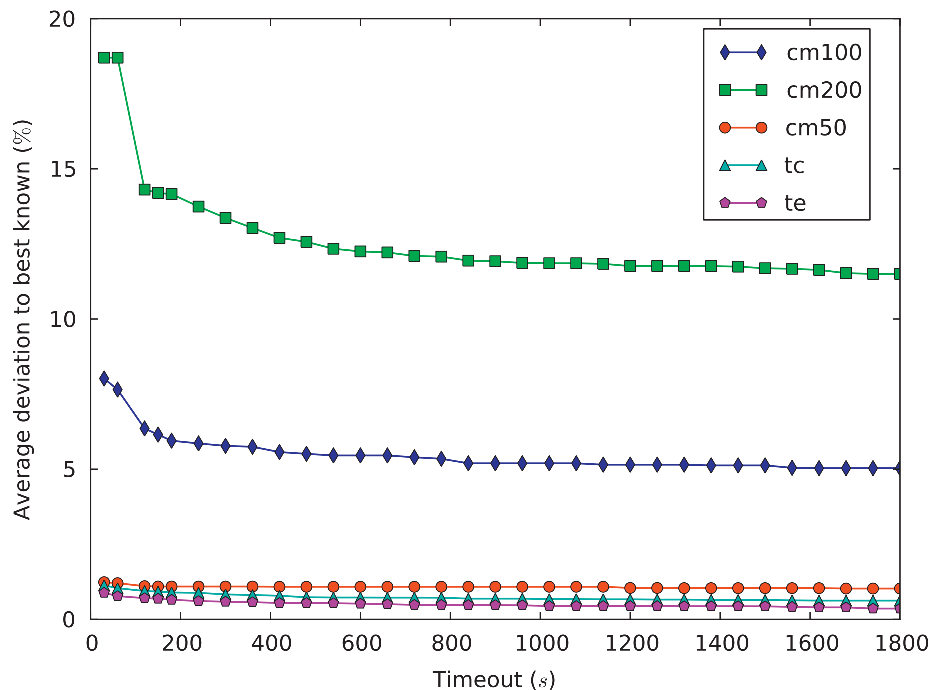

Fig. 6. Solution quality over time for PARAMS.200.9.3 (CMSTP). One data-point for each timeout value. Lines are mere guides to the eyes.

standard deviations are reported for each test set and for the PARAMS.200.9.3 and the MULTISTART-CMST algorithms. Moreover, the last line of the table displays the column averages.

The PARAMS.200.9.3 algorithm performs better than the MULTISTART-CMST approach, on average. The deviations on each instance set are smaller than those reported in Öncan and Altınel (2009) and in Battarra et al. (to appear), where the parametric EW algorithm is tuned by a genetic-based heuristic.

We observe that the PARAMS.200.9.3 algorithm does not perform better than the MULTISTART-CMST when applied to the *cm200* instances in the case of computing time equal to 5 min. The MULTISTART-CMST algorithm is not able to make any genetic algorithm iteration in executions of 1 min, because the slave algorithm needs a longer computing time on large-sized instances. The single minute is completely used up to evaluate the fitness of the initial population and the PARAMS.200.9.3 approach turns out to be a randomized iterative algorithm itself (i.e., the performance is set by the quality of the initial population).

When the computing time of each execution is increased to 5 min (i.e., 25 min in total), the PARAMS.200.9.3 algorithm improves its performance by roughly 1%, whereas the MULTISTART-CMST algorithm only by about 0.3% (note that the improvement is mostly to be ascribed to the performance achieved on the large-sized

instances). Even a few iterations of the genetic algorithm (i.e., roughly 150–250 iterations per 5-min execution, on the *cm200* instances) improve the solution quality substantially. In smaller instances, the Params.200.9.3 algorithm can iterate a sufficient number of times even in 1 min of computing time (i.e., roughly 700–800 and 4200–4500 iteration per min execution, on *cm50* and *cm100*, respectively) and the algorithm drastically outperforms Multistart-cmst.

A runtime analysis of Params.200.9.3 concludes this section. The experimental procedure is the same as the one employed for the previous two problems. Fig. 6 summarizes these results. The *x*-axis reports the timeout, in seconds, for the experiments. The *y*-axis reports percentage deviation with respect to the best known solution averaged over five test sets as shown in Fig. 6. In the chart we put together all *tc* instances because performances do not significantly differ from one another over the possible capacity values. The same is true for *te* instances. On the contrary, *cm* instances show greater variability as both capacity Q and computation time increase.

Even with short execution times, Params.200.9.3 already attains a good performance on *tc*, *te* and *cm50* instances. Longer timeouts do not allow it to close the gap to optimality. The noticeable sudden increase in solution quality in *cm100* and *200* instances around 150 s timeout is a consequence of the large computation time required by those instances. As stated before, a short runtime does not allow Params.200.9.3 to perform any iterations of the high-level genetic algorithm and performances are the same as a simple randomized multistart. As soon as runtime grows, the genetic algorithm has the possibility to perform some iterations, so the performance quickly increases. Longer execution times also mean a sizable improvement: around 7% on *cm200* and around 3% on *cm100*.

6. Conclusions

In this paper, we have introduced an MSG algorithm able to improve the performance of any saving algorithm. The genetic *master* algorithm evolves toward the most suitable saving lists, or chromosomes, and the saving *slave* algorithm computes the solution quality, based on the saving list. This approach is general and can be applied to any saving algorithm.

We have tested the performance of the MSG algorithm by considering three NP-hard problems from the literature, namely the CVRP, the FSMF, and the CMSTP. Extensive experimental testings and comparison against the state-of-the-art results have showed the robustness of the approach and demonstrated that the master algorithm evolves to high quality solutions within reasonable computing times.

Acknowledgement

The authors would like to thank the Writing Center of Kadir Has University for their precious contribution in revising this manuscript.

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.engappai.2011.01.007.

References

Ahuja, R.K., Orlin, J.B., Sharma, D., 2003. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. Operations Research Letters 31, 185–194.

Amberg, A., Domschke, W., Voss, S., 1996. Capacitated minimum spanning trees: algorithms using intelligent search. In: Combinatorial Optimization: Theory and Practice, pp. 9–39.

Augerat, P., Belenguer, J.M., Benavent, E., Corberan, A., Naddef, D., Rinaldi, G., 1995. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report RR 949-M, University Joseph Fourier, Grenoble, France.

Baldacci, R., Mingozzi, A., 2009. A unified exact method for solving different classes of vehicle routing problems. Mathematical Programming 120, 347–380.

Baldacci, R., Battarra, M., Vigo, D., 2008a. Routing a Heterogeneous Fleet of Vehicles. Springer US (Chapter 1, pp. 3–27).

Battarra, M., Golden, B.L., Vigo, D., 2008b. Tuning a parametric Clarke–Wright heuristic for vehicle routing through a genetic algorithm. Journal of the Operational Research Society 59, 1568–1572.

Battarra, M., Öncan, T., Altınel, K.İ., Golden, B., Vigo, D., Phillips, E., in press. An evolutionary approach for tuning parametric Esau and Williams heuristics. Journal of the Operational Research Society.

Begur, S.V., Miller, D.M., Weaver, J.R., 1997. An integrated spatial DSS for scheduling and routing home-health-care nurses. Interfaces 27, 35–48.

Cetinkaya, S., Uster, H., Easwaran, G., Keskin, B.B., 2009. An integrated outbound logistics model for Frito-lay: coordinating aggregate-level production and distribution decisions. Interfaces 39, 460–475.

Chan, Y., Baker, S.F., 2005. The multiple depot, multiple traveling salesmen facility-location problem: vehicle range, service frequency, and heuristic implementations. Mathematical and Computer Modelling 41, 1035–1053.

Chau, K.W., 2004. A two-stage dynamic model on allocation of construction facilities with genetic algorithm. Automation in Construction 13, 481–490.

Christofides, N., Eilon, S., 1969. An algorithm for the vehicle routing dispatching problem. Operational Research Quarterly 20, 309–318.

Christofides, N., Mingozzi, A., Toth, P., 1979. The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (Eds.), Combinatorial Optimization. Wiley, Chichester.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12, 568–581.

Conover, W.J., 1999. Practical Nonparametric Statistics, third ed. John Wiley & Sons.

Corominas, A., Garcia-Villoria, A., Pastor, R., 2010. Fine-tuning a parametric Clarke and Wright heuristic by means of EAGH (empirically adjusted greedy heuristics). Journal of the Operational Research Society 61, 1309–1314.

Daskin, M.S., 2002. Private communication.

de Souza, M.C., Duhamel, C., Ribeiro, C.C., 2004. A grasp heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In: Metaheuristics: computer decision-making. Kluwer Academic Publishers, Norwell, MA, USA, pp. 627–657.

Doyuran, T., Catay, B., 2011. A robust enhancement to the Clarke–Wright savings algorithm. Journal of the Operational Research Society 62, 223–231.

El-Araby, E.E., Yorino, N., Sasaki, H., 2003. A two level hybrid ga/slp for facts allocation problem considering voltage security. International Journal of Electrical Power & Energy Systems 25, 327–335.

Erkut, E., Myroon, T., Strangway, K., 2000. Transalta redesigns its service-delivery network. Interfaces 30, 54–69.

Esau, L.R., Williams, K.C., 1966. On teleprocessing system design. IBM Systems Journal 5, 142–147.

Faulin, J., Sarobe, P., Simal, J., 2005. The DSS LOGDIS optimizes delivery routes for FRILAC's frozen products. Interfaces 35, 202–214.

Faulin, J., Gilibert, M., Juan, A.A., Vilajosana, X., Ruiz, R., 2008. Sr-1: a simulation-based algorithm for the capacitated vehicle routing problem. In: WSC '08: Proceedings of the 40th Conference on Winter Simulation. Winter Simulation Conference, pp. 2708–2716.

Fleischmann, B., 1990. The vehicle routing problem with multiple use of vehicles. Technical report, Fachbereich Wirschaftswissenschaften, Universitat Hamburg.

Gajpal, Y., Abad, P., 2010. Saving-based algorithms for vehicle routing problem with simultaneous pickup and delivery. Journal of the Operational Research Society 61, 1498–1509.

Gaskell, T.J., 1967. Bases for vehicle fleet scheduling. Operational Research Quarterly 18, 281–295.

Georgopoulou, C.A., Giannakoglou, K.C., 2009. Two-level two-objective evolutionary algorithms for solving unit commitment problems. Applied Energy 86, 1229–1239.

Golden, B.L., Magnanti, T.L., Nguyen, H.Q., 1977. Implementing vehicle routing algorithms. Networks 2, 113–148.

Golden, B.L., Assad, A.A., Levy, L., Gheysens, F., 1984. The fleet size and mix vehicle routing problem. Computers & Operations Research 11, 49–66.

Gronalt, M., Hartl, R.F., Reimann, M., 2003. New savings based algorithms for time constrained pickup and delivery of full truckloads. European Journal of Operational Research 151, 520–535.

Husband, P., Mill, F., 1991. Simulated coevolution as the mechanism for emergent planning and scheduling. In: Belew, R., Booker, L. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 264–270.

Jothi, R., Raghavachari, B., 2004. Revisiting Esau–Williams' algorithm: on the design of local access networks. In: Proceedings 7th INFORMS Telecommunications Conference (Telecom), Boca Raton, Florida, pp. 104–107.

Juan, A.A., Faulin, J., Jorba, J., Riera, D., Masip, D., Barrios, B., 2010. On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright savings heuristics. Journal of the Operational Research Society, published online, doi:10.1057/jors.2010.29.

Kant, G., Jacks, M., Aantjes, C., 2008. Coca-cola enterprises optimizes vehicle routes for efficient product delivery. Interfaces 38, 40–50.
Khajavirad, A., Michalek, J., Simpson, T., 2009. An efficient decomposed multi-objective genetic algorithm for solving the joint product platform selection and product family design problem with generalized commonality. Structural and Multidisciplinary Optimization 39, 187–201.
Lacomme, P., Prins, C., Ramdane-Cherif, W., 2005. Evolutionary algorithms for periodic arc routing problems. European Journal of Operational Research 165, 535–553.
Van Landeghem, H.R.G., 1988. A bi-criteria heuristic for the vehicle routing problem with time windows. European Journal of Operational Research 36, 217–226.
Laporte, G., Gendreau, M., Potvin, J.-Y., Semet, F., 2000. Classical and modern heuristics for the vehicle routing problem. International Transactions in Operational Research 7, 285–300.
Lee, Y.-J., Atiquzzaman, M., 2005. Least cost heuristic for the delay-constrained capacitated minimum spanning tree problem. Computer Communications 28, 1371–1379.
Öncan, T., Altınel, I.K., 2009. Parametric enhancements of the Esau–Williams heuristic for the capacitated minimum spanning tree problem. Journal of the Operational Research Society 60, 259–267.
Öncan, T., Altınel, K., 2005. A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. Journal of the Operational Research Society 56, 954–961.
Papadimitriou, C.H., 1978. The complexity of the capacitated tree problem. Networks 8, 217–230.
Patterson, R., Pirkul, H., 2000. Heuristic procedure neural networks for the CMST problem. Computers and Operations Research 27, 1171–1200.
Patterson, R., Pirkul, H., Rolland, E., 1999. A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. Journal of Heuristics 5, 159–180.
Potter, M.A., De Jong, K.A., 2000. Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evolutionary Computation 8 (1), 1–29.
Santiago-Mozos, R., Salcedo-Sanz, S., DePrado-Cumplido, M., Bousoño-Calzón, C., 2005. A two-phase heuristic evolutionary algorithm for personalizing course timetables: a case study in a spanish university. Computers & Operations Research 32, 1761–1776.
Shestak, V., Chong, E.K.P., Siegel, H.J., Maciejewski, A.A., Benmohamed, L., Wang, I.-J., Daley, R., 2008. A hybrid branch-and-bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing 68, 410–426.
Toth, P., Vigo, D., 2001. The Vehicle Routing Problem. SIAM, Philadelphia, PA.
Uchoa, E., Fukasawa, R., Lysgaard, J., Pessoa, A., Poggi da Aragão, M., Andrade, D., 2008. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. Mathematical Programming, Series A 112, 443–472.
Yang, J., Zhang, M., He, B., Yang, C., 2009. Bi-level programming model and hybrid genetic algorithm for flow interception problem with customer choice. Computers & Mathematics with Applications 57, 1985–1994.
Yellow, P., 1970. A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly 21, 281–283.