Cite as: L.F. Morán-Mirabal, J.L. González-Velarde, and M.G.C. Resende, "Randomized heuristics for the family traveling salesperson problem," International Transactions in Operational Research, vol. 21, pp. 41-57, 2014.

RANDOMIZED HEURISTICS FOR THE FAMILY TRAVELING SALESPERSON PROBLEM

L.F. MORÁN-MIRABAL, J.L. GONZÁLEZ-VELARDE, AND M.G.C. RESENDE

ABSTRACT. This paper introduces the FAMILY TRAVELING SALESPERSON PROBLEM (FTSP), a variant of the generalized traveling salesman problem. In the FTSP, a subset of nodes must be visited for each node cluster in the graph. The objective is to minimize the distance traveled. We describe an integer programming formulation for the FTSP and show that the commercial-grade integer programming solver CPLEX 11 can only solve small instances of the problem in reasonable running time. We propose two randomized heuristics for finding optimal and near-optimal solutions of this problem. These heuristics are a biased random-key genetic algorithm and a GRASP with evolutionary path-relinking. Computational results comparing both heuristics are presented.

1. INTRODUCTION

This paper presents a variant of the well-known generalized traveling salesperson problem (GTSP) (Srivastava et al., 1970) which we call the *family traveling salesperson problem* (FTSP). Despite the fact that many TSP-related problems have been widely studied in the literature (Applegate et al., 2011; Feremans et al., 2003; Gutin and Punnen, 2002; Tsitsiklis, 1992), the FTSP variant is new and appears to be computationally challenging. Golden et al. (2012) recently proposed the generalized covering salesman problem (GCSP). In the GCSP a minimum length tour is sought, such that a subset of nodes visited on the tour "covers" all the nodes in a graph, i.e. are within a given covering distance from them. The GTSP is a special case of the GCSP and consequently the FTSP is also a variant of this problem.

The FTSP is motivated by an application for order picking in warehouses. Although extensive work has been done concerning warehouse design and control, order picking is usually modeled using a traveling salesman problem or a Steiner traveling salesman problem, where similar products are stored together (Koster et al., 2007; Ratliff and Rosenthal, 1983; Theys et al., 2010). This may not be the case in modern warehouses, where technologies such as RFID allow for product localization, and hence, similar products may be stored separately.

In the FTSP we wish to visit a prescribed number nv_l of nodes that must be selected from a set $\mathcal{F}_l$, where l is the index of K families. We are interested in finding a minimum distance tour that visits all the prescribed nodes for all families.

For any given FTSP instance, consider the complete graph $G = (\mathcal{N} \cup \{0\}, E)$ where each family l has nf_l nodes, and 0 is the origin of the tour. The edge-set E

Date: May 4, 2013.

Key words and phrases. Family traveling salesperson problem, distance minimization, randomized heuristics, biased random-key genetic algorithm, GRASP.

Technical report, Tecnológico de Monterrey and AT&T Labs Research.

indexes the distances between pairs of nodes i and j such that $d(i,j) > 0$ where $i \neq j$. Additionally, a number of required visits to each family nv_l is given such that $KN = \sum_{l=1 \to K} nv_l$. Note that the family traveling salesperson problem can be reduced to the NP-hard generalized traveling salesperson problem (Garey and Johnson, 1979) when the number of visits to each family is exactly one, and hence, the FTSP is also NP-hard. Figure 1 shows the optimal solution for an example instance with $|\mathcal{N}| = 29$, $K = 4$, and $KN = 18$. The 18 visits are such that $nv_1 = 6$, $nv_2 = 6$, $nv_3 = 1$, and $nv_4 = 5$.

This problem can be formulated as a binary integer program (BIP). Define the binary variables $x_{i,j} = 1$ if node j is visited immediately after node i, and $x_{i,j} = 0$ otherwise. The objective (1) is to minimize the distance traveled when visiting the selected family members. Constraints (2–3) force the tour to start and end at node 0. Constraints (4–5) reflect the fact that at most one arc must enter and leave each node. The number of arcs used in the solution is established by constraint (6), while constraints (7–8) establish the number of arcs to enter and leave each of the families. Flow conservation is expressed by constraint (9) and constraints (10) are the sub-tour elimination constraints. The binary nature of the variables is established in (11).

$$\min \sum_{i \in \mathcal{N} \cup \{0\}} \sum_{j \in \mathcal{N} \cup \{0\}} d_{i,j} \times x_{i,j} \tag{1}$$

$$\sum_{i \in \mathcal{N}} x_{0,i} = 1 \tag{2}$$

$$\sum_{j \in \mathcal{N}} x_{j,0} = 1 \tag{3}$$

$$\sum_{i \in \mathcal{N} \cup \{0\}} x_{i,j} \leq 1 \quad \forall j \in \mathcal{N} \tag{4}$$

$$\sum_{j \in \mathcal{N} \cup \{0\}} x_{i,j} \leq 1 \quad \forall i \in \mathcal{N} \tag{5}$$

$$\sum_{i \in \mathcal{N} \cup \{0\}} \sum_{j \in \mathcal{N} \cup \{0\}} x_{i,j} = KN + 1 \tag{6}$$

$$\sum_{i \in \mathcal{N} \cup \{0\}} \sum_{j \in \mathcal{F}_l} x_{i,j} = nv_l \quad for \;\; l = 1, 2, \ldots, K \tag{7}$$

$$\sum_{i \in \mathcal{F}_l} \sum_{j \in \mathcal{N} \cup \{0\}} x_{i,j} = nv_l \quad for \;\; l = 1, 2, \ldots, K \tag{8}$$

$$\sum_{i \in \mathcal{N} \cup \{0\}} x_{i,j} - \sum_{i \in \mathcal{N} \cup \{0\}} x_{j,i} = 0 \quad \forall j \in \mathcal{N} \tag{9}$$

$$\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} x_{i,j} \leq |\mathcal{S}| - 1 \quad \mathcal{S} \subseteq \mathcal{N} \cup \{0\} \tag{10}$$

$$x_{i,j} \in 0, 1 \quad \forall i, j \in \mathcal{N} \tag{11}$$

The remainder of this paper is organized as follows. In Section 2 we use a well-known integer programming solver to address a set of benchmark instances. Then, in Section 3 we describe two randomized heuristics for the FTSP, and experimental

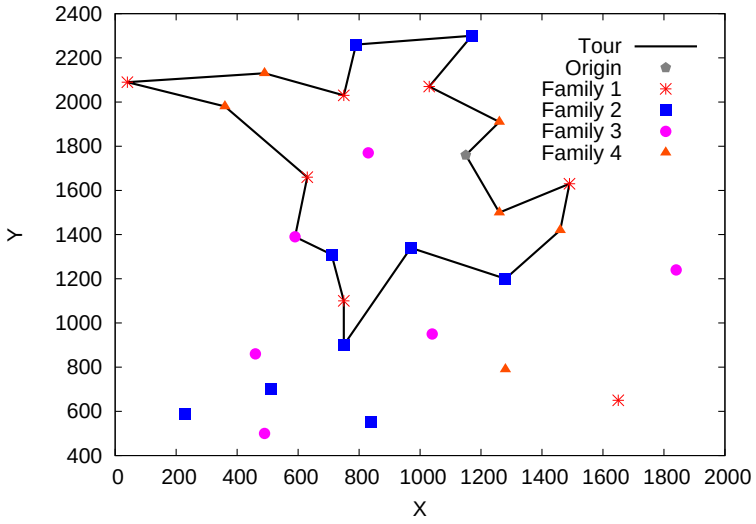

FIGURE 1. Example of instance with 29 nodes 4 families and 18 visits.

results are presented in Section 4. Finally, in Section 5 we present concluding remarks.

2. Solving the mixed integer programming formulation

A subset of the benchmark instances described later in Section 4 are used to evaluate the BIP formulation presented in Section 1. Although the TSP solver *Concorde* (D.L. Applegate and R.E. Bixby and V. Chvátal and W.J. Cook, 2013) has proven to be a state-of-the-art TSP solver (Hahsler and Hornik, 2007), it cannot solve an FTSP instance directly. Hence, the integer programming solver CPLEX 11 is used in the BIP evaluations.

Table 1 shows the results of running CPLEX 11 on 13 FTSP instances having the following triads of $|\mathcal{N}|$ nodes, K families and KN visits: (14,3,6), (14,3,10), (14,3,4), (29,4,16), (29,4,17), (29,4,18), (48,5,34), (48,5,25), (48,5,15), (127,10,62), (127,10,85), (127,10,60), and (280,20,179). For each instance, the table lists the instance name, the number of nodes, the number of families, the number of visits, the total distance in the best known solution for that instance, the total distance in the best solution found by CPLEX, and the time in CPU seconds taken by CPLEX. All CPLEX experiments were run on a computer with an Intel Core 2 Duo processor running at 2.2 GHz, and 2.0 GB of RAM on Microsoft Windows XP.

On instances with $|\mathcal{N}| \leq 127$ a maximum running time of 18000 seconds was allowed. This limit was set to 30000 seconds for the instance with $|\mathcal{N}| = 280$. CPLEX found optimal solutions for all instances with 14 and 29 nodes and for one instance with 48 nodes. However, for the remaining instances with 48 or more nodes, CPLEX either ran out of time and returned solutions that were quite far from the best known solutions or ran out of memory before the process was finished. Running times increased with both number of nodes and number of visits. These examples indicate that FTSP instances with 127 or more nodes are beyond the reach of the current version of CPLEX.

TABLE 1. CPLEX runs

Instance				Solution (# distance)		Soln. Time
Name	Nodes	Families	Visits	Best Known	CPLEX	(seconds)
burma14_3_1001_1001_2	14	3	6	13.93	13.93	0.43
burma14_3_1001_1002_2			10	25.65	25.65	0.34
burma14_3_1001_1003_2			4	11.88	11.88	0.19
bayg29_4_1001_1001_2	29	4	16	5345.86	5345.86	4.50
bayg29_4_1001_1002_2			17	5791.02	5791.02	27.94
bayg29_4_1001_1003_2			18	5544.33	5544.33	4.68
att48_5_1001_1001_2	48	5	34	23686.02	23686.02	3032.53
att48_5_1001_1002_2			25	20609.09	20609.09	3223.75
att48_5_1001_1003_2			15	9024.58	9024.58	1131.12
bier127_10_1001_1001_2	127	10	62	34161.23	**46453.61**	*18000.00
bier127_10_1001_1001_2			85	92784.43	**95658.72**	*18000.00
bier127_10_1001_1001_2			60	49044.66	**61369.67**	*18000.00
a280_20_1001_1001_2	280	20	179	1826.24	**4907.66**	*30000.00

* CPLEX reached maximum solution time.
Boldface indicates CPLEX failed to find best known solution.

3. Randomized heuristics

Noting the limitations found in Section 2, we propose in this section two randomized heuristics to find good-quality (optimal or near-optimal) solutions to the FTSP: a biased random-key genetic algorithm and a GRASP with evolutionary path-relinking.

3.1. **Biased random-key genetic algorithm for FTSP.**

A biased random-key genetic algorithm (BRKGA) is a metaheuristic for optimization that works by evolving a population of individuals (encoded as encoded as vectors of random keys) by applying genetic operations such as mutation and crossover. For a recent survey of BRKGAs, the reader is referred to Gonçalves and Resende (2011). BRKGAs are an extension of the random-key genetic algorithm (RKGA) proposed by Bean (1994). Both methods search the solution space indirectly by searching the space of random keys and use a decoder to map solutions from the space of random keys to the solution space of the optimization problem. An application of a RKGA for the GTSP can be found in Snyder and Daskin (2006). We are unaware of any application of a BRKGA to the GTSP.

A BRKGA begins with an initial population $\mathcal{P}_0$ of n_p vectors (individuals), each of size n_c. Each vector λ_i with $i \in 1, \ldots, n_p$ is made up of randomly generated numbers (random keys) in the real interval $(0, 1]$. A *decoder* then translates each individual λ_i into a feasible solution to the optimization problem being solved and a fitness value (solution value) $f(\lambda_i)$ associated to it is computed.

At the g-th BRKGA iteration (generation), population $\mathcal{P}_g$ is partitioned into a set $\mathcal{P}_g^e$ of n_e elite individuals (the most fit) and a set $\mathcal{P}_g^{\bar{e}}$ with the remaining non-elite individuals of the population. To evolve a population into the next iteration, first all elite individuals in $\mathcal{P}_g^e$ are copied into the population of the next generation $\mathcal{P}_{g+1}$. Then, a set $\mathcal{P}^m$ of n_m mutant individuals is inserted into $\mathcal{P}_{g+1}$. A *mutant* is a new randomly generated individual (i.e. vector of random keys). Note that the sizes of $\mathcal{P}_g^e$ and $\mathcal{P}^m$ are restricted to be such that $2 \times n_e < n_p$ and $n_e + n_m \leq n_p$.

Data: n_p, n_c, n_e, n_m, p_e
Generate $\mathcal{P}_0$ with n_p individuals having n_c random-keys $\in (0, 1]$;
$g = 0$;
while *stopping criterion is not satisfied* **do**
    Evaluate fitness of each new individual in $\mathcal{P}_g$;
    Partition $\mathcal{P}_g$ into $\mathcal{P}_g^e$ and $\mathcal{P}_g^{\bar{e}}$;
    Initialize next population: $\mathcal{P}_{g+1} \leftarrow \mathcal{P}_g^e$;
    Generate n_m mutants $\mathcal{P}^m$ each having n_c random-keys $\in (0, 1]$;
    $\mathcal{P}_{g+1} \leftarrow \mathcal{P}_{g+1} \cup \mathcal{P}^m$;
    for $k \leftarrow 1$ **to** $n_p - n_e - n_m$ **do**
        Select parent a at random from $\mathcal{P}_g^e$;
        Select parent b at random from $\mathcal{P}_g^{\bar{e}}$;
        for $i \leftarrow 1$ **to** n_c **do**
            Toss biased coin having probability of heads, $p_e > 0.5$;
            if *Toss results in heads* **then** $c[i] \leftarrow a[i]$;
            else $c[i] \leftarrow b[i]$;
        end
        $\mathcal{P}_{g+1} \leftarrow \mathcal{P}_{g+1} \cup \{c\}$;
    end
    $g++$;
end
return $\lambda^* \leftarrow$ **argmin**$\{f(\lambda_g) \mid \lambda_g \in \mathcal{P}_{n_g}\}$

Algorithm 1: Biased random-key genetic algorithm

Finally, the remaining $n_o = n_p - n_e - n_m$ individuals that make up $\mathcal{P}_{g+1}$ are found by mating n_o pairs of individuals (parents) from $\mathcal{P}_g$, one from $\mathcal{P}_g^e$ and another from $\mathcal{P}_g^{\bar{e}}$. Parents are chosen at random with replacement.

Let $a \in \mathcal{P}_g^e$ and $b \in \mathcal{P}_g^{\bar{e}}$ denote the elite and non-elite parents respectively, and let c denote the individual (offspring) that results by mating a and b. The mating operator uses parameterized uniform crossover (Spears and DeJong, 1991). When mating takes place, a biased coin toss is repeated n_c times (i.e. once for each random key), to determine from which parent will the offspring inherit each key. For each coin toss $i \in 1, \ldots, n_c$ the offspring inherits the i-th key from a with probability $p_e > 0.5$ and from parent b with probability $p_{\bar{e}} = 1 - p_e$. This ensures that an offspring has a greater chance to inherit keys from its elite parent. BRKGAs differ from the RKGA proposed by Bean (1994) in the way that pairs of parents are selected at random from the entire population and therefore the coin toss is not necessarily biased towards the fittest of the two parents. This small difference between a BRKGA and a RKGA usually leads to an improved performance of BRKGAs with respect to RKGAs (Gonçalves et al., 2012).

Once the population of the next generation $\mathcal{P}_{g+1}$ is complete, it is set as the current population and all BRKGA operations are repeated until a stopping criterion is met. Denote n_g as the total number of iterations (generations) produced by the algorithm. An individual with the best fitness in the last population $\mathcal{P}_{n_g}$ is returned as the result of the BRKGA. The BRKGA is summarized in the pseudo-code of Algorithm 1.

3.1.1. *Encoding and decoding random keys.* Solutions to the family traveling salesperson problem are encoded as an $|\mathcal{N}| + KN$ vector λ of random keys, where $KN = \sum_{j \in 1,\ldots,K} nv_j$. The first $|\mathcal{N}|$ keys are used to determine which subset of nodes will be visited from each family while the last KN keys are used to define the tour to be followed. To produce a tour that defines a feasible solution to the FTSP, the vector λ is decoded in the following steps:

(1) **Separate keys:** The first $|\mathcal{N}|$ keys are divided into K sets of families $\mathcal{R}_i$, $i = 1, 2, \ldots, K$, each of size nf_i.
(2) **Select family members:** For each family set of random keys $\mathcal{R}_i$, $i = 1, 2, \ldots, K$, sort the nf_i keys in the set in increasing order and select the nv_j smallest key indices where $j = 1, 2, \ldots, K$. The chosen indices will define the nodes to be visited in the solution.
(3) **Define tour:** The remaining KN keys are sorted in increasing order. Their sorted indices are used to define the order (tour) in which the nodes selected in step (2) will be visited.
(4) **Local search:** Let a solution Λ be the tour defined in the previous steps. Local search is applied, scanning the nodes in Λ. Nodes are scanned in increasing order of their indices. Suppose that a visited node i is in position k and another visited node j is in position l. If a reduction in the total distance traveled is achieved by swapping the position of both nodes, then i is set to position l and j to position k. The tour is repeatedly checked for swaps until no further improvement is found or $s_{lim} = KN$ successful swaps are made. The maximum number successful swaps s_{lim} is fixed so that the time spent doing local search is limited for low-quality solutions.
(5) **Update chromosome:** If a tour improvement was found in step (4), the we adjust the KN keys of output vector λ so that they reflect the local optimum found in the previous step. Otherwise, the output vector λ remains unchanged. Since local search is unrelated to the selected family members, there is no need to adjust the first $|\mathcal{N}|$ keys of λ.
(6) **Compute fitness:** Finally, $f(\lambda)$, the total distance traveled in the tour defined by the output vector λ is set as its fitness.

3.2. **GRASP with evolutionary path-relinking for FTSP.** A greedy randomized adaptive search procedure (GRASP) is a multi-start metaheuristic, where in each iteration a feasible solution is constructed with a randomized greedy algorithm and refined afterwards with a local search algorithm (Feo and Resende, 1989; 1995). Path-relinking (PR) is a search intensification method first introduced by Glover (1996). PR is based in the premise that paths that connect two good-quality solutions can be explored to search for better solutions.

Laguna and Martí (1999) proposed the first hybridization of GRASP and PR (GRASP+PR). Their version of GRASP+PR keeps a pool of elite (good-quality) solutions found during the search. At each GRASP iteration, a path-relinking operator is applied between the GRASP local search solution and a solution randomly chosen from the elite pool. Recent surveys of PR can be found in Ribeiro and Resende (2012) and Resende et al. (2010).

Evolutionary path-relinking (evPR) tries to improve an elite pool of solutions by applying a PR operator between pairs of elite solutions, updating the pool if better solutions are found in the process (Festa et al., 2002; Resende and Werneck, 2004; Aiex et al., 2005; Morán-Mirabal et al., 2012b).

```
1  f* = ∞;
2  E ← ∅;
3  ev_count = i_e;
4  while stopping criterion is not satisfied do
5  |  x ← GreedyRandomized();
6  |  x ← LocalSearch(x);
7  |  E ← UpdateElitePool(E, x);
8  |  if |E| > 1 then
9  |  |  x_p ← SelectEliteSolution(E, x);
10 |  |  x ← PathRelinking(x, x_p);
11 |  |  E ← UpdateElitePool(E, x);
12 |  end
13 |  if ev_count == 0 then
14 |  |  E ← EvolutionaryPR(E, x);
15 |  |  ev_count ← i_e + 1;
16 |  end
17 |  ev_count ← ev_count − 1;
18 end
19 f* = argmin{f(x) | x ∈ E};
20 return f*
```

Algorithm 2: GRASP with evolutionary path-relinking

The pseudo-code in Algorithm 2 illustrates a GRASP+evPR metaheuristic for a minimization problem. The algorithm begins by initializing the incumbent solution f^* to a large number in line 1, and the pool of elite solutions E to an empty set in line 2. The variable ev_{count}, which measures the number of iterations left until evolutionary path-relinking is invoked, is initialized in line 3. All GRASP+evPR iterations take place between lines 4 and 18, until some stopping criterion is met. Each iteration begins by constructing a solution using a greedy randomized procedure in line 5, and then applying local search to it in line 6. The resulting local minimum is tested for inclusion in the elite pool E in line 7. If E is full, x is accepted if it is better than at least one solution in the pool. If x is better than all pool solutions, then it replaces the worst pool solution. Otherwise, if it is better than at least one solution but not all, then it replaces the least different solution having higher cost. If E is not full, then x is accepted if it differs from all current pool solutions. PR is applied once there are at least two elite solutions in E. When this is met, a solution x_p is selected from E in line 9, and PR is applied between x and x_p in line 10. The resulting solution x returned by path-relinking is tested for inclusion in the elite pool in line 11. Evolutionary path-relinking is called every i_e iterations. This condition is checked in line 13 and, if fulfilled, evPR is done in line 14 where an evolved pool is returned. The counter ev_{count} is then re-initialized in line 15. At the end of each GRASP+evPR iteration in line 17, this counter is reduced by one unit. The algorithm finishes in line 19 by setting the incumbent solution f^* to a best solution in the last elite pool considered. As a result, the GRASP+evPR returns the incumbent solution in line 20.

We propose a customized GRASP+evPR for the FTSP. It consists of three different construction procedures, seven local search algorithms, twelve path-relinking and evolutionary path-relinking strategies, and one restart method. We describe each GRASP+evPR component separately, and briefly discuss how parameters and configurations are tuned.

3.2.1. *Construction.* The randomized greedy algorithm constructs a feasible solution one node at a time. At any time, all remaining node candidates consist of all the feasible non-visited family members, i.e. if family j has already been visited the prescribed number of times nf_j, then all remaining nodes in j are discarded as future candidates. Let $\hat{F}_l$ consist of the feasible non-visited nodes in family l. The construction phase begins by initializing a set $\mathcal{M} = \varnothing$ which will hold the nodes to be selected in each family, a set $\mathcal{T} = \varnothing$ which will hold the order indices in which the members of $\mathcal{M}$ will be visited in the tour, and a set $\mathcal{C} = \hat{F}_1 \cup \hat{F}_2 \cup \ldots \cup \hat{F}_K$ with all the candidate nodes available to visit at each move.

Let $\nu = 0$ be the current node visited, τ_ν be the tour position of ν, and define the greedy function associated with a candidate node $i \in \mathcal{C}$ to be $h(i) = d(\nu, i)$. A restricted candidate list (RCL) is formed such that

$$\text{RCL} = \{i \in \mathcal{M} \mid h(i) \leq h_* - \alpha(h_* - h^*)\},$$

where $h_* = \min\{h(i) \mid i \in \mathcal{M}\}$, $h^* = \max\{h(i) \mid i \in \mathcal{M}\}$, and $\alpha \in \mathbb{R}$ such that $0 \leq \alpha \leq 1$. The next node to be visited in the tour ν, is selected uniformly at random from the RCL, and the candidate list $\mathcal{C}$, the selected members set $\mathcal{M}$ and the tour indices set $\mathcal{T}$ are updated, i.e. $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\nu\}$, $\mathcal{M} \leftarrow \mathcal{M} \cup \{\nu\}$, and $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau_\nu\}$. The procedure is repeated until all nv_j visits required, with $j \in 1, \ldots, K$, are fulfilled. The total distance traveled by the tour defined in solution $\Lambda = (\mathcal{M}, \mathcal{T})$ is set as the objective function value Z of the current solution.

In this paper we use three variants of this construction procedure, which we call *Fixed*, *Random*, and *Reactive*. In *Fixed* construction we use a fixed value for the RCL parameter α, whereas in *Random*, fixed lower and upper bounds for α are given respectively (i.e. α_l and α_u) and a value within these bounds is selected uniformly at random at each GRASP iteration. *Reactive* construction uses a value of α selected at random from a finite set consisting of n_α values with probabilities favoring those values that produced better solutions in previous constructions. This approach, called *Reactive GRASP* (Prais and Ribeiro, 2000), recalculates these probabilities over time. The number n_α of distinct α values varies from 2 to 8 and the values go from 0 to 0.8.

3.2.2. *Local Search.* Once the construction phase produces a feasible solution $\Lambda = (\mathcal{M}, \mathcal{T})$, a local search algorithm (LS) attempts to improve the solution by making changes in sets $\mathcal{M}$ and $\mathcal{T}$. We propose three LS algorithms, *change-members*, *tour-swap-2*, and *iterative*. Each algorithm scans either the elements of $\mathcal{M}$, the elements of $\mathcal{T}$ or the elements of both sets. The order in which elements are scanned is at random.

In *change-members*, the procedure scans each selected member $i \in \mathcal{M}$ and tries to change it for a node family member k which is not currently used in the solution. If a decrease in the current total distance traveled is found, then k replaces member i, Λ and Z are updated, and the procedure continues to the next element in $\mathcal{M}$.

Alternatively, in *tour-swap-2*, pairs of tour indices in $\mathcal{T}$ are considered for swapping, i.e. if element i and j from $\mathcal{M}$ are in positions k and l respectively, the swap would change their indices so that element i is in position l and element j in position k. If a swap results in a reduction of the total distance traveled, then the swap is made, Λ and Z are updated, and LS proceeds to check the next pair of indices in $\mathcal{T}$. The number of pairs considered for swapping is limited by the value

$$\varphi = \beta \times 0.5 \times |\mathcal{T}| \times (|\mathcal{T}| - 1),$$

where $0.01 \leq \beta \leq 1.00$ is an input parameter.

Finally in *iterative*, the *change-members* and *tour-swap-2* procedures are applied iteratively (i.e. one element in $\mathcal{M}$ and one element in $\mathcal{T}$ at a time), in an attempt to extend the solution space explored by LS. In the case of all three LS procedures, the resulting local minimum solution is denoted as $\Lambda_l(\mathcal{M}_l, \mathcal{T}_l)$.

Aside from the three LS algorithms proposed, three LS options are considered to determine the extensiveness of the search. We propose three LS options, *First*, *Best*, and *Variable*. In *First*, elements in $\mathcal{M}$ or $\mathcal{T}$ (depending on the algorithm) are scanned until a first solution improvement is found. Then the search is restarted and repeated until no further improvement is found. In the case of *Best*, all elements are scanned and the move that produces the best improvement is selected. The search is then restarted and repeated until no further improvement is found. Finally in *Variable*, each time LS is called, an algorithm is chosen at random from among *Change-members*, *Tour-swap-2*, and *Iterative*. In addition, either option *First* or *Best* is selected at random.

3.2.3. *Path-relinking and evolutionary path-relinking.* Besides with local search, our algorithm does intensification with both path-relinking (PR) and evolutionary path-relinking (evPR). A survey of the many variants of these path-relinking based search intensification methods can be found in Ribeiro and Resende (2012).

During the search of the solution space carried out by GRASP+evPR, the algorithm maintains a pool of elite solutions E. In each iteration, a solution $\Lambda_e = (\mathcal{M}_e, \mathcal{T}_e)$ is chosen at random from the elite set and a PR operator is applied, exploring a path in the solution space spanned by Λ_e and the locally optimal solution $\Lambda_l = (\mathcal{M}_l, \mathcal{T}_l)$ found during the local search phase. Let the symmetric difference $\Gamma(\Lambda_1, \Lambda_2)$ between two solutions denoted by $(\mathcal{M}_1, \mathcal{T}_1)$ and $(\mathcal{M}_2, \mathcal{T}_2)$ be the set of node family members and tour indices that differ between the two solutions. The method selects a solution Λ_e from the elite set with probability proportional to the cardinality of its symmetric difference with respect to Λ_l.

The basic PR operator used in this paper proceeds as follows. PR is given a starting solution $\Lambda_s = (\mathcal{M}_s, \mathcal{T}_s)$ and a guiding solution $\Lambda_g = (\mathcal{M}_g, \mathcal{T}_g)$. Depending on the variant of PR used, Λ_s and Λ_g can vary. Let $Z(\Lambda_i)$ denote the total distance traveled by the tour defined in solution Λ_i. In *Forward* PR, for example, Λ_s would be the solution with larger Z between Λ_e and Λ_l, whereas Λ_g would be the solution with shorter Z.

At each iteration of PR, the method randomly selects an element k in the symmetric difference $\Gamma(\Lambda_s, \Lambda_g)$ and reassigns this element in Λ_s with the corresponding value in Λ_g. If an improvement in $Z(\Lambda_s)$ is found then this solution is saved as Λ_{pr}. The procedure continues until $|\Gamma(\Lambda_s, \Lambda_g)| = 1$. This way, PR generates a sequence of neighboring solutions, and selects among them, a solution Λ_{pr} with the smallest

Z value. LS is applied on this solution and the local minimum $\Lambda_{pr\text{-}l}$ is returned as the path-relinking solution.

Since each solution Λ_i consists of a set $\mathcal{M}_i$ and a set $\mathcal{T}_i$ we propose three PR move options, which we call *M-T*, *T-M*, and *I-MT*. In *M-T*, all differences regarding $\mathcal{M}_i$ are considered first and all differences regarding $\mathcal{T}_i$ are considered afterwards. Inversely, *T-M* considers all tour differences first and all member differences afterwards. Finally, the *I-MT* option iteratively considers one member difference and one tour difference.

We allow four flavors of PR in our procedure: *Forward*, *Backward*, *Back&Forth*, and *Mixed*. In *Forward* PR, Λ_s is the solution with larger Z between Λ_e and Λ_l whereas in *Backward* PR, Λ_s is the solution with shorter Z. In *Back&Forth* PR, one path is produced with $\Lambda_s = \Lambda_l$ and $\Lambda_g = \Lambda_e$, and another with $\Lambda_s = \Lambda_e$ and $\Lambda_g = \Lambda_l$. Lastly, with *Mixed* PR, we move from both Λ_e and Λ_l, alternating at each step which solution plays the role of starting and guiding solution. In this variant the solutions eventually meet when their symmetric difference differs by one.

For each variant of PR, we consider full and truncated versions. In the full version, the entire path linking the two solutions is explored while in the truncated version $\gamma\%$ of the path is explored, where γ is an input parameter such that $20\% \leq \gamma \leq 60\%$.

Evolutionary path-relinking (evPR) is done every i_e iterations in an attempt to improve the pool E of elite solutions (i_e is a given input parameter). Whenever evPR takes place, a PR operator is applied to each pair of elite solutions, and every improved solution Λ_{evPR} found in the process is tested for inclusion into the elite pool following the rules mentioned in Algorithm 2. The procedure is repeated until no further improvement is found after applying the PR operator to all new pairs of solutions in E.

3.2.4. *Tuning.* We described a number of construction, local search, path-relinking, and evolutionary path-relinking procedures that can be combined in different ways to result in a GRASP+evPR heuristic. Logical and numerical parameters can be associated with these procedures and used to configure the heuristic. In the experiments described in Section 4 we tune both the logical and numerical parameters with an automatic parameter tuning procedure described in Morán-Mirabal et al. (2012a).

4. Experimental results

In this section we compare both BRKGA and GRASP+evPR algorithms on benchmark instances.

4.1. **Instances.** To test the performance of both heuristics, we propose a set of benchmark instances for the FTSP. A set of 21 benchmark instances were generated by using 7 instances from TSPLIB (Reinelt, 1991). To generate an instance, a number of families K, smaller than the number of nodes in the instance $|\mathcal{N}|$, was chosen. Next, the amount of family members nf_l were established uniformly at random such that $\sum_{l=1,\ldots,K} nf_l = |\mathcal{N}|$. Finally, the number of prescribed visits to each family nv_l was determined at random such that $1 \leq nv_l \leq nf_l$.

The set of 21 instances consists of three instances for each of the following combinations $(|\mathcal{N}|, K, KN)$ of nodes, families and visits: $(14, 3, *)$, $(29, 4, *)$, $(48, 5, *)$,

$(127, 10, *)$, $(280, 20, *)$, $(666, 30, *)$, $(1002, 40, *)$. The number of visits varies from one instance to the next in the same group.

4.2. **The experiments.** The objective of the computational experiments was to study the performance of the heuristics described in Section 3. The experiments were divided into those run on CPLEX 11, and those using `GRASP+evPR-FTSP` and `BRKGA-FTSP`.

Both `GRASP+evPR-FTSP` and `BRKGA-FTSP` were coded in C++ and compiled with the `g++` compiler using optimization flag `-O3`. Both algorithms used for random-number generator an implementation of the Mersenne Twister algorithm (Matsumoto and Nishimura, 1998). `BRKGA-FTSP` was implemented using the API described in Toso and Resende (2012). All experiments other than those with CPLEX were on an Intel Core i5-2450M processor at 2.50 GHz and 6GB of RAM.

As mentioned in Subsection 3.2, `GRASP+evPR-FTSP` was tuned using the procedure of Morán-Mirabal et al. (2012a). The average time taken for tuning each instance with 14, 29, 48, 127, 280, 666, and 1002 nodes was approximately 46, 158, 236, 1453, 7557, 15068, and 24365 seconds, respectively. The tuned parameters and configurations for each instance are shown in Table 2. All `GRASP+evPR-FTSP` experiments were run using the tuned settings, and an elite pool size $|E| = 15$.

`BRKGA-FTSP` was run with a population size of $|\mathcal{P}| = 1000$ for instances with $|\mathcal{N}| \leq 29$, $|\mathcal{P}| = 800$ for instances with $48 \leq |\mathcal{N}| \leq 127$, $|\mathcal{P}| = 600$ for instances with $280 \leq |\mathcal{N}| \leq 666$, and $|\mathcal{P}| = 400$ for instances with $|\mathcal{N}| = 1002$. Since the time required to measure an individual's fitness increases with the number of nodes considered, the population size was decreased for `BRKGA-FTSP` to be able to consider more generations. For each population, a fraction of 0.15 elite members and 0.20 mutants was considered. A reset was triggered every n_r generations without improvement of the incumbent solution, replacing the population with new randomly generated individuals. n_r was set to 200, 150, 100, and 50 generations without improvement for instances with $|\mathcal{N}| \leq 29$, $48 \leq |\mathcal{N}| \leq 127$, $280 \leq |\mathcal{N}| \leq 666$, and $|\mathcal{N}| = 1002$, respectively. The probability that an offspring inherits the key from the elite parent during each step of crossover was set to $p_h = 0.7$.

CPLEX was tested on all instances having 127 nodes or less, and one instance with 280 nodes. As mentioned in Section 2, CPLEX was able to prove optimality only for instances having less than or equal to 48 nodes. Instances having more than 48 nodes reached the time limit and returned a low-quality solution.

To compare the performance of `BRKGA-FTSP` and `GRASP+EvPR-FTSP`, first a set of 5 independent runs of 36000 seconds, were done for each instance. After collecting the results from both heuristics, the best objective function value found in all runs was established as the best known solution for each instance (which in turn were used when testing CPLEX). Next, target values with gaps between 0% and 11% from the best known solutions were selected, and both heuristics were run for a new set of five independent runs with a limited run-time. The maximum run-time of `GRASP+EvPR-FTSP` was set to 7200 seconds for all instances. However, the maximum run-time of `BRKGA-FTSP` was increased to compensate for the average time taken when `GRASP+EvPR-FTSP` was tuned. Concisely, the maximum run-time of `BRKGA-FTSP` was set to 7246, 7358, 7436, 8653, 14757, 22268, and 31565 seconds for instances with 14, 29, 48, 127, 280, 666, and 1002 nodes, respectively.

Each run stopped when a solution with objective function value less than or equal to the target solution was found, or the time limit was reached. Table 3

TABLE 2. Tuned parameters and configurations of GRASP+evPR-FTSP

Instance				GRASP			Local Search		Path-relinking			evPR
Name	Nodes	Families	Visits	Type	α		Options	φ	Direction	Type	γ	i_e
burma14_3_1001_1001_2	14	3	6	Random	[0.21,0.65]		Variable	0.15	Back&Forth	M-T	0.28	Off
burma14_3_1001_1002_2	14	3	10	Random	[0.45,0.59]		Variable	0.70	Backward	T-M	0.43	100
burma14_3_1001_1003_2	14	3	4	Random	[0.15,0.54]	First	Change-members		Backward	M-T	1.00	Off
bayg29_4_1001_1001_2	29	4	16	Random	[0.16,0.40]	First	Iterative	0.99	Backward	T-M	0.46	50
bayg29_4_1001_1002_2	29	4	17	Fixed	0.76	First	Iterative	0.62	Forward	I-MT	1.00	66
bayg29_4_1001_1003_2	29	4	18	Random	[0.00,0.17]		Variable	1.00	Mixed	I-MT	1.00	100
att48_5_1001_1001_2	48	5	34	Fixed	0.26	First	Change-members		Forward	M-T	1.00	Off
att48_5_1001_1002_2	48	5	25	Reactive	6 alphas	First	Iterative	0.52	Backward	T-M	0.55	40
att48_5_1001_1003_2	48	5	15	Fixed	0.03		Variable	0.97	Backward	T-M	1.00	200
bier127_10_1001_1001_2	127	10	62	Fixed	0.02	First	Iterative	0.81	Backward	T-M	0.24	100
bier127_10_1001_1002_2	127	10	85	Fixed	0.02	First	Iterative	0.95	Mixed	I-MT	0.39	100
bier127_10_1001_1003_2	127	10	60	Fixed	0.03	First	Tour-swap-2	0.76	Mixed	T-M	1.00	40
a280_20_1001_1001_2	280	20	179	Reactive	8 alphas	First	Iterative	0.29	Mixed	I-MT	1.00	200
a280_20_1001_1002_2	280	20	156	Reactive	8 alphas	Best	Iterative	0.73	Back&Forth	T-M	0.48	66
a280_20_1001_1003_2	280	20	141	Reactive	4 alphas	First	Iterative	0.80	Mixed	T-M	1.00	50
gr666_30_1001_1001_2	666	30	357	Random	[0.01,0.20]	First	Iterative	0.22	Forward	T-M	1.00	66
gr666_30_1001_1002_2	666	30	328	Reactive	7 alphas		Variable	0.98	Back&Forth	T-M	0.30	Off
gr666_30_1001_1003_2	666	30	328	Reactive	7 alphas		Variable	0.98	Mixed	I-MT	1.00	40
pr1002_40_1001_1001_2	1002	40	486	Reactive	3 alphas		Variable	0.87	Mixed	M-T	1.00	50
pr1002_40_1001_1002_2	1002	40	538	Reactive	4 alphas	Best	Iterative	0.77	Back&Forth	T-M	1.00	66
pr1002_40_1001_1003_2	1002	40	463	Reactive	3 alphas		Variable	0.60	Back&Forth	T-M	0.54	50

TABLE 3. Time to target performance on 5 independent runs of BRKGA-FTSP and GRASP+EvPR-FTSP

Instance					BRKGA-FTSP			GRASP+EvPR-FTSP		
Name	Nodes	Families	Visits	Target (distance)	Avg. solution (distance)	Avg. time$^{\nabla}$ (s.)	Best solution (distance)	Avg. solution (distance)	Avg. time (s.)	Best solution (distance)
burma14_3_1001_1001_2	14	3	6	13.93	13.93	0.01	13.93	13.93	0.01	13.93
burma14_3_1001_1002_2	14	3	10	25.66	25.66	†0.01	25.66	25.66	0.03	25.66
burma14_3_1001_1003_2	14	3	4	11.89	11.89	0.01	11.89	11.89	0.01	11.89
bayg29_4_1001_1001_2	29	4	16	5345.86	5345.86	†3.40	5345.86	5345.86	8.05	5345.86
bayg29_4_1001_1002_2	29	4	17	5791.01	5791.01	†1.40	5791.01	5791.01	75.74	5791.01
bayg29_4_1001_1003_2	29	4	18	5544.33	5544.33	1.60	5544.33	5544.33	†0.03	5544.33
att48_5_1001_1001_2	48	5	34	23686.02	**23686.02**	143.40	23686.02	23709.622	2938.77	23686.02
att48_5_1001_1002_2	48	5	25	20609.09	**20609.09**	62.80	20609.09	20635.57	7199.22	20635.57
att48_5_1001_1003_2	48	5	15	9024.58	9024.58	1.80	9024.58	9024.58	†0.05	9024.58
bier127_10_1001_1001_2	127	10	62	36990.04	36950.75	498.80	*36913.74	**36856.17**	3.65	*36800.39
bier127_10_1001_1002_2	127	10	85	98410.69	**98333.46**	1413.80	*98216.10	98370.63	2966.56	*97615.41
bier127_10_1001_1003_2	127	10	60	51079.03	**50891.36**	1048.60	*50513.10	50920.77	11.26	*50715.49
a280_20_1001_1001_2	280	20	179	1902.67	2164.40	14760.40	2126.34	**1898.17**	218.28	*1891.16
a280_20_1001_1002_2	280	20	156	1701.75	1980.84	14759.20	1925.28	**1702.33**	2700.88	*1697.48
a280_20_1001_1003_2	280	20	141	1601.14	17740.78	14758.40	1720.23	**1598.49**	6.91	*1597.25
gr666_30_1001_1001_2	666	30	357	1835.08	2733.62	22271.60	2625.69	**1848.50**	6610.22	*1817.06
gr666_30_1001_1002_2	666	30	328	1453.35	2390.46	22269.00	2275.80	**1451.10**	4005.06	*1443.05
gr666_30_1001_1003_2	666	30	328	1358.12	3062.17	22269.75	2426.59	**1403.00**	7199.99	1384.18
pr1002_40_1001_1001_2	1002	40	486	165947.46	432665.72	31568.00	421061.63	**164721.66**	20.72	*163461.79
pr1002_40_1001_1002_2	1002	40	538	187306.74	445954.00	31566.75	421761.00	**184440.18**	8.86	*182144.13
pr1002_40_1001_1003_2	1002	40	463	150068.78	326561.61	31565.50	284856.22	**149838.13**	227.56	*149456.63

Boldface indicates one heuristic did better on average solution values.
† indicates both heuristics had same average solutions, but one heuristic did better on average time.
* indicates the best solution found in all runs was better than the target solution.
$^{\nabla}$ The maximum run-time of BRKGA-FTSP is 7200 seconds plus the average time taken when GRASP+EvPR-FTSP was tuned.

summarizes these experiments. For each instance, the table lists the instance name and size, the target solution value, and for each heuristic, the average best solution value, the average run-time (in seconds), and the best solution value found in all runs.

The table shows that both heuristics perform similarly for instances with 14 and 29 nodes. Both of them achieve the target solution in all runs, which in turn is the optimal solution found previously by CPLEX. Both heuristics have short average times per run, but `BRKGA-FTSP` performs faster than `GRASP+EvPR-FTSP` in 3 out of 6 instances.

For instances with 48 and 127 nodes the table indicates that `BRKGA-FTSP` has a better average solution value than `GRASP+EvPR-FTSP` in 4 out of 6 instances. For remaining two instances, either both achieve the target solution (i.e. optimal solution) in all runs, or `GRASP+EvPR-FTSP` has a better average solution value. Despite the fact that one heuristic does better than the other, the average percentage gap difference between both procedures is 0.12%, which denotes a small difference between solution values and the target solutions. `BRKGA-FTSP` also performs faster in 3 out of 6 instances. An interesting observation is that both `BRKGA-FTSP` and `GRASP+EvPR-FTSP` found at least one solution that had a lower solution value than the target in all 3 instances with 127 nodes.

Finally for the remaining instances, the table shows that `GRASP+EvPR-FTSP` has a better average solution value than `BRKGA-FTSP`. In this case the the average percentage gap difference between both procedures for instances with 280, 666, and 1002 nodes are 13.79%, 78.34%, and 139.61% respectively. Moreover, `BRKGA-FTSP` did not reach the target solution in all runs, whereas `GRASP+EvPR-FTSP` found at least one solution that had a lower solution value than the target in 8 out of 9 instances. This denotes that `GRASP+EvPR-FTSP` outperforms `BRKGA-FTSP` considerably as instances become larger.

5. Concluding remarks

This paper introduced the family traveling salesperson problem (FTSP), a variant of the generalized traveling salesperson problem, which can be applied to order-picking in warehouses. Considering that families of similar products can be located in scattered positions in a warehouse, the FTSP can be used to optimize the routes followed when picking a collection of products of an incoming order.

We also present a binary integer programming model for the FTSP, and propose two randomized heuristics for this problem: a biased random-key genetic algorithm and a GRASP with evolutionary path-relinking heuristic. After showing that the integer programming solver CPLEX 11 can only handle small instances of the FTSP, we compare the performance of the proposed heuristics on a set of benchmark instances.

The experiments show that both heuristics can find optimal and good quality near-optimal solutions. `GRASP+evPR-FTSP` seems to perform much better on large instances, even if the time taken for fine-tuning the heuristic is considered. On the other hand, `BRKGA-FTSP` tends to perform better in smaller instances, and find equal or better solutions than `GRASP+evPR-FTSP` in less time.

References

R.M. Aiex, P.M. Pardalos, M.G.C. Resende, and G. Toraldo. GRASP with path-relinking for three-index assignment. *INFORMS J. on Computing*, 17:224–247, 2005.

D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, USA, 2011.

J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 2:154–160, 1994.

D.L. Applegate and R.E. Bixby and V. Chvátal and W.J. Cook. Concorde TSP Solver, 2013. `www.tsp.gatech.edu/concorde`, last visited on February 5, 2013.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

C. Feremans, M. Labbé, and G. Laporte. Generalized network design problems. *European J. of Operational Research*, 148:1–13, 2003.

P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

M.R. Garey and D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

F. Glover. Tabu search and adaptive memory programming – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.

B. Golden, Z. Naji-Azimi, S. Raghavan, M. Salari, and P. Toth. The generalized covering salesman problem. *INFORMS J. on Computing*, 24:534–553, 2012.

J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *J. of Heuristics*, 17:487–525, 2011.

J.F. Gonçalves, M.G.C. Resende, and R.F. Toso. Biased and unibiased random key genetic algorithms: An experimental analysis. Technical report, AT&T Labs Research, Florham Park, NJ, 2012.

G. Gutin and A.P. Punnen. *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization. Springer, Dordrecht, The Netherlands, 2002.

M. Hahsler and K. Hornik. TSP Infrastructure for the Traveling Salesperson. *J. of Statistical Software*, 23, 2007.

R. Koster, T. LeDuc, and K.J. Roodbergen. Design and control of warehouse order picking: A literature review. *European J. of Operational Research*, 182:481–501, 2007.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. on Computing*, 11:44–52, 1999.

M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

L.F. Morán-Mirabal, J.L. González-Velarde, and M.G.C. Resende. Automatic parameter tuning of GRASP with evolutionary path-relinking. Technical report, AT&T Labs Research, Florham Park, New Jersey, June 2012a.

L.F. Morán-Mirabal, J.L. González-Velarde, and M.G.C. Resende. Randomized heuristics for handover minimization in mobility networks. Technical report, AT&T Labs Research, Florham Park, New Jersey, August 2012b.

M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS J. on Computing*, 12: 164–176, 2000.

H.D. Ratliff and A.S. Rosenthal. Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 31:507–521, 1983.

G. Reinelt. TSPLIB – A Traveling Salesman Problem Library. *ORSA J. on Computing*, 3:376–384, 1991.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *J. of Heuristics*, 10:59–88, 2004.

M.G.C. Resende, C.C. Ribeiro, F. Glover, and R. Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 87–107. Springer, 2nd edition, 2010.

C.C. Ribeiro and M.G.C. Resende. Path-relinking intensification methods for stochastic local search algorithms. *J. of Heuristics*, 18:193–214, 2012.

L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European J. of Operational Research*, 174:38–53, 2006.

W.M. Spears and K.A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

S. S. Srivastava, S. Kumar, R. C. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *J. of the Canadian Operational Research Society*, 7:97–101, 1970.

C. Theys, O. Bräysy, W. Dullaert, and B. Raa. Using a TSP heuristic for routing order pickers in warehouses. *European J. of Operational Research*, 200:755–763, 2010.

R.F. Toso and M.G.C. Resende. A C++ application programming interface for biased random-key genetic algorithms. Technical report, AT&T Labs Research, Florham Park, NJ, 2012.

J.N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

(Luis F. Morán-Mirabal) Tecnológico de Monterrey, Monterrey, Mexico.
E-mail address: luismoranm@gmail.com

(José Luis González-Velarde) Tecnológico de Monterrey, Monterrey, Mexico.
E-mail address: gonzalez.velarde@itesm.mx

(Mauricio G.C. Resende) Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.
E-mail address, M.G.C. Resende: mgcr@research.att.com