A GRASP with path-relinking for the p-median problem [*]

Mauricio G. C. Resende[†] Renato F. Werneck[‡]

Abstract

Given n customers and a set F of m potential facilities, the p-median problem consists in finding a subset of F with p facilities such that the cost of serving all customers is minimized. This is a well-known NP-complete problem with important applications in location science and classification (clustering). We present here a GRASP (Greedy Randomized Adaptive Search Procedure) with path-relinking to find near-optimal solutions to this problem. Empirical results on instances from the literature suggest that this is a very robust algorithm, performing at least as well as other methods, and often better in terms of both running time and solution quality.

1 Introduction

The p-median problem is defined as follows. Given a set F of m potential facilities, a set U of n users (or customers), a distance function $d : U \times F \to \mathcal{R}$, and a constant $p \leq m$, determine which p facilities to open so as to minimize the sum of the distances from each user to its closest open facility.

Several algorithms for the p-median problem have been proposed, including exact methods based on linear programming [2, 3, 8, 30], constructive algorithms [3, 17, 38], dual-based algorithms [8, 21], and local search procedures [13, 16, 19, 24, 27, 35, 38]. More recently, metaheuristics capable of obtaining solutions of near-optimal quality have been devised. Tabu search procedures have been proposed by Voss [37] and Rolland et al. [26]. The latter method was compared in [29] with Rosing and ReVelle's *heuristic concentration* method [28], which obtained comparatively superior results. In [14], Hansen and Mladenović propose a VNS (variable neighborhood search) for the problem, later parallelized by García-López et al. in [10]. A variation of this method, VNDS (variable neighborhood decomposition search), was proposed by Hansen, Mladenović, and Perez-Brito [15]. Heuristics based on linear programming have been proposed by du Merle et al. [4] and by Senne and Lorena [32, 33].

In this paper, we propose a hybrid GRASP (Greedy Randomized Adaptive Search Procedure) for the p-median problem. GRASP is a randomized multistart iterative metaheuristic [5, 6, 7, 23], in which each iteration consists of a constructive phase (when a greedy randomized algorithm is performed) followed by a local search phase. The best solution obtained over all iterations is taken as the result of the whole algorithm.

Our algorithm enhances this basic approach with some intensification strategies. We keep a pool with some of the best solutions found in previous iterations, the so-called *elite solutions*. In each iteration of the GRASP, the solution obtained after the local search phase is combined with one elite solution through a process called *path-relinking*. Furthermore, after all iterations are completed, we have a second, post-optimization phase in which elite solutions are combined with each other. Figure 1 shows the pseudo-code for our overall strategy.

[*]AT&T Labs Research Technical Report TD-5E53XL, September 18, 2002.

[†]AT&T Labs Research, Florham Park, NJ 07932. E-mail: mgcr@research.att.com.

[‡]Princeton University, Princeton, NJ 08544. E-mail: rwerneck@cs.princeton.edu. The results presented in this paper were obtained while this author was a summer intern at AT&T Labs Research.

```
function GRASP ($seed, maxit, elitesize$)
    randomize($seed$);
    init($elite, elitesize$)
    for $i = 1$ to $maxit$ do
        $S \leftarrow$ randomizedBuild();
        $S \leftarrow$ localSearch($S$);
        $S' \leftarrow$ select($elite, S$);
        if ($S' \neq$ NULL) then
            $S' \leftarrow$ pathRelinking($S, S'$);
            add($elite, S'$);
        endif
        add($elite, S$);
    endfor
    $S \leftarrow$ postOptimize($elite$);
    return $S$;
end GRASP
```

Figure 1: Pseudo-code for the GRASP procedure

Of course, much remains to be specified to turn this outline into an actual algorithm. We study each individual component (constructive algorithm, local search, and intensification) separately in Sections 3, 4, and 5, respectively. In Section 6, we present the results obtained by the final version of our method and compare them with those obtained by other methods in the literature. But first, in Section 2, we discuss important issues related to the experiments we conducted both to evaluate individual components and to produce the final results.

2 Testing

2.1 Instances

We tested our algorithm on five classes of problems: TSP, ORLIB, SL, GR, and RW.

Instances in class TSP are just sets of points on the plane. Originally proposed for the traveling salesman problem, they are available from the TSPLIB [22]. In the context of the p-median problem, these instances were first used by Hansen et al. [14, 15]. Every point is considered both a potential facility and a customer, and the cost of assigning customer c to facility f is simply the Euclidean distance between the points representing c and f. As in [15], three instances are considered (fl1400, pcb3038, and rl5934, with 1400, 3038, and 5934 nodes, respectively), each with several different values for p (number of facilities to open).

Class ORLIB (short for OR-Library) was introduced by Beasley in [2]. Each of the 40 instances (pmed01, pmed02, ..., pmed40) in the class is a graph with a corresponding value for p. Every node is a customer and a potential facility, and the cost of assigning a customer to a facility is the length of the shortest path between the corresponding nodes. The number of nodes in this class varies from 100 to 900, and the value of p from 5 to 200.

The third class we consider is SL, a slight extension to ORLIB proposed by Senne and Lorena in [32]. It contains three new instances, all based on graphs from ORLIB: sl700 uses the same graph as pmed34, but uses $p = 233$; sl800 is the same as pmed37, with $p = 267$; and sl900 is pmed40 with $p = 300$ [31].

The fourth class studied is GR, introduced by Galvão and ReVelle [9] and used for the p-median problem by Senne and Lorena [32]. This class contains two graphs, with 100 and 150 nodes (named gr100 and gr150, respectively). Eight values of p (between 5 and 50) were considered in each case.

The fifth class we study, RW, was originally used in [24] and corresponds to completely random distance matrices. The distance between each facility and each customer has an integer value taken uniformly at random from the interval $[1, n]$, where n is the number of customers.[1] In all instances, the number of potential facilities (m) is equal to the number of customers (n). Four different values of n were considered: 100, 250, 500, and 1000 (instance names are rw100, rw250, rw500, and rw1000, respectively). In each case, several different values of p were tested. The program that created these instances, which uses the random number generator by Matsumoto and Nishimura [20], is available from the authors upon request.

Costs are integral in all classes except TSP, in which distances are, in theory, real values. We did not explicitly round nor truncate values, which were kept with `double` precision throughout the algorithm.

Results obtained by the final version of our algorithm on all instances are shown in Section 6. We also conducted experiments with several variations of the algorithm to assess how each individual component (constructive algorithm, path-relinking, local search, among others) affects the overall performance of the method. In those experiments, however, we used only a *restricted set* of instances. This set was chosen with two goals in mind. First, its instances should be hard enough to make evident the differences between various parameters and components. Some instances, especially in class ORLIB, can be solved to optimality by local search alone, thus making it pointless to include them in these experiments. Our second goal was to keep the set small enough so as to allow statistically meaningful experiments (i.e., with several random seeds for each instance) on a relatively small amount of CPU time (no more than a few days per experiment). Given those goals and our experience from early versions of our algorithm, we defined the restricted set with 10 instances: pmed15 and pmed40, both from class ORLIB; sl700, from class SL; fl1400 (with $p = 150$ and $p = 500$) and pcb3038 (with $p = 30$ and $p = 250$), from class TSP; gr150 (with $p = 25$), from class GR; and rw500 (with $p = 25$ and $p = 75$), from class RW.

2.2 Testing Environment

Tests were performed on an SGI Challenge with 28 196-MHz MIPS R10000 processors (with each execution of the program limited to only one processor) and 7.6 GB of memory. The algorithm was coded in C++ and compiled with the SGI MIPSpro C++ compiler (v. 7.30) with flags `-O3 -OPT:Olimit=6586`. All running times shown in this paper are CPU times, measured with the `getrusage` function. Unless noted otherwise, running times do not include the time spent reading instances from disk (in particular, in classes SL and ORLIB, they do include the time necessary to compute the all-pairs shortest paths from the graph read). The random number generator we use is Matsumoto and Nishimura's *Mersenne Twister* [20].

3 Constructive Algorithms

The standard greedy algorithm for the p-median problem [3, 38] starts with an empty solution and adds facilities one at a time; in each iteration, it chooses the most profitable facility, i.e., the one whose addition causes the greatest drop in solution cost. Evidently, this method cannot be used directly within the GRASP framework: being completely deterministic, it would yield identical solutions in all iterations. We considered the following randomized variations in our experiments:

- random (random solution): Select p facilities uniformly at random. The selection itself requires $O(m)$ time, and determining which facility should serve each customer requires $O(pn)$ operations.[2] Therefore, the overall complexity of the algorithm is $O(m + pn)$.
- rpg (random plus greedy): Select a fraction α (an input parameter) of the p facilities at random, then complete the solution in a greedy fashion. The worst-case complexity of the algorithm is $O((1 - \alpha)(pmn) + \alpha(m + pn))$, which corresponds to $O(pmn)$ if α is not very close to 1. In our tests, a value for α was chosen uniformly at random in the interval $[0; 1]$ in every GRASP iteration.

[1] In particular, unlike all other series, the distance from facility i to user i is not zero. Moreover, the distance between facility i and user j need not be equal to the distance between facility j and user i.

[2] This can be made faster in some settings, like sparse graphs or points in the Euclidean plane. The results in this paper, however, do not use any such metric-specific accelerations.

- rgreedy (randomized greedy): Similar to the greedy algorithm, but in each step i, instead of selecting the best among all $m-i+1$ options, choose randomly from the $\lceil\alpha(m-i+1)\rceil$ best options, where $0<\alpha\leq 1$ is an input parameter. Note that if $\alpha\rightarrow 0$, this method degenerates into the greedy algorithm; if $\alpha\rightarrow 1$, it turns into the random algorithm. In our tests, we selected α uniformly at random in the interval $(0;1]$ in each GRASP iteration. This algorithm takes $O(pmn)$ time.

- pgreedy (proportional greedy): Yet another variation of the greedy algorithm. In each iteration, compute, for every candidate facility f_i, how much would be saved if f_i were added to the solution. Let $s(f_i)$ be this value. Then pick a candidate at random, but in a biased way: the probability of a given facility f_i being selected is proportional to $s(f_i)-\min_j s(f_j)$. If all candidates are equally good (they would all save the same amount), select one uniformly at random. This method also takes $O(pmn)$ time.

- pworst (proportional worst): In this method, the first facility is selected uniformly at random. Other facilities are added one at a time as follows. Compute, for each customer, the difference between how much its current assignment costs and how much the best assignment would cost; then select a customer at random, with probability proportional to this value, and open the closest facility. Customers for which the current solution is particularly bad have a greater chance of being selected. This method, also used in [34], runs in $O(mn)$ time.

- sample (sample greedy): This method is similar to the greedy algorithm, but instead of selecting the best among all possible options, it only considers $q<m$ possible insertions (chosen uniformly at random) in each iteration. The most profitable among those is selected. The running time of the algorithm is $O(m+pqn)$. The idea is to make q small enough so as to significantly reduce the running time of the algorithm (when compared to the pure greedy one) and to ensure a fair degree of randomization. In our tests, we used $q=\lceil\log_2(m/k)\rceil$.

It was not clear at first which of these methods would be most adequate as a building block of our heuristic. To better analyze this issue, we conducted an experiment on the restricted set of instances (defined in Section 2.1). For every instance in the set and every constructive procedure, we ran our heuristic 10 times, with 10 different random seeds. In every case, we ran 32 GRASP iterations, with 10 elite solutions, using up:down as the criterion to determine the direction of path-relinking (this criterion is defined in Section 5.4.2).

To explain the results shown in Table 1, we need some definitions. For each instance I, we compute $avg(I)$, the average solution value (obtained by the entire GRASP procedure with post-optimization) considering all 60 executions (6 different methods, each with 10 random seeds). Then, for each method, we determine the *relative percentage deviation* for that instance: how much the average solution value obtained by that method is above (or below) $avg(I)$, in percentage terms. By taking the average of these deviations over all 10 instances, we obtain the *average relative percentage deviation* for each method; these are the values shown in column 2 of Table 1. Column 4 was computed in a similar fashion, but considering running times instead of solution values.

Columns 3 and 5 were computed as follows. For each instance, the methods were sorted according to their relative percentage deviations; the best received one point, the second two points, and so on, until the sixth best method, with six points. When there was a tie, points were divided equally between the methods involved. For example, if the deviations were -0.03, -0.02, -0.02, 0.01, 0.03, and 0.03, the methods would receive $1, 2.5, 2.5, 4, 5.5$, and 5.5 points, respectively. The number of points received by a method according to this process is its *rank* for that particular instance. Its *normalized rank* was obtained by linearly mapping the range of ranks (1 to 6, in this case) to the interval $[-1,1]$. In the example above, the normalized ranks would be $-1, -0.4, -0.4, 0.2, 0.8$, and 0.8. The normalized must add up to zero (by definition). If a method is always better than all the others, its normalized rank will be -1; if always worse, it will be 1. What columns 3 and 5 of Table 1 show are the *average normalized ranks* of each method, taken over the set of 10 instances. Column 3 refers to solution quality, and column 5 to running times.

The correlation between these measures is higher when one method is obviously better (or worse) than other methods. In general, however, having a lower average normalized rank does not imply having a better

Table 1: Average GRASP results with different constructive procedures: Average relative percentage deviations (%DEV) and average normalized ranks (NRANK) for solution qualities and running times (both referring to the entire GRASP procedure with post-optimization). Smaller values are better.

METHOD	QUALITY		TIME	
	DEV%	NRANK	DEV%	NRANK
pgreedy	-0.009	0.160	39.6	0.920
pworst	-0.006	-0.400	-18.7	-0.480
rgreedy	0.020	-0.160	35.8	0.400
random	0.015	0.000	-24.9	-0.840
rpg	0.009	0.620	-12.3	-0.300
sample	-0.029	-0.220	-19.5	-0.600

average relative percentage deviation, as the table shows.

The table makes it evident that the methods are distinguishable much more by the time they take than the quality of the solutions they provide. As the analysis of their worst case complexities suggests, rgreedy and pgreedy are much slower than the other methods. In fact, they are so much slower that, as shown in the table, they make the entire GRASP heuristic take twice as long on average than when using faster methods. The other method with $O(pmn)$ worst-case performance, rpg, is much faster than rgreedy and pgreedy in practice, but it is still slower than other methods without finding better solutions on average. We therefore tend to favor the three relatively fast methods: pworst, sample, and random. Among those, sample and pworst seem to be able to find solutions of slightly better quality. We chose sample for the final version of our algorithm, although pworst would probably find very similar results.

One cannot help but notice that this experiment reveals an unusual feature of the p-median problem. Generally, in the GRASP framework, the running time of the constructive heuristic is not an issue. It is usually a good idea to come up with randomized constructive methods that produce a variety of high-quality solutions, since this would reduce the number of iterations of the generally much slower local search procedure. In our case, the local search is so fast (and the constructive algorithms relatively so slow), that investing extra time building the solution can actually make GRASP much slower without any significant gain in terms of solution quality. We could not use the randomization strategy normally used in GRASP, represented here by rgreedy; instead, we had to develop a faster alternative based on sampling.

4 Local Search

The standard local search procedure for the p-median problem, originally proposed by Teitz and Bart [35] and studied or used by several authors [10, 14, 15, 16, 24, 38], is based on swapping facilities. Given an initial solution S, the procedure determines, for each facility $f \notin S$, which facility $g \in S$ (if any) would improve the solution the most if f and g were interchanged (i.e., if f were opened and g closed). If there is one such "improving" move, f and g are interchanged. The procedure continues until no improving interchange can be made, in which case a *local minimum* will have been found.

In [38], Whitaker describes an efficient implementation of this method, which he called *fast interchange*. A similar implementation is used by Hansen and Mladenović in [14] (and later, by several authors, in [10] and [15]). The only minor difference between them is the fact that Whitaker prefers a *first improvement* strategy (the algorithm moves to a neighboring solution as soon as it finds an improving one), while in the other cases the preferred strategy is *best improvement* (all neighbors are checked and the very best is chosen). In either case, the running time of each iteration is bounded by $O(mn)$.

In [24], Resende and Werneck propose an alternative implementation for the local search procedure. Although it has the same worst case complexity as Whitaker's, it can be substantially faster in practice. The speedup (of up to three orders of magnitude) results from the use of information gathered in early iterations of the algorithm to reduce the amount of computation performed in later stages. This, however, requires a greater amount of memory. While Whitaker's implementation requires $O(n)$ memory in the worst case (not considering the distance matrix), the implementation in [24] may use up to $O(mn)$ memory positions.

In any case, we believe that the speedup is well worth the extra memory requirement. This is especially true for methods that rely heavily on local search procedures, which includes not only the method described here (GRASP), but also VNS [14] and tabu search [26, 37], for example. Furthermore, one should also remember that while the extra memory is asymptotically relevant when the distance function is given implicitly (as in the case of Euclidean instances), it is irrelevant when there is an actual $O(mn)$ distance matrix (as in series RW). Given these considerations, we opted for using in this paper the faster version described in [24], which requires $\Theta(mn)$ memory positions.

Since it is hardly trivial, we abstain from describing the implementation here. The reader is referred to [24] for details and for an experimental comparison with Whitaker's implementation.

5 Intensification

In this section, we describe the intensification aspects of our heuristic. It works through a pool of *elite solutions*, high-quality solutions found during the execution. Intensification occurs in two different stages, as Figure 1 shows. First, every GRASP iteration contains an intensification step, in which the newly generated solution is combined with a solution from the pool. Then, in the post-optimization phase, solutions in the pool are combined among themselves. In both stages, the strategy used to combine a pair of solutions is the same: *path-relinking*. Originally proposed for tabu search and scatter search [11, 12], this procedure was first applied within the GRASP framework by Laguna and Martí [18], and widely applied ever since (see [23] for some examples). Subsection 5.1 briefly describes how path-relinking works. Subsection 5.2 explains the rules by which the pool is updated and solutions are taken from it. Finally, Subsection 5.3 describes the post-optimization phase.

5.1 Path-relinking

Let S_1 and S_2 be two valid solutions, interpreted as sets of (open) facilities. The path-relinking procedure starts with one of the solutions (say, S_1) and gradually transforms it into the other (S_2) by swapping in elements that are in $S_2 \setminus S_1$ and swapping out elements that are in $S_1 \setminus S_2$. The total number of swaps made is $|S_2 \setminus S_1|$, which is equal to $|S_1 \setminus S_2|$; this value is known as the *symmetric difference* between S_1 and S_2. The choice of which swap to make in each stage is greedy: we always perform the most profitable (or least costly) move.

As mentioned in [23], the outcome of the method is usually the best solution found in the path from S_1 to S_2. Here we use a slight variation: the outcome is the best *local minimum* in the path. A local minimum in this context is a solution that is both succeeded (immediately) and preceded (either immediately or through a series of same-value solutions) in the path by strictly worse solutions. If the path has no local minimum, one of the original solutions (S_1 or S_2) is returned with equal probability. When there is an improving solution in the path, our criterion matches exactly the traditional one (returning the best element in the path). It is different only when path-relinking is unsuccessful, in which case we try to increase diversity by selecting some solution other than the extremes of the path.

In our implementation, we augmented the intensification procedure by performing a full local search on the solution produced by path-relinking. This is usually much faster than applications on randomized constructive solutions produced during GRASP, since it starts from a solution that is either a local optimum or very close to being one. A "side effect" of applying local search at this point is increased diversity, since we are free to use facilities that did not belong to any of the original solutions.

It is interesting to note that path-relinking itself is remarkably similar to the local search procedure described in Section 4, with two main differences. First, the number of allowed moves is restricted: only elements in $S_2 \setminus S_1$ can be inserted, and the ones in $S_1 \setminus S_2$ can be removed. Second, non-improving moves are allowed. However, these differences are subtle enough to be easily incorporated into the basic implementation of the local search procedure. In our implementation, both procedures share much of their code.

5.2 Pool Management

An important aspect of the algorithm is the management of the pool of elite solutions. Empirically, we observed that an application of path-relinking to a pair of solutions is less likely to be successful if the solutions are very similar. The longer the path between the solutions, the greater the probability that an entirely different local optimum (as opposed to the original solutions themselves) will be found. It is therefore reasonable to take into account not only solution quality, but also diversity when dealing with the pool of elite solutions.

The pool must support two essential operations: addition of new solutions (represented by the `add` function in Figure 1) and selection of a solution for path-relinking (the `select` function in the pseudo-code). We describe each of these in turn.

Updates. For a solution S with cost $v(S)$ to be added to the pool, two conditions must be met. First, its symmetric difference from all solutions in the pool whose value is less than $v(S)$ must be at least four (after all, path-relinking between solutions that differ in fewer than four facilities cannot produce solutions that are better than the original ones). Second, if the pool is full, the solution must be at least as good as the worst elite solution (if the pool is not full, this is obviously not necessary).

If these conditions are met, the solution is inserted. If the pool is not full and the new solution is not within distance four of any other elite solution (including worse ones), it is simply added. Otherwise, it replaces the most similar solution among those of equal or higher value.

Selection. In every iteration of the algorithm, a solution is selected from the pool (Figure 1, line 6) and combined with S, the solution most recently found. An approach that has been applied to other problems with some degree of success is to select a solution uniformly at random [23]. However, this often means selecting a solution that is too similar to S, therefore making the procedure unlikely to find good new solutions. Therefore, instead of picking solutions from the pool uniformly, we take them with probabilities proportional to their difference with respect to S. In Section 5.4.3, we show that this strategy does pay off.

5.3 Post-optimization

In the process of looking for a good solution, a GRASP heuristic produces not one, but several different local optima, whose values are often not too far from the value of the best solution found. The *post-optimization phase* in our algorithm tries to combine these solutions in order to obtain even better ones. This phase takes as input the pool of elite solutions, whose construction was described in previous sections. Every solution in the pool is combined with each other by path-relinking. The solutions generated by this process are added to a new pool of elite solutions (following the same constraints described in Section 5.2), representing a new *generation*. This process is repeated until it creates a generation in which the best solution is not better than the best found in previous generations. A similar multi-generation path-relinking strategy has been used successfully in [1, 25].

5.4 Empirical Analysis

In this section, we analyze empirically some details related to intensification that were mentioned in the previous subsections. First, in Section 5.4.1 we show how the execution of path-relinking during the first stage of the algorithm (and not only in the post-optimization stage) can help us find good solutions faster. Then, in Section 5.4.2, we examine the question of which *direction* to choose when performing path-relinking between two solutions S_1 or S_2: from S_1 to S_2, from S_2 to S_1, or both? Finally, in Section 5.4.3, we examine which selection strategy should be used.

5.4.1 Path-relinking within GRASP

Our implementation is such that the randomized constructive solution produced in each GRASP iteration depends only on the initial random seed, regardless of whether path-relinking is executed or not. Therefore, if the number of iterations is the same, the addition of path-relinking cannot decrease solution quality when compared to a "pure" GRASP. It could be the case, however, that the extra time spent on path-relinking, if used for extra standard GRASP iterations instead, would lead to even better results.

To test this hypothesis, we took a few representative instances and ran both versions of the heuristic (with and without path-relinking) for a period 100 times as long as the average time it takes to execute one iteration (construction followed by local search) *without* path-relinking. We were therefore able to compare the qualities of the solutions obtained as the algorithm progressed. The constructive algorithm used was sample. Results in this test do not include post-optimization. We selected one instance from each class (fl1400 from TSP, pmed40 from ORLIB, and rw500 from RW), and tested each with seven values of p, from 10 to roughly one third of the number of facilities (m). The test was run 10 times for each value of p, with 10 different seeds.

Figure 2 refers to one of the instances tested, fl1400 with $p = 500$. The graph shows how solution quality improves over time. Both quality and time are normalized. Times are given in multiples of the average time it takes to perform one GRASP iteration without path-relinking (this average is taken over all iterations of all 10 runs). Solution quality is given as a fraction of the average solution value found by the first GRASP iteration (without path-relinking).[3]

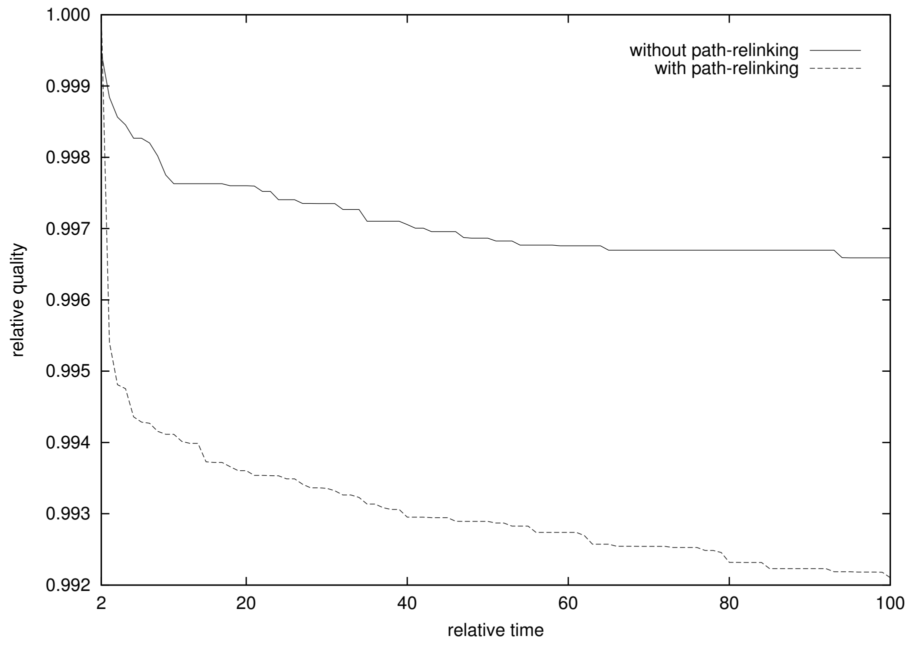

Figure 2: Instance fl1400, $p = 500$: Quality of the best solution found as a fraction of the average value of the first solution found. Times are given as multiples of the average time required to perform one GRASP iteration. Smaller values are better.

Figures 3, 4 and 5 refer to the same experiment. Each curve in those graphs represents an instance with a particular value of p. Times are normalized as before, and *quality ratio* (shown in the vertical axis) is simply

[3]Note that the first time value shown is 2; At time 1, not all ratios are defined, since in some cases the first iteration takes more than average time to execute.

the ratio of the average solution quality obtained with path-relinking to the one obtained without it, both considering the same running times. Values smaller than 1.000 favor path-relinking.

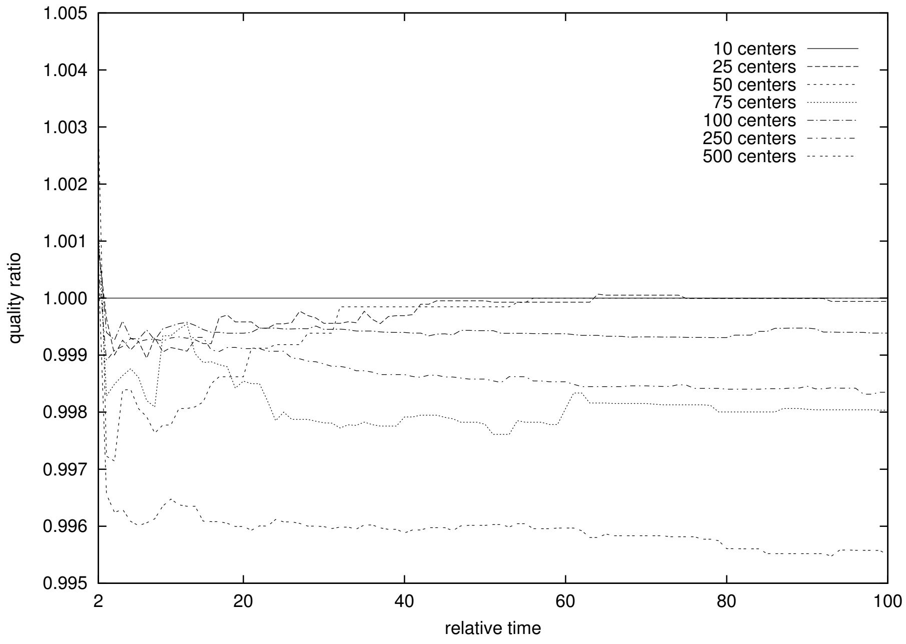

Figure 3: Instance fl1400 (class TSP): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one GRASP iteration. Values smaller than 1.000 suggest that it is worthwhile using path-relinking.

These results confirm what should be expected. If only very a few iterations are performed, path-relinking is not especially helpful; solutions of comparable quality (or even better) can be found using GRASP alone. However, if more time is to be spent, using path-relinking is a good strategy, consistently leading to solutions of superior quality within the same time frame. This is especially true for more difficult instances, those in which p has a relatively high value. The case of instance rw500 seems to be an exception; as p becomes greater than 75, the instance apparently becomes easier.

5.4.2 Direction

An important aspect of path-relinking is the *direction* in which it is performed. Given two solutions S_1 and S_2, we must decide in which direction to perform the relink: from S_1 to S_2, from S_2 to S_1, or both. We tested the following criteria:

- random: Direction picked uniformly at random.
- up: From the best to the worst solution (among the two); this has the potential advantage of exploring more carefully the most promising vicinity.
- down: From the worst to the best solution; by exploring more carefully the vicinity of the worst solution, it may be able to find good solutions that are relatively far from the best known solutions, thus favoring diversity.
- new: Always start from the newly generated solution, not from the one already in the pool (this strategy applies only to the first stage of the algorithm, not to the post-optimization phase). Again, the goal is to obtain greater solution diversity.

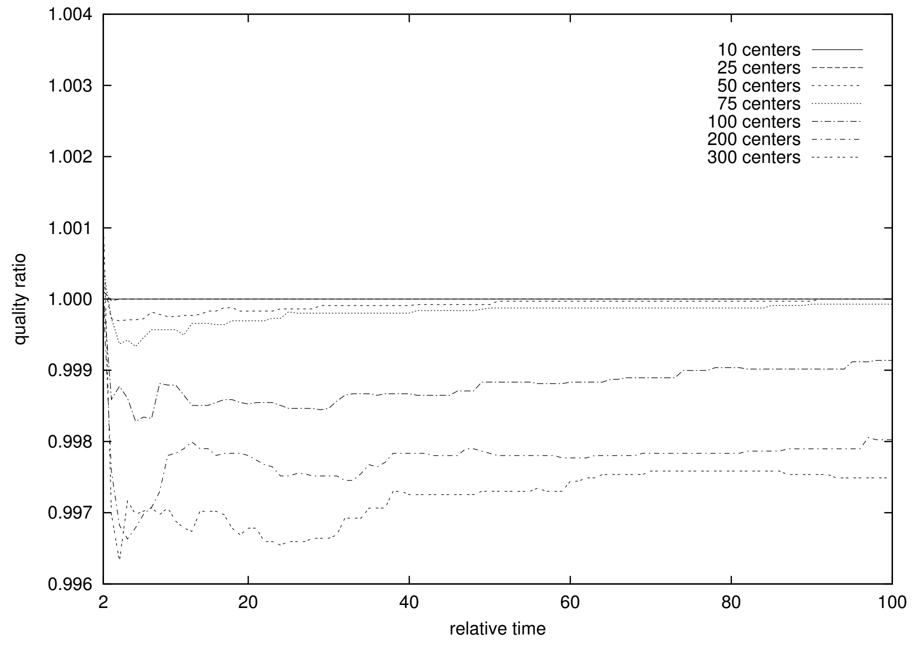

Figure 4: Instance pmed40 (class ORLIB): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one GRASP iteration. Values smaller than 1.000 suggest that it is worthwhile using path-relinking.

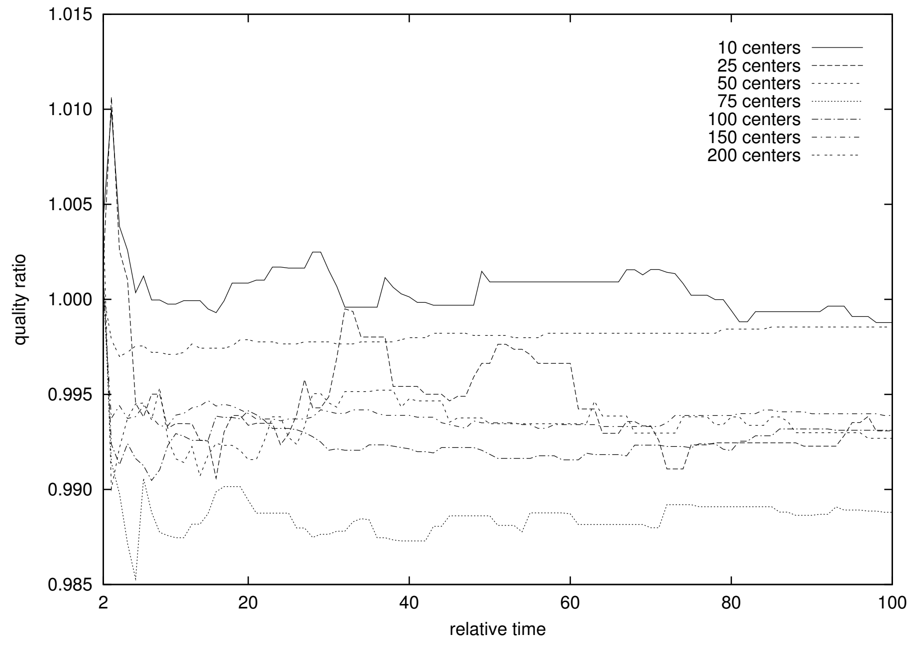

Figure 5: Instance rw500 (class RW): Ratios between partial solutions found with and without path-relinking. Times are normalized with respect to the average time it takes to execute one GRASP iteration. Values smaller than 1.000 suggest that it is worthwhile using path-relinking.

- none: Do not perform path-relinking (does not apply to post-optimization).
- both: Perform path-relinking in both directions and return the best result. This method finds the best possible results in each case, but it takes twice as long as the other methods to run.

We tested all valid combinations of these methods on the 10 instances of the restricted set defined in Section 2.1, each with 10 different random seeds. In each case, we ran GRASP with 32 iterations and 10 elite solutions, using sample as the constructive method. Tables 2, 3, and 4 show the results obtained in the experiment. (The definitions of *average relative percentage deviation* and *normalized relative rank*, used in these tables, are given in Section 3.)

Table 2: GRASP solution quality with different path-relinking strategies: Average relative percentage deviations. Each value represents how much the average solution value found by each method is above (or below) the average found by all methods. Smaller values are better.

GRASP	POST-OPTIMIZATION METHOD			
METHOD	both	down	random	up
none	0.056	0.056	0.033	0.024
both	0.005	0.009	-0.030	-0.007
down	-0.010	0.007	-0.009	-0.012
random	0.001	0.004	-0.002	0.001
new	-0.008	0.004	-0.007	-0.011
up	-0.029	-0.032	-0.019	-0.022

Table 3: GRASP solution quality with different path-relinking strategies: Average normalized ranks. Smaller values are better.

GRASP	POST-OPTIMIZATION METHOD			
METHOD	both	down	random	up
none	-0.017	0.565	0.448	0.465
both	-0.117	-0.143	-0.270	0.174
down	-0.357	0.270	-0.265	0.004
random	-0.183	0.209	-0.100	0.161
new	-0.387	-0.030	-0.135	0.078
up	-0.283	-0.209	-0.061	0.183

Table 4: GRASP running times with different path-relinking strategies: Average relative percent deviation with respect to the average.

GRASP	POST-OPTIMIZATION METHOD			
METHOD	both	down	random	up
none	33.3	-12.2	-7.3	-6.9
both	27.8	-2.3	-0.5	-2.3
down	22.7	-9.4	-7.8	-9.4
random	20.1	-12.0	-9.7	-10.9
new	20.3	-8.7	-8.0	-11.1
up	23.1	-9.3	-10.0	-9.6

Note that some strategies can be discarded for being too slow without any clear improvement in solution quality; that is the case of those that use strategy both in the post-optimization phase (and, also during the first stage of the algorithm, although the extra time in this case is far less relevant). Furthermore, it seems clear that using path-relinking during the first stage is important; even though it is still possible to obtain above-average solutions eventually if none is used in the first stage, this can only happen if both is the strategy used in the post-post-optimization phase — which results in much longer running times.

When the other strategies are considered, the results do not show any clearly dominant one. Several combinations of new, up, down, and random seem like reasonable choices for the algorithm. Of all combinations that do not use the more time consuming method both, five have better-than-average quality according to both measures used: up:down (the first method refers to the first stage of the algorithm, the second to the post-optimization stage), down:random, random:random, new:random, and up:random (see Tables 2 and 3). We decided to use up:down in the final version of our algorithm, since this was the method with the best average relative percentage deviation and a good average rank. This method has the interesting feature of favoring quality when dealing with lower-quality solutions (during the first stage of the algorithm), and diversity when the overall solution quality is higher (during the post-optimization stage).

5.4.3 Selection Strategy

We have shown that applying path-relinking during the first stage of the algorithm helps finding good solutions faster. Here, we analyze the criterion used to choose the elite solutions to be combined with S, the solution obtained after local search. Recall that the usual method is to select the solution uniformly at random, and that we propose picking solutions with probabilities proportional to their symmetric difference with respect to S. We will call these methods uniform and biased, respectively.

When performing path-relinking between a pair of solutions, our goal is to obtain a third solution of lower cost. We will consider the combination *successful* when this happens. The ultimate goal of the selection scheme is to find, among the elite solutions, one that leads to a successful combination. Better selection schemes will find one such solution with higher probability.

We devised the following experiment, run for each of the 10 instances in the restricted set defined in Section 2.1, to determine which method is better according to this criterion. First, run GRASP (without path-relinking) until a pool with capacity for 110 solutions is filled. Then, take the top 10 solutions (call them $E_1, E_2, \dots, E_{10}$) obtained and create a new pool. Denote the remaining 100 solutions by $S_1, S_2, \dots, S_{100}$. Perform path-relinking between each of these 100 solutions and each solution in the pool, and decide based on the results which selection method (biased or uniform) would have a greater probability of success if we had to select one of the 10 instances.

To calculate the probability of success of each method, we need some definitions. Let $s(i, j)$ be 1 if the path-relinking between S_i and E_j is successful, and 0 otherwise; also, let $\Delta(i, j)$ be the symmetric difference between S_i and E_j. For a given solution S_i, the probability of success for uniform, if it were applied, would be

$$u_i = \frac{\sum_{j=1}^{10} s(i, j)}{10}.$$

On the other hand, the probability of success of biased would be

$$b_i = \frac{\sum_{j=1}^{10} [s(i, j) \cdot \Delta(i, j)]}{\sum_{i=1}^{10} \Delta(i, j)}.$$

For each of the 10 instances, the procedure described above was executed 10 times, with 10 random seeds, always using sample as the constructive algorithm and up as the path-relinking direction. Therefore, for each instance, 1,000 selections were simulated (100 for each random seed).

The results are summarized in Table 5. For each instance, we show the percentage of the cases in which a method has greater probability of success than the other (when the probabilities are equal, we consider the experiment a tie).

Note that in all cases biased has superior performance, sometimes by a significant margin. It is interesting to note that in two cases the probability of a tie was almost 100%; this is due to the fact that path-relinking

Table 5: Comparison between the uniform and biased selection schemes. Values represent percentage of cases in which one method has greater probability of leading to a successful relink than the other.

INSTANCE		SELECTION METHOD		
NAME	p	uniform	TIE	biased
fl1400	150	38.7	10.8	50.5
fl1400	500	0.0	99.9	0.1
gr150	25	34.9	5.6	59.5
pcb3038	30	45.2	5.4	49.4
pcb3038	250	0.2	98.5	1.3
pmed15	100	14.5	4.5	81.0
pmed40	90	14.2	5.2	80.6
rw500	25	39.8	10.3	49.9
rw500	75	32.0	11.3	56.7
sl700	233	6.4	56.2	37.4

almost always works for those particular instances — any selection scheme would be successful. In the few situations in which there were "wrong" selections, biased was better in avoiding them.

6 Final Results

This section presents detailed results obtained by the final version of our method, built based on the experiments reported in previous sections. It uses sample as the randomized constructive heuristic (see Section 3); path-relinking is executed in both stages of the algorithm (Section 5.4.1); path-relinking is performed from the best to the worst solution during the first stage of the algorithm, and from the worst to the best during post-optimization (Section 5.4.2); and solutions are selected from the pool in a biased way during the first stage of the algorithm (Section 5.4.3). The results reported here refer to runs with 32 GRASP iterations and 10 elite solutions — of course, these numbers can be changed to make the algorithm faster (if they are reduced) or to obtain better solutions (if they are increased).

We present results for all instances mentioned in Section 2.1. Since ours is a randomized algorithm, different runs can yield different results. In order to avoid anomalous results, we ran it nine times for each instance tested. All solution values presented here refer to the *median* of the results obtained (this ensures that the value shown is indeed the value of a valid solution to the problem), whereas times are arithmetic means taken over all executions. The results are presented in Tables 6 to 12.

For reference, the tables contain the lowest upper bounds on solution values presented in the literature (to the best of our knowledge) for each of the instances tested. The optimum values are known for all instances in three classes: ORLIB (provided in [2]), SL [32], and GR [32]. For class TSP, we present the best upper bounds listed in the literature, as well as references to the papers that first presented the bounds shown (they are presented in the SOURCE column in Tables 6, 7, and 8). Although that is often the case, the bounds do not necessarily correspond to solutions found by the main heuristics described in those papers — in some cases, they were found by other, more time-consuming methods. For several instances, at least one of the nine runs of our GRASP procedure was able to improve the best bound known. When that was the case, the improved bound is presented, and the source column is marked by a dash (—). These values should not be considered the "final result" of our method when compared to others, since it represents especially successful runs; the meaningful results are the *medians* listed in the tables. Because class RW was recently introduced [24], there are no results obtained by heuristics in the literature. Therefore, the BEST column in Table 12 presents the best solutions found by our algorithm when all nine runs are considered.

Note that our method found solutions within 0.2% of the previous best known solutions in all cases. There were greater deviations for class RW, but note that they are calculated with respect to solutions found by GRASP itself. However, it does suggest that our method is capable of obtaining better results — in

Table 6: Final results for fl1400, an Euclidean instance from class TSP with 1400 nodes.

	BEST KNOWN		SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
p	VALUE	SOURCE	VALUE	%ERR	TIME	VALUE	%ERR	TIME
10	101249.47	[15]	101249.55	0.000	117.1	101249.55	0.000	118.5
20	57857.55	[15]	57857.94	0.001	76.8	57857.94	0.001	83.5
30	44013.48	—	44013.48	0.000	76.0	44013.48	0.000	106.2
40	35002.52	—	35002.60	0.000	68.6	35002.60	0.000	101.3
50	29089.78	[15]	29090.23	0.002	58.8	29090.23	0.002	73.9
60	25161.12	—	25166.91	0.023	57.0	25164.02	0.012	91.5
70	22125.53	[15]	22126.03	0.002	50.6	22126.03	0.002	70.2
80	19872.72	—	19878.45	0.029	49.8	19876.57	0.019	78.1
90	17987.94	[15]	18006.83	0.105	48.7	17988.60	0.004	74.2
100	16551.20	[14]	16567.01	0.096	47.3	16559.82	0.052	82.4
150	12026.47	—	12059.12	0.271	48.7	12036.00	0.079	132.5
200	9359.15	—	9367.98	0.094	49.4	9360.67	0.016	101.3
250	7741.51	—	7754.50	0.168	54.5	7746.31	0.062	130.3
300	6620.92	—	6637.81	0.255	57.8	6623.98	0.046	167.1
350	5720.91	—	5749.51	0.500	59.6	5727.17	0.109	177.6
400	5006.83	—	5033.96	0.542	64.1	5010.22	0.068	157.5
450	4474.96	—	4485.16	0.228	68.3	4476.68	0.038	170.7
500	4047.90	—	4059.16	0.278	71.9	4049.56	0.041	210.9

Table 7: Final results for pcb3038, an Euclidean instances from class TSP with 3038 nodes.

	BEST KNOWN		SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
p	VALUE	SOURCE	VALUE	%ERR	TIME	VALUE	%ERR	TIME
10	1213082.03	—	1213082.03	0.000	1115.8	1213082.03	0.000	1806.3
20	840844.53	—	840844.53	0.000	647.9	840844.53	0.000	943.4
30	677306.76	—	678108.52	0.118	426.7	677436.66	0.019	847.0
40	571887.75	—	572012.44	0.022	312.6	571887.75	0.000	492.6
50	507582.13	—	507754.72	0.034	251.7	507663.80	0.016	472.4
60	460771.87	—	461194.61	0.092	218.2	460797.55	0.006	481.4
70	426068.24	—	426933.75	0.203	201.3	426153.31	0.020	470.9
80	397529.25	—	398405.57	0.220	188.5	397585.89	0.014	555.9
90	373248.08	—	374152.75	0.242	182.3	373488.82	0.064	380.8
100	352628.35	—	353576.86	0.269	174.0	352755.13	0.036	448.1
150	281193.96	[34]	282044.70	0.303	163.3	281316.82	0.044	402.5
200	238373.26	—	238984.42	0.256	162.0	238428.35	0.023	406.9
250	209241.25	[34]	209699.36	0.219	171.8	209326.83	0.041	407.5
300	187712.12	—	188168.32	0.243	184.4	187763.64	0.027	395.8
350	170973.34	[34]	171443.87	0.275	200.0	171048.03	0.044	412.0
400	157030.46	[34]	157414.79	0.245	203.4	157073.20	0.027	436.3
450	145384.18	—	145694.26	0.213	216.3	145419.81	0.025	462.3
500	135467.85	[34]	135797.08	0.243	231.1	135507.73	0.029	478.5
550	126863.30	—	127207.83	0.272	243.8	126889.89	0.021	514.0
600	119107.99	[15]	119428.60	0.269	258.3	119135.62	0.023	595.8
650	112063.73	—	112456.15	0.350	271.0	112074.74	0.010	619.0
700	105854.40	—	106248.00	0.372	284.0	105889.22	0.033	637.3
750	100362.55	[15]	100713.79	0.350	296.4	100391.53	0.029	649.3
800	95411.78	—	95723.00	0.326	286.6	95432.66	0.022	677.8
850	91003.62	—	91268.56	0.291	296.1	91033.10	0.032	689.3
900	86984.10	—	87259.78	0.317	306.4	87022.59	0.044	730.4
950	83278.78	—	83509.58	0.277	314.3	83299.22	0.025	780.5
1000	79858.79	—	80018.33	0.200	321.7	79869.98	0.014	806.2

Table 8: Final results for instance rl5934, an Euclidean instance from class TSP with 5934 nodes.

	BEST KNOWN		SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
p	VALUE	SOURCE	VALUE	%ERR	TIME	VALUE	%ERR	TIME
10	9794951.00	[15]	9794973.65	0.000	5971.1	9794973.65	0.000	8687.1
20	6718848.19	—	6719116.39	0.004	3296.8	6719026.03	0.003	4779.6
30	5374936.14	—	5379979.09	0.094	2049.8	5376040.45	0.021	4515.1
40	4550364.60	—	4550843.75	0.011	1470.4	4550518.95	0.003	2499.3
50	4032379.97	—	4033758.13	0.034	1195.3	4032675.94	0.007	2280.6
60	3642397.88	—	3646198.03	0.104	996.1	3642949.30	0.015	2244.0
70	3343712.45	—	3348834.92	0.153	872.5	3344888.24	0.035	2138.3
80	3094824.49	—	3099917.93	0.165	778.8	3095442.55	0.020	1792.4
90	2893362.39	—	2898721.66	0.185	708.8	2894954.78	0.055	1844.2
100	2725180.81	—	2730313.90	0.188	671.2	2725580.72	0.015	1892.6
150	2147881.53	—	2151985.53	0.191	560.2	2148749.47	0.040	1209.2
200	1808179.07	—	1812249.63	0.225	526.6	1808658.73	0.027	1253.0
250	1569941.34	—	1573800.83	0.246	526.2	1570445.77	0.032	1203.8
300	1394115.39	—	1397064.23	0.212	550.1	1394361.41	0.018	1042.7
350	1256844.04	—	1259733.85	0.230	575.6	1257098.17	0.020	1246.4
400	1145669.38	[15]	1148386.49	0.237	583.8	1145961.13	0.025	1157.6
450	1053363.64	—	1055756.67	0.227	619.2	1053729.79	0.035	1236.9
500	973995.18	—	975940.78	0.200	641.7	974242.08	0.025	1236.7
600	848283.85	—	849765.46	0.175	703.7	848499.21	0.025	1439.4
700	752068.38	[15]	753522.21	0.193	767.3	752263.82	0.026	1566.6
800	676795.78	—	678300.99	0.222	782.1	676956.64	0.024	1574.9
900	613367.44	[15]	614506.49	0.186	834.5	613498.64	0.021	1722.0
1000	558802.38	[15]	559797.83	0.178	877.7	558943.93	0.025	1705.3
1100	511813.19	[15]	512793.56	0.192	931.4	511928.86	0.023	1893.4
1200	470295.38	[15]	471486.76	0.253	988.1	470411.12	0.025	2082.0
1300	433597.44	[15]	434688.75	0.252	1033.4	433678.02	0.019	2147.8
1400	401853.00	[15]	402796.80	0.235	1072.4	401934.24	0.020	2288.7
1500	374014.57	—	374803.24	0.211	1029.7	374056.40	0.011	2230.3

Table 9: Final results obtained for class ORLIB, graph-based instances introduced by Beasley [2].

INSTANCE				SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
NAME	n	p	OPT	VALUE	%ERR	TIME	VALUE	%ERR	TIME
pmed01	100	5	5819	5819	0.000	0.5	5819	0.000	0.5
pmed02	100	10	4093	4093	0.000	0.4	4093	0.000	0.5
pmed03	100	10	4250	4250	0.000	0.4	4250	0.000	0.5
pmed04	100	20	3034	3034	0.000	0.4	3034	0.000	0.5
pmed05	100	33	1355	1355	0.000	0.4	1355	0.000	0.5
pmed06	200	5	7824	7824	0.000	1.8	7824	0.000	1.8
pmed07	200	10	5631	5631	0.000	1.4	5631	0.000	1.4
pmed08	200	20	4445	4445	0.000	1.2	4445	0.000	1.2
pmed09	200	40	2734	2734	0.000	1.2	2734	0.000	1.5
pmed10	200	67	1255	1255	0.000	1.3	1255	0.000	1.6
pmed11	300	5	7696	7696	0.000	3.5	7696	0.000	3.5
pmed12	300	10	6634	6634	0.000	2.9	6634	0.000	2.9
pmed13	300	30	4374	4374	0.000	2.4	4374	0.000	2.5
pmed14	300	60	2968	2968	0.000	2.9	2968	0.000	3.5
pmed15	300	100	1729	1729	0.000	3.3	1729	0.000	4.3
pmed16	400	5	8162	8162	0.000	8.1	8162	0.000	8.2
pmed17	400	10	6999	6999	0.000	6.1	6999	0.000	6.3
pmed18	400	40	4809	4809	0.000	5.5	4809	0.000	6.7
pmed19	400	80	2845	2845	0.000	6.3	2845	0.000	7.5
pmed20	400	133	1789	1789	0.000	7.1	1789	0.000	8.6
pmed21	500	5	9138	9138	0.000	12.2	9138	0.000	12.2
pmed22	500	10	8579	8579	0.000	10.7	8579	0.000	11.3
pmed23	500	50	4619	4619	0.000	9.4	4619	0.000	11.0
pmed24	500	100	2961	2961	0.000	11.4	2961	0.000	13.1
pmed25	500	167	1828	1828	0.000	13.4	1828	0.000	16.2
pmed26	600	5	9917	9917	0.000	20.5	9917	0.000	20.5
pmed27	600	10	8307	8307	0.000	16.4	8307	0.000	16.4
pmed28	600	60	4498	4498	0.000	14.6	4498	0.000	17.4
pmed29	600	120	3033	3033	0.000	18.0	3033	0.000	21.0
pmed30	600	200	1989	1989	0.000	21.1	1989	0.000	26.9
pmed31	700	5	10086	10086	0.000	28.8	10086	0.000	28.8
pmed32	700	10	9297	9297	0.000	22.8	9297	0.000	22.9
pmed33	700	70	4700	4700	0.000	20.6	4700	0.000	23.7
pmed34	700	140	3013	3013	0.000	25.8	3013	0.000	30.8
pmed35	800	5	10400	10400	0.000	36.7	10400	0.000	36.7
pmed36	800	10	9934	9934	0.000	31.7	9934	0.000	34.4
pmed37	800	80	5057	5057	0.000	28.8	5057	0.000	32.4
pmed38	900	5	11060	11060	0.000	52.9	11060	0.000	52.9
pmed39	900	10	9423	9423	0.000	36.5	9423	0.000	36.5
pmed40	900	90	5128	5129	0.020	36.6	5128	0.000	43.4

Table 10: Final results for class SL, graph-based instances introduced by Senne and Lorena [32].

INSTANCE				SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
NAME	n	p	OPT	VALUE	%ERR	TIME	VALUE	%ERR	TIME
sl700	700	233	1847	1848	0.054	30.2	1847	0.000	39.5
sl800	800	267	2026	2027	0.049	41.8	2026	0.000	53.2
sl900	900	300	2106	2107	0.047	54.1	2106	0.000	68.2

Table 11: Final results for class GR, graph-based instances introduced by Galvão and ReVelle [9].

INSTANCE			SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
NAME	p	OPT	VALUE	%ERR	TIME	VALUE	%ERR	TIME
gr100	5	5703	5703	0.000	0.5	5703	0.000	0.5
	10	4426	4426	0.000	0.6	4426	0.000	1.0
	15	3893	3893	0.000	0.5	3893	0.000	0.8
	20	3565	3565	0.000	0.4	3565	0.000	0.7
	25	3291	3291	0.000	0.4	3291	0.000	0.7
	30	3032	3032	0.000	0.4	3032	0.000	0.6
	40	2542	2542	0.000	0.4	2542	0.000	0.6
	50	2083	2083	0.000	0.4	2083	0.000	0.6
gr150	5	10839	10839	0.000	1.3	10839	0.000	1.3
	10	8729	8729	0.000	1.1	8729	0.000	2.0
	15	7390	7390	0.000	1.0	7390	0.000	1.7
	20	6454	6462	0.124	0.9	6462	0.124	1.5
	25	5875	5887	0.204	0.9	5875	0.000	1.7
	30	5495	5502	0.127	0.8	5495	0.000	1.5
	40	4907	4907	0.000	0.8	4907	0.000	1.2
	50	4374	4375	0.023	0.8	4375	0.023	1.2

Table 12: Final results for class RW, random instances introduced in [24].

INSTANCE			SINGLE-STAGE GRASP			WITH POST-OPTIMIZATION		
NAME	p	BEST	VALUE	%ERR	TIME	VALUE	%ERR	TIME
rw100	10	530	530	0.000	0.7	530	0.000	1.3
	20	277	277	0.000	0.5	277	0.000	0.7
	30	213	213	0.000	0.4	213	0.000	0.5
	40	187	187	0.000	0.3	187	0.000	0.5
	50	172	172	0.000	0.3	172	0.000	0.4
rw250	10	3691	3691	0.000	6.1	3691	0.000	10.4
	25	1364	1370	0.440	3.3	1364	0.000	5.8
	50	713	718	0.701	2.1	713	0.000	3.9
	75	523	523	0.000	1.9	523	0.000	2.6
	100	444	444	0.000	1.8	444	0.000	2.2
	125	411	411	0.000	1.5	411	0.000	2.0
rw500	10	16108	16259	0.937	33.1	16108	0.000	76.9
	25	5681	5749	1.197	20.8	5683	0.035	46.9
	50	2628	2657	1.104	14.1	2635	0.266	27.7
	75	1757	1767	0.569	11.6	1757	0.000	20.5
	100	1380	1388	0.580	11.5	1382	0.145	20.4
	150	1024	1026	0.195	11.1	1024	0.000	15.4
	200	893	893	0.000	11.8	893	0.000	14.4
	250	833	833	0.000	9.6	833	0.000	11.6
rw1000	10	67811	68202	0.577	153.6	68136	0.479	256.3
	25	24896	25192	1.189	111.1	24964	0.273	293.5
	50	11306	11486	1.592	77.7	11360	0.478	169.1
	75	7161	7302	1.969	60.2	7207	0.642	160.1
	100	5223	5297	1.417	55.5	5259	0.689	109.8
	200	2706	2727	0.776	57.5	2710	0.148	100.4
	300	2018	2021	0.149	55.2	2018	0.000	71.5
	400	1734	1734	0.000	61.8	1734	0.000	73.5
	500	1614	1614	0.000	47.9	1614	0.000	55.9

absolute terms — for instances with well-defined metrics (graphs and Euclidean instances), than on random instances (such class RW). This is hardly surprising, it just reflects the behavior of the basic building blocks of our method, the local search procedure and the path-relinking method.

Other Methods. We now analyze our methods in relative terms: how it behaves in comparison with other methods in the literature. For simplicity, we refer to our own method as GRASP, even though the results reported here include path-relinking and the post-optimization phase. The results taken into account in the comparison are those presented in the three last columns of Tables 6 and 12.

Other methods referred to in the comparison are:

- VNS: Variable Neighborhood Search, by Hansen and Mladenović [14]; results for this method are available for the ORLIB series (all 40 instances were tested, with running times given for only 22 of them), for fl1400 (all 18 values of p), and pcb3038 (with only 10 values of p: $50, 100, 150, \ldots, 500$). The values used here were computed from those reported in Tables 1, 2, and 3 of [14].
- VNDS: Variable Neighborhood Decomposition Search, by Hansen, Mladenović, and Perez-Brito [15]. Results are available for all ORLIB and TSP instances.[4]
- LOPT: Local Optimization method, proposed by Taillard in [34]. The method works by heuristically solving locally defined subproblems in integrating them into a solution to the general problem. The author provides detailed results (in Table 7 of [34]) only for instance pcb3038, with nine values of p, all multiples of 50 between 100 to 500.
- DEC: Decomposition Procedure, also studied by Taillard in [34] and based on the decomposition of the original problem. Results are provided by the same nine instances as LOPT (in Table 7 of [34]).
- LSH: Lagrangean-Surrogate Heuristic, described by Senne and Lorena in [32]. The paper contains results for six ORLIB instances (pmed05, pmed10, pmed15, pmed20, pmed25, pmed30), for nine values of p for pcb3038 (the same nine used to test LOPT), and for all instances in classes SL and GR. The values used in our comparison were taken from Tables 1, 2, and 3 of [32].
- CGLS: Column Generation with Lagrangean/Surrogate Relaxation, studied by Senne and Lorena in [33]. Results are available for 15 ORLIB instances (pmed01, pmed05, pmed06, pmed07, pmed10, pmed11, pmed12, pmed13, pmed15, pmed16, pmed17, pmed18, pmed20, pmed25, and pmed30), for all three SL instances, and for five values of p in instance pcb3038 (300, 350, 400, 450, and 500). We consider here the results found by method *CG(t)*, taken from Tables 1, 2, and 4 of [33].

We start our comparison by presenting, for each of the methods studied, the average percentage deviation with respect to the best solutions known (as given by Tables 6 to 12 above). The results are presented in Table 13. Each of the instances of class TSP is shown separately to allow a more precise analysis of the algorithms. Some values are in *slanted font* as a reminder that not all instances in the set were considered in the paper describing the method. A dash (—) is shown when no result for the class is available. Class RW is not shown, since the only results available are those obtained by our method.

The table shows that our method is the only one within 0.04% of the best values known for all instances. Futhermore, even though our method did not obtain the best results for every single instance, it did obtain the best results on average for all six sets of instances presented in the table. It must be said, however, that the difference is often very small. Many of the methods are virtually as good as ours in one or another class: that is the case of VNDS for all three TSP instances; of VNS and LSH for ORLIB instances; and of CGLS for pcb3038. This reveals the greatest strength of our method: *robustness*. It was able to obtain competitive results for all classes of instances. None of the methods tested has shown such degree of consistency.

Of course, we also have to consider the running times of the methods involved. Finding good solutions would be hardly surprising if the running times of our method were significantly greater than those of others.

[4]The authors of [15] also tested instances from [26]; unfortunately, we were unable to obtain these instances at the time of writing.

Table 13: Mean percentage deviations for each method with respect to the best solution known. Values in *slanted font* indicate that not all instances in the set were tested by the method. Smaller values are better.

series	GRASP	CGLS	DEC	LOPT	LSH	VNDS	VNS
GR	0.009	—	—	—	0.727	—	—
SL	0.000	0.691	—	—	0.332	—	—
ORLIB	0.000	*0.101*	—	—	*0.000*	0.116	0.007
fl1400	0.031	—	—	—	—	0.071	0.191
pcb3038	0.025	*0.043*	*4.120*	*0.712*	*2.316*	0.117	*0.354*
rl5934	0.022	—	—	—	—	0.142	—

To present a meaningful comparison between the methods, we adopted the following strategy. For each instance in which a method was tested, we computed the ratio between the time it required and the running time of our method. In Table 14, we present the *geometric means* of these ratios taken over the instances in each set (once again, only instances tested by the relevant method are considered). We believe this makes more sense than the usual arithmetic mean in this case: if a method is twice as fast as another for 50% of the instances and half as fast for the other 50%, intuitively the methods should be considered equivalent. The geometric mean reflects that, whereas the arithmetic mean does not.

Table 14: Mean ratios between the running times obtained by methods in the literature and those obtained by ours. Although values greater than 1.0 indicate that our method is faster on average, differences within the same order of magnitude should be disregarded. Values in *slanted font* indicate that there are instances in the set for which times are not available.

series	GRASP	CGLS	DEC	LOPT	LSH	VNDS	VNS
GR	1.00	—	—	—	1.11	—	—
SL	1.00	0.51	—	—	24.20	—	—
ORLIB	1.00	*55.98*	—	—	*4.13*	0.46	*5.47*
fl1400	1.00	—	—	—	—	0.58	19.01
pcb3038	1.00	*9.55*	*0.21*	*0.35*	*1.67*	2.60	*30.94*
rl5934	1.00	—	—	—	—	2.93	—

One important observation regarding the values presented should be made: for VNS and VNDS, the times taken into consideration are times in which the best solution was found (as in the papers that describe these methods [14, 15]); for all other algorithms (including ours), the *total* running time is considered. The values reported for our algorithm also include the time necessary to precompute all pairwise vertex distances in graph-based classes (ORLIB and SL).[5]

Values greater than one in the table favor our method, whereas values smaller than one favor others. One should not take these results too literally, however. Since results were obtained on different machines (see Table 15), small differences in running time should not be used to draw any conclusion regarding the relative effectiveness of the algorithms. Running times within the same order of magnitude should be regarded as indistinguishable.

7 Concluding Remarks

In this paper, we presented a GRASP with path-relinking for the p-median problem. We have shown that it is remarkably robust, handling a wide variety of instances and obtaining results competitive with those

[5]GR is also a graph-based class, but the instances we obtained, kindly provided by E. Senne, were already represented as distance matrices.

Table 15: Machines in which times used in Table 14 were obtained.

METHOD	MACHINE
CGLS	Sun Ultra 30
DEC	SGI Workstation (195 MHz MIPS R10000)
GRASP	SGI Challenge (196 MHz MIPS R10000)
LOPT	SGI Workstation (195 MHz MIPS R10000)
LS	Sun Ultra 30
VNDS	Sun Ultra I (143 MHz UltraSparc)
VNS	Sun SparcStation 10

obtained by the best heuristics in the literature in each case. This makes our heuristic a valuable candidate for a general-purpose solver for the p-median problem.

We do not claim, however, that our method is the best in every circumstance. Other methods described in the literature are able to produce results of remarkably good quality, often at the expense of a somewhat higher running times. VNS [14] is especially successful for graph instances; VNDS [15] is particularly strong for Euclidean instances, and is often faster than our method (especially when the number of facilities to open is very small); and CGLS [33] can obtain very good results for Euclidean instances, and has the additional advantage of providing good lower bounds. An interesting research topic would be to use some of the ideas explored in this paper (such as a fast implementation of the local search procedure and the combination of elite solutions through path-relinking) with those methods to obtain even better results.

The goal of our algorithm is to produce close-to-optimal solutions. Therefore, it should be said that our method does not handle well really large instances. If the input is a graph with millions of vertices, simply computing all-pairs shortest paths would be prohibitively slow. For that purpose, one would probably be better of relying on methods based on sampling techniques like the one proposed by Thorup [36]. Methods like this aim to find solutions that are "good", not near-optimal, in a reasonable (quasi-linear) amount of time.

However, if one is interested in solving instances large enough to preclude the application of exact algorithms, but not so large so as to make anything worse than quasi-linear prohibitive, our method has proven to be a very useful alternative.

References

[1] R. M. Aiex, M. G. C. Resende, P. M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. Technical Report TD-4RR83X, AT&T Labs Research, 2000.

[2] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[3] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.

[4] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[5] 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.

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

[7] P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer, 2002.

[8] R. D. Galvão. A dual-bounded algorithm for the p-median problem. *Operations Research*, 28:1112–1121, 1980.

[9] R. D. Galvão and C. S. ReVelle. A Lagrangean heuristic for the maximal covering problem. *European Journal of Operational Research*, 18:114–123, 1996.

[10] F. García-López, B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, 8(3):375–388, 2002.

[11] F. Glover. Tabu search and adaptive memory programing: Advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Scinece and Operations Research*, pages 1–75. Kluwer, 1996.

[12] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[13] M. F. Goodchild and V. Noronha. Location-allocation for small computers. Monograph 8, Department of Geography, University of Iowa, 1983.

[14] P. Hansen and N. Mladenović. Variable neighborhood search for the p-median. *Location Science*, 5:207–226, 1997.

[15] P. Hansen, N. Mladenović, and D. Perez-Brito. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(3):335–350, 2001.

[16] M. J. Hodgson. Toward more realistic allocation in location-allocation models: An interaction approach. *Environment and Planning A*, 10:1273–85, 1978.

[17] A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

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

[19] F. E. Maranzana. On the location of supply points to minimize transportation costs. *Operations Research Quarterly*, 15(3):261–270, 1964.

[20] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[21] R. M. Nauss and R. E. Markland. Theory and application of an optimizing procedure for lock box location analysis. *Management Science*, 27:855–865, 1981.

[22] G. Reinelt. TSPLIB: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991. *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.*

[23] M. G. C. Resende and C. C. Ribeiro. Greedy randomized adpative search procedures. In F. Glover and G. Kochenberger, editors, *State of the Art Handbook in Metaheuristics*. Kluwer, 2002. To appear.

[24] M. G. C. Resende and R. Werneck. On the implementation of a swap-based local search procedure for the p-median problem. Technical Report TD-5E4QKA, AT&T Labs Research, 2002.

[25] C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002.

[26] E. Rolland, D. A. Schilling, and J. R. Current. An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research*, 96:329–342, 1996.

[27] K. E. Rosing. An empirical investigation of the effectiveness of a vertex substitution heuristic. *Environment and Planning B*, 24:59–67, 1997.

[28] K. E. Rosing and C. S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, 97:75–86, 1997.

[29] K. E. Rosing, C. S. ReVelle, E. Rolland, D. A. Schilling, and J. R. Current. Heuristic concentration and tabu search: A head to head comparison. *European Journal of Operational Research*, 104:93–99, 1998.

[30] K. E. Rosing, C. S. ReVelle, and H. Rosing-Vogelaar. The p-median and its linear programming relaxation: An approach to large problems. *Journal of the Operational Research Society*, 30(9):815–823, 1979.

[31] E. Senne, 2002. Personal communication.

[32] E. L. F. Senne and L. A. N. Lorena. Langrangean/surrogate heuristics for p-median problems. In M. Laguna and J. L. Gonzalez-Velard, editors, *Computing Tools for Modeling Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 115–130. Kluwer, 2000.

[33] E. L. F. Senne and L. A. N. Lorena. Stabilizing column generation using Lagrangean/surrogate relaxation: an application to p-median location problems. *European Journal of Operational Research*, 2002. To appear.

[34] E. D. Taillard. Heuristc methods for large centroid clustering problems. Technical Report 96-96, IDSIA, Lugano, Switzerland, 1998. Revised from an original version published in 1996.

[35] M. B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16(5):955–961, 1968.

[36] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*, volume 2076 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2001.

[37] S. Voss. A reverse elimination approach for the p-median problem. *Studies in Locational Analysis*, 8:49–58, 1996.

[38] R. Whitaker. A fast algorithm for the greedy interchange of large-scale clustering and median location prolems. *INFOR*, 21:95–108, 1983.