Journal of the Operational Research Society

ISSN: 0160-5682 (Print) 1476-9360 (Online) Journal homepage: https://www.tandfonline.com/loi/tjor20

An evolutionary approach for tuning parametric Esau and Williams heuristics

M Battarra, T Öncan, I K Altınel, B Golden, D Vigo & E Phillips

To cite this article: M Battarra, T Öncan, I K Altınel, B Golden, D Vigo & E Phillips (2012) An evolutionary approach for tuning parametric Esau and Williams heuristics, Journal of the Operational Research Society, 63:3, 368-378, DOI: 10.1057/jors.2011.36

To link to this article: https://doi.org/10.1057/jors.2011.36

Published online: 21 Dec 2017.

Journal of the Operational Research Society (2012) 63, 368–378 © 2012 Operational Research Society Ltd. All rights reserved. 0160-5682/12

www.palgrave-journals.com/jors/

An evolutionary approach for tuning parametric Esau and Williams heuristics

M Battarra[1]*, T Öncan[2], IK Altınel[3], B Golden[4], D Vigo[1] and E Phillips[5]

[1]Kadir Has University, Istanbul, Turkey; [2]Galatasaray University, Istanbul, Turkey; [3]Boğazici University, Istanbul, Turkey; [4]R.H. Smith School of Business, University of Maryland, MD, US; and [5]Department of Mathematics, University of Maryland, MD, US

Owing to its inherent difficulty, many heuristic solution methods have been proposed for the capacitated minimum spanning tree problem. On the basis of recent developments, it is clear that the best metaheuristic implementations outperform classical heuristics. Unfortunately, they require long computing times and may not be very easy to implement, which explains the popularity of the Esau and Williams heuristic in practice, and the motivation behind its enhancements. Some of these enhancements involve parameters and their accuracy becomes nearly competitive with the best metaheuristics when they are tuned properly, which is usually done using a grid search within given search intervals for the parameters. In this work, we propose a genetic algorithm parameter setting procedure. Computational results show that the new method is even more accurate than an enumerative approach, and much more efficient.

Journal of the Operational Research Society (2012) **63,** 368–378. doi:10.1057/jors.2011.36
Published online 1 June 2011

Keywords: capacitated minimum spanning tree problem; evolutionary algorithms; parameter tuning

1. Introduction

The *capacitated minimum spanning tree problem* (CMSTP) can be defined as finding a minimal-cost tree that spans all vertices such that the sum of the demands in each subtree does not exceed the given capacity limitation Q. When the demands of all vertices are equal to unity, the problem reduces to finding a minimal-cost rooted spanning tree in which each subtree contains at most Q vertices. This is the unit demand or *homogeneous* case and is referred to as the CMSTP in the literature. We are concerned with both homogeneous and heterogeneous cases in this paper. To learn more about telecommunication network design problems, which can be formulated as CMSTPs, see the survey by Gavish (1991). The CMSTP is a difficult combinatorial optimization problem. It has been shown to be NP-Hard even for the unit demand case when $2<Q<n/2$ (Papadimitriou, 1978). As a result of the inefficiency of exact methods in solving large instances, heuristics are widely used in practice. CMSTP heuristics can be classified as classical ones and metaheuristics. Classical heuristics are straightforward and perform a limited exploration of the search space compared with metaheuristics. However, they are simple, which makes them easy to understand and implement, and they can find good solutions quickly. Some of them are flexible and can be extended easily to solve many variants of the CMSTP. The performance of the classical heuristics does not usually compare with that of metaheuristics, but the parameter setting and the execution of the latter demands a much higher computational cost. Besides, the large set of parameters makes metaheuristics context dependent and difficult to extend to other instances even though it increases their flexibility.

In their work on vehicle routing heuristics, Cordeau *et al* (2002) introduce *accuracy, speed, simplicity*, and *flexibility* as the most important characteristics of a good heuristic and compare well-known classical heuristics and best available metaheuristics for the capacitated vehicle routing problem (CVRP) according to these four criteria. They report that none of the classical heuristics is as accurate and flexible as any one of the metaheuristics, but Clarke and Wright's saving heuristic (CW) (Clarke and Wright, 1964) is very fast and extremely simple to implement, which probably explains its popularity. The same principles can be extended to the CMSTP and the Esau and Williams heuristic (EW) (Esau and Williams, 1966), as well. The EW heuristic is also based on a savings criterion and ends up with a feasible solution using an extremely modest computational effort, which makes it a widely used solution approach for the CMSTP. It is also embedded in many metaheuristics as a slave local search (LS) procedure

**Correspondence*: *M Battarra, DEIS, University of Bologna, Via Venezia 52, Cesena, FC 47023, Italy.*
E-mail: maria.battarra@khas.edu.tr

(Amberg *et al*, 1996; Patterson *et al*, 1999; Patterson and Pirkul, 2000; Ahuja *et al*, 2003), and it is regarded as a benchmark by several researchers. In their early work, Amberg *et al* (1996) emphasize the superiority of EW's average performance when compared with other heuristics in the literature having similar computational cost. As a result, the improvement of EW's solution quality without harming its simplicity and efficiency becomes an interesting issue. Recently, Öncan and Altınel (2009) have suggested a quick and intuitive approach by introducing additional parameters and terms related to the capacity restrictions in its savings criterion. The enhanced EW procedure can control edge selections and combine distance and demand information in its savings function. The authors obtained significant improvement in the solution quality relative to the original EW heuristic after running these new enhancements with several thousands of different parameter combinations.

In short, the improvement in the solution quality depends on the appropriate tuning of three parameters, which comes at a cost of extra computing time. This is a weakness shared by many heuristic procedures with multiple parameters, and the efficient determination of parameter values that allows best possible performance over all problem instances is a critical issue. Trial and error algorithms require long tuning experiments; a systematic and simple way to determine parameters would increase algorithmic efficiency.

In the literature, Van Breedam (1996) tested the influence of the parameters of a genetic algorithm (GA) and a simulated annealing (SA) algorithm, by considering a Automatic Interaction Detection technique (Morgan and Sonquist, 1963). Robertson *et al* (1998) considered a fractional factorial experiment to set the parameters of a neural network for a finance application, whereas Park and Kim (1998) used the simplex method (for nonlinear programming) to define the parameters of a SA algorithm. Parson and Johnson (1997) used statistical design to set the parameters in a GA. Since these early attempts, many algorithms have proved to be effective in tuning parameters. We can mention Backpropagation (Rumelhart *et al*, 1986), K-Nearest Neighbour (Aha *et al*, 1991), support vector machines (Cortes and Vapnik, 1995), and C4.5 (Quinlan, 1993), as some of the most effective and widely used algorithms in practice.

An alternative approach is to use a GA for tuning the parameters, which was first applied by Golden *et al* (1998) to determine the parameters of a CVRP heuristic, by considering a *two-stage* GA. Pepper *et al* (2002) applied a similar technique to the parameter setting of an annealing-based heuristic for the travelling salesman problem, and Chandran *et al* (2003) further analyzed the possibility of applying a genetic-based parameter-setting algorithm by introducing a simpler *single-stage* procedure.

In this paper, we follow this line of research and propose an evolutionary algorithm to search for the best parameters of the EW enhancements. The genetic search helps to determine locally promising areas in the parameter space, and the following local search and randomized prohibition stages refine the search in these promising areas. On the basid of the experimental results, we can say that the new method is able to compute solutions of better quality than that of Öncan and Altınel (2009) in significantly less computing time. We summarize the parametric EW enhancements in the next section, for the sake of completeness, which is followed by a brief explanation of the evolutionary parametric search in Section 3. Computational results are provided in Section 4 and conclusions are presented in Section 5.

2. The Esau and Williams heuristic and a three-parameter enhancement

Suppose we have a given complete graph $G=(V, E)$, where $V=\{0, 1, \ldots, n\}$ is the set of vertices with 0 as the central vertex, and $E=\{\{i,j\}: i,j \in V, i \neq j\}$ is the set of edges. Each noncentral vertex has a known nonnegative demand d_i. A cost c_{ij}, which is the cost of joining vertices i and j, is associated with edge $\{i,j\}$. The subtree containing vertex i is called the *subtree of i* and the set V_i denotes its vertices. The edge connecting the subtree of vertex i to the central vertex is the *gate* of i and has cost c_{0i}. What makes the problem capacitated is the limitation on the sum of the individual vertex demands in a subtree: it is not allowed to be larger than a given bound Q. When the demands of the vertices are all equal to unity, the problem reduces to finding a minimal-cost rooted spanning tree in which each subtree includes at most Q vertices.

Ordinarily, the EW starts with a *star tree;* that is, every vertex is directly connected to the central vertex and there are as many subtrees as the number of vertices. The star tree is a feasible topology, otherwise the problem is infeasible. EW attempts to reduce the cost as much as possible by modifying the current feasible solution without violating capacity constraints. This is done first by identifying and merging two vertices i and j belonging to two different subtrees that yield the largest savings, so that the total demand of subtree i and subtree j does not exceed Q. Then, the edge $\{i, j\}$ replaces the most expensive of the edges $\{0, i\}$ and $\{0, j\}$, resulting in the largest saving

$$s_{ij} := \max\{c_{oi}, c_{oj}\} - c_{ij}. \tag{1}$$

This operation joins the subtrees of vertices i and j.
EW is a myopic and greedy algorithm. Once two vertices are joined, it is not possible to disconnect them later, which can produce results with low solution quality. In order to remedy this drawback, EW can be modified to exercise greater control over the merge operations. A concise review of such attempts can be found in the recent work by Öncan

and Altınel (2009), where they introduce several parametric EW enhancements.

The most accurate of the proposed enhancements consists of a parametric savings expression in which the *tree shape parameter* α, a new term considering asymmetry between vertices i and j with respect to the central vertex, weighted by β, and a third term considering bin packing aspects, weighted by γ, are included. The resulting savings expression is

$$s_{ij} = \max\{c_{0i}, c_{0j}\} - \alpha c_{ij} + \beta|c_{0i} - c_{0j}| + \gamma\Big(\sum_{k \in V_i \cup V_j} d_k\Big)/\bar{d}, \tag{2}$$

where d_i is the demand of vertex i and d is the average demand. Note that this savings formula mimics the one considered by Altınel and Öncan (2005), while enhancing the performance of CW.

3. Evolutionary search for the best parameters

The idea of considering GAs for tuning the parameters of the heuristics is first introduced by Golden *et al* (1998) in their work on the CVRP in order to calculate the parameters of a Lagrangean heuristic. This is a *two-phase procedure*. During the *first phase*, the procedure is trained on a small set of representative problem instances called the analysis set by determining a parameter vector **p** that guarantees a good performance. During the *second phase*, the generated parameter vectors are linearly combined into an overall vector with the weights of the linear combination determined by the genetic procedure, so that the best overall performance is reached.

Previous results show that the obtained parameter values are more robust than the ones obtained by traditional experimental design methods. Pepper *et al* (2002) use a similar technique to set the parameter of an annealing-based heuristic for the travelling salesman problem (TSP), and Chandran *et al* (2003) analyze further the possibility of applying a genetic parametric search procedure by introducing a simpler *single-stage* procedure. Their new single-stage algorithm considers the analysis set as a whole, and produces a single parameter vector resulting in high quality results. They apply this approach on a variant of the so-called *demon algorithm* for the TSP. In a more recent paper, Battarra *et al* (2008) propose a single-stage procedure for tuning the parametric CW enhancement of Altınel and Öncan (2005), and report substantial improvements in the efficiency. In this work, we also proceed in this direction and propose a single-stage genetic search procedure for finding the best parameter values of the three-parameter EW enhancement (EW3) with the saving expression (2). It is originally proposed in Öncan and Altınel (2009), where they determine the best values of the parameters using a brute force evaluation procedure within given intervals. We call this enhancement EWBF3 in the sequel.

3.1 Genetic search

The global genetic search procedure is trained with a suitable *analysis set,* consisting of a few instances as a representative subset of the benchmark test instances. Different strategies can be used while forming the analysis set. The choice of the analysis set affects the overall performance of the algorithm. Hence, we performed an investigation based on different strategies. The results of this investigation are reported in Appendix A. According to these results, we can argue that the more 'difficult' the instances in the analysis set are, the better the algorithm performs. However, the choice of a suitable analysis set is not crucial for the overall algorithm behaviour and a larger number of representative instances in the analysis set does not remarkably improve the solution quality.

The performance of a parameter vector **p** is evaluated by means of the fitness function

$$F(\mathbf{p}) = 100\sqrt{\frac{1}{|\mathscr{A}|}\sum_{i \in \mathscr{A}}((D(\mathbf{p}, i)/B(i)) - 1)^2} \tag{3}$$

where $\mathscr{A}$ is the analysis set. For each instance, $i \in \mathscr{A}$, the value $D(\mathbf{p}, i)$ is computed by considering the parameter set **p**; it is the final objective value computed by running the parametric EW with given α, β, and γ values. Then, the ratio between $D(\mathbf{p}, i)$ and the best-known solution value, $B(i)$, is evaluated. A parameter vector **p** with a smaller fitness value $F(\mathbf{p})$ is accepted to be more promising.

The GA begins with an initial population constructed using 21 random individual generated according to a uniform distribution within $[0.1, 2]$ for α, and within $[0, 2]$ for β and γ, which are the ranges considered by Öncan and Altınel (2009) in their tree-parameter brute force evaluation enhancement EWBF3. The fitness function (3) is evaluated for each individual in the initial population over the instances in the analysis set, and these values are stored as the initial fitness values. Then, the algorithm iteratively performs the following three steps until a stopping criterion is satisfied:

1. *Crossover*. The best parameter set in terms of fitness value is selected as the *queen mother*. Each one of the remaining parameter vectors is a *father* and produces a new *child* by combining his vector with that of hers. Each offspring parameter vector is generated using a uniform distribution in the interval bounded by the values in the parameter vectors of the queen mother and of the father. For example, let us assume the queen mother is the parameter vector $(\alpha_1, \beta_1, \gamma_1)$ and the current father is the parameter vector $(\alpha_2, \beta_2, \gamma_2)$. Then,

the offspring parameters are random numbers generated in the interval $[\alpha_1, \alpha_2]$, $[\beta_1, \beta_2]$, and $[\gamma_1, \gamma_2]$.

2. *Mutation*. For each offspring vector, the value of each parameter is mutated with probability $r = 0.2$. The new parameter values are generated by considering a uniform distribution in the original range of each parameter, therefore the parameter value is essentially reset. The fitness function of each child is evaluated by means of (3) and the fitness values are saved.
3. *Selection*. The 21 best individuals, in terms of fitness value, are selected from the 20 new offspring and the current population of 21 parameter vectors, thereby maintaining the same population size.

The steps 1–3 are repeated until the stopping criterion is met. In our implementation, we imposed a 12 min time limit (as in Battarra *et al*, 2008), but if the average fitness value does not decrease for 35 consecutive iterations, the bottom half of the population (consisting of the individuals with higher fitness values) is refreshed randomly. This is done by randomly generating $\mathbf{p} = (\alpha, \beta, \gamma)$ from a uniform distribution within $[0.1, 2]$ for α, and $[0, 2]$ for β and γ.

Note that the mutated offspring generated during the genetic algorithm execution is saved in a pool. At the end of the execution, the best individual in terms of the fitness function is chosen from this pool and included in the parameter set $\mathscr{P}$, where $|\mathscr{P}| = 5$. In addition, $|\mathscr{P}| = 1$ other individuals are selected from the pool and included in $\mathscr{P}$. The selection criterion is based both on the fitness value of the individuals (the parameter sets in $\mathscr{P}$ cannot have fitness value larger than 1.5 times that of the minimum fitness) and their *distances* to the best individual. More specifically, in addition to the one with the best fitness value, say $\mathbf{p_0}$, the $|\mathscr{P}| - 1$ parameter vectors farthest away from $\mathbf{p_0}$ that are within 50% of the fitness of $\mathbf{p_0}$ are also included in $\mathscr{P}$. We calculate the distance $dist(\mathbf{p_1}, \mathbf{p_2})$ between the parameter vectors $\mathbf{p_1} = (\alpha_1, \beta_1, \gamma_1)$ and $p_2 = (\alpha_2, \beta_2, \gamma_2)$ as

$$dist(\mathbf{p}_1, \mathbf{p}_2) = |\alpha_1 - \alpha_2| + |\beta_1 - \beta_2| + |\gamma_1 - \gamma_2|. \tag{4}$$

3.2 Local search improvements

The set $\mathscr{P}$ contains several interesting parameter vectors, having a good fitness value and being far away in parameter space from the best one. For each problem in our benchmark set, the parameter vectors in $\mathscr{P}$ are 'potential starting points' of a LS within the parameter space. More precisely, each of the parameter vectors in the set $\mathscr{P}$ is used to initialize EW3. The neighbourhood is a cube centered around $\mathbf{p} \in \mathscr{P}$, having edge length $l = 0.2$. The performance of EW3 is tested by setting its parameters to 8 vertices and 6 face centers of the cube. The resulting 14 parameter vectors in the neighbourhood are then evaluated. The LS can be iterated by considering the best incumbent in the neighbourhood as the new cube center. An example of iterated LS is depicted in Figure 1, where the parameter vector **p** is shown by the solid black circle.

In the first cube, 14 parameter vectors are considered (6 are the face centers and 8 are the cube vertices) plus the original parameter vector **p**. The best incumbent (the light cross in the bottom figure) becomes the center of a new cube. Note that, the number of parameter vectors to be evaluated in the subsequent cubes is 13. The previous cube's best incumbent becomes one of the vertices of the new search cube (ie the black circle in the figure below is the center of the previous search cube, which is shown with dashed lines). If the number of cubes explored is 1, the overall number of EW3 runs is $1 + 14$ for each starting parameter vector, whereas if the number of explored cubes is $n > 1$, the number of EW3 runs becomes $1 + 14 + 13(n-1)$ for each $\mathbf{p} \in \mathscr{P}$. For example, when $|\mathscr{P}| = 5, 75, 140, 205, 270$, and 335 EW3 runs for $n = 1, 2, 3, 4$, and 5 cubes, respectively. Note the drastic decrease in the number of EW3 runs when $n \leqslant 5$ compared to 8820 runs required by Öncan and Altınel's three-parameter brute force enhancement EWBF3 (Öncan and Altınel, 2009).

This simple LS operator substantially improves the solution quality. Figure 2 represents the average percent improvement with respect to the EW solutions, by considering up to five cubes of LS. The first one or two cubes substantially improve the algorithm performance, whereas a larger number of cubes provides marginal improvement. The GA detects promising areas in the parameters' space fairly well (corresponding to zero cubes in Figure 2). However, when the number of cubes is high the search

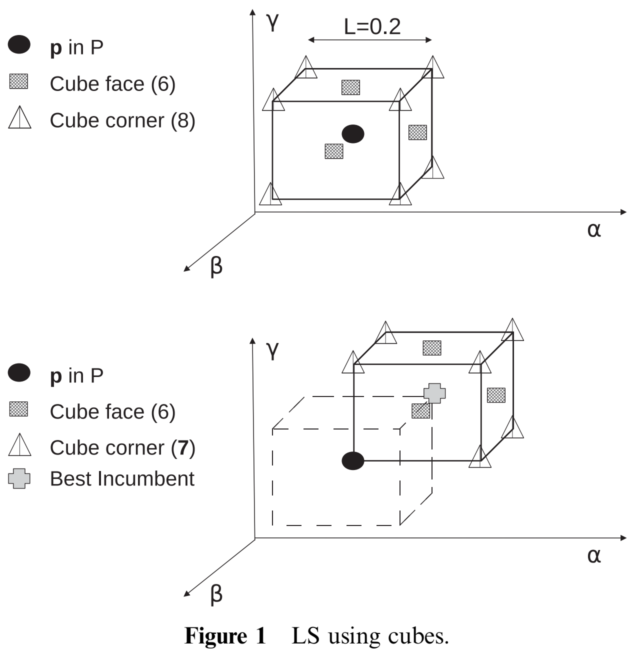

Figure 1 LS using cubes.

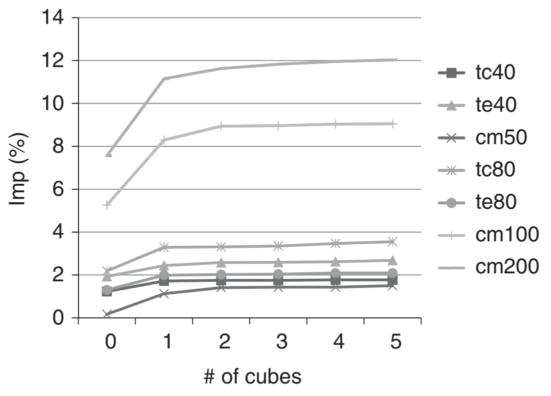

Figure 2 The effect of the number of cubes on the solution quality.

moves away from these regions, which makes the improvement marginal. We present the effect of the number of LS cubes and the choice of analysis set in Table A1 of Appendix A.

On the basis of the results, we can say that it make sense to use two cubes for LS for each parameter set.

In order to confirm the robustness of our LS operator, we compared its performance with two well-known LS algorithms, namely the algorithms by Hooke and Jeeves (1961) and Nelder and Mead (1964). Both algorithms have been implemented as described in Bazaraa *et al* (2005) and were executed for the same number of iterations as the cubic LS operator, so as to perform a fair comparison. The improvement obtained by considering these two LS operators is quite small (ie, at most an average improvement of 0.38% over all instances). A more detailed description of our computational experience can be found in Battarra (2010). Moreover, the two alternative LS algorithms require the tuning of their own parameters, in contrast to the simplicity of our evolutionary approach. They also increase the overall computing time. Moreover, the unimodality assumption of most of the interval search methods is not satisfied by our parametric search. Therefore, our cubic LS operator represents a good compromise between the algorithm's simplicity and the solution quality.

3.3 Further improvements using randomized prohibition

The storage of the savings allows the consideration of different selection rules. One approach is to adopt a random selection strategy from the elements of the list as done previously in the randomized enhancements of CW by Daskin (2002) and Bard *et al* (1998).

The *randomized prohibition* (RP) is a multi-start EW algorithm, in which, given a saving list, the indicated merger at each step (determined by the best savings) is accepted with 95% probability, otherwise it is skipped. The disregarded savings are not likely the same from one iteration to the next, giving rise to an effective iterative approach. The algorithm is straightforward to code and it can be slightly faster since the initial saving list has to be evaluated and ordered once, not at each iteration as a regular savings algorithm does.

We thought that a fair comparison with the LS algorithm would be obtained by comparing the same number of EW iterations. As reported in Section 3.2, the LS algorithm marginally benefits from the latter three cubes. We substitute these cubes with the same number of RP executions, such as $13 \times 3 \times 5 = 195$ EW iterations, and we compared the performance of these two methods.

For each instance, the best three parameter sets found during the first two LS cubes are collected. These parameter sets are defined with the same criteria used for defining $\mathscr{P}$, such a combination of fitness function and distance with respect to the best parameter (see Section 4). Each parameter set produces an EW3 saving list, and the RP algorithm is run for $195/3 = 65$ iterations, for each list.

Computational results are reported in Section 4, however, in order to verify the consistency of RP, we conducted additional experiments. We proved that RP is effective if initialized with high quality saving lists such as the ones produced during the LS step. A summary of these experiments is given in Appendix B.

4 Experimental results

In order to demonstrate the improvement in efficiency while preserving the simplicity and solution quality of the EW enhancement, we conducted computational experiments with EWBF3, the new genetic enhancement (EWG), the new genetic enhancement combined with LS (EWLS), and the new genetic enhancement combined with both LS and RP (EWR).

4.1 Test environment

The instances we consider are from the *tc40*, *te40*, *tc80*, *te80*, *cm50*, *cm100*, and *cm200* test problem sets downloaded from Beasley's ORLIB library, at http://people.brunel.ac.uk/~mastjjb/jeb/info.html. The *tc* instances have the central vertex (ie, the root of the final tree) located in a central position with respect to the other vertices. The *te* instances have the central vertex in an eccentric position with respect to the other vertices (ie, in a corner). In both cases, the vertices have unit demands, which is not the case for the *cm* instances. All of the vertices do not necessarily have unit demands. Besides the capacity bounds are significantly larger. The last two or three digits are the number of vertices. Each set consists of three groups of five instances of the same capacity limitations. This makes 15 test problems per set and 105 test problems in total. The characteristics of the instances are summarized in Table 1.

Table 1 Summary of the instances characteristics

	n	*Q*	*d*	*Central vertex*
tc	40	3, 5, 10	Unit weights	Central location
	80	5, 10, 20	Unit weights	Central location
te	40	3, 5, 10	Unit weights	Eccentric location
	80	5, 10, 20	Unit weights	Eccentric location
cm	50	200, 400, 800	Non-unit weights	Random location
	100	200, 400, 800	Non-unit weights	Random location
	200	200, 400, 800	Non-unit weights	Random location

The best known solutions are obtained from Uchoa *et al* (2008), except *tc80* and *te80*, for which the ones given in Öncan and Altınel (2009) are considered. In fact, it is noticed in the ORLIB that some of the 80 vertex instances have been slightly modified by Uchoa *et al* (2008) in order to obtain symmetric cost matrices. We did not consider neither these modified instances, nor the new ones proposed by Uchoa *et al* (2008), in order to be consistent with Öncan and Altınel (2009). Finally, the experimental testing is conducted on an AMD Athlon 64X2 Dual, 2.20 GHz PC, and our algorithms are coded in C++ programming language.

4.2 Validation of the re-implementation

We have re-implemented the EW algorithm and its parametric enhancement EW3, in order to provide an objective assessment of the algorithms' performance. The new codes are more efficient, since they use the *component oriented* data structures, by Dai and Fujino (2000) during the tree merge operations, which cause a substantial decrease in the number of saving updates. Moreover, an early stopping criterion for the EW algorithm has been established, based on a bin packing lower bound. In fact, the number of subtrees in any feasible solution is greater than or equal to

$$LB = \left\lceil \frac{\sum_{i \in V} d_i}{Q} \right\rceil. \tag{5}$$

During the EW execution, if the number of subtrees is equal to LB, the algorithm can be stopped. When customers require unit demand (ie, *tc* and *te* sets), this early termination condition occurs quite early and most of the $(n-1)^2/2$ merge operations can be disregarded.

In order to verify our codes, we compared the EW implementations with the parametric enhancements by Öncan and Altınel (2009). In Table 1, results are reported for each problem set. The 'Dev.(%)' columns are the average percent deviations over each set with respect to the best-known solutions, calculated according to the formula $100 \times ((z-z^*)/z^*)$, where z is the current solution value and z^* is the best known value reported in the literature. The 'Imp(%)' columns report the average percentage improvement introduced by a parametric enhancement over EW, for each set. The 'Time (s)' columns are the total CPU times in seconds, for each set. The computing time is the sum of $20, 20 \times 21 = 420$ and $20 \times 21 \times 21 = 8820$ EW executions, when $\beta = \gamma = 0$ (EWBF1), when $\gamma = 0$ (EWBF2), and when EWBF3 is applied, respectively. Finally, the last row of the table includes column averages.

The overall average percent deviations from the best-known solution values is 11.89% for the new EW implementation, compared to 10.15% for the Oncan and Altınel's implementation. This becomes 8.16%, 6.60%, and 5.99% for the new implementations of the brute force parametric enhancements EWBF1, EWBF2, and EWBF3. They are, respectively, 9.083%, 6.456%, and 5.876% according to Öncan and Altınel (2009). Notice that the parametric enhancements improve the solution quality performance of the EW algorithm quite substantially. In fact, the deviation with respect to the best-known solutions is usually halved. However, the 'brute force' parameter search procedure increases the computing times considerably. For example, the total computing time required to solve a single instance in the largest set (ie, *cm200*) using EWBF3 is almost 1 h. Moreover, Table 2 reports the maximum (ie, Max) and the standard deviation (ie, Std. Dev.) over the Dev. (%), the Imp. (%), and the Time (s). As expected, the maximum deviations and the standard deviations become smaller as the number of iterations increases (ie, EWBF3), whereas the maximum improvements become larger.

4.3 Evolutionary versus brute force parametric search

The results obtained by considering EWBF3, EWG, EWLS, and EWR are summarized in Table 3. For each test set, the average percentage improvements with respect to EW (Imp (%)) and the total CPU time in seconds (Time (s)) are reported.

The bottom lines report the average values (Average), the maximum values (Max), and the standard deviations (Std. Dev.) of the average improvements and the computing times, over all instances. The last line reports the overall computing time (Total).

The re-implementation of EWBF3 results in an average improvement of 4.82% *versus* EW, but it is obtained by executing EW3 with 8820 distinct parameter vectors. The total computing time is 59403.50 s. The average percentage improvement is 2.81%, by executing EW3 with the 5 parameter vectors in $\mathscr{P}$, namely EWG. Note that EWG does not consider any refinement step and the general/deterministic analysis set is chosen (see Appendix A). The EWG execution requires 5 EW3 runs, resulting in a total computing time of 753.73 s (note that 720 s of the 753.73 are spent on the genetic parametric search). By introducing five cubes of LS for each $\mathbf{p} \in \mathscr{P}$ and maintaining the same analysis set, the EWLS average improvement increases to 4.67%, which is very close to the improvement by the brute

Table 2 The solution quality and efficiency of the EW re-implementation

Instance set	*EW*		*EWBF1*			*EWBF2*			*EWBF3*		
	Dev. (%)	*Time (s)*	*Dev. (%)*	*Imp. (%)*	*Time (s)*	*Dev. (%)*	*Imp. (%)*	*Time (s)*	*Dev. (%)*	*Imp. (%)*	*Time (s)*
tc 40	3.08	0.03	1.83	1.20	0.83	1.25	1.77	17.52	1.14	1.87	371.22
te 40	5.98	0.03	3.86	1.98	0.80	3.11	2.67	17.56	2.91	2.86	372.42
tc 80	6.73	0.22	4.89	1.71	5.41	2.83	3.60	132.59	2.67	3.76	3102.33
te 80	7.81	0.24	6.52	1.18	5.74	5.24	2.34	137.20	5.01	2.55	3180.34
cm 50	3.68	0.06	2.94	0.70	1.42	2.44	1.18	33.38	1.95	1.66	748.22
cm 100	24.68	0.44	15.36	7.32	11.95	13.40	8.78	262.38	11.70	10.09	5680.78
cm 200	31.24	3.17	21.75	7.14	81.52	17.95	9.86	1921.72	16.57	10.91	45948.19
Average	11.89	0.60	8.16	3.03	15.38	6.60	4.31	360.34	5.99	4.82	8486.22
Max	48.53	0.25	35.77	17.26	6.06	25.83	17.59	141.58	25.00	18.57	3390.50
Std. dev.	12.93	0.07	8.96	3.59	1.83	7.32	4.16	43.05	6.69	4.48	1031.98

Table 3 The solution quality and the efficiency of the evolutionary enhancements

Instance set	*EWBF3*		*EWG*		*EWLS*		*EWR*	
	Imp (%)	*Time (s)*	*Imp (%)*	*Time (s)*	*Imp (%)*	*Time (s)*	*Imp (%)*	*Time (s)*
tc 40	1.87	371.22	1.23	0.20	1.77	14.27	2.13	12.72
te 40	2.86	372.42	1.93	0.22	2.68	14.28	3.22	12.81
tc 80	3.76	3102.33	2.19	1.55	3.55	110.30	3.95	104.81
te 80	2.55	3180.34	1.30	1.55	2.09	110.36	2.75	101.91
cm 50	1.66	748.22	0.17	0.41	1.50	26.23	2.28	24.00
cm 100	10.09	5680.78	5.26	3.44	9.05	243.75	11.20	231.11
cm 200	10.91	45948.19	7.56	26.36	12.03	1867.36	13.00	1786.73
Average	4.82	8486.21	2.81	4.82	4.67	340.94	5.50	324.87
Max	18.57	3390.50	17.59	2.08	19.87	143.17	21.82	138.41
Std. Dev.	4.48	1031.98	4.27	0.60	4.98	42.33	5.86	40.57
Total	—	59403.50	—	753.73	—	3106.55	—	2994.09

force approach. The overall number of EW3 runs is 335 for each instance, resulting in a total computing time of 3106.55 s, 720 s of which are spent on the genetic search. In our last experiment, namely EWR, the last three cubes of LS are replaced by the RP method, by preserving the same overall number of EW3 iterations of EWLS. The average improvement is 5.50% and the overall computing time is 2994.09 s. Note that the EWLS computing time is larger than that of EWR, even if the number of iterations is the same. This is probably due to the fact that the RP algorithm computes the initial saving list once, whereas the saving list has to be computed and ordered every time a new parameter set is considered in EWLS (ie, a vertex or face of a cube). With respect to both time and solution quality, EWR outperforms brute force parametric enhancement EWBF3.

The variability of EWR, because of its random nature, has been tested by executing the algorithm with 10 different randomly generated seeds. The mean average improvement of the Imp (%) over the 10 executions is 5.38%, the maximum average improvement is 5.59%, and the minimum average improvement is 5.19%. These results confirm the robustness of the EWR approach.

5. Conclusions

The EW heuristic has low solution quality, very high speed, and considerable simplicity. In the recent work by Öncan and Altınel (2009), the authors attempted to increase the solution quality of the EW without harming its speed and simplicity very much by introducing new parameters and terms. The result was a substantial increase in the solution quality without affecting its simplicity. However, there was a considerable decrease in the efficiency because of the search effort spent on the determination of good parameter values. In this work, we proposed a GA-based procedure to tune efficiently a three-parameter enhancement of the EW heuristic.

GAs also proved in this context to be effective tools in tuning parametric algorithms. In fact, this paper shows not only the robustness of the GA proposed in Battarra *et al*

(2008), but it also demonstrates that high quality results can be achieved by combining the parameters proposed by a genetic approach, or any other effective algorithm from the literature, with some effective refinement steps.

Note that a few iterations of a LS algorithm and a prohibition algorithm increased the solution quality substantially, without compromising the simplicity or the speed of the overall method. These refinements steps are able to capture the specific characteristics of each problem, not considered in the genetic search.

Moreover, it is interesting to observe that only a few steps of LS are sufficient; in fact, the third, the fourth, and the fifth cube provide a negligible improvement. The LS algorithm plays a subtle role in our evolutionary approach, it does not guide the search far away in the parameter space, but it leads to slight improvements. The solutions proposed by the GA proved to be promising 'seeds' in the solution space.

Finally, we observe that the sequence of two simple refinement steps (ie, LS and RP) provided a more exhaustive search in the parameter space. The strength of the two approaches is synergistic and the overall method benefits from both.

Possible directions of further research could be the analysis of the performance of alternative algorithms for obtaining promising seeds in the parameter space. For example, neural networks could be employed instead of our GA. Moreover, it would be interesting to analyze the performance of our evolutionary approach, when applied to new and different tuning problems.

In conclusion, we proposed a three-stage evolutionary approach for tuning a parametric variant of the EW algorithm. A genetic algorithm provides robust 'seeds' in the parameter space for a large set of instances, whereas the sequence of a LS algorithm and a RP algorithm provides the refinements necessary to capture the characteristics of each problem and improve the solution quality. The evolutionary approach is able to produce high quality results in a limited amount of computing time and without harming the simplicity of the method.

Acknowledgements—The authors acknowledge the supports of Ministero dell'Istruzione, dell'Università e della Ricerca, Italy, Galatasaray University Scientific Research Projects Grant no: 07.402.014, and Boğazici University Scientific Research Projects Grant no: 08A304D.

References

Aha DW, Kibler D and Albert MK (1991). Instance-based learning algorithms. *Mach Learn* **6**: 37–66.

Ahuja RK, Orlin JB and Sharma D (2003). A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Opns Res Lett* **31**: 185–194.

Altınel IK and Öncan T (2005). A new enhancement of the Clarke and Wright savings heuristic for the capacitated vehicle routing problem. *J Opl Res Soc* **56**: 954–961.

Amberg A, Domschke W and Voss S (1996). Capacitated minimum spanning trees: Algorithms using intelligent search. *Comb Optim: Theory Pract* **1**: 9–39.

Bard JF, Huang L, Jaillet P and Dror M (1998). A decomposition approach to the inventory routing problem with satellite facilities. *Transportat Sci* **32**: 189–203.

Battarra M (2010). *Exact and heuristic algorithms for routing problems*. PhD thesis, Università di Bologna.

Battarra M, Golden BL and Vigo D (2008). Tuning a parametric Clarke-Wright heuristic via a genetic algorithm. *J Opl Res Soc* **59**: 1568–1572.

Bazaraa MS, Sherali HD and Shetty CM (2005). *Nonlinear Programming: Theory and Algorithms*. Wiley—Interscience: Hoboken, NJ.

Chandran B, Golden BL and Wasil E (2003). A computational study of three demon algorithm variants for solving the traveling salesman problem. In: Bhargava HK (eds). *Computational Modeling and Problem Solving in the Networked World*. Kluwer Academic Publisher: NY, pp 155–175.

Clarke G and Wright JW (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Opns Res* **12**: 568–581.

Cordeau J-F, Gendreau M, Laporte G, Potvin J-Y and Semet F (2002). A guide to vehicle routing heuristics. *J Opl Res Soc* **53**: 512.

Cortes C and Vapnik V (1995). Support-vector networks. *Mach Learn* **20**: 273–297.

Dai HK and Fujino S (2000). On designing constrained local access networks. In: Sudborough IH and Hsu DF (eds). *Proceedings of the 2000 International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN00)*. IEEE Computer Society Press: Los Alamitos, CA, USA, pp 167–176.

Daskin MS (2002). Randomized saving algorithms. Private Communication.

Esau LR and Williams KC (1966). On teleprocessing system design, part II: A method for approximating the optimal network. *IBM Syst J* **5**: 142–147.

Gavish B (1991). Topological design of telecommunication networks-local access design methods. *Ann Opns Res* **33**: 17–71.

Golden BL, Pepper J and Vossen T (1998). Using genetic algorithms for setting parameter values in heuristic search. In: Dagli C, Akay A, Buczak M, Ersoy O and Fernandez B (eds). *Intelligent Engineering System through Artificial Neural Networks*. Vol. 8, ASME Press: NY, pp 239–245.

Hooke R and Jeeves TA (1961). Direct search solution of numerical and statistical problems. *J Assoc Comput Mach* **8**: 212–229.

Morgan J and Sonquist J (1963). Problems in the analysis of survey data, and a proposal. *J Am Stat Assoc* **58**: 415–434.

Nelder JA and Mead R (1964). A simplex method for function minimization. *Comput J* **7**: 308–313.

Öncan T and Altınel IK (2009). Parametric enhancements of the Esau-Williams heuristic for the capacitated minimum spanning tree problem. *J Opl Res Soc* **60**: 259–267.

Papadimitriou CH (1978). The complexity of the capacitated tree problem. *Networks* **8**: 217–230.

Park M-W and Kim Y-D (1998). A systematic procedure for setting parameters in simulated annealing algorithms. *Comput Opns Res* **24**: 207–217.

Parson R and Johnson M (1997). A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly. *Am J Math Manage Sci* **17**: 369–396.

Patterson R and Pirkul H (2000). Heuristic procedure neural networks for the CMST problem. *Comput Opns Res* **27**: 1171–1200.

Patterson R, Pirkul H and Rolland E (1999). A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. *J Heuristics* **5**: 159–180.

Pepper J, Golden B and Wasil E (2002). Solving the traveling salesman problem with annealing-based heuristics: a computational study. *IEEE Trans Syst Man Cybern A* **32**: 72–77.

Quinlan JR (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann: San Francisco, CA.

Robertson W, Golden BL, Runger GC and Wasil E (1998). Neural network models for initial public offerings. *Neurocomputing* **18**: 165–182.

Rumelhart DE, Hinton GE and Williams RJ (1986). Learning internal representations by error propagation. *Parallel Distrib Process* **1**: 318–362.

Uchoa E, Fukasawa R, Lysgaard J, Pessoa A, Poggi da Aragão M and An-drade D (2008). Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Math Program, Ser. A* **112**: 443–472.

Van Breedam A (1996). *An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem*. Technical report, RUCA Working Paper 96/14, Faculty of Applied Economics, University of Antwerp, Belgium.

Appendix A

Analysis set choice

The strategy used to select the analysis set can affect the performance of a GA. We test four analysis set selection strategies for EWG, EWLS, and EWR.

The first analysis set is obtained through the *general and deterministic* (G/D) strategy. It is general, because it includes instances from all test problem sets. The first two instances with the two smallest capacities are deterministically selected from the sets *cm50*, *cm100*, and *cm200*, which totally makes six instances. We qualify these three sets as more *difficult* than the problem sets *tc40*, *tc80*, *te40*, and *te80*, since the EW and its parametric enhancement end up with larger deviations on them.

The remaining three strategies are *dedicated* in the sense that they determine the analysis instances particularly from one of the three problem groups *tc*, *te*, and *cm*. We believe that this can give the genetic search procedure a higher chance to learn better the set's structural characteristics, but at the expenses of three executions of the GA. Moreover, we use three rules to determine the dedicated instances in the analysis set. The first rule is *deterministic*. We choose the first two instances from each one of the two smallest capacity groups, as in G/D; this is what we call D/D. The second rule selects the *most difficult instances* and it is denoted by D/M. The two instances for which EW3 produces the smallest improvements over EW are selected from the test problem groups. Finally, the third rule is based on the *random* selection of two instances from each group, and it is referred as D/R. Notice that the size of the analysis sets are four for *tc* and *te,* and six for the *cm* instances, for any analysis set strategy. The five parameter vectors forming the set $\mathscr{P}$ are listed in Table A1, for the four analysis set selection strategies.

The average percentage deviations and CPU times computed running EWBF3 and EWLS are summarized in Table A2. Note that they are obtained according to four distinct analysis set strategy for $0, 1, 2, 3, 4$, and 5 search

Table A1 Results of the genetic search: parameters ($\mathcal{P}$) *versus* analysis sets

Strategy	*tc*			*te*			*cm*		
	α	β	γ	α	β	γ	α	β	γ
	1.8176	0.0654	0.5100	1.8176	0.0654	0.5100	1.8176	0.0654	0.5100
	1.1590	0.1470	1.9512	1.1590	0.1470	1.9512	1.1590	0.1470	1.9512
G/D	0.6423	0.1143	0.0098	0.6423	0.1143	0.0098	0.6423	0.1143	0.0098
	1.5894	0.2391	1.7056	1.5894	0.2391	1.7056	1.5894	0.2391	1.7056
	1.5636	0.1470	1.8867	1.5636	0.1470	1.8867	1.5636	0.1470	1.8867
	1.3868	0.3828	0.3057	1.2439	0.3036	1.6855	1.6605	0.2555	0.0831
	1.9040	0.7834	0.3977	1.6494	0.2161	0.0727	0.7322	0.2525	0.6862
D/D	0.9468	0.2382	0.0093	0.9374	0.2828	0.0150	0.5470	0.2556	0.0835
	1.9263	0.3654	0.6699	0.9218	0.2104	0.1178	0.5471	0.2555	0.0830
	1.8313	0.5427	0.0991	0.9374	0.3045	0.0150	0.5591	0.2555	0.0832
	1.4759	0.1569	0.2416	0.7883	0.2868	0.6258	1.9530	0.3279	0.7855
	1.9043	0.7340	1.7159	1.9286	1.4390	1.3930	0.6804	0.0864	1.7958
D/M	1.9893	1.5519	0.7553	1.5384	0.7017	1.7770	0.6357	0.1584	0.0357
	0.6111	0.2016	1.7518	1.9326	1.0202	0.2324	0.6357	0.1584	0.0357
	0.4866	0.1553	1.5659	1.7314	0.2875	1.8910	0.6357	0.1584	0.0357
	1.8176	0.0654	0.5100	1.8339	0.1445	0.7756	1.7862	0.1201	0.7431
	1.1590	0.1470	1.9512	0.7443	0.1141	0.2393	0.6423	0.1143	0.0098
D/R	0.6423	0.1143	0.0098	1.5395	0.2159	1.9500	1.1590	0.1152	1.9512
	1.5894	0.2391	1.7056	1.0072	0.0910	1.4902	1.0871	0.1087	1.8619
	1.5636	0.1470	1.8867	1.8898	0.1364	1.9852	1.3512	0.1894	1.7056

Table A2 The effect of the analysis set and the number of search cubes on the performance of EWLS

		0 *cubes*	1 *cube*	2 *cubes*	3 *cubes*	4 *cube*	5 *cubes*	*Average*
	EWBF3				1.87 (371.22)			
	G/D	1.23	1.72	1.75	1.75	1.77	1.77	1.66
	D/D	1.29	1.64	1.69	1.69	1.71	1.81	1.64
tc40	D/M	1.26	1.68	1.76	1.76	1.78	**1.87**	1.69
	D/R	1.27	1.65	1.73	1.78	1.78	1.78	1.67
	Time (s)	0.21	3.10	5.97	8.85	11.73	14.53	7.40
	EWBF3				2.86 (372.42)			
	G/D	1.93	2.44	2.58	2.59	2.62	2.68	2.47
	D/D	2.16	2.40	2.48	2.55	2.58	2.58	2.46
te40	D/M	2.12	2.54	2.71	2.71	2.71	2.72	2.59
	D/R	1.60	2.42	2.48	2.61	2.61	2.67	2.40
	Time (s)	0.21	3.11	6.03	9.00	11.79	14.72	7.48
	EWBF3				3.76 (3102.33)			
	G/D	2.19	3.29	3.31	3.35	3.47	3.55	3.19
	D/D	2.94	3.39	3.47	3.48	3.48	3.54	3.39
tc80	D/M	2.77	3.44	3.50	3.58	3.60	3.62	3.42
	D/R	2.54	3.34	3.42	3.45	3.47	3.49	3.28
	Time (s)	1.60	23.95	46.23	68.20	90.41	112.34	57.12
	EWBF3				2.55 (3180.34)			
	G/D	1.30	1.99	2.02	2.04	2.07	2.09	1.92
	D/D	1.35	2.07	2.32	2.40	2.44	2.46	2.17
te80	D/M	1.58	2.07	2.24	2.46	2.52	2.54	2.23
	D/R	1.42	2.02	2.15	2.27	2.31	2.36	2.09
	Time (s)	1.59	23.84	46.06	68.23	90.29	112.71	57.12
	EWBF3				1.66 (748.22)			
	G/D	0.17	1.13	1.41	1.43	1.43	1.50	1.18
cm50	D/D	0.17	1.13	1.41	1.43	1.43	1.50	1.18
	D/M	0.30	1.17	1.43	1.46	1.54	1.54	1.24
	D/R	0.19	1.37	1.44	1.65	**1.70**	**1.70**	1.34
	Time (s)	0.39	5.54	10.75	15.94	21.12	26.28	13.34
	EWBF3				10.09 (5680.78)			
	G/D	5.26	8.29	8.94	8.96	9.04	9.05	8.26
cm100	D/D	5.26	8.29	8.94	8.96	9.04	9.05	8.26
	D/M	3.43	8.89	9.30	9.33	9.33	9.63	8.32
	D/R	3.17	7.95	8.61	8.91	9.23	9.23	7.85
	Time (s)	3.47	51.82	99.37	147.79	195.47	243.61	123.59
	EWBF3				10.91 (45948.19)			
	G/D	7.56	**11.15**	**11.63**	**11.83**	**11.96**	**12.03**	**11.03**
cm200	D/D	7.56	**11.15**	**11.63**	**11.83**	**11.96**	**12.03**	**11.03**
	D/M	4.59	**11.19**	**11.77**	**12.07**	**12.07**	**12.22**	10.65
	D/R	4.26	9.72	10.30	**11.38**	**11.68**	**11.77**	9.85
	Time (s)	26.33	393.87	760.88	1129.28	1499.22	1867.87	946.24

cubes. The first column denotes the sets of test problem sets. The second column includes the four analysis set selection strategies. Notice that the average relative percent improvements made on EW and total CPU time in seconds are reported; they are computed with the new re-implementations.

The results show that a dedicated analysis set can produce slightly better results than the general analysis set. In some cases, the dedicated algorithm produces even better results than EWBF3; these are marked in bold in Table A2. Moreover, the improvements seem to be larger when the instances solved are large (namely, *cm200*). The most effective analysis set choice seems to be the D/M, but the results obtained with other strategies are not substantially different.

Moreover, results obtained by increasing the number of cubes considered in the LS step are quite interesting. In fact, the first two cubes introduce major improvements in the solution quality, whereas a higher number of cubes result in marginal improvements for a relatively higher computational cost. This behaviour shows that EWG is able to detect promising areas in parameter space fairly well. Whenever we move unnecessarily far away from these regions, by considering a higher number of cubes, the performance of the method does not improve anymore.

Appendix B

Initialization of the randomized prohibition heuristic

The RP heuristic is effective even by considering a very limited number of EW3 iterations, as reported in Section 4. However, we think we should test the performance of the standalone method and we should verify if the solution quality is dependent or not on the saving list quality.

The first experiment consists in initializing RP with a random saving list. Three random parameters are generated in $[0, 2]$ and the corresponding EW3 saving list initializes 195 iterations of RP. The solution quality obtained is definitely not satisfactory, in fact a single run of EW is on the average better than 195 iterations of the randomly initialized RP algorithm.

On the other hand, if the EW3 saving list is initialized with the EW parameters (ie, $\alpha = 1$, $\beta = \gamma = 0$), the RP's average improvement is 3.86% with respect to a single EW execution. If the saving list is computed by considering higher quality parameters, such as the three best parameter vectors obtained after the genetic algorithm (ie, the first three parameter sets in $\mathscr{P}$) and each saving list initializes $195/3 = 65$ RP iterations, the average improvement increases to 4.16%. RP proves to be strongly dependent on the quality of the saving list considered. When the saving list is initialized by even better parameters, as the best parameter set found by EWLS, even with 65 RP iterations are performed, the average improvement is 5.11%. If the parameter vectors are the best three found and 65 RP iterations are run for each corresponding saving list, the average improvement increases to 5.50%.

For the sake of completeness, we pushed this latter configuration by considering a larger number of iterations. We executed 1500 and 3000 iterations of the RP algorithm. The average percentage improvement grows to 5.99% for 3000 iterations, but the improvements beyond 1500 iterations are considerably smaller.

Received May 2010;
accepted November 2010