# COMPUTING APPROXIMATE SOLUTIONS OF THE MAXIMUM COVERING PROBLEM WITH GRASP

MAURICIO G.C. RESENDE

ABSTRACT. We consider the maximum covering problem, a combinatorial optimization problem that arises in many facility location problems. In this problem, a potential facility site covers a set of demand points. With each demand point, we associate a nonnegative weight. The task is to select a subset of $p > 0$ sites from the set of potential facility sites, such that the sum of weights of the covered demand points is maximized. We describe a greedy randomized adaptive search procedure (GRASP) for the maximum covering problem that finds good, though not necessarily optimum, placement configurations. We describe a well-known upper bound on the maximum coverage which can be computed by solving a linear program and show that on large instances, the GRASP can produce facility placements that are nearly optimal.

## 1. INTRODUCTION

We consider the maximum covering problem (MCP) [11], a combinatorial optimization problem that has been applied to numerous facility location problems, including rural health centers [1], emergency vehicles [6], and commercial bank branches [12], as well as other applications [2, 4, 5].

The maximum covering problem can be stated as: Let $J = \{1,2,\ldots,n\}$ denote the set of $n$ potential facility locations. Define $n$ finite sets $P_1, P_2, \ldots, P_n$, each corresponding to a potential facility location, such that $I = \cup_{j\in J} P_j = \{1,2,\ldots,m\}$ is the set of the $m$ demand points that can be covered by the $n$ potential facilities. With each demand point $i \in I$, we associate a weight $w_i \geq 0$. A *cover* $J^* \subseteq J$ covers the demand points in set $I^* = \cup_{j\in J^*} P_j$ and has an associated weight $w(J^*) = \sum_{i\in I^*} w_i$. Given the number $p > 0$ of facilities to be placed, we wish to find the set $J^* \subseteq J$ that maximizes $w(J^*)$, subject to the constraint that $|J^*| = p$.

The MCP has an compact integer programming formulation, first described by Church and ReVelle [3]. For $i = 1,\ldots,m$ and $j = 1,\ldots,n$, let $x_j$ and $y_i$ be $(0,1)$ variables such that

$$x_j = \begin{cases} 1 & \text{if } j \in J^* \\ 0 & \text{otherwise} \end{cases}$$

and

$$y_i = \begin{cases} 1 & \text{if } i \in I^* \\ 0 & \text{otherwise.} \end{cases}$$

Define the constraint coefficient

$$a_{ij} = \begin{cases} 1 & \text{if } i \in P_j \\ 0 & \text{otherwise.} \end{cases}$$

*Key words and phrases.* Maximum covering problem, facility location, heuristic, GRASP, linear programming bound.

The following is an integer programming formulation for the maximum covering problem:

$$\max \sum_{i=1}^{m} w_i y_i$$

subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \geq y_i, \;\; i = 1, \ldots, m,$$

$$\sum_{j=1}^{n} x_j = p,$$

$$x_j = (0,1), \;\; j = 1, \ldots, n$$

$$0 \leq y_i \leq 1, \;\; i = 1, \ldots, m,$$

where we observe that the integrality of the $y_i$ variables can be relaxed.

The solution to the linear programming relaxation of the above integer program produces as its optimal objective function value, an upper bound on the maximum coverage. We shall call this bound, the LP upper bound, denoted by

$$\text{UB} = \max\{w^\top y \mid Ax \geq y,\ e^\top x = p,\ 0 \leq x \leq 1,\ 0 \leq y \leq 1\},$$

where $w = (w_1, w_2, \ldots, w_m)$, $y = (y_1, y_2, \ldots, y_m)$, $A = [a_{\cdot 1}, a_{\cdot 2}, \ldots, a_{\cdot n}]$, $x = (x_1, x_2, \ldots, x_n)$, and $e = (1, 1, \ldots, 1)$ of dimension $n$.

We describe a greedy randomized adaptive search procedure (GRASP) for the MCP that finds approximate, i.e. good though not necessarily optimum, facility placement configurations. GRASP [7] is a metaheuristic that has been applied to a wide range of combinatorial optimization problems, including set covering [8], maximum satisfiability [10], and $p$-hub location [9], all three of which have some similarities with the MCP. GRASP is an iterative process, with a feasible solution constructed at each independent GRASP iteration. Each GRASP iteration consists of two phases, a construction phase and a local search phase. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

Since the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions, it is usually beneficial to apply a local search to attempt to improve each constructed solution. While local optimization can require exponential time from an arbitrary starting point, empirically its efficiency significantly improves as the initial solutions improve. The result is that many GRASP solutions can be generated in the same amount of time required for the local optimization procedure to converge from a single random start. In addition, the best of these GRASP solutions is generally better than the solution obtained from a random starting point.

```
procedure grasp(α,MaxIter,RandomSeed)
1    BestSolutionFound = ∅;
2    do k = 1,...,MaxIter →
3        ConstructGreedyRandomizedSoln(α,RandomSeed,p,J*);
4        LocalSearch(J*);
5        if w(J*) > w(BestSolutionFound) → BestSolutionFound = J*;
6    od;
7    return(BestSolutionFound)
end grasp;
```

FIGURE 1. A generic GRASP pseudo-code

```
procedure ConstructGreedyRandomizedSoln(α,RandomSeed,p,J*)
1    J* = ∅;
2    do k = 1,..., p →
3        RCL = MakeRCL(α, J, J*, γ);
4        s = SelectFacility(RCL,RandomSeed,J*);
5        J* = J* ∪ {s};
6        AdaptGreedyFunction(s, J, J*, Γ, Γ^-1, γ);
7    od;
end ConstructGreedyRandomizedSoln;
```

FIGURE 2. GRASP construction phase pseudo-code

The paper is organized as follows. In Section 2, we describe the GRASP. In Section 3, we show how the GRASP solution is better than the pure random or pure greedy alternatives. On a large instance arising from a real-world application, we show how the GRASP solution is near optimal. Parallelization of GRASP is also illustrated. Concluding remarks are made in Section 4.

## 2. GRASP FOR MAXIMUM COVERING

As outlined in Section 1, a GRASP possesses four basic components: a greedy function, an adaptive search strategy, a probabilistic selection procedure, and a local search technique. These components are interlinked, forming an iterative method that, at each iteration, constructs a feasible solution, one element at a time, guided by an adaptive greedy function, and then searches the neighborhood of the constructed solution for a locally optimal solution. Figure 1 shows a GRASP in pseudo-code. The best solution found so far (`BestSolutionFound`) is initialized in line 1. The GRASP iterations are carried out in lines 2 through 6. Each GRASP iteration has a construction phase (line 3) and a local search phase (line 4). If necessary, the solution is updated in line 5. The GRASP returns the best solution found.

In the remainder of this section, we describe in detail the ingredients of the GRASP for the MCP, i.e. the GRASP construction and local search phases. To describe the construction phase, one needs to provide a candidate definition (for the restricted candidate list) and an adaptive greedy function, and specify the candidate restriction mechanism. For the local search phase, one must define the neighborhood and specify a local search algorithm.

```
procedure MakeRCL(α, J, J*, γ)
1    RCL = ∅;
2    γ* = max{γ_j | j ∈ J \ J*};
3    do s ∈ J \ J* →
4        if γ_s ≥ α × γ* →
5            RCL = RCL ∪ {s};
6        fi;
7    od;
8    return(RCL);
end MakeRCL;
```

FIGURE 3. `MakeRCL` pseudo-code

```
procedure AdaptGreedyFunction(s, J, J*, Γ, Γ^{-1}, γ)
1    do i ∈ Γ_s →
2        do j ∈ Γ_i^{-1} ∩ {J \ J*} (j ≠ i) →
3            Γ_j = Γ_j − {i};
4            γ_j = γ_j − w_i;
5        od;
6    od;
end AdaptGreedyFunction;
```

FIGURE 4. `AdaptGreedyFunction` pseudo-code

```
procedure LocalSearch(J^0, N(·), w(·), J*)
1    J* = J^0;
2    do ∃ J^+ ∈ N(J*) ∋ w(J^+) > w(J*) →
3        J* = J^+;
4    od;
end LocalSearch;
```

FIGURE 5. A generic local search algorithm

```
procedure LocalSearch(J*)
1    do local maximum not found →
2        do s ∈ J* →
3            do t ∈ J \ J* →
4                if WeightGain(J*, t) > WeightLoss(J*, s) →
5                    J* = J* ∪ {t} \ {s};
6                fi;
7            od;
8        od;
9    od;
end LocalSearch;
```

FIGURE 6. The local search procedure in pseudo-code

2.1. **Construction phase.** The construction phase of a GRASP builds a solution, around whose neighborhood a local search is carried out in the local phase, producing a locally optimal solution. This construction phase solution is built, one element at a time, guided by a greedy function and randomization. Figure 2 describes in pseudo-code a GRASP construction phase. Since in the MCP there are $p$ facility locations to be chosen, each construction phase consists of $p$ iterations, with one location chosen per iteration. In `MakeRCL` the restricted candidate list of facility locations is set up. The index of the next facility location to be chosen is determined in `SelectFacility`. The facility location selected is added to the set $J^*$ of chosen facility locations in line 5 of the pseudo-code. In `AdaptGreedyFunction` the greedy function that guides the construction phase is changed to reflect the choice just made. As before, let $J = \{1,2,\ldots,n\}$ be the set of indices of the sets of potential facility locations. Solutions are constructed by selecting one facility location at a time to be in the set $J^*$ of chosen facility locations. To define a restricted candidate list, we must rank the yet unchosen facility locations according to an adaptive greedy function.

The greedy function used in this algorithm is the total weight of yet-uncovered demand points that become covered after the selection in each construction phase iteration. Let $J^*$ denote the set (initially empty) of chosen facility locations being built in the construction phase. At any construction phase iteration, let $\Gamma_j$ be the set of additional uncovered demand points that would become covered if facility location $j$ (for $j \in J \setminus J^*$) were to be added to $J^*$. Define the *greedy function*

$$\gamma_j = \sum_{i \in \Gamma_j} w_i$$

to be the incremental weight covered by the choice of facility location $j \in J \setminus J^*$. The greedy choice is to select the facility location $k$ having the largest $\gamma_k$ value. Note that with every selection made, the sets $\Gamma_j$, for all yet unchosen facility location indices $j \in J \setminus J^*$, change to reflect the new selection. This consequently changes the values of the greedy function $\gamma_j$, characterizing the adaptive component of the heuristic.

We describe next the restriction mechanism for the restricted candidate list (RCL) used in this GRASP. The RCL is set up in `MakeRCL` of the pseudo-code of Figure 3. A value restriction mechanism is used. Value restriction imposes a parameter based *achievement level*, that a candidate has to satisfy to be included in the RCL. Let

$$\gamma^* = \max\{\gamma_j \mid \text{facility location } j \text{ is yet unselected, i.e. } j \in J \setminus J^*\}$$

and $\alpha$ be the restricted candidate parameter ($0 \leq \alpha \leq 1$). We say a facility location $j$ is a *potential candidate*, and is added to the RCL, if $\gamma_j \geq \alpha \times \gamma^*$. `MakeRCL` returns the set `RCL` with the indices of all potential facility locations that have greedy function values within $\alpha \times 100\%$ of the value of the greedy choice. Note that by varying the parameter $\alpha$ the heuristic can be made to construct a set of $p$ random facility locations ($\alpha = 0$) or act as a greedy algorithm ($\alpha = 1$).

Once the RCL is set up, a candidate from the list must be selected and made part of the solution being constructed. `SelectFacility` selects, at random, the facility location index $s$ from the RCL. In line 5 of `ConstructGreedyRandomizedSoln`, the choice made in `SelectFacility` is added to the set of facility locations $J^*$.

The greedy function $\gamma_j$ is changed in `AdaptGreedyFunction` to reflect the choice made in `SelectFacility`. This requires that some of the sets $\Gamma_j$ as well as the values $\gamma_j$ be updated. Let $\Gamma_i^{-1}$ denote the set of facility locations that cover demand point $i$. Let $s$ be the newly added facility location. The potential facility locations $j$ whose elements $\Gamma_j$ need to

be updated are those not yet in the facility location set $J^*$ for which demand points in $P_s$ are covered by facility location $j$.

2.2. **Local search phase.** Given a solution neighborhood structure $N(\cdot)$ and a weight function $w(\cdot)$, a local search algorithm takes an initial solution $J^0$ and seeks a locally optimal solution with respect to $N(\cdot)$. For a maximization problem, such as the MCP, a local optimum is a solution $J^*$ having weight $w(J^*)$ greater than or equal to the weight $w(J^+)$ for any $J^+ \in N(J^*)$. The local search algorithm examines a sequence of solutions $J^0, J^1, \ldots, J^k = J^*$, where $J^{i+1} \in N(J^i)$, i.e. immediately after examining solution $J^i$, it can only examine a solution $J^{i+1}$ that is a neighbor of $J^i$. Figure 5 illustrates a generic local search algorithm that finds a local maximum of the function $w(\cdot)$. If in line 2 there exists a solution $J^+$ in the neighborhood of the current solution $J^*$ with a weight greater than that of the current solution, then in line 3 the improved solution is made the current solution. The loop from line 2 to 4 is repeated until no local improvement is possible.

A combinatorial optimization problem can have many different neighborhood structures. For the MCP, a simple structure is 2-exchange. Two solutions (sets of facility locations) $J^1$ and $J^2$ are said to be neighbors in the 2-exchange neighborhood if they differ by exactly one element, i.e. $| J^1 \cap \Delta J | = | J^2 \cap \Delta J | = 1$, where $\Delta J = (J^1 \cup J^2) \setminus (J^1 \cap J^2)$. The local search starts with a set $J^*$ of $p$ facility locations, and at each iteration attempts to find a pair of locations $s \in J^*$ and $t \in J \setminus J^*$ such that $w(J^* \setminus \{s\} \cup \{t\}) > w(J^*)$. If such a pair exists, then location $s$ is replaced by location $t$ in $J^*$. A solution is locally optimal with respect to this neighborhood if there exists no pairwise exchange that increases the total weight of $J^*$. This local search algorithm is described in the pseudo-code in Figure 6. Though it is not the objective of this paper to delve into implementation details, it is interesting to observe that the total weight of the neighborhood solutions need not be computed from scratch. Rather, in line 4 of the pseudo-code, procedures `WeightGain` and `WeightLoss` compute, respectively, the weight gained by $J^*$ with the inclusion of facility location $j$ and the weight loss by $J^*$ with the removal of facility location $i$ from $J^*$. The weight gained can be computed by adding the weights of all demand points not covered by any facility location in $J^*$ that is covered by $j$, while the weight loss can be computed by adding up the weights of the demand points covered by facility location $i$ and no other facility location in $J^*$.

The GRASP construction phase described in Subsection 2.1 computes a feasible set of chosen facility locations that is not necessarily locally optimal with respect the 2-exchange neighborhood structure. Consequently, local search can be applied with the objective of finding a locally optimal solution that may be better than the constructed solution. In fact, the main purpose of the construction phase is to produce a good initial solution for the local search. It is empirically known that simple local search techniques perform better if they start with a good initial solution. This is illustrated in Section 3, where experiments indicate that local search applied to a solution generated by the construction phase, rather than random generation, produces better overall solutions, and GRASP converges faster to an approximate solution.

## 3. Computing large coverings with GRASP

In this section, we illustrate the use of GRASP on a large real facility location problem from the telecommunications industry as well as randomnly generated test problems. All runs were carried out on a Silicon Graphics Challenge computer (196MHz IPS R10000 processor) running the IRIX 6.2 operating system. The GRASP code is written in Fortran and was compiled with the SGI Fortran compiler `f77` using compiler flags `-O3 -r4 -64`.

```
procedure LocalSearch(indin,indout)
1    flag = 1;
2    while  flag == 1  →
3         flag = 0;
4         do i = 1,...,p  →
5              wloss = ComputeWeightLoss(indin(i));
6              do j = 1,...,n − p  →
7                   wgain = ComputeWeightGain(indout(j));
8                   if  wgain − wloss > 0  →
9                        tmp = indin(i);
10                       indin(i) = indout(j);
11                       indout(j) = tmp;
12                       flag = 1;
13                       goto 7;
14                  fi;
15             od;
16        od;
17   elihw;
end LocalSearch;
```

FIGURE 7. A local search algorithm used in experiments

Before we describe the experimental results, we must comment on the implementation of the local search, since the local search described in Section 2 can be implemented in many ways. For these experiments we have adopted the following strategy. Two arrays store the facility location indices: `indin` has the $p$ indices of the chosen facilties, and `indout` has the $n-p$ indices of the unchosen facilities. An array `ncov` is used to count the number of chosen facilities that cover a specific demand region, i.e. `ncov(i)` is the number of chosen facilities that cover demand region `i`. With `ncov` it is easy to compute weight loss and gain due to a facility swap. Figure 7 illustrates the local search used in the experiments described in this section.

The local search tries to improve the covering defined by arrays `indin` and `indout` by seeking a swap of a facility index in `indin` with one in `indout`. The search ends only when no swap can improve the solution. For each facility index `indin(i)` in the current solution, the weight loss (`wloss`) by removing `indin(i)` from the covering is computed. For each facility index `indout(j)` not in the covering, the weight gained (`wgain`) is computed and compared with `wloss`. If the gain is greater than the loss, the swap improves the solution. After a swap is made, the search continues from the same position in array `indin`. We exprimented with several other implementations of this local search. Some were more efficient in reaching a local minimum, but none produced local optimal solutions as good as those produced with this implementation.

3.1. **Large telecommunications facility location problem.** The telecommunications facility location problem has over 18 thousand demand regions and over 27 thousand locations to which locate facilities. The sum of the weights over all demand regions is over 27 million. We compare an implementation of the GRASP described in Section 2 with implementations of an algorithm having a purely greedy construction phase and one having

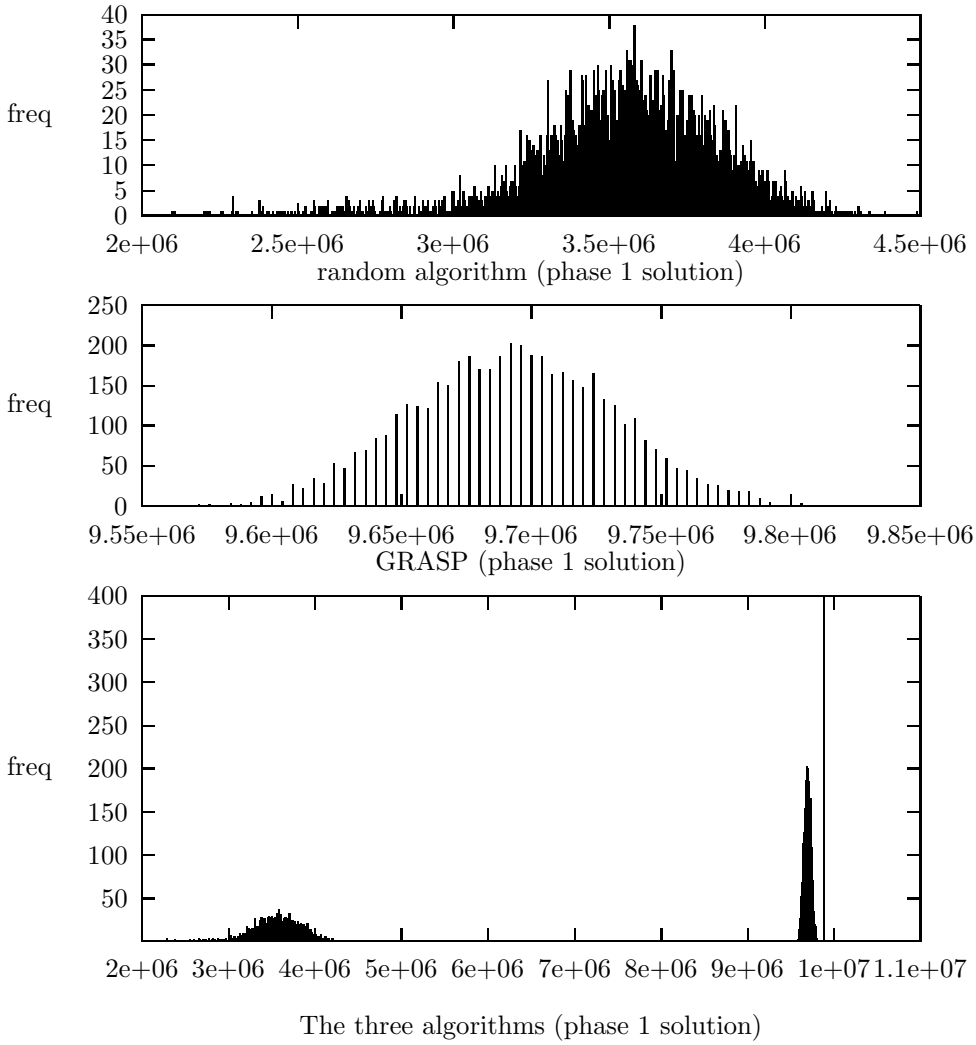


FIGURE 8. Phase 1 solution distribution for random algorithm (RCL parameter $\alpha = 0$), GRASP ($\alpha = 0.85$), and greedy algorithm ($\alpha = 1$)

purely random construction. All three algorithms use the same local search procedure, described in Section 2.2. Furthermore, since pure greedy and pure random are special cases of GRASP construction, all three algorithms are implemented using the same code, simply by setting the RCL parameter value $\alpha$ to appropriate values. For GRASP, $\alpha = 0.85$, while for the purely greedy algorithm, $\alpha = 1$, and for the purely random algorithm, $\alpha = 0$.

Two experiments are done. In the first, the number of facilities to be place is fixed at $p = 146$ and the three implementations are compared. Each code is run on 10 processors, each using a different random number generator seed for 500 iterations of the build–local search cycle, thus each totaling 5000 iterations. Because of the long processing times associated with the random algorithm, the random algorithm processes were interrupted before completing the full 500 iterations on each processor. They did 422, 419, 418, 420, 415, 420, 420, 412, 411, and 410 iterations on each corresponding processor, totaling

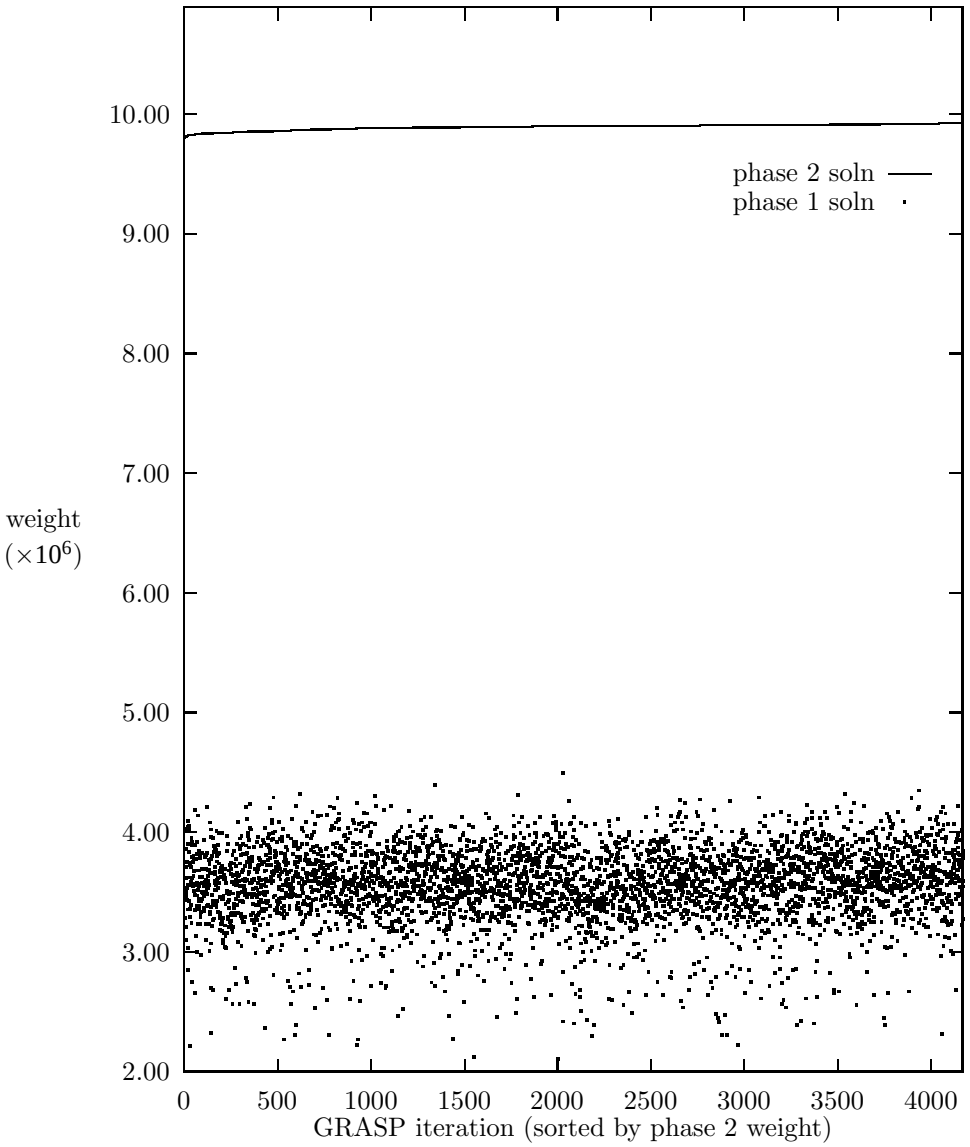


FIGURE 9. GRASP phase 1 and phase 2 solutions, sorted by phase 2, then phase 1 solutions. RCL parameter $\alpha = 0.0$ (purely random construction)

4167 iterations. In the second experiment, GRASP was run 300 times on facility location problems defined by varying $p$, the number of facilities, from 1 to 300 in increments of 1. Instead of running the algorithm for a fixed number of iterations, the LP upper bound was computed for each instance and the GRASP was run until it found a facility assignment within one percent of the LP upper bound.

Figure 8 illustrates the relative behavior of the three algorithms. The top and middle plots in Figure 8 show the frequency of the solution values generated by the purely random

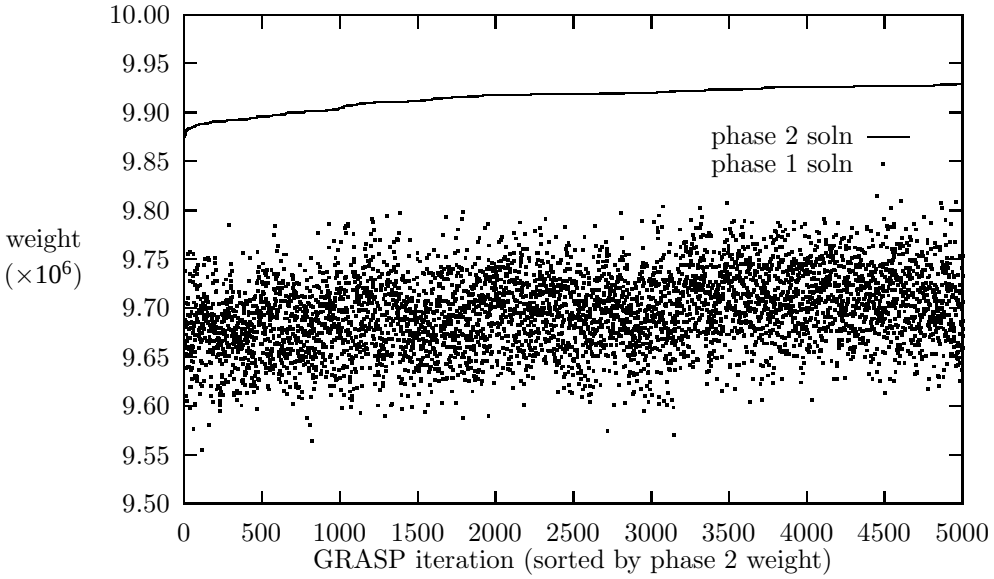


FIGURE 10. GRASP phase 1 and phase 2 solutions, sorted by phase 2, then phase 1 solutions. RCL parameter $\alpha = 0.85$

construction and GRASP construction respectively. The plot on the bottom of Figure 8 compares the constructed solutions of the three algorithms. As can be observed, the purely greedy algorithm constructs the best quality solution, followed by the GRASP, and then by the purely random algorithm. On the other hand, the purely random algorithm produces the largest amount of variance in the constructed solutions, followed by the GRASP and then the purely greedy algorithm, which generated the same solution on all 5000 repetitions. High quality solutions as well as large variances are desirable characteristics of constructed solutions. Of the three algorithms, GRASP captures these two characteristics in its phase 1 solutions. As we will see next, the tradeoff between solution quality and variance plays an important role in designing a GRASP.

The solutions generated by the purely random algorithm and the GRASP are shown in Figures 9 and 10, respectively. The solution values on these plots are sorted according to local search phase solution value. As one can see, the differences between the values of the construction phase solutions and the local search phase solutions are much smaller for the GRASP than for the purely random algorithm. This suggests that the purely random algorithm requires greater effort in the local search phase than does GRASP. This indeed is observed and will be shown next. Figures 11 and 12 illustrate how the three algorithms compare in terms of best solution found so far, as a function of the algorithm's iteration and running time. Figure 11 shows local search phase solution for each algorithm, sorted by increasing value for each algorithm. The solution produced by applying local search to the solution constructed with the purely greedy algorithm is constant. Its value is only better than the worst 849 GRASP solutions and the worst 2086 purely random solutions. This figure illustrates well the effect of the tradeoff between greediness and randomness in terms of solution quality as a function of the number of iterations.

Figures 13 and 14 correspond to the second experiment, where GRASP was run until a solution within one percent of the LP upper bound was produced. For all 300 problems, the

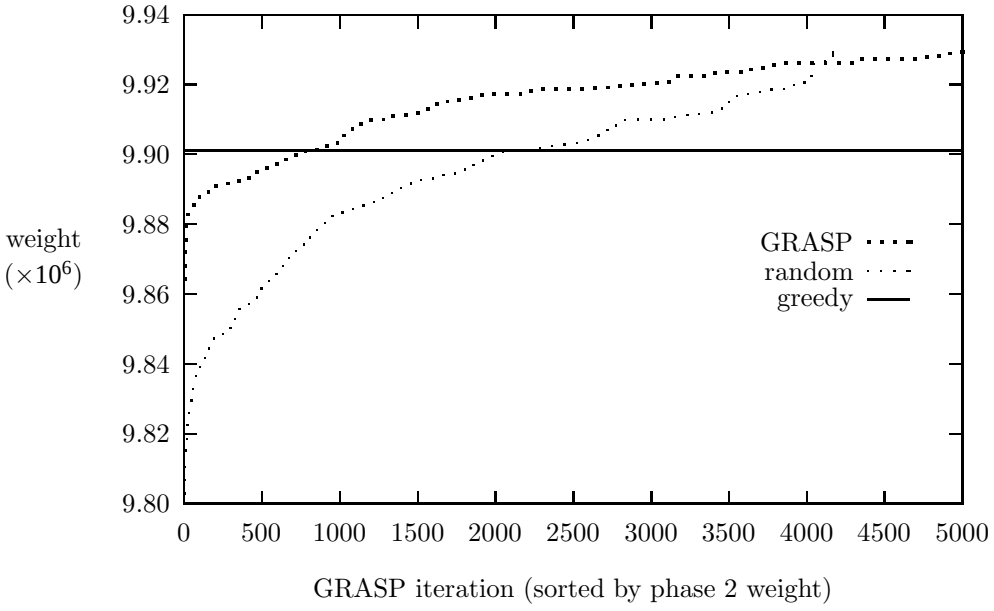


FIGURE 11. Phase 2 solutions, sorted by phase 2 for random, GRASP, and greedy algorithms

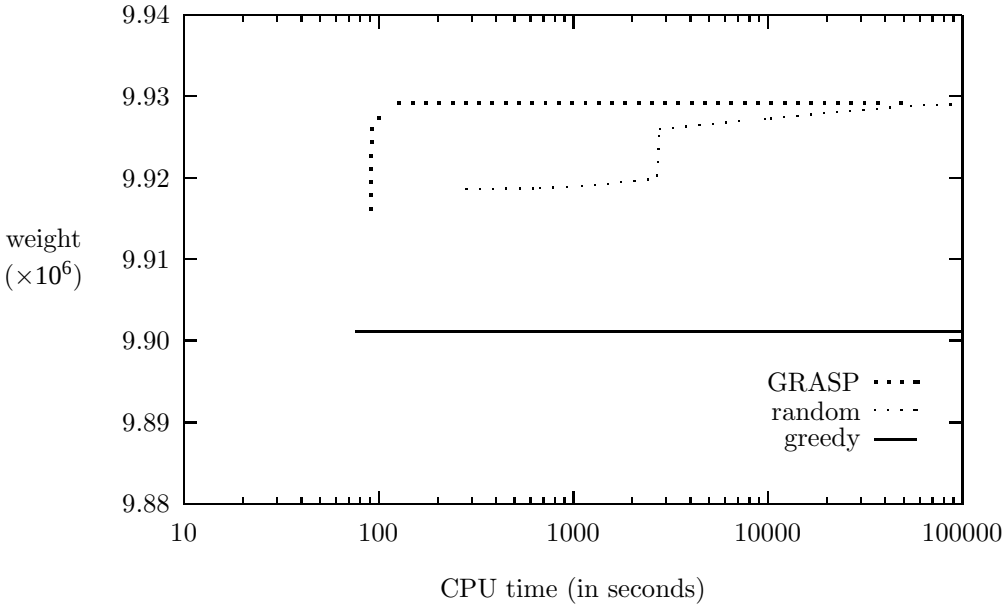


FIGURE 12. Incumbent phase 2 solution of random algorithm ($\alpha = 0$), GRASP ($\alpha = 0.85$), and greedy algorithm ($\alpha = 1$) as a function of CPU time (in seconds), running 10 processes in parallel.

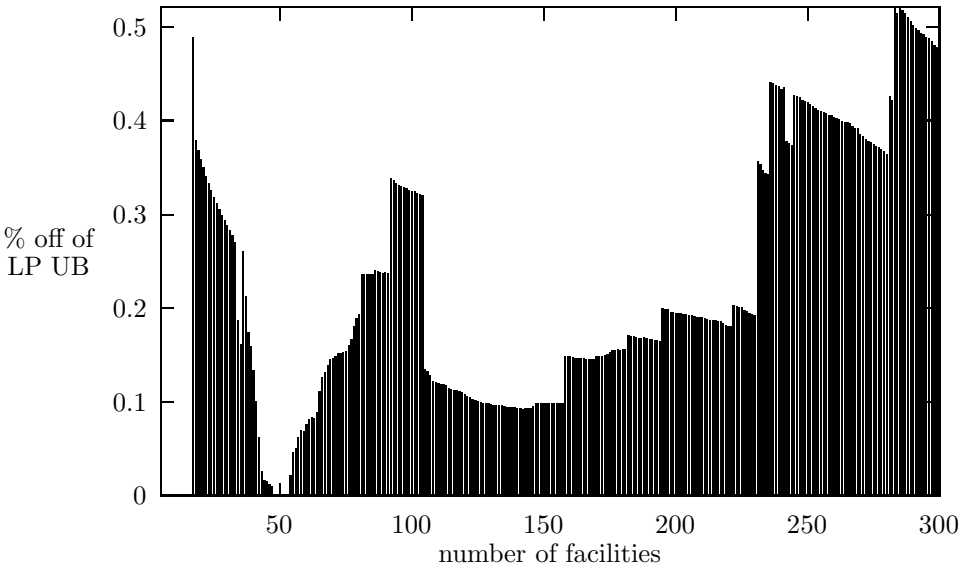


FIGURE 13. Percentage off of LP upper bound when stopping with a solution at most 1% off of bound

GRASP produced a design within 1% of the LP upper bound. Figure 13 shows the error of the GRASP solution as a percentage off of the LP upper bound when the algorithm was terminated. As can be observed, GRASP found tight solutions (GRASP solution equal to LP upper bound) for several instances and almost always produced a solution less than .5% off of the LP upper bound the first time it found a solution less than 1% of the upper bound. Figure 14 shows CPU times for each of the runs. CPU time grows with the complexity of the problem, as measured by the number of facilities in the assignment. For up to about 50 facilities (a number of facilties found in practical incremental designs) the GRASP solution takes less than 30 seconds on a 196MHz Silicon Graphics Challenge. The longest runs took a little less than 3 minutes to conclude.

3.2. **Randomly generated maximum covering problems.** We next focus on randomly generated test problems to evaluate the performance of GRASP on problems with different characteristics.

We first describe the problem generator. The dimension of the problem is determined by:

- $m$: the number of demand points,
- $n$: the number of potential facility locations.

We assume that facility locations are limited to demand point locations and hence require that $m \geq n$. Three other parameters are input to the generator:

- $w_{\min}$: smallest possible demand point weight,
- $w_{\max}$: largest possible demand point weight,
- $r_{\max}$: maximum distance between a facility location and any demand point covered by the facility location.

Demand point weights are distributed uniformly in the interval $[w_{\min}, w_{\max}]$ and demand points are located uniformly in the unit square. $n$ of the $m$ demand points are selected at

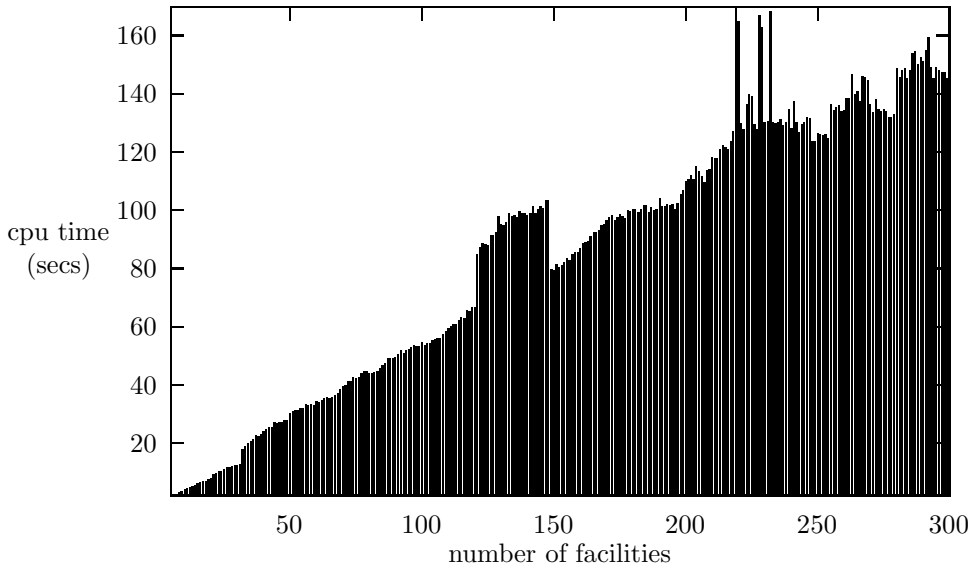


FIGURE 14. CPU time to stop when stopping with a solution at most 1% off of bound

TABLE 1. Randomly generated test problems

| prob | $r_{\max}$ | demand points | % density | sum of weights |
|---|---|---|---|---|
| r11 | 0.01 | 3381 | 0.04071 | 16819721 |
| r12 | | 3402 | 0.04113 | 17001325 |
| r13 | | 3411 | 0.04079 | 16961172 |
| r14 | | 3412 | 0.04105 | 16919527 |
| r15 | | 3443 | 0.04046 | 17250082 |
| r21 | 0.02 | 7401 | 0.13139 | 36668594 |
| r22 | | 7419 | 0.13662 | 37181210 |
| r23 | | 7420 | 0.13236 | 36937682 |
| r24 | | 7425 | 0.13459 | 37498327 |
| r25 | | 7455 | 0.13267 | 37434928 |
| r51 | 0.05 | 9980 | 0.75190 | 49775404 |
| r52 | | 9985 | 0.75879 | 49683624 |
| r53 | | 9991 | 0.76406 | 50478996 |
| r54 | | 9994 | 0.75665 | 49957445 |
| r55 | | 9996 | 0.75653 | 50314787 |

random to be potential facility locations. A potential facility location located at $(x_f, y_f)$ covers a demand point located at $(x_d, y_d)$ if they are within a distance $r_{\max}$ of each other, i.e. if $\sqrt{(x_f - x_d)^2 + (y_f - y_d)^2} \leq r_{\max}$.

The experiment was done on problems of dimension $n = 1000$ and $m = 10000$. Demand point weights were generated at random uniformly in the interval $[1, 10000]$. To allow for different levels of coverage, 3 levels of $r_{\max}$ were used: .01, .02, and .05. Five instances

TABLE 2. Summary of results on randomly generated test problems

| $r_{\max}$ | $p$ | avg itr to best | avg time to best | % proven optimal | max % error |
|---|---|---|---|---|---|
| 0.01 | 10 | 1.0 | 0.07 | 100 | |
| | 100 | 1.0 | 0.19 | 100 | |
| | 250 | 1.4 | 0.43 | 100 | |
| | 500 | 30.0 | 9.89 | 100 | |
| | 750 | 2.6 | 0.71 | 80 | 0.0007 |
| | 900 | 1.2 | 0.22 | 100 | |
| 0.02 | 10 | 1.0 | 0.21 | 100 | |
| | 100 | 2.6 | 1.54 | 100 | |
| | 250 | 246.2 | 314.14 | 20 | 0.0474 |
| | 500 | 150.8 | 261.26 | 0 | 0.1949 |
| | 750 | 135.6 | 137.98 | 0 | 0.0113 |
| | 900 | 1.0 | 0.35 | 100 | |
| 0.05 | 10 | 1.0 | 1.46 | 100 | |
| | 100 | 337.8 | 3670.61 | 0 | 1.6832 |
| | 250 | 1.0 | 5.50 | 100 | |
| | 500 | 1.0 | 3.83 | 100 | |
| | 750 | 1.0 | 3.00 | 100 | |
| | 900 | 1.0 | 1.76 | 100 | |

were generated for each problem combination $n, m$, and $r_{\max}$, totalling 15 instances. Each instance was solved for six levels of the number $p$ of facilities to be located: 10, 100, 250, 500, 750, and 900. Therefore, the total number of test problems considered is 90. Table 3.2 summarizes the problems used. For each problem, the table reports its name, the $r_{\max}$ value used to generate it, the number of demand points (of the total of 10,000) that can be covered by at least one potential facility site, the density (average percentage of demand points covered by a potential facility site), and the sum of weights over all demand points. These test problems as well as the generator are available from the author.

To solve each instance, we initially compute the linear programming upper bound and stop GRASP if a covering with the weight of the linear programming bound is found. A maximum of 500 GRASP iterations are done.

Table 3.2 summarizes the runs. We take averages over the 5 instances in each density class. All running times are given in CPU seconds. We make the following observations about the experiment:

- The LP bound is tight on the majority of the test problems. Of the 90 problems considered in the experiment, the LP bound was tight (i.e. equal to the optimal integer solution) in at least 70 cases.
- GRASP found provably optimal solutions in 70 of the 90 test problems.
- For the most sparse problems in the test set (those with $r_{\max} = .01$), in only one instance (`r15` with $p = 750$) did GRASP not find a provably optimal solution.
- For the most dense problems in the test set (those with $r_{\max} = .05$), GRASP found provably optimal solutions for all values of $p$, except $p = 100$.
- In each density class, varying the value of $p$ affects the difficulty of the problem for GRASP. In the class with $r_{\max} = .01$, the hardest instances were for $p = 500$ and p$= 750$. For the class with $r_{\max} = .02$, the difficult instances had $p = 250, 500,$ and

TABLE 3. Solutions found (on randomly generated test problems) not proven to be optimal.

| prob | $r_{\max}$ | $p$ | iters to best | time to best | total time | % error |
|---|---|---|---|---|---|---|
| r15 | 0.01 | 750 | 1 | 0.30 | 105.05 | 0.0007 |
| r22 | 0.02 | 250 | 286 | 373.85 | 641.03 | 0.0232 |
| r23 | | | 400 | 506.80 | 621.00 | 0.0372 |
| r24 | | | 44 | 55.40 | 625.94 | 0.0211 |
| r25 | | | 182 | 240.70 | 644.29 | 0.0475 |
| r21 | 0.02 | 500 | 1 | 1.82 | 796.45 | 0.1803 |
| r22 | | | 320 | 552.26 | 844.12 | 0.0656 |
| r23 | | | 114 | 216.60 | 942.19 | 0.1949 |
| r24 | | | 300 | 503.78 | 818.57 | 0.0864 |
| r25 | | | 19 | 31.85 | 828.48 | 0.1099 |
| r21 | 0.02 | 750 | 215 | 181.15 | 400.51 | 0.0044 |
| r22 | | | 12 | 14.27 | 566.54 | 0.0024 |
| r23 | | | 5 | 5.63 | 538.72 | 0.0047 |
| r24 | | | 123 | 139.00 | 536.26 | 0.0053 |
| r25 | | | 323 | 349.84 | 519.77 | 0.0113 |
| r51 | 0.05 | 100 | 343 | 3947.98 | 5675.26 | 1.0180 |
| r52 | | | 256 | 2762.40 | 5275.84 | 1.2225 |
| r53 | | | 418 | 4427.86 | 5269.05 | 1.6832 |
| r54 | | | 249 | 2614.62 | 5193.63 | 1.2620 |
| r55 | | | 423 | 4600.17 | 5380.23 | 1.3924 |

750. In the most dense class, with $r_{\max} = .05$, the hardest instances had $p = 100$. This indicates that as the problems become more dense, the more difficult problems are those with small values of $p$.

- The percentage error is defined as

$$\frac{(\text{LP upper bound} - \text{GRASP solution})}{\text{LP upper bound}} \times 100.$$

  Optimal solutions have 0% error. The average percentage error increased with problem density. Of the instances on which GRASP did not find an optimal solution, the percentage error never exceeded 1.68%. On the $r_{\max} = .02$ instances, the maximum percentage error was 0.19%, while for the $r_{\max} = .01$ instances it was .0007%.
- Table 3.2 lists all test problems for which GRASP did not find provably optimal solutions. Besides iterations and CPU time to find the best solution, the table lists the total time to run the 500 iterations, as well as the percentage error in the GRASP solution.
- It should be noted that the percentage error is actually a maximum percentage error, since the LP bound may not be tight and the GRASP solutions found may be closer to the optimal than to the LP bound.

## 4. CONCLUDING REMARKS

In this paper, we have studied the maximum covering problem.

A heuristic method for solving this problem was proposed. The method, a GRASP or greedy randomized adaptive search procedure, was described in detail.

The quality of the heuristic solutions can be measured with an upper bound produced by solving a linear programming problem. Computational results on both a real-world facility location problem, as well as on randomly generated test problems, indicate that the method finds solutions of very good quality. In fact, on the majority of the problems tested, the GRASP found solutions that could be verified to be optimal by the LP upper bound. On those that were not verifiable, the error never exceeded 2%.

## REFERENCES

[1] V. L. Bennett, D. J. Eaton, and R. L. Church. Selecting sites for rural health workers. *Social Science Medicine*, 16:63–72, 1982.

[2] C. H. Chung. Recent applications of the maximal covering location planning (M. C. L. P.) model. *Journal of the European Operational Research Society*, 37:735–746, 1986.

[3] R. Church and C. ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32:101–118, 1974.

[4] M. S. Daskin, P. C. Jones, and T. J. Lowe. Rationalizing tool selection in a flexble manufacturing system for sheet metal products. *Operations Research*, 38:1104–1115, 1990.

[5] F. P. Dwyer and J. R. Evans. A branch and bound algorithm for the list selection problem in direct mail advertising. *Management Science*, 27:658–667, 1981.

[6] D. J. Eaton, M. S. Daskin, D. Simmons, B. Bulloch, and G. Jansma. Determining medical service vehicle deployment in Austin, Texas. *Interfaces*, 15:96–108, 1985.

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

[8] 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.

[9] J. G. Klincewicz. Avoiding local optima in the *p*-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

[10] M. G. C. Resende, L. S. Pitsoulis, and P. M. Pardalos. Approximate solution of weighted MAX-SAT problems using GRASP. In D.-Z. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, R.I., 1996.

[11] D. Schilling, V. Jayaraman, and R. Barkhi. A review of covering problems in facility location. *Location Science*, 1:25–55, 1993.

[12] D. J. Sweeney, R. L. Mairose, and R. K. Martin. Strategic planning in bank location. In *AIDS Proceedings*, November 1979.

INFORMATION SCIENCES RESEARCH, AT&T LABS RESEARCH, FLORHAM PARK, NJ 07932 USA.
*E-mail address*: mgcr@research.att.com