RANDOMIZED HEURISTICS FOR HANDOVER MINIMIZATION IN MOBILITY NETWORKS

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

Abstract. A mobile device connects to the cell tower (base station) from which it receives the strongest signal. As the device moves it may connect to a series of towers. The process in which the device changes the base station it is connected to is called handover. A cell tower is connected to a radio network controller (RNC) which controls many of its operations, including handover. Each cell tower handles an amount of traffic and each radio network controller has capacity to handle a maximum amount of traffic from all base stations connected to it. Handovers between base stations connected to different RNCs tend to fail more often than handovers between base stations connected to the same RNC. Handover failures result in dropped connections and therefore should be minimized. The Handover Minimization Problem is to assign towers to RNCs such that RNC capacity is not violated and the number of handovers between base stations connected to different RNCs is minimized. We describe an integer programming formulation for the handover minimization problem and show that state-of-the-art integer programming solvers can solve only very small instances of the problem. We propose several randomized heuristics for finding approximate solutions of this problem, including a GRASP with path-relinking for the generalized quadratic assignment problem, a GRASP with evolutionary path-relinking, and a biased random-key genetic algorithm. Computational results are presented.

1. Introduction

A cellular (or mobility) network consists of fixed base stations (cell towers) and mobile transceivers (e.g., mobile phones and tablet computers). A radio signal between the mobile transceiver and the base station allows communication between the transceiver and other transceivers as well as with other devices in the network. Each base station covers an area called a *cell*. As a mobile transceiver moves between cells, it may need to connect over time to several base stations. The transfer of connection from one base station to another is called a *handover*.

Each base station is controlled by a *radio network controller* or RNC. Each base station is connected to one RNC. The amount of traffic between transceivers and each base station depends strongly on the location of the base station. For example, a base station located in a city center will usually have more traffic than one located in a rural area distant from the city center. Each RNC can handle a maximum amount of traffic. This constraint limits the subsets of base stations that can connect to each RNC.

Date: August 2, 2012.

Key words and phrases. Mobility networks, handover minimization, randomized heuristics, GRASP, biased random-key genetic algorithm.

AT&T Labs Research Technical Report.

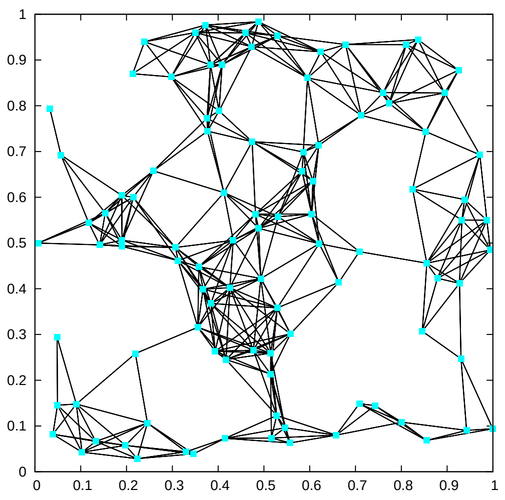

FIGURE 1. Example of 100 base station instance. Nodes correspond to the 100 base stations and edges to pairs of base stations with handover.

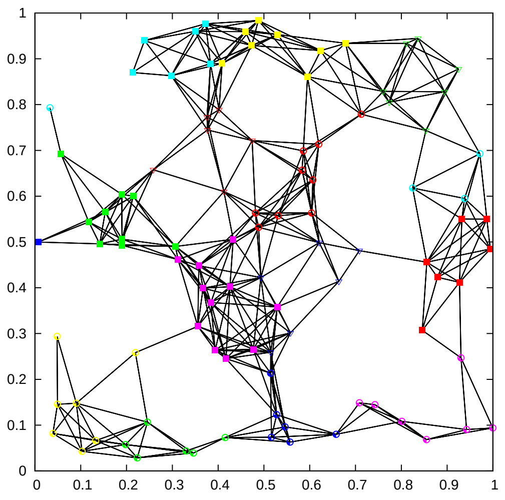

FIGURE 2. Example of solution found for 100 base station and 15 RNCs. Nodes corresponding to the 100 base stations are colored according to RNC they are assigned to.

Let $\mathcal{T}$ be the set of base stations and let t_i be the total traffic between base station $i \in \mathcal{T}$ and the transceivers connected to it. Let $\mathcal{R}$ be the set of RNCs and let c_j be the maximum amount of traffic (capacity) RNC $j \in \mathcal{R}$ can handle. Finally, let $h_{i,j}$ be the total number of handovers between base stations i and j $(i, j \in \mathcal{T}, i \neq j)$. Note that $h_{i,j}$ and $h_{j,i}$ may differ.

In the *handover minimization problem* (HMP) we wish to find an assignment of each base station in set $\mathcal{T}$ to some RNC in set $\mathcal{R}$. Let π_i be the index of the RNC that base station $i \in \mathcal{T}$ is assigned to and let Φ_j be the set of indices of the base stations assigned to RNC $j \in \mathcal{R}$. The assignments must also satisfy the capacity constraints of the RNCs, i.e. for each RNC $j \in \mathcal{R}$, we require that $\sum_{k\in\Phi_j} t_k \leq c_j$. Among all feasible assignments, we seek one that minimizes the sum of handovers between base stations assigned to different RNCs, i.e. we want to minimize $\sum_{i,j\in\mathcal{T}\times\mathcal{T}\ |\ \pi_i\neq\pi_j} h_{i,j}$.

Consider the graph $G = (\mathcal{T}, E)$ where the node-set $\mathcal{T}$ represents the base stations and the edge-set E is such that if base stations i, j are such that $h(i,j) + h(j,i) > 0$, then $e = (i,j) \in E$. Figure 1 shows an example of graph G with 100 base stations. Each node represents a base station and an edge is present between a pair of base stations if there is a positive number of handovers between the base stations. Figure 2 shows a solution for this example when there are 15 RNCs. In the figure, base stations assigned to the same RNC are grouped by color and shape. The HMP can be formulated as a mixed integer program (MIP). Let the binary variable $x_{e,k} = 1$ if and only if edge e has both of its endpoints assigned to RNC k. Furthermore, let the binary variable $y_{i,k} = 1$ if and if RNC $\pi_i = k$. We first require that each base station $i \in \mathcal{T}$ must be assigned to one and only one RNC, i.e.

$$\sum_{k=1}^{|\mathcal{R}|} y_{i,k} = 1, \forall i \in \mathcal{T}.$$

The next set of constraints forces $x_{e,k} = 0$ if either one of the endpoints of edge e is not assigned to RNC k, i.e.

$$x_{e,k} \leq y_{i,k}, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R} \tag{1}$$

$$x_{e,k} \leq y_{j,k}, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R}. \tag{2}$$

$$x_{e,k} \geq y_{i,k} + y_{j,k} - 1, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R}. \tag{3}$$

The final set of constraints limits the set of base stations that can be assigned to each RNC. Since RNC j has capacity for at most c_j units of base station traffic, the sum of the the traffic values of all base stations assigned to RNC j cannot exceed c_j, i.e.

$$\sum_{i\in\mathcal{T}} t_i y_{i,k} \leq c_k, \quad \forall k \in \mathcal{R}.$$

Since minimizing the total handover between base stations assigned to different RNCs is equivalent to maximizing the total handover between base stations assigned to the same RNC, we can formulate the following mixed integer program:

$$\begin{aligned}
\max & \sum_{k \in \mathcal{R}} \sum_{e=(i,j) \in E} h(i,j) x_{e,k} \\
& \sum_{k=1}^{|\mathcal{R}|} y_{i,k} = 1, \forall i \in \mathcal{T} \\
& x_{e,k} \leq y_{i,k}, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R} \\
& x_{e,k} \leq y_{j,k}, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R} \\
& \sum_{i \in \mathcal{T}} t_i y_{i,k} \leq c_k, \quad \forall k \in \mathcal{R} \\
& 0 \leq x_{e,k} \leq 1, \quad \forall e = (i,j) \in E, \quad k \in \mathcal{R} \\
& y_{i,k} \in (0,1), \quad \forall i \in \mathcal{T}, \quad k \in \mathcal{R}.
\end{aligned}$$

Note that constraints (1–2) together with the objective function make constraint (3) redundant. For the same reason, the integrality of decision variable $x_{e,k}$ can be relaxed.

The handover minimization problem is also known as the *node capacitated graph partitioning* problem. Ferreira et al. (1998) were first to study this problem, proposing strong valid inequalities for a branch and cut algorithm. They test their algorithm on three applications of partitioning: compiler design, finite element computations associated with meshes, and design of electronic circuits. The largest instances they were able to solve to optimality had 61 nodes and 187 edges for compiler design, 48 nodes and 81 edges for design of electronic circuits, and 274 nodes and 469 edges for finite element computations on a mesh. Mehrotra and Trick (1997) proposed a branch and price algorithm for the node capacitated graph partitioning problem. They test their algorithm on instances having 30 to 61 nodes and 47 to 187 edges. The largest instances they were able to solve to optimality had 61 nodes and 187 edges. Deng and Bard (2011) recently proposed a reactive GRASP with path-relinking post-processing for the node capacitated graph partitioning problem. They test their heuristic on instances varying in size from 30 to 82 nodes and 65 to 540 edges. They used CPLEX 11 to solve to optimality instances having 30 nodes and most of the instances having 40 nodes. They show that their heuristic found solutions having the same objective function value than those found by CPLEX on most instances, but in less time than CPLEX. On larger instances, CPLEX failed to prove optimality and found worse solutions than those found by the reactive GRASP with path-relinking for all but one instance where they were tied. They also compare their reactive GRASP heuristic with the combinatorial algorithm of Mehrotra and Trick (1997) showing that their heuristic matched the optimal solutions found by the combinatorial algorithm on all but one instance where it found a suboptimal solution.

The remainder of this paper is organized as follows. In Section 2 we describe three heuristics for the HMP. Experimental results are presented in Section 3. Finally, in Section 4 we present concluding remarks.

2. Randomized heuristics

In this section we motivate the need for heuristics by first showing that the mixed integer programming approach fails even for small instances. We then propose three randomized heuristics: a GRASP with path-relinking for generalized quadratic assignment, a GRASP with evolutionary path-relinking, and a biased random-key genetic algorithm.

2.1. **Solving the integer programming model.** In Section 3 we will describe a procedure to generate synthetic instances of the HMP and propose a set of benchmark instances on which to test our algorithms. We use here a few instances of the benchmark set to evaluate CPLEX 11, a state-of-the-art integer programming solver, on the MIP model proposed in Section 1.

Table 1 show results of running CPLEX on instances having the following pairs of base stations and RNCs: (20,5), (20,10), (30,5), (30,10), (30,15), (40,5), (40,10), (40,15), (100,15), (100,25), and (100,50). CPLEX was run on multiple instances with (base stations, RNCs) pairs (20,*), (30,*), and (40,*) and on a single instance of sizes (100,15), (100,25), and (100,50). For each instance, the table lists the instance name, the number of base stations, the number of RNCs, the number of handovers in the best known solution for that instance, the number of handovers in the best solution found by CPLEX, and the time in CPU seconds taken by CPLEX. On the instances with 20, 30, and 40 base stations a maximum running time of 10 hours was allowed while for the instances with 100 base stations the maximum running time was 24 hours. CPLEX found optimal solutions for all instances have 20 and 30 base stations and all instances having 40 base stations and 5 and 10 RNCs. For two instances with 40 base stations and 15 RNCs, it could not prove optimality in spite of having found the best known solutions. On the other three instances of the same size, it proved optimality. Running times increased with both number of base stations and number of RNCs. On the three largest instances, with 100 base stations, CPLEX found solutions that were quite far from the best known solutions even though it ran for 24 hours. These examples indicate that problems with 100 or more base stations are beyond the reach of the current version of CPLEX.

2.2. **GRASP for generalized quadratic assignment.** In the generalized quadratic assignment problem (GQAP) we are given a set $\mathcal{F}$ of facilities and a set $\mathcal{L}$ of locations. Each facility $i \in \mathcal{F}$ has a demand a_i and each location $j \in \mathcal{L}$ has a capacity c_i. Each pair of facilities $\{i,j\} \in \mathcal{F} \times \mathcal{F}$ has an associated flow $f_{i,j} \geq 0$. Furthermore, the distance between each pair of locations $\{k,l\} \in \mathcal{L} \times \mathcal{L}$ is $d_{k,l} \geq 0$. We want to find an assignment π of facilities to locations. Let π_i be the location to which facility $i \in \mathcal{F}$ is assigned and let Φ_j be the set of facilities assigned to location $j \in \mathcal{L}$. We want to find an assignment such that the capacity constraints of the locations are not violated, i.e. for each location $k \in \mathcal{L}$ we have

$$\sum_{i \in \Phi_k} a_i \leq c_k.$$

Among all feasible assignments, the generalized quadratic assignment problem seeks one that minimizes

$$\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{F}} f_{i,j} \; d_{\pi_i, \pi_j},$$

the sum of products of flows between pairs of facilities and distances between locations these facilities are assigned to. A pair of facilities with heavy inter-facility flow should be assigned to nearby locations.

TABLE 1. CPLEX runs on small instances.

Instance			Solution (# handovers)		Soln. Time
Name	Base Stations	RNCs	Best Known	CPLEX	(seconds)
20_5_270001	20	5	540	540	0.33
20_5_270002			54	54	0.19
20_5_270003			816	816	0.23
20_5_270004			126	126	0.27
20_5_270005			372	372	0.20
20_10_270001	20	10	2148	2148	4.11
20_10_270002			1426	1426	0.94
20_10_270003			2458	2458	11.53
20_10_270004			1570	1570	2.22
30_5_270001	30	5	772	772	0.64
30_5_270002			136	136	0.66
30_5_270003			920	920	0.50
30_5_270004			52	52	0.30
30_5_270005			410	410	0.56
30_10_270001	30	10	3276	3276	18.63
30_10_270002			1404	1404	1.84
30_10_270003			2214	2214	316.89
30_10_270004			2150	2150	15.97
30_10_270005			2540	2540	48.45
30_15_270001	30	15	6178	6178	25882.84
30_15_270002			4042	4042	19.64
30_15_270003			4126	4126	8.50
30_15_270004			3920	3920	11.02
40_5_270001	40	5	610	610	1.02
40_5_270002			136	136	0.94
40_5_270003			234	234	0.75
40_5_270004			232	232	1.08
40_5_270005			774	774	1.22
40_10_270001	40	10	4544	4544	44.59
40_10_270002			2068	2068	28.14
40_10_270003			2090	2090	67.02
40_10_270004			1650	1650	548.61
40_10_270005			4316	4316	3245.55
40_15_270001	40	15	8646	8646	9146.02
40_15_270002			4586	4586	*36000.00
40_15_270003			5396	5396	*36000.00
40_15_270004			4800	4800	19250.03
40_15_270005			6272	6272	863.89
100_15_270001	100	15	19000	**49270**	*86400.00
100_25_270001	100	25	36412	**58637**	*86400.00
100_50_270001	100	50	60922	**70740**	*86400.00

* – Reached maximum solution time in seconds.
Boldface indicates CPLEX failed to find best known solution.

Note that the HMP is a special case of the GQAP. If we let

- facilities in the GQAP be the base stations in the HMP,
- locations in the GQAP be the RNCs of the HMP,
- flows in the GQAP be the handovers in the HMP,
- distances in the GQAP (between each pair of RNCs in the HMP) be one,

then the objective function of the GQAP corresponds to that of the HMP.

Mateus et al. (2011) proposed a GRASP with path-relinking heuristic for the GQAP and its Java implementation. In Section 3 we describe computational results solving instances of the HMP with this algorithm.

2.3. **GRASP with evolutionary path-relinking for handover minimization.** We next describe a specially tailored GRASP with evolutionary path-relinking for the HMP. We first give an overview of the basic structure of a GRASP with evolutionary path-relinking heuristic. Then, we describe construction procedures, local search algorithms, and strategies for path-relinking and evolutionary path-relinking. We then briefly address how heuristic parameters are set and how the appropriate variants are configured.

GRASP, or greedy randomized adaptive search procedures (Feo and Resende, 1989; 1995), is a multi-start stochastic search method, where at each iteration local search is applied to a solution generated with a randomized greedy algorithm. Path-relinking was proposed by Glover (1996) as a search intensification procedure for tabu search and scatter search. Paths in the solution space, connecting two good-quality solutions, are explored for better solutions. Laguna and Martí (1999) introduced the hybridization of GRASP with path-relinking. Their algorithm maintains a pool of elite, or high-quality, solutions found during the search, and at each GRASP iteration applies path-relinking between the GRASP local search solution and a solution chosen at random from the pool. See Resende et al. (2010) and Ribeiro and Resende (2012) for recent surveys of path-relinking. Evolutionary path-relinking (Festa et al., 2002; Resende and Werneck, 2004; Aiex et al., 2005) uses the path-relinking operator to attempt to improve the pool of elite solutions. Given a pool, evolutionary path-relinking applies path-relinking between pairs of pool solutions, updating the pool if better solutions are found.

Algorithm 1 shows pseudo-code for a GRASP with evolutionary path-relinking heuristic for a minimization problem. In line 2 the value f^* of the best solution found is initialized to a large number while in line 3 the pool P of elite solutions is initialized empty. The variable `it2evPR` which measures the number of GRASP iterations left until evolutionary path-relinking is carried out is initialized in line 4. The main loop of the algorithm goes from line 5 to line 22. It is repeated until some stopping criterion is satisfied. In line 6, a randomized greedy solution x is constructed. If the construction phase fails to produce a feasible solution, a repair procedure is applied to x in line 8 with the objective of making x feasible. In line 10 local search is applied, starting at x and the resulting local minimum is tested for inclusion in the elite pool in line 11. If the current pool of elite solutions is not yet full, then solution x is accepted if it differs from all solutions currently in the pool. It is added to the pool and does not replace any solution already in the pool. Otherwise, if the pool is full, x is accepted if it is better than at least one solution currently in the pool. In this case, if accepted, x replaces the worst pool solution if x is better than all pool solutions. Otherwise, if it is better than at least one solution but not all solutions, then it replaces the least different solution among those having worse cost. In the first GRASP iteration, path-relinking is not applied. From then on, in line 13 a solution x_p is selected from the pool and path-relinking is applied between x and x_p in line 14. The resulting solution x returned by path-relinking is tested for inclusion in the elite pool in line 15. If x is accepted into the pool, the pool is updated to include x. Evolutionary path-relinking is done once every i_e GRASP iterations. This condition is tested in line 17 and if triggered

```
1  GRASP+evPR
2  $f^* \leftarrow \infty$;
3  $P \leftarrow \emptyset$;
4  it2evPR $\leftarrow i_e$;
5  while stopping criterion is not satisfied do
6      $x \leftarrow$ GreedyRandomized($\cdot$);
7      if $x$ is not feasible then
8          $x \leftarrow$ Repair($x$);
9      end
10     $x \leftarrow$ LocalSearch($x$);
11     $P \leftarrow$ UpdateElite($P, x$);
12     if $|P| \geq 2$ then
13         $x_p \leftarrow$ SelectPoolSolution($P, x$);
14         $x \leftarrow$ PathRelinking($x, x_p$);
15         $P \leftarrow$ UpdateElite($P, x$);
16     end
17     if it2evPR $= 0$ then
18         $P \leftarrow$ evPathRelinking($P, x$);
19         it2evPR $\leftarrow i_e + 1$;
20     end
21     it2evPR $\leftarrow$ it2evPR $- 1$;
22 end
23 return argmin$\{f(x) \mid x \in P\}$
```

Algorithm 1: GRASP with evolutionary path-relinking.

evolutionary path-relinking is done in line 18 where the updated pool is returned and the counter of iterations to the next application of evolutionary path-relinking is re-initialized in line 19. At end of each iteration, this counter is decremented by one in line 21. A pool solution having minimum objective function value is returned by the procedure in line 23.

The randomized greedy algorithm for handover minimization constructs a solution one base station to RNC assignment at a time. RNCs are initially permuted at random and the algorithm scans the RNCs in the permuted order, dealing with only one RNC at a time. Base stations are assigned to the current RNC being scanned until this RNC does not have enough leftover capacity to accept another base station. When an RNC being scanned is empty, a first base station must be assigned to it. Let $\bar{\mathcal{T}}$ be the set of all yet-unassigned base stations for which the RNC capacity is greater than the traffic associated with the base station. We select a base station $i \in \bar{\mathcal{T}}$ to assign to the RNC with probability proportional to $\sum_{j \in \mathcal{T},\ (j \neq i)} h(i,j) + h(j,i)$. The greedy choice would be to select the yet-unassigned base station

$$i^* = \text{argmax}\{ \sum_{j \in \mathcal{T}\ (j \neq i)} h(i,j) + h(j,i) \mid i \in \bar{\mathcal{T}} \}.$$

The selection process proposed for this heuristic is therefore randomized greedy since the choice is not deterministic. After each base station is assigned to the RNC, the RNC's available capacity is adjusted to reflect the assignment just made. Base stations are assigned to the RNC while the RNC has available capacity. Suppose now that one or more base stations have been assigned to the current RNC and again let $\bar{\mathcal{T}}$ denote the set of base stations that can be accommodated by the RNC and let Φ denote the set of base stations already assigned to the RNC. Define the greedy function associated with base station $i \in \bar{\mathcal{T}}$ to be $g(i) = \sum_{j\in\Phi} h(i,j)+h(j,i)$. A restricted candidate list (RCL) is formed such that

$$\text{RCL} = \{i \in \bar{\mathcal{T}} \mid g(i) \geq g^* - \alpha(g^* - g_*)\},$$

where $g_* = \min\{g(i) \mid i \in \bar{\mathcal{T}}\}$, $g^* = \max\{g(i) \mid i \in \bar{\mathcal{T}}\}$, and $\alpha \in \mathbb{R}$ such that $0 \leq \alpha \leq 1$. A base station in the RCL is selected uniformly at random and is assigned to the RNC. In this paper we use two variants of this construction procedure, which we call *fixed* and *random*. 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 and a value within these bounds is selected uniformly at random at each GRASP iteration.

After scanning all available RNCs, it may occur that not all base stations are assigned. In such a case, a repair procedure is applied to attempt to achieve feasibility. Let $\hat{\mathcal{T}}$ be the set of base stations that have not be assigned. Let $u_k = \sum_{i\in\Phi_k} t_i$ be the total utilization of RNC k, where Φ_k is the set of base stations assigned to RNC k. For base station $i \in \hat{\mathcal{T}}$, let $\hat{u}_k = t_i + u_k$ be the utilization of RNC k if base station i were to be assigned to it. Clearly $\hat{u}_k > c_k$. The repair procedure scans the base stations in $\hat{\mathcal{T}}$ and assigns each base station to RNC $k = \mathbf{argmin}\{\hat{u}_j \mid j \in \mathcal{R}\}$ and updates the utilization u_k of RNC k, i.e. $u_k \leftarrow \hat{u}_k$. This is continued until all base stations are assigned to an RNC. The resulting base station to RNC assignment is clearly infeasible since at least one RNC has an utilization that exceeds its capacity. Let π_i be the RNC to which base station i is assigned. To try to achieve feasibility, we attempt to swap RNC assignments of base station $i \in \hat{\mathcal{T}}$ assigned to RNC π_i with some base station $j \in \mathcal{T} \setminus \hat{\mathcal{T}}$ assigned to RNC $\pi_j \neq \pi_i$. If the swap results in feasible assignments, i.e. if $\hat{u}_{\pi_i} - t_i + t_j \leq c_{\pi_i}$ and $\hat{u}_{\pi_j} - t_j + t_i \leq c_{\pi_j}$, then the swap is made and the set $\hat{\mathcal{T}}$ is updated, i.e. $\hat{\mathcal{T}} \leftarrow \hat{\mathcal{T}} \setminus \{i\}$. Otherwise, if $t_i > t_j$ and $\hat{u}_{\pi_j} - t_j + t_i \leq c_{\pi_j}$ then by making the swap, RNC π_j remains underutilized and the over-utilization of RNC π_i is reduced. The swap is made and the set $\hat{\mathcal{T}}$ is updated, i.e. $\hat{\mathcal{T}} \leftarrow \hat{\mathcal{T}} \setminus \{i\} \cup \{j\}$. This repair procedure continues until a feasible assignment is found, i.e. $\hat{\mathcal{T}} = \emptyset$, or a maximum number of swap attempts is reached and we declare the repair procedure to have failed and a new GRASP iteration begins.

Once the randomized greedy construction procedure produces an assignment vector $\{\pi_1, \pi_2, \ldots, \pi_{|\mathcal{T}|}\}$, where π_i is the RNC to which base station $i \in \mathcal{T}$ is assigned to, a local search algorithm attempts to improve the assignment by making changes in the assignment vector. We propose three local search algorithms, *move-1*, *move-max*, and *swap-2*. The three algorithms scan the base stations in increasing order of their total traffic. For base station i, assigned to the RNC π_i, the procedure checks if there is any RNC $k \neq \pi_i$ with enough capacity to accommodate base station i such that the reassignment of base station i from RNC π_i to RNC k reduces the total handover count. If such RNC is found, base station i is reassigned to RNC k. In the case of move-1, the procedure is restarted at the first base station in the permutation (i.e. the base station with the smallest traffic), whereas in the case of move-max it proceeds to the next station in the permutation (i.e. the station with least traffic among those with more traffic than the just reassigned station).

The procedure ends when the all base stations are scanned and no improving move is found. In the case of swap-2, pairs of base station assignments are considered for swapping, i.e. if base station i is assigned to RNC π_i and base station j is assigned to RNC π_j, the swap would assign i to π_j and j to π_i. This is only feasible if the available capacities of π_i and π_j allow and if $\pi_i \neq \pi_j$. If a feasible swap can be made and this results in a reduction of the handover count, then the swap is made and the local search proceeds to the base station pair i and $j+1$ in case $j \leq n-1$, otherwise the pair $i+1$ and $i+2$ is considered. The number of pairs considered for swapping is limited by the value

$$\beta \times \sum_{i \in \mathcal{R}} \left(\sum_{j \in \mathcal{R} \setminus \{i\}} |\Phi_i| \times |\Phi_j| \right),$$

where $0.01 \leq \beta \leq 0.3$ is an input parameter and Φ_k is the set of base stations assigned to RNC k. In the case of all three local search procedure, the resulting local minimum is denoted by Π_l.

Besides local search, intensification is done with both path-relinking and evolutionary path-relinking. We refer the reader to Ribeiro and Resende (2012) for descriptions of the many variants of these search intensification methods.

During the search, the algorithm maintains a pool, or set, of elite solutions. In each GRASP iteration, a solution Π_e is chosen at random from the elite set and a path-relinking operator is applied, exploring a path in the solution space spanned Π_e and the local minimum Π_l. Let the symmetric difference $\Gamma(\Pi_1, \Pi_2)$ between two solutions Π_1 and Π_2 be the set of base stations assigned to different RNCs in these assignments. The method selects a solution Π_e from the elite set with probability proportional to the cardinality of its symmetric difference with respect to Π_l.

In what follows we describe the basic path-relinking operator used in this paper. Path-relinking is given a starting solution $\Pi_s = \{\pi^s_1, \pi^s_2, \ldots, \pi^s_{|\mathcal{T}|}\}$ and a guiding solution $\Pi_g = \{\pi^g_1, \pi^g_2, \ldots, \pi^g_{|\mathcal{T}|}\}$. Depending on the flavor of path-relinking used, Π_s and Π_g can vary. In forward path-relinking, for example, $\Pi_s = \Pi_l$ and $\Pi_g = \Pi_e$. At each iteration of path-relinking we consider each base station in the symmetric difference $\Gamma(\Pi_s, \Pi_g)$ and reassign each base station $i \in \Gamma(\Pi_s, \Pi_g)$ to π^g_i and evaluate the ratio of cost improvement to the resulting capacity deficit. Let $\Pi_s \oplus \pi^g_i$ denote the assignment obtained by reassigning base station i in Π_s to RNC π^g_i and let $\Delta_h(i)$ denote the change in the total number of handovers due to this reassignment (an increase in the number of handovers corresponds to a value $\Delta_h(i) < 0$). Denote the capacity utilization of RNC π^g_i to be $\sigma_{\pi^g_i} = t_i + \sum_{j \in \Phi_{\pi^g_i}} t_j$ and let $\Delta_c(i) = 1 + \max\{0, \sigma_{\pi^g_i} - c_{\pi^g_i}\}$. The cost function used by path-relinking to select a move is $\Delta_h(i)/\Delta_c(i)$. The greedy choice for a move in path-relinking is to choose the reassignment (of base station i) that maximizes the greedy function $g(i) = \Delta_h(i)/\Delta_c(i)$. This cost function penalizes moves that lead to capacity deficits in the RNC. Our algorithm defines a restricted candidate list (RCL) such that

$$\text{RCL} = \{i \in \Gamma(\Pi_s, \Pi_g) \mid g(i) \geq g^* - \alpha_p(g^* - g_*)\},$$

where $g_* = \min\{g(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, $g^* = \max\{g(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, and $0 \leq \alpha_p \leq 1$. A move is selected from the RCL uniformly at random. If the move does not lead to any infeasibility, then the selected move is only based on the difference in handover count. Once a move is selected, say base station i is reassigned to RNC π_i^g, we then redefine Π_s to be $\Pi_s \oplus \pi_i^g$. The procedure continues until $|\Gamma(\Pi_s, \Pi_g)| = 1$. This way, path-relinking generates a sequence of neighboring solutions, each of which may be feasible or infeasible, and selects, among the feasible solutions, a solution with the fewest handovers. On this solution, local search is applied and the local minimum Π_p is returned as the path-relinking solution.

The GRASP with path-relinking heuristic attempts to add all local minimum solutions (Π_l produced by local search after construction and Π_p produced by path-relinking) into the elite set. Let us call this local minimum simply Π. If the elite set is not full, then Π is placed in the set as long as it is different from all elite set solutions. Next, we describe the case when the elite set if full. If the number of handovers of solution Π is less than that of the best solution in the elite set, Π replaces a worst solution in the set. If the number of handovers of solution Π is greater than that of the best solution in the elite set but less than that of the worst, then Π replaces the elite set solution most similar to it (that minimizes the symmetric difference with respect to it) among all elite set solution having greater number of handovers than it does.

We allow four flavors of path-relinking: forward, backward, back and forth, and mixed. Suppose path-relinking is to be applied between solutions Π_l and Π_e. In forward path-relinking, we have $\Pi_s = \Pi_l$ and $\Pi_g = \Pi_e$ and in backward path-relinking, we have $\Pi_s = \Pi_e$ and $\Pi_g = \Pi_l$. In back and forth path-relinking, one path is produced with $\Pi_s = \Pi_l$ and $\Pi_g = \Pi_e$ and another with $\Pi_s = \Pi_e$ and $\Pi_g = \Pi_l$. Finally, with mixed path-relinking, we move both Π_s and Π_g, alternating at each step which solution plays the role of starting and guiding solution. The solutions eventually meet when their symmetric difference differs by one. Finally for each flavor of path-relinking we have 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 $20\% \leq \gamma \leq 70\%$.

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 with evolutionary path-relinking heuristic. Logical parameters can be associated with these procedures and used to configure the heuristic. Furthermore, each procedure has associated with it a number of numerical parameters that need to be tuned. In the experiments described in Section 3 we tune both the logical and numerical parameters with an automatic parameter tuning procedure described in Morán-Mirabal et al. (2012).

2.4. **Biased random-key genetic algorithm for handover minimization.** Biased random-key genetic algorithms (BRKGAs) evolve a set (or population) of vectors of random keys (or individuals) applying Darwin's principle of survival of the fittest. See Gonçalves and Resende (2011) for a recent survey of BRKGAs.

A BRKGA works with a fixed-size population $\mathcal{P}$ made up of $|\mathcal{P}|$ vectors $\chi_1, \chi_2, \ldots, \chi_{|\mathcal{P}|}$ of randomly generated numbers (random keys) in the real interval $(0, 1]$. Each vector has n random keys. The initial population therefore consists of $|\mathcal{P}|$ vectors of n random keys each. A *decoder* is a deterministic algorithm that takes as input a vector of random keys χ and outputs a solution to the optimization problem and a fitness value (or objective function cost) $f(\chi)$ for the vector.

At each iteration (or generation) of the algorithm, the population is partitioned into a smaller set $\mathcal{P}_e$ of elite individuals and a larger set $\mathcal{P}_{\bar{e}}$ with the remaining individuals of $\mathcal{P}$. The evolutionary dynamics of a BRKGA are as follows. First, all elite individuals are copied, without change, to the population of the next generation. Then, a set $\mathcal{P}_m$ of mutant individuals (a mutant is simply a vector of random keys, generated in the same way as an individual of the initial population) is inserted into the population of the next generation. We restrict the sizes of sets $\mathcal{P}_e$ and $\mathcal{P}_m$ to be such that $|\mathcal{P}_e| + |\mathcal{P}_m| \leq |\mathcal{P}|$ and $2 \times |\mathcal{P}_e| \leq |\mathcal{P}|$.

The first two steps of the evolutionary dynamics account for $|\mathcal{P}_e| + |\mathcal{P}_m|$ individuals and therefore $p_x = |\mathcal{P}| - |\mathcal{P}_e| - |\mathcal{P}_m|$ individuals will still need to be inserted into the population of the next generation. This is done through the combination (or mating) of p_x pairs of individuals from the current population, one from $\mathcal{P}_e$ and another from $\mathcal{P}_{\bar{e}}$. Individuals are selected for mating at random and with replacement. Therefore a single individual may mate more than once in each generation. Let $a \in \mathcal{P}_e$ and $b \in \mathcal{P}_{\bar{e}}$ denote the elite and non-elite individuals, respectively, and let c denote the offspring that results from the combination of a and b. Mating is done through parameterized uniform crossover (Spears and DeJong, 1991) where a biased coin is flipped n times, once for each random key, to determine from which parent the offspring will inherit each key. The coin has probability $p_h > \frac{1}{2}$ to result in heads. For $i = 1, \ldots, n$, the i-th component of c receives the i-th component of a if the coin toss results in heads or the i-th component of b otherwise. This way an offspring has a greater chance to inherit the keys of its elite parent. Furthermore, since a given elite parent is selected from the smaller set $\mathcal{P}_e$, it will have a greater chance of being selected for mating than will a given non-elite parent.

```
BRKGA($|\mathcal{P}|, |\mathcal{P}_e|, |\mathcal{P}_m|, n, p_h$)
Generate population $\mathcal{P}$ with individuals having $n$ random-keys $\in (0,1]$;
while stopping criterion is not satisfied do
    Evaluate fitness of each new individual in $\mathcal{P}$;
    Partition $\mathcal{P}$ into sets $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$;
    Initialize next population: $\mathcal{P}^+ \leftarrow \mathcal{P}_e$;
    Generate mutants $\mathcal{P}_m$ each having $n$ random-keys $\in (0,1]$;
    $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \mathcal{P}_m$;
    for $i \leftarrow 1$ to $|\mathcal{P}| - |\mathcal{P}_e| - |\mathcal{P}_m|$ do
        Select parent $a$ at random from $\mathcal{P}_e$;
        Select parent $b$ at random from $\mathcal{P}_{\bar{e}}$;
        for $j \leftarrow 1$ to $n$ do
            Toss biased coin having probability $p_h > 0.5$ of heads;
            if Toss is heads then  $c[j] \leftarrow a[j]$;
            else $c[j] \leftarrow b[j]$;
        end
        $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \{c\}$;
    end
    $\mathcal{P} \leftarrow \mathcal{P}^+$;
end
return $\chi^* \leftarrow$ argmin$\{f(\chi) \mid \chi \in \mathcal{P}\}$
```

Algorithm 2: Biased random-key genetic algorithm.

The BRKGA is summarized in the pseudo-code of Algorithm 2. It takes as input the size of population $\mathcal{P}$, the sizes of the elite set $\mathcal{P}_e$ and mutant set $\mathcal{P}_m$, where $|\mathcal{P}_e| + |\mathcal{P}_m| \leq |\mathcal{P}|$ and $2 \times |\mathcal{P}_e| \leq |\mathcal{P}|$, the size of the random-key vector (n), and the probability that the toss results in heads ($p_h > 0.5$). In line 2 the initial population is generated, consisting of $|\mathcal{P}|$ vectors, each with n real-valued keys randomly generated in the interval $(0, 1]$. The iterations of the algorithm correspond to the loop in lines 3 to 20. In line 4, the finesses of all newly added individuals of population $\mathcal{P}$ are evaluated (this can be done in parallel). Population $\mathcal{P}$ is partitioned into a smaller set $\mathcal{P}_e$ of elite individuals (those with the best overall fitness values) and a larger set $\mathcal{P}_{\bar{e}}$ with the remaining population. The population of the next generation is initialized with the elite set of the current population in line 6. In line 7 the mutant set $\mathcal{P}_m$ is generated in the same way that the initial population was generated. It is added to the population of the next generation in line 8. The remainder of the population of the next generation is completed in lines 9 to 18. Parents a and b are selected at random in lines 10 and 11, respectively, and parameterized uniform crossover is applied in lines 12 to 16 to produce the offspring c which is added to the population of the next generation in line 17. An iteration is completed in line 19 by making the population of the current generation that of the next generation. Finally, after the stopping criterion has been satisfied, the fittest individual of population $\mathcal{P}$ is returned by the algorithm in line 21.

BRKGA are based on a similar algorithm proposed by Bean (1994). Bean's algorithm differs from BRKGA in that in Bean's algorithm parents are selected at random from the entire population and the coin flip is not necessarily biased in favor of the more fit parent.

2.4.1. *Encoding and decoding.* Solutions of the handover minimization problem are encoded as a $2 \times |\mathcal{T}|$ vector χ of random keys. The first $|\mathcal{T}|$ keys dictate the order in which base stations are considered for assignment and the last $|\mathcal{T}|$ keys are used to make the assignments.

To decode the vector χ and produce an assignment of base stations to RNCs, the following steps are computed:

(1) **Initialization:** Initialize empty the set τ of unassigned base stations, i.e. $\tau \leftarrow \emptyset$, and for RNC $k = 1, \ldots, |\mathcal{R}|$, initialize empty the set of base stations assigned to RNC k, i.e. $\Phi_k \leftarrow \emptyset$.

(2) **Order base stations:** Sort the first $|\mathcal{T}|$ keys to produce a permutation of the base stations defining the order in which they will be tentatively assigned, i.e. the index of the i-th smallest key is the i-th base station to be tentatively assigned.

(3) **Assign initial base stations:** Consider the base stations in the order determined in step (2). The target RNC for base station i is $k = \lceil \chi[|\mathcal{T}|+i] \times |\mathcal{R}| \rceil$. If RNC k can accommodate base station i, i.e. if $\sum_{j \in \Phi_k} t_j + t_i \leq c_k$, then assign base station i to RNC k, i.e. $\Phi_k \leftarrow \Phi_k \cup \{i\}$. Otherwise, add base station i to τ, the set of base stations to be assigned later, i.e. $\tau \leftarrow \tau \cup \{i\}$.

(4) **Assign remaining base stations:** Now consider the remaining base stations, i.e. those in set τ. Scan base stations in set τ in the order determined in step (2). For each base station $i \in \tau$, if there exists an RNC that can accommodate base station i, consider only RNCs that can accommodate base station i. Otherwise, consider all RNCs. Assign base station $i \in \tau$ to the RNC which hosts the set of base stations which has maximum total handovers with base station i. Note that if no RNC can accommodate base

station i, then an infeasible assignment will be made. Such assignments are penalized with a high handover cost.

(5) **Local search:** Apply local search, scanning base stations in the order determined in step (2). Scan RNCs in increasing order of their indices. If moving base station i from its current RNC (say RNC j) to another RNC (say RNC k) that can accommodate it reduces the total number of handovers, then move the base station from RNC j to RNC k. Repeat until no reduction in number of handovers can be obtained by moving a base station to a different RNC.

(6) **Adjust chromosomes:** Adjust the keys in the second half of vector χ so that they can be decoded directly into the final locally optimal assignment in step (3) of the decoder. For example, if tower i was initially assigned to RNC $j = \lceil \chi[\mathcal{T} + i] \rceil$ and was reassigned to RNC k during decoding, then adjust the chromosome: $\chi[|\mathcal{T}| + i] \leftarrow \chi[|\mathcal{T}| + i] + (k - j) \times |\mathcal{R}|^{-1}$.

3. Experimental results

In this section we compare the algorithms on benchmark synthetic instances and on a real-world instance from a large wireless services provider.

3.1. Instances.

To test the performance of our heuristics, we propose a set of synthetically generated instances of handover minimization that mimic problems encountered in practice. We also test the heuristics on one instance from a large wireless provider.

We first describe how instances are generated. The idea is to randomly locate base stations in the unit square, each with a randomly generated amount of traffic, and associate with closely located pairs of stations an amount of handover that is inversely proportional to the distance between the stations. Pairs of base stations located beyond a given maximum distance do not have handover between themselves. RNC capacities are generated with the goal of the RNCs being able to accommodate all of the base station traffic. To generate an instance we are given as input parameters $|\mathcal{T}|$ (number of base stations), $|\mathcal{R}|$ (number of RNCs), r (maximum handover distance), u_t and l_t (upper and lower bound on base station traffic), u_h and l_h (upper and lower bounds on number of handovers), and u_c and l_c (upper and lower bounds on capacity slacks) such that $u_c > l_c > 1$. The steps of the generation process are:

(1) Generate uniformly at random in the unit square the x and y coordinates of the $|\mathcal{T}|$ base stations.

(2) For each base station $i \in \mathcal{T}$, generate its traffic uniformly at random between its lower and upper bounds, i.e. $t_i = \texttt{randunif}(l_t, u_t)$.

(3) Let $d_{i,j}$ be the distance between stations $i, j \in \mathcal{T} \times \mathcal{T}$. For each pair of base stations $i, j \in \mathcal{T} \times \mathcal{T}$ such that $d_{i,j} \leq r$, generate an amount of handover $h_{i,j}$ equal to $(l_h - u_h)/r^2 \times d_{i,j}^2 + u_h$. For all $i, j \in \mathcal{T} \times \mathcal{T}$ such that $d_{i,j} > r$, we have that the handover $h_{i,j} = 0$.

(4) For each RNC $j \in \mathcal{R}$, generate a capacity c_j equal to $\texttt{randunif}(l_c, u_c) \times \bar{t}$, where $\bar{t} = (\sum_{i \in \mathcal{T}} t_i)/|\mathcal{T}|$.

Note that step 4 goes not guarantee that a feasible solution will be generated since it is possible that some base station has more traffic associated with it than the capacity of the largest RNC. In an attempt to relax somewhat the capacity constraint, we reassign to each RNC the largest capacity generated, i.e. $c_j \leftarrow \max\{ c_k : k \in \mathcal{R} \}$, for all $j \in \mathcal{R}$.

The benchmark set of 83 synthetic instances consists of five instances for each of the following combinations (b, r) of base stations and RNCs: $(20, 5)$, $(30, 5)$, $(30, 10)$, $(40, 5)$, $(40, 10)$, $(40, 15)$, $(100, 15)$, $(100, 25)$, $(100, 50)$, $(200, 15)$, $(200, 25)$, $(200, 50)$, $(400, 15)$, $(400, 25)$, $(400, 50)$, and four instances with the following combinations: $(20, 10)$, $(30, 15)$. [1] These last two problem classes only have four instances because one of five of each was infeasible. For all instances we used the following parameters: $r = 0.17$, $(l_t, u_t) = (5, 50)$, $(l_h, u_h) = (5, 200)$, and $(l_c, u_c) = (1.05, 1.15)$.

The real instance from a large wireless services provider has about 1000 base stations, 30 RNCs, and about 2% of the ordered base station pairs have handover between them.

3.2. **The experiments.** The goal of the computational experiments was to study the behavior of three heuristics: `GQAP`, the GRASP with path-relinking for the generalized quadratic assignment problem of Mateus et al. (2011), `GevPR-HMP`, the GRASP with evolutionary path-relinking described in Subsection 2.3, and `BRKGA`, the biased random-key genetic algorithm described in Subsection 2.4.

Instances were categorized into two sets: *small*, i.e. those with the following combinations (b, r) of base stations and RNCs: $(20, 10)$, $(30, 15)$, $(20, 5)$, $(30, 5)$, $(30, 10)$, $(40, 5)$, $(40, 10)$, $(40, 15)$, and *large*, i.e. those with the following combinations (b, r) of base stations and RNCs: $(100, 15)$, $(100, 25)$, $(100, 50)$, $(200, 15)$, $(200, 25)$, $(200, 50)$, $(400, 15)$, $(400, 25)$, and $(400, 50)$.

Both `GevPR-HMP` and `BRKGA` were written in C++ and compiled with the `g++` compiler using optimization flag `-O3`. `GQAP` was implemented in Java and compiled into bytecode with `javac` version 1.6.0_05. All three implementations used for random-number generator an implementation of the Mersenne Twister algorithm (Matsumoto and Nishimura, 1998). `GQAP` used the implementation of the Mersenne Twister from the COLT[2] library. `BRKGA` was implemented using the API described in Toso and Resende (2012). All experiments other than those with CPLEX were done using the Condor job control system (University of Wisconsin, 2012) which submitted jobs to either a cluster running Intel Xeon X5650 processors at 2.67 GHz or a cluster running Intel Xeon E5530 processors at 2.4 GHz.

`GevPR-HMP` was run on the large instances with parameters tuned automatically using the procedure described in Morán-Mirabal et al. (2012), where those heuristic configurations and parameters are listed. On all small instances `GevPR-HMP` was run using identical configuration and parameter settings as follows. The construction randomly chose an RCL α parameter in the interval $[0, 0.5]$, local search was move-max with all base stations scanned at most once, and greedy, complete, back & forth path-relinking was used. Evolutionary path-relinking was complete and back & forth. It used a greedy randomized scheme with $\alpha_p = 0.2$. Evolutionary path-relinking was triggered every 200 GRASP iterations. The size of the elite set was 10 for both small and large instances.

`GQAP` used the default configuration and parameter settings used in the experiments described in Subsection 3.2 of Mateus et al. (2011).

[1]Available at `http://www.research.att.com/~mgcr/data/handover-minimization`.

[2]COLT is a open source library for high performance scientific and technical computing in Java. See `http://acs.lbl.gov/~hoschek/colt/`.

BRKGA was run with three independent populations, each of size $|\mathcal{P}| = 1000$ ($|\mathcal{P}_e| = 300$ elite and $|\mathcal{P}_m| = 200$ mutants). Every 500 generations populations pass to each other their two best individuals, replacing the worst individuals of the populations. Every 500 generations without improvement of the incumbent solution, a reset was triggered, replacing each population with new randomly generated individuals. The probability that an offspring inherits the key from the elite parent during each step of crossover was set to $p_h = 0.7$.

The experiments were divided into two types: those on small instances and those on large instances. On the small instances the mixed integer programming solver CPLEX 11.1 was run with a time limit of 10 hours on an Intel Core 2 Duo processor running at 2.2 GHz on Microsoft Windows XP. As we saw in Subsection 2.1, CPLEX was able to prove optimality for all instances having up to 40 base stations and 10 RNCs as well as for three of the five instances having 40 base stations and 15 RNCs. For each small instance, the three heuristics (GevPR-HMP, GQAP, and BRKGA) were each run independently five times for one hour each run. Tables 2 and 3 show the results for these runs. For each instance, the tables show the name and size of the instance, and for each heuristic the average solution value over the five runs, the average time taken to find the best solution in each run, and the value of the best known solution for this instance.

The tables show that GevPR-HMP found the best known solution on all small instances on all five independent runs. GQAP and BRKGA did so on all but two and three instances, respectively. On each of the instances where GQAP failed to find the best known solution on all runs, it failed on only one of the five runs. On the instances where BRKGA failed to find the best solution on all five runs, it failed once on 40_15_270001, twice on 40_15_270002, and all five times on 40_15_270004. With respect to time taken to find the best solution, GevPR-HMP was clearly the fastest, only having higher average running times than GQAP on two of the 48 instances and was faster than BRKGA on all but one instance. BRKGA was faster than GQAP on 25 of the 48 instances. Note that our implementation of BRKGA allows decoding to be done in parallel. The table reports user times for BRKGA, i.e. the sum of the times taken by all processors not taking into account any parallel processing. Real time (wall clock time) for BRKGA were actually less than reported in the tables. As shown in Figure 3, real time to find the best solution using 16 processors was, on average, about a factor 11 times smaller than the reported running times. Parallel efficiency with 16 processors (the ratio of running time with a single processor to 16 times the running time with 16 processors) varied from 60% to 80%. Only with 32 processors did parallel efficiency drop to about 30%.

For the 45 large instances, CPLEX was run on only three instances (100_15_270001, 100_25_270001, and 100_50_270001) always failing to find a near-optimum solution in 24h of running time, as shown in Table 1. On the entire set of large instances the three randomized heuristics were run independently five times for 24 hours each and the best solution of each run was recorded. Tables 4 and 5 summarize these results. For each instance the tables list the instance name and size, and the average and best solution values for each of the three heuristics and the best known solution value for the instance. Note that the best known solution may correspond to independent runs done by the authors that were longer than 24h. GevPR-HMP was the best of the three randomized heuristics. It found solutions with lower average number of handovers than BRKGA on 44 of the 45 instances. With respect to

Table 2. Algorithm performance for small instances on one hour runs (part 1)

Instance			`GevPR-HMP`		`GQAP`		`BRKGA`		
Name	Stations	RNCs	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Best known solution (handover)
20_5_270001	20	5	540	0.017	540	1.030	540	0.217	540
20_5_270002			54	0.001	54	0.690	54	0.270	54
20_5_270003			816	0.001	816	0.600	816	0.282	816
20_5_270004			126	0.001	126	0.650	126	0.284	126
20_5_270005			372	0.001	372	0.870	372	0.292	372
20_10_270001	20	10	2148	0.001	2148	1.950	2148	0.230	2148
20_10_270002			1426	0.001	1426	1.230	1426	0.234	1426
20_10_270003			2458	0.001	2458	2.500	2458	0.487	2458
20_10_270004			1570	0.004	1570	0.920	1570	0.471	1570
30_5_270001	30	5	772	0.006	772	2.790	772	0.384	772
30_5_270002			136	0.001	136	3.500	136	0.380	136
30_5_270003			920	0.261	920	3.050	920	0.542	920
30_5_270004			52	0.003	52	2.830	52	0.377	52
30_5_270005			410	0.005	410	3.480	410	0.530	410
30_10_270001	30	10	3276	0.087	3276	3.580	3276	1.457	3276
30_10_270002			1404	0.046	1404	3.880	1404	1.409	1404
30_10_270003			2214	0.119	2214	3.370	2214	14.263	2214
30_10_270004			2150	1.306	2150	4.870	2150	34.784	2150
30_10_270005			2540	0.531	2540	3.970	2540	3.303	2540

Table 3. Algorithm performance for small instances on one hour runs (part 2)

Instance			GevPR-HMP		GQAP		BRKGA		
Name	Stations	RNCs	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Best known solution (handover)
30_15_270001	30	15	6178	0.082	6178	3.830	6178	21.735	6178
30_15_270002			4042	0.003	4042	5.450	4042	7.763	4042
30_15_270003			4126	0.035	4126	3.850	4126	4.306	4126
30_15_270004			3920	0.292	3920	3.740	3920	14.697	3920
40_5_270001	40	5	610	0.003	610	6.140	610	0.910	610
40_5_270002			136	0.016	136	7.740	136	0.922	136
40_5_270003			234	0.006	234	6.470	234	1.650	234
40_5_270004			232	12.998	232.01	85.140	232	0.818	232
40_5_270005			774	0.010	774	5.340	774	0.709	774
40_10_270001	40	10	4544	0.019	4544	8.790	4544	0.516	4544
40_10_270002			2068	0.269	2068	9.120	2068	111.121	2068
40_10_270003			2090	0.637	2090	7.740	2090	5.446	2090
40_10_270004			1650	39.318	1650	9.410	1650	84.193	1650
40_10_270005			4316	11.273	4316	11.940	4316	584.453	4316
40_15_270001	40	15	8646	2.225	8646.52	10.950	8677.2	1194.792	8646
40_15_270002			4586	7.991	4586	22.070	4614	1945.570	4586
40_15_270003			5396	18.355	5396	8.880	5396	1254.844	5396
40_15_270004			4800	2.628	4800	3.920	4894	524.678	4800
40_15_270005			6272	0.508	6272	6.490	6272	5.159	6272

Table 4. Algorithm performance for large instances on one day runs (part 1)

Instance			GevPR-HMP		GQAP		BRKGA		
Name	Stations	RNCs	Avg. solution (handover)	Best solution (handover)	Avg. solution (handover)	Best solution (handover)	Avg. solution (handover)	Best solution (handover)	Best known solution (handover)
100_15_270001	100	15	19174.0	19174	19114.8	19000	19533.2	19000	19000
100_15_270002			22686.0	22686	22805.6	22686	23478.0	23288	22686
100_15_270003			14558.0	14558	14568.8	14558	14655.2	14616	14558
100_15_270004			19762.0	19762	19711.2	19700	20177.6	19882	19700
100_15_270005			22892.0	22892	22885.2	22746	23430.8	23092	22746
100_25_270001	100	25	36665.2	36412	36608.0	36448	37107.2	36752	36412
100_25_270002			39199.2	39144	38677.2	38608	39515.6	39256	38608
100_25_270003			33098.0	32966	32717.6	32686	33356.0	32708	32686
100_25_270004			35801.2	35678	35433.2	35322	36068.4	35954	35322
100_25_270005			36911.2	36906	36968.0	36878	37363.6	37100	36878
100_50_270001	100	50	61074.0	60922	61234.8	61172	61845.2	61554	60922
100_50_270002			62065.2	62046	62090.8	62022	62684.8	62524	62022
100_50_270003			54707.6	54618	54661.6	54596	55390.8	55192	54596
100_50_270004			57906.4	57894	57903.6	57894	58358.0	58208	57894
100_50_270005			61283.2	61088	61318.0	61318	63042.0	62784	61088
200_15_270001	200	15	81915.2	81558	84327.2	82834	81946.0	81558	81558
200_15_270002			90949.6	89810	93462.0	90620	92372.4	90506	89810
200_15_270003			79232.0	79232	81716.8	80980	79584.0	79548	79232
200_15_270004			78324.0	78324	84737.6	80538	81019.2	80026	78324
200_15_270005			96429.2	95998	100146.8	98826	99065.6	98830	95998
200_25_270001	200	25	133674.0	133168	141961.6	138454	141938.8	140492	133168
200_25_270002			137514.0	136038	141666.4	140066	141012.8	140690	136038
200_25_270003			139962.4	139438	145647.6	144120	144409.6	143724	139438
200_25_270004			129508.0	128554	136128.4	134054	133894.0	131786	128554
200_25_270005			149298.8	148402	157307.2	154260	154275.2	152934	148402

Table 5. Algorithm performance for large instances on one day runs (part 2)

Instance			GevPR-HMP		GQAP		BRKGA		
Name	Stations	RNCs	Avg. solution (handover)	Best solution (handover)	Avg. solution (handover)	Best solution (handover)	Avg. solution (handover)	Best solution (handover)	Best known solution (handover)
200_50_270001	200	50	221821.6	221550	223556.8	223096	224034.0	223098	219672
200_50_270002			218761.2	218254	221346.8	219910	221131.2	219834	216444
200_50_270003			222315.6	221500	223175.2	222404	221568.0	221110	221348
200_50_270004			212626.8	212044	213880.8	212544	213509.2	213170	211832
200_50_270005			232938.0	231890	238228.8	236136	237939.6	237156	231890
400_15_270001	400	15	375429.2	372694	475012.0	456158	378718.0	375650	370314
400_15_270002			373304.8	370274	465669.2	460232	386282.8	383096	370274
400_15_270003			360152.8	358684	456513.6	448830	369552.8	366314	358684
400_15_270004			336826.4	334430	447753.2	406834	349110.0	346282	334430
400_15_270005			365974.0	361904	476186.0	457274	380204.4	377094	361904
400_25_270001	400	25	571930.0	570852	694715.6	663908	584584.8	579130	568830
400_25_270002			547953.6	544568	679772.4	658440	560692.0	554840	543182
400_25_270003			554179.6	548000	680754.8	667982	555433.2	553162	548000
400_25_270004			504474.4	501750	633011.2	607672	517828.0	516416	501750
400_25_270005			561315.2	556044	703440.0	679848	589444.0	585070	556044
400_50_270001	400	50	854656.0	851412	957526.0	951882	881239.6	879438	851412
400_50_270002			848217.6	845496	953687.6	949562	877662.0	874226	845496
400_50_270003			824118.8	819242	927517.6	919140	850384.8	843242	819242
400_50_270004			777953.6	774564	885893.2	878912	810186.8	806690	774564
400_50_270005			857133.2	854726	950535.6	940358	885058.8	882060	854726

Table 6. Time to target performance of GevPR-HMP, GQAP, and BRKGA (part1)

Instance				GevPR-HMP		GQAP		BRKGA	
Name	Stations	RNCs	Target (handover)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)
100_15_270001	100	15	19806	19620.06	5.328	20662.36	638.939	19902.53	403.877
100_15_270002			23366	23174.25	11.378	24462.48	660.282	24137.02	418.112
100_15_270003			14684	14652.25	88.405	15703.73	672.222	14927.27	378.447
100_15_270004			20228	20167.62	39.133	21229.3	682.959	20575.55	415.186
100_15_270005			23674	23447.49	14.367	25363.04	679.104	23872.55	403.012
100_25_270001	100	25	37078	37058.2	139.643	38100.18	679.237	37810.72	399.842
100_25_270002			40150	40011.04	143.075	40181.71	632.307	40344.89	393.189
100_25_270003			34246	34058.03	103.451	34281.69	622.673	34167.63	386.389
100_25_270004			37178	37079.93	166.298	37376.2	644.268	36963.23	173.671
100_25_270005			38428	38171.05	81.539	38673.09	660.364	38169.68	203.611
100_50_270001	100	50	62000	61873.06	17.064	62259.75	551.270	62925.32	440.113
100_50_270002			62700	62605.34	54.887	62848.58	537.100	63485.57	454.600
100_50_270003			55500	55386.18	14.483	55653.1	534.460	56178.85	422.793
100_50_270004			58600	58481.38	8.971	58527.49	339.130	59005.59	470.755
100_50_270005			62400	62288.8	60.725	62285.03	301.390	64476.55	404.792
200_15_270001	200	15	85249	84881.44	112.937	135174	759.537	84549.84	313.546
200_15_270002			92670	92922.22	148.077	138469.52	741.627	95791.04	407.303
200_15_270003			79455	79391.4	76.038	131023.3	754.658	81723.72	410.758
200_15_270004			79151	79049.84	191.834	131143.9	757.408	84192.58	374.751
200_15_270005			99753	99398.22	93.715	148374.34	750.687	102008.12	399.623
200_25_270001	200	25	136888	136985.08	164.352	199407.98	768.219	145775.02	430.870
200_25_270002			142496	142158.64	82.973	200546.6	762.716	144337.88	428.857
200_25_270003			145173	144430.38	88.706	205867.48	757.659	146618.14	464.378
200_25_270004			132899	133138.72	202.187	199655.6	738.460	137325.12	426.084
200_25_270005			155801	155751.84	133.340	212062.98	756.682	157598.46	455.711

Table 7. Time to target performance of GevPR-HMP, GQAP, and BRKGA (part2)

Instance				GevPR-HMP		GQAP		BRKGA	
Name	Stations	RNCs	Target (handover)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)	Avg. solution (handover)	Avg. time (sec)
200_50_270001	200	50	226713	226015.22	69.906	257332.98	755.506	226413.86	416.312
200_50_270002			221356	221203.06	119.822	253743.32	751.607	223162.9	419.161
200_50_270003			224419	224107.78	336.053	251997.68	746.041	224256.8	333.639
200_50_270004			216710	216101.96	104.803	244904.66	755.018	216107.06	134.643
200_50_270005			238376	237984.48	89.017	268692.52	761.601	239762.5	392.378
400_15_270001	400	15	402527	399649.52	321.422	945799.60	688.419	392628.54	174.726
400_15_270002			386927	385339.46	228.210	817270.62	669.215	398998.96	367.328
400_15_270003			376442	374582.30	184.832	794048.56	670.530	383126.52	383.002
400_15_270004			364988	361189.44	104.066	851717.10	656.314	364816.84	368.051
400_15_270005			387876	386195.32	143.762	840390.58	670.427	393153.64	361.039
400_25_270001	400	25	576035	578068.20	332.992	1108569.74	673.703	600535.5	410.892
400_25_270002			569055	564824.74	186.245	1123872.38	674.358	578253.92	440.243
400_25_270003			573570	570633.50	127.711	1072120.50	691.000	571058.72	334.239
400_25_270004			519279	516601.28	157.849	1049194.58	651.094	532885.44	463.086
400_25_270005			602675	599008.74	162.387	1125957.00	718.975	602253.58	371.044
400_50_270001	400	50	863436	862929.66	378.383	1211789.38	779.589	886941.26	417.197
400_50_270002			864789	861970.84	76.637	1193964.44	695.899	884644.42	346.493
400_50_270003			842378	840511.60	88.194	1177311.02	753.713	860251.82	420.258
400_50_270004			800980	799154.16	181.951	1131179.18	706.823	819416.4	448.579
400_50_270005			863916	863747.04	368.984	1208322.08	765.271	892137.36	363.091

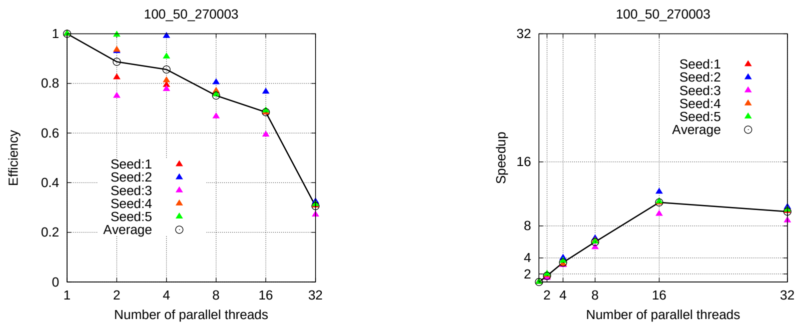

FIGURE 3. Parallel runtime efficiency and speedup of `BRKGA`

`GQAP`, it found solutions with lower average number of handovers on all but eight instances. Compared to `GQAP`, `BRKGA` found better average solutions on 29 of the 45 instances. In 32 of the 45 instances `GevPR-HMP` found a best solution that was strictly better than the solutions found by the other heuristics. In 8 of the 45 instances `GQAP` found a best solution that was strictly better than the solutions found by the other heuristics. On a single instance (200_50_270003), `BRKGA` found a best solution that was strictly better than the solutions found by the other heuristics. On this instance its average solution value was also smaller than those of the other two heuristics.

To compare the three randomized heuristics, we ran each heuristic on each of the 45 large instances 300 times for instances with 100 base stations and 100 times for instances with 200 or 400 base stations. Each run was independent of the other and was limited to a maximum of 800 seconds. Each run was stopped when a solution with handover count less than or equal to a target solution value was found. Each target value was determined by running `GevPR-HMP` independently five times for 150 seconds. The target value for each instance was set as the average best solution value, rounded up to the nearest integer, over the five runs. Tables 6 and 7 summarize these experiments. For each instance, the tables list the instance name and size, the target solution value, and for each heuristics, the average best solution value found and the average time to find the best solution value. `GevPR-HMP` found the smallest average solution values on all but five instances. On those five, `BRKGA` found the smallest on four and `GQAP` found the smallest on one.

Figure 6 shows time-to-target plots (Aiex et al., 2007) for nine of the 45 large instances: 100_15_270001, 100_25_270003, 100_50_270004, 200_15_270001, 200_25_270002, 200_50_270005, 400_15_270002, 400_25_270003, and 400_50_270002. For each instance let K_h be the number of the K (100 or 300) trials that heuristic h (`GevPR-HMP`, `GQAP`, or `BRKGA`) found the target solution in at most 800 seconds and let $t_1, t_2, \ldots, t_{K_n}$ be the times, sorted in increasing order, that the target solution was found in the K_h successful trials of heuristic h. The time-to-target plot for heuristic h consists of the points

$$(t_1, 1/K), (t_2, 2/K), \ldots (t_{K_h}, K_h/K).$$

The point $(t_i, i/K)$ is such that i/K corresponds to the cumulative probability that the heuristic will find the target solution in at most t_i seconds. In these figures, a heuristic that finds the target solution in all trials in at most 800 seconds will

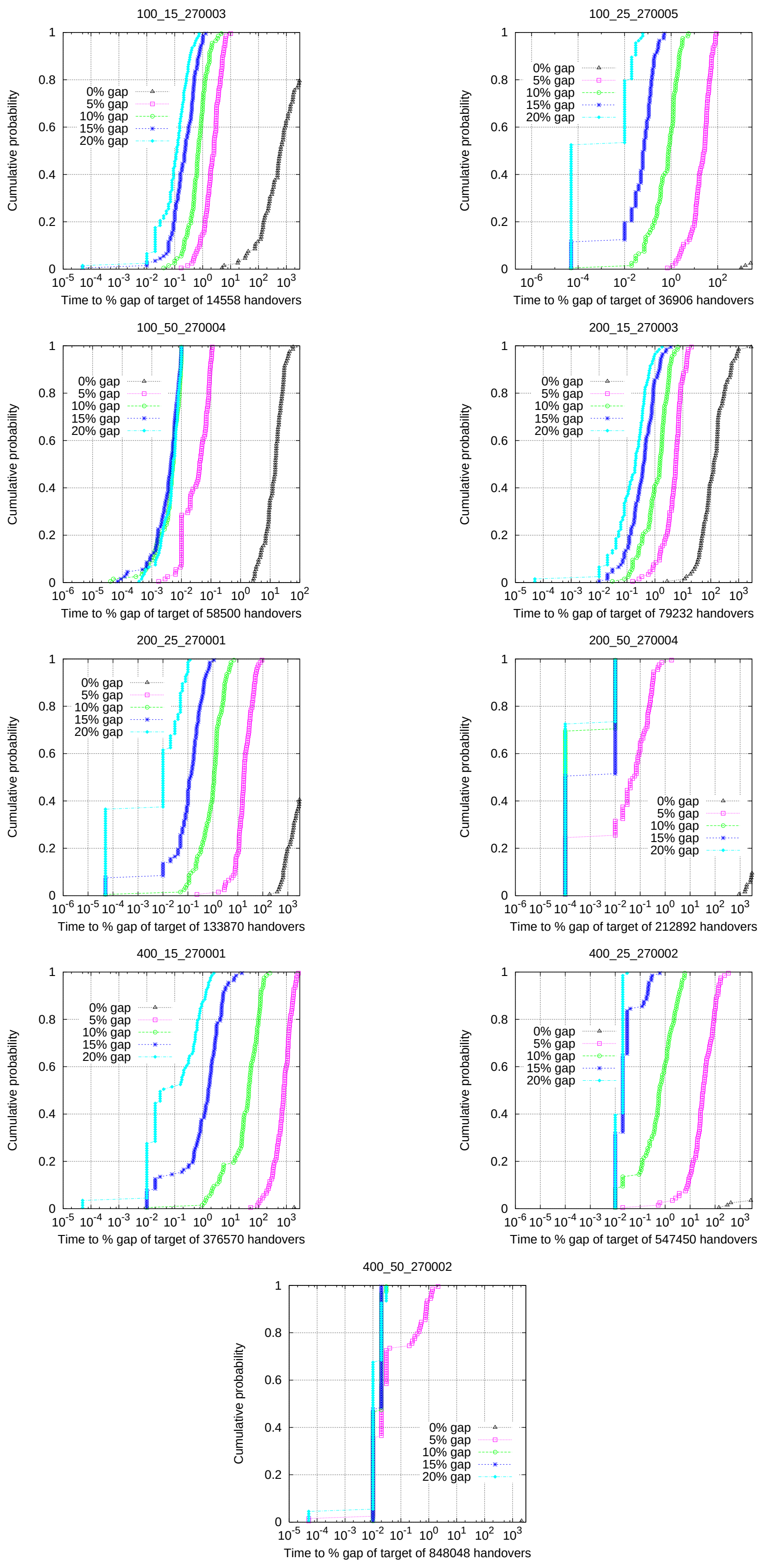

FIGURE 4. Performance profiles of `GevPR-HMP` (time in seconds)

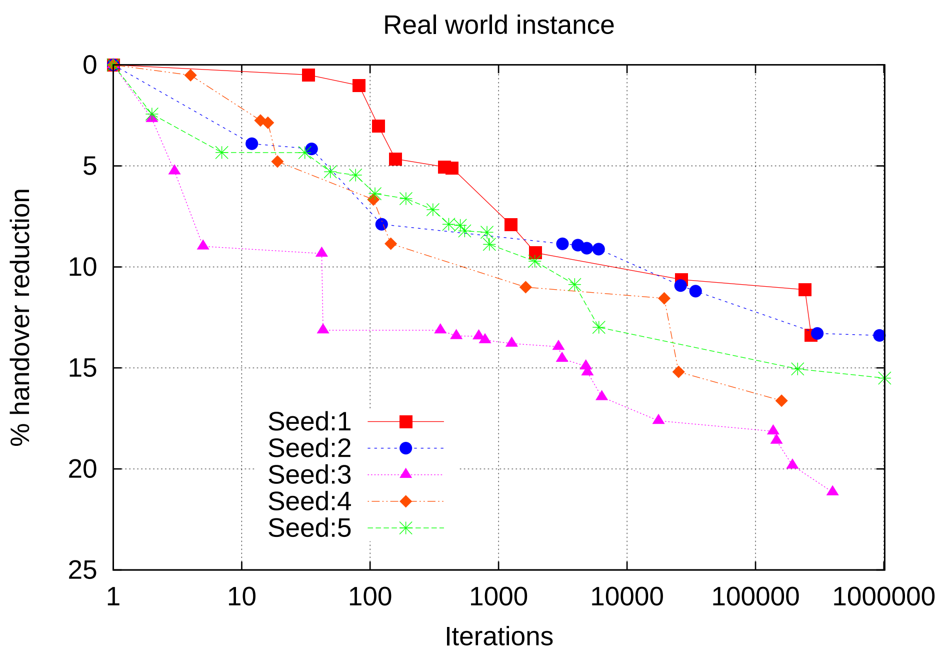

FIGURE 5. Progress of best solution for five independent runs of `GevPR-HMP`: Percentage handover reduction as a function of number of iterations.

have a time-to-target plot with cumulative probability going from 0 to 1, whereas a heuristic which, for example only finds the target in half of the trials, will have a time-to-target plot with cumulative probability going from 0 to 0.5.

Figure 6 shows time-to-target plots for nine of the 45 large instances. We make the following observations regarding these plots:

- In all but one instance, `GevPR-HMP` found the target solution on all trials. In that instance (200_15_270001) it failed to find the target on 32 of 100 trials.
- In two of the instances, `GevPR-HMP` was the only heuristic that found the target solution in any trial.
- In all but two instances, `BRKGA` found the target solution in at least one trial. On 200_15_270001, it was the only heuristic to find the target solution on all trials.
- `GQAP` only found the target solution on instances with 100 base stations, the most successful of which was on 100_50_270004 on which it failed on only four trials. On this instance, `GevPR-HMP` also found the target solution on all trials, but `BRKGA` found the target solution on only 16 of the 300 trials.
- The time-to-target plots show the dominance of `GevPR-HMP` over `BRKGA` as well as the dominance of `BRKGA` over `GQAP`.
- The time-to-target plots of `BRKGA` show running times for the sequential version of the algorithm. The parallel version would show a left shift in the plot.

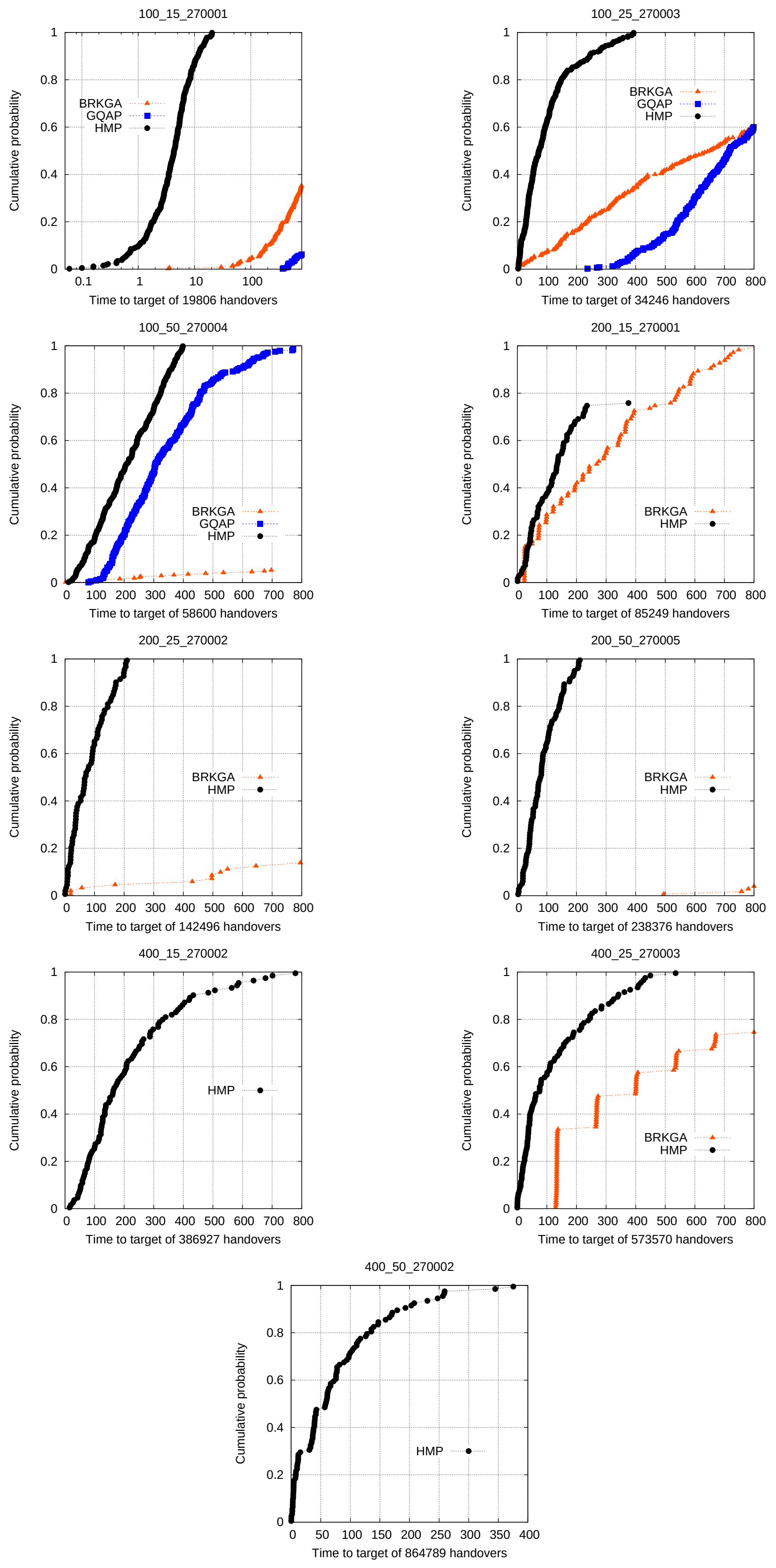

FIGURE 6. Runtime distributions of `GevPR-HMP`, `GQAP`, and `BRKGA` for specific target solutions (time in seconds)

- GQAP was implemented in Java while both GevPR-HMP and BRKGA were implemented in C++. It is not clear that this affected the results. One issue that may have impacted the performance of GQAP is that fact that it is for solving generalized quadratic assignment problems. For example GQAP sorts the inter-facility distances at each iteration even though all inter-facility distances are one.

Figure 4 shows time-to-target plots of GevPR-HMP for nine instances of different sizes. Each plot shows five time-to-target plots, one for each different target value. In each case a 0% gap target value $f^{\oplus}$ was determined to be an integer value at most 1% greater than the average solution found by GevPR-HMP in the one-day run experiments. Four other targets were computed as $\lceil 1.05 \times f^{\oplus} \rceil$, $\lceil 1.10 \times f^{\oplus} \rceil$, $\lceil 1.15 \times f^{\oplus} \rceil$, and $\lceil 1.20 \times f^{\oplus} \rceil$. For each target, 100 independent runs of GevPR-HMP were carried out. The maximum allowed time per run was 3000 seconds, which explains why several 0% gap runs failed to find the target solution. We make the following observations regarding these experiments:

- Only for two instances (100_50_270004 and 200_15_270003) did GevPR-HMP find the 0% gap target on all 100 trials.
- On all instances, GevPR-HMP found the 5% gap target (as well as the 10%, 15%, and 20% targets) on all 100 trials.
- As expected the easier the targets get, the further to the left the time-to-target plots are shifted.
- The plots show that most runs find in the initial iterations solutions that are within 15% to 20% of the 0% gap target. In the largest instance (400_50_270002), around 50% of the initial solutions are within 5% of the 0% gap target value.

Finally, Figure 5 shows percent reduction of number of handovers with respect to the solution value of the first GRASP iteration for handover minimization problem taken from a real instance from a large wireless service provider. This instance has about 1000 base stations and 30 RNCs. The figure shows reduction as a function of the number of GRASP iterations for five independent runs of GevPR-HMP, each using a different seed for the random number generator. GevPR-HMP was allowed to run for up to one million iterations. Maximum percent reductions varied from 13.4% to 21.1%. This is in agreement with our observations on the synthetic instances.

4. Concluding remarks

The handover minimization problem (HMP) arises in the operation of mobility networks. Base stations which connect to mobile devices handover connections to other base stations as devices change their locations. The handover operation is not actually done by the base stations but rather by a radio network controller (RNC). Each base station is assigned to an RNC and uses up part of the RNC's processing capacity with its traffic. Handovers between base stations assigned to different RNCs tend to fail more often than those between base stations assigned to the same RNC leading to more dropped connections between mobile devices and the network. In the HMP we wish to assign base stations to RNCs such that the RNC capacity is not violated and the number of handovers between base stations assigned to different RNCs is minimized. Since traffic and handover patterns change and new base stations and RNCs are introduced, the HMP must be solved on an ongoing basis.

In this paper we present a mixed integer programming formulation for the HMP and introduce three randomized heuristics for this problem: a GRASP with path-relinking for the generalized quadratic assignment problem, a GRASP with evolutionary path-relinking, and a biased random-key genetic algorithm.

After showing that state-of-the-art mixed integer programming solvers can only handle small instances of the HMP, we compare the three heuristics on a set of 83 synthetic instances of the HMP and show how one of the heuristics performs on a real-world instance from a large wireless services provider.

The experiments show that the three heuristics can find optimal and near-optimal solutions. Of the three, the GRASP with evolutionary path-relinking appears to perform best even though there are instances where the other two heuristics find solutions with fewer handovers.

Acknowledgment

This research was done while L.F. Morán-Mirabal and R.M.A. Silva were Visiting Scholars at AT&T Labs Research in Florham Park, New Jersey. The research of R.M.A Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais (FAPEMIG), Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES), and Foundation for the Support of Development of the Federal University of Pernambuco (FADE).

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.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.

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

Y. Deng and J.F. Bard. A reactive GRASP with path relinking for capacitated clustering. *J. of Heuristics*, 17:119–152, 2011.

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.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81:229–256, 1998.

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.

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.

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

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

G.R. Mateus, M.G.C. Resende, and R.M.A. Silva. GRASP with path-relinking for the generalized quadratic assignment problem. *J. of Heuristics*, 17:527–565, 2011.

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.

A. Mehrotra and M.A. Trick. Cliques and clustering: A combinatorial approach. *Operations Research Letters*, 22:1–12, 1997.

L.F. Morán-Mirabal, J.L. Gonzalez-Velarde, and M.G.C. Resende. Automatic parameter tuning of grasp with evolutionary path-relinking heuristics with a biased random-key genetic algorithm. Technical report, AT&T Labs Research, Florham Park, New Jersey, June 2012.

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.

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.

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.

University of Wisconsin. Condor high throughput computing, 2012. `research.cs.wisc.edu/condor`, last visited on June 25, 2012.

(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`

(Ricardo M. A. Silva) Centro de Informática (CIn), Federal University of Pernambuco, Av. Prof. Luís Freire s/n, Cidade Universitária, Recife, PE, Brazil.
E-mail address: `rmas@cin.ufpe.br`