RC 21515 (06/24/99)
Computer Science/Mathematics

IBM Research Report

Solving Large Scale Uncapacitated Location Problems

Francisco Barahona, Fabian Chudak

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems (P. M. Pardalos, Editor),

Solving Large Scale Uncapacitated Facility Location Problems

Francisco Barahona (barahon@watson.ibm.com)
IBM T.J. Watson Research Center,
PO Box 218, Yorktown Heights, New York

Fabián A. Chudak (chudak@watson.ibm.com)
IBM T.J. Watson Research Center,
PO Box 218, Yorktown Heights, New York

Abstract

We investigate the solution of instances of the uncapacitated facility location problem with at most 3000 potential facility locations and similar number of customers. We use heuristics that produce a feasible integer solution and a lower bound on the optimum. In particular, we present a new heuristic whose gap from optimality was generally below 1%. The heuristic combines the volume algorithm and a recent approximation algorithm based on randomized rounding. Our computational experiments show that our heuristic compares favorably against DUALOC.

Keywords: Volume algorithm, randomized rounding, facility location.

1 Introduction

The study of location of facilities to serve clients at minimum cost has been one of the most studied themes in the field of Operations Research (see, e.g., [13]). In this paper, we focus on one of its simplest variants, *the uncapacitated facility location problem* (UFLP), which has been extensively treated in the literature (see [7]). The problem can be described as follows. There is a set of potential facility locations $\mathcal{F}$; building a facility at location $i \in \mathcal{F}$ has an associated nonnegative fixed cost f_i, and any open facility can provide an unlimited amount of a certain commodity. There also is a set of customers or demand points $\mathcal{D}$ that require service; customer $j \in \mathcal{D}$ must be serviced from exactly one of the open facilities. If a facility at location $i \in \mathcal{F}$ is used to satisfy the demand of client $j \in \mathcal{D}$, the service or transportation cost incurred is proportional to the distance from i to j, c_{ij} . The goal is to determine a subset of the set of potential facility locations at which to open facilities and an assignment of clients to these facilities so as to minimize the overall total cost, that is, the fixed costs of opening the facilities plus the total service cost. In this paper we present a new heuristic for solving large scale instances of the problem.

The uncapacitated facility location problem is known to be NP-complete and due to its widely broad area of applications many heuristics have been devised to solve it. Among these, the most recognized in the literature is the one due to Erlenkotter [8], called DUALOC, which combines simple dual heuristics in a branch and bound framework. Typically, the computational experience that has been reported dealt with problems with several hundreds of potential facility locations as well as several hundreds of customers. In contrast, in this paper we present a heuristic designed to deal with larger instances. We report our computational experience with problems with up to 3000 potential facility locations and similar number of customers.

Most of the previous computational work on the UFLP problem focused on finding *optimal* solutions. For the larger instances we investigated, however, we focused on finding *approximate* solutions, with a relative error, say, of no more than 1%. On one hand, in practice, the data itself is not error-free. On the other hand, enumerative methods such as branch and bound, may require a prohibitive amount of resources (such as time and/or memory). For example, DUALOC spent 60 hours to find an optimal solution to an instance with 1500 points, while our heuristic found a solution within 1% in about 20 minutes.

Our new heuristic is based on the *Volume Algorithm*, introduced in [3], to approximately solve a linear programming relaxation and a sophisticated variant of randomized rounding, presented in [5, 6], to find a feasible solution to the UFLP. Our algorithm provides both an integer solution to the UFLP and a lower bound on the optimum value. In most of our experiments, the gap between the value of the integer solution and the lower bound was less than 1%. Our results compare favorably against partial outputs of DUALOC. A feature of our new heuristic is that it can be easily parallelized at almost optimal speed-up, thus it can be used to solve efficiently much larger instances than the ones reported here.

One of the simplest linear programming relaxation for the UFLP is the one due to Balinsky [2], and is as follows (for future reference we call it P for primal):

$$\text{Minimize} \sum_{j\in\mathcal{D}}\sum_{i\in\mathcal{F}} c_{ij}x_{ij} + \sum_{i\in\mathcal{F}} f_i y_i \tag{1}$$

$$\begin{aligned}
(\mathsf{P}) \qquad \text{subject to} \qquad & \sum_{i\in\mathcal{F}} x_{ij} = 1\ , && \text{for each } j \in \mathcal{D}, && (2)\\
& x_{ij} \le y_i\ , && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, && (3)\\
& x_{ij} \ge 0\ , && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}. && (4)
\end{aligned}$$

Any 0-1 feasible solution corresponds to a feasible solution to the uncapacitated facility location problem: $y_i = 1$ indicates that a facility at location $i \in \mathcal{F}$ is open, whereas $x_{ij} = 1$ means that client $j \in \mathcal{D}$ is serviced by the facility built at location $i \in \mathcal{F}$. Inequalities (2) state that each demand point $j \in \mathcal{D}$ must be assigned to some facility, whereas inequalities (3) say that clients can only be assigned to open facilities. Thus the linear program P is indeed a relaxation of the problem. Throughout, we will use n to denote the number of clients (that is, $n = |\mathcal{D}|$), and m to denote the number of potential facility locations ($m = |\mathcal{F}|$).

The linear programming relaxation P is known to provide excellent lower bounds in practice. Our results seem to confirm this hypothesis for large instances: starting from a primal solution of P we derive a "close" to optimum integer solution. However, since there are nm inequalities (3), solving P becomes prohibitive for commercial LP solvers for instances with, say, $n, m \ge 500$. Many approaches have been taken to deal with this problem (see [7]), and one of the most successful ones is based on subgradient optimization to obtain tight lower bounds. Previous work, however, only provided lower bounds, more concretely, a "good" dual solution, failing to provide "good" primal solutions. To tackle this difficulty, the volume algorithm [3] not only provides primal solutions, but also exhibits enhanced convergence properties.

Recently a significant amount of work has been devoted to investigate the *worst-case* analysis of algorithms for the UFLP. A ρ-approximation algorithm is a polynomial time algorithm that delivers a feasible solution within a factor of ρ of optimum, for *any* instance of the problem; and ρ is usually called the *performance guarantee* of the algorithm. When the distance function c is a metric, recent positive results on approximation algorithms for the UFLP began with the work of Shmoys, Tardos & Aardal [14], who presented the first constant factor approximation algorithm, with a performance guarantee of 3.16. The guarantee was subsequently improved to 2.41 by Guha & Khuller [10] and to 1.74 by Chudak & Shmoys [5, 6]. On the negative side, no performance guarantee less than 1.463 exists unless P=NP (see [10], and [15]).

2 Solving the linear programming relaxation

In this section we describe how to use the volume algorithm to approximately solve the linear programming relaxation P. First of all, since we use a Lagrangian method, to improve the quality of the subsequent relaxation, we need to add to P the following trivial upper bounds on the $y's$:

$$y_i \le 1, \quad \text{for each } i \in \mathcal{F}. \tag{5}$$

Let u_j be a dual multiplier for equation j in (2), and $\bar{c}_{ij} = c_{ij} - u_j$. If we dualize equations (2), a lower bound $L(u)$ is

$$L(u) = \text{Min} \quad \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}} \bar{c}_{ij} x_{ij} + \sum_{i \in \mathcal{F}} f_i y_i \tag{6}$$

$$\begin{aligned}
\text{s.t.} \qquad & x_{ij} \le y_i, && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, && (7)\\
& x_{ij} \ge 0, && \text{for each } i \in \mathcal{F},\ j \in \mathcal{D}, && (8)\\
& y_i \le 1, && \text{for each } i \in \mathcal{F}. && (9)
\end{aligned}$$

One can observe that this decomposes into m independent problems (one for each $i \in \mathcal{F}$). After dropping the index i, their generic form is

$$\begin{aligned}
& \text{Min} \ \ fy + \sum_j \bar{c}_j x_j \\
& x_j \le y, \quad j \in \mathcal{D}, \\
& 0 \le y \le 1, \quad 0 \le x.
\end{aligned}$$

This can be solved as follows. If $\bar{c}_j > 0$ then x_j should be 0 ($j \in \mathcal{D}$). Let

$$\mu = \sum_{j:\ \bar{c}_j \le 0} \bar{c}_j.$$

If $f + \mu < 0$ then we set $y = 1$ and $x_j = 1$ if $\bar{c}_j \le 0$. If $f + \mu \ge 0$ then all variables should be 0.

Since computing $L(u)$ is very easy, one can iterate this using the subgradient method to improve $L(u)$. This would give a good lower bound, but it does not give a primal solution that is needed for the approximation algorithms described in the next section. For this we use the *Volume Algorithm* [3], this is an extension of the subgradient method that produces primal solutions. We describe this method below.

Volume Algorithm

Step 0. Start with a vector $\bar{u}$ and solve (6)-(9) to obtain $(\bar{x}, \bar{y})$ and $L(\bar{u})$. Set $t = 1$.

Step 1. Compute v^t, where $v^t_j = 1 - \sum_i \overline{x}_{ij}$, and $u^t = \overline{u} + sv^t$ for a step size s given by (11) below.
Solve (6)-(9) with u^t. Let (x^t, y^t) be the solution thus obtained. Then $(\overline{x}, \overline{y})$ is updated as

$$(\overline{x}, \overline{y}) \leftarrow \alpha(x^t, y^t) + (1 - \alpha)(\overline{x}, \overline{y}), \tag{10}$$

where α is a number between 0 and 1. This is discussed later.

Step 2. If $L(u^t) > L(\overline{u})$ update $\overline{u}$ as

$$\overline{u} \leftarrow u^t.$$

Let $t \leftarrow t + 1$ and go to Step 1.

Notice that in Step 2 we update $\overline{u}$ only if $L(u^t) > L(\overline{u})$, so this is an ascent method, it has some similarities with the bundle method [12], one difference is that we do not solve a quadratic problem at each iteration.

One difference with the subgradient algorithm is the use of formula (10). If $(x^0, y^0), \ldots, (x^t, y^t)$ is the sequence of vectors produced by (6)-(9), then

$$(\overline{x}, \overline{y}) = \alpha(x^t, y^t) + (1 - \alpha)\alpha(x^{t-1}, y^{t-1}) + \cdots + (1 - \alpha)^t (x^0, y^0).$$

The assumption that this sequence approximates an optimal solution of (1)-(5) is based on a theorem in linear programming duality that appears in [3]. Notice the exponential decrease of the coefficients of this convex combination, thus later vectors receive a much larger weight than earlier ones. At every iteration the direction depends upon this convex combination, so this is a method with "memory" that does not have the zig-zagging behavior of the subgradient method.

As for the subgradient method [11], the formula for the step size is

$$s = \lambda \frac{UB - L(\overline{u})}{\|v\|^2}, \tag{11}$$

where λ is a number between 0 and 2, and UB is an upper bound for the optimal value.

In order to set the value of λ we define three types of iterations:

- Each time that we do not find an improvement we call this iteration *red*. A sequence of red iterations suggests the need for a smaller step-size.

- If $L(u^t) > L(\overline{u})$ we compute $w_j = 1 - \sum_i x^t_{ij}$ for all j and

  $$d = v^t \cdot w.$$

  If $d < 0$ it means that a longer step in the direction v^t would have given a smaller value for $L(u^t)$, we call this iteration *yellow*.

- If $d \geq 0$ we call this iteration *green.* A green iteration suggests the need for a larger step-size.

At each green iteration we would multiply λ by 1.1. After a sequence of 20 consecutive red iterations we would multiply λ by 0.66.

In order to set the value of α in (10), we solve the following 1-dimensional problem:

$$\begin{aligned}&\text{minimize } \|\alpha w + (1-\alpha)v^t\| \\ &\text{subject to} \\ &\frac{b}{10} \leq \alpha \leq b.\end{aligned}$$

Here w is defined as $w_j = 1 - \sum_i x^t_{ij}$ for all j. We try to minimize the norm of the new vector v^{t+1}, while using bounds to control α. The value b was originally set to 0.1 and then every 100 iterations we would check if $L(u^t)$ had increased by at least 1%, if not we would divide b by 2. When b becomes less than 10^{-5} we keep it constant. Each time that we would decrease b we would notice a decrease in the sum of the primal infeasibilities. This choice of α bears great similarities with the one proposed in [16], the difference is in the bounds $b/10$ and b.

3 Randomized Rounding

In this section we briefly describe the randomized rounding procedures of [5, 6].

Suppose that (x^*, y^*) is an optimal solution to the linear programming relaxation P. Consider first a simple randomized rounding (RR) that opens facility $i \in \mathcal{F}$ with *probability* y^*_i, and assigns each demand point to the closest open facility. Even though this algorithm performs well in practice, no worst-case performance guarantee is known for it. However, when the distance function c is a metric, a sophisticated variant of randomized rounding, *randomized rounding with clustering* (RRWC), presented in [5, 6], achieves a worst-case performance guarantee of 1.74. Remarkably, this algorithm generally outperforms the simple randomized rounding (see Section 5, and [4]). The success of the new algorithm relies on limiting the choices of randomized rounding by introducing dependencies in such a way that it uses additional structural information from the optimal solution to the linear programming relaxation P .

We briefly describe the algorithm of [5, 6]. As before, let (x^*, y^*) be an optimal solution to P. For each demand point $j \in \mathcal{D}$, the *neighborhood* of j, $\mathsf{N}(j)$, is the set of facilities $i \in \mathcal{F}$ for which $x^*_{ij} > 0$. The algorithm also makes use of the dual linear program, that can be written as follows: maximize $\{\sum_{j\in\mathcal{D}} v_j : \sum_{j\in\mathcal{D}} w_{ij} \leq f_i \ (i \in \mathcal{F})$; $v_j - w_{ij} \leq c_{ij} \ (i \in \mathcal{F},\ j \in \mathcal{D})$; $w_{ij} \geq 0 \ (i \in \mathcal{F},\ j \in \mathcal{D})\}$. Let (v^*, w^*) be an optimal dual solution.

First we break the set of points $\mathcal{F} \cup \mathcal{D}$ into clusters such that each demand point belongs to exactly one cluster, but facilities belong to at most one cluster. Also, each cluster has a center j such that all the facilities in the cluster are the neighbors of j; in addition, if k is any demand point in the cluster, and i is any facility location in the cluster, the distance c_{ik} is at most $2v_k^* + \sum_{i \in \mathcal{F}} c_{ik} x_{ik}^*$. More concretely, the clustering procedure works as follows. Let $\mathcal{S}$ be the set of demand points that have not yet been assigned to any cluster; initially, $\mathcal{S} = \mathcal{D}$. We find the unassigned demand point $j_\circ$ with smallest $(v_j^* + \sum_{i \in \mathcal{F}} c_{ij} x_{ij}^*)$-value and create a new cluster *centered* at $j_\circ$. Then all of the unassigned demand points that are fractionally serviced by facilities in the neighborhood of $j_\circ$ (that is, all the demand points $k \in \mathcal{S}$ with $\mathsf{N}(k) \cap \mathsf{N}(j_\circ) \neq \varnothing$) are assigned to the cluster centered at $j_\circ$; the set $\mathcal{S}$ is updated accordingly. We repeat the procedure until all of the demand points are assigned to some cluster (i.e., $\mathcal{S} = \varnothing$). The set of facilities that are in the neighborhood of a some center are called "central" facilities, and all the others "noncentral".

Now the algorithm of [5, 6] is a modification of the simple randomized rounding, that makes sure that always there is a facility open in each cluster. If facility i is noncentral, we open facility i with probability y_i^*. Next we treat central facilities as follows. We open exactly one facility per cluster: if j is the center, open neighbor $i \in \mathsf{N}(j)$ with probability x_{ij}^* (note that $\sum_{i \in \mathsf{N}(j)} x_{ij}^* = 1$). Notice next that it is possible that for a central facility i that belongs to the cluster centered at j, $x_{ij}^* < y_i^*$, and thus if facility i has not been opened by center j, we can open it now independently with probability $y_i^* - x_{ij}^*$. Finally, the algorithm assigns each demand point to the closest open facility. It was shown in [5, 6] that this algorithm produces a solution with expected value no more than a factor of 1.74 of optimum. Furthermore, the algorithm can be derandomized.

4 A new heuristic for the UFLP

In this section we describe a new algorithm that combines both the volume algorithm of [3] and the algorithm of [5, 6].

The intuition behind the new algorithm is based on the following two simple observations:

- the randomized rounding based procedure described in Section 3 takes substantially less time than the volume algorithm

- if we find an integral solution that is close to our current lower bound we can stop the run of the volume algorithm altogether.

The new heuristic exploits these facts by just running the procedure of Section 3 whenever the volume algorithm finds a "good" primal fractional solution. More precisely, we run the randomized heuristic of Section 3 (RRWC) when the current primal solution violates equations (2) by less than 20%. We call this new heuristic V&RRWC.

5 Computational experiments

In this section we present a representative of our computational experiments.

All of our experiments were conducted on an IBM RISC 6000/7043P-240, with a cpu running at 166MHz.

We implemented the volume algorithm as described in Section 1. In addition, using the solution returned by the volume algorithm, we implemented the simple randomized, RR, that open facility i with probability y_i^* (here y_i^* is the value returned by the volume algorithm for variable y_i) and assigns each demand point to the closest open facility, and the more sophisticated randomized rounding algorithm described in Section 3, RRWC. For each we generated 4000 random trials and recorded the best solution.

To implement randomized rounding with clustering, we used the following observation. If (x^*, y^*) is an optimal primal solution, and if we set $u_j = \min_{i \in \mathcal{F}: x_{ij}^* > 0} c_{ij}$, $(j \in \mathcal{D})$, then the v^*'s in RRWC can be replaced by the u's and this variant of the algorithm also achieves a performance guarantee of 1.74 . Thus we just implemented this simpler algorithm. In addition, to make sure that our primal solution was indeed feasible, we only extracted the y_i values from the volume algorithm and computed the x_{ij}'s from them in a straightforward way.

We implemented the new heuristic of Section 4, V&RRWC, by running RRWC inside the volume algorithm, every 10 iterations, whenever the violation of the current primal solution was less than 20%. In each run of the algorithm of Section 3, we generated 1000 trials and kept the solution with smallest cost. Whenever the gap between the value of the current best integer solution and the current lower bound was less than 1%, the algorithm stops.

We compared our algorithms against the two heuristics that are the key components of DUALOC [8]: the *dual ascent* procedure (ASCENT) and the more elaborated *dual adjustment* procedure (ADJUST). These two subroutines were extracted from DUALOC-II, provided to us by Erlenkotter [9]. We next describe the dual ascent procedure of Erlenkotter [8]. First note that the v_j's in the dual of P as in Section 3 completely determine a dual solution. Starting with $v_j = \min_{i \in \mathcal{F}} c_{ij}$ $(j \in \mathcal{D})$, the dual ascent procedure produces a maximal dual solution (v_j); that is, a solution for which none of the v_j's can be increased without losing feasibility; this is done by increasing one v_j value at a time, when v_j is changed from $c_{i'j}$ to the next largest $c_{i''j}$. Once a maximal dual solution is reached, all of the facilities that are tight in the dual are opened, and each client is assigned to the closest open facility. In addition, some facilities are closed if the objective function value of the solution improves (see [8] and [9] for details). For the dual adjustment procedure, the idea is the following: given a maximal dual solution, decrease one of the v_j's to its previous value and try to recover the loss in the objective function value of the dual using other v_j's, through applications of the dual ascent procedure. If the amount recovered exceeds the loss, the dual objective function value improves (see [8], [9]). In our experiments, we repeat the dual adjustment procedure until all the demand points are examined.

To test our new heuristic we considered instances in which both facility and demand points are distributed uniformly at random in the unit square $[0,1] \times [0,1]$, and the facility costs are all the same within some range of values. These instances are not very specialized, are easy to generate, are easy to solve for problems with up to 500 points, and exhibit some interesting properties as described by Ahn, Cooper, Cornuéjols and Frieze [1]. More concretely, n points are chosen independently uniformly at random in the unit square, and each point is simultaneously a potential facility location and a demand point. The distances correspond to the usual Euclidean distances in the plane. It was shown in [1] that, when n is large, any enumerative method based on LP lower bounds would require the exploration of an exponential number of solutions; also the value of the linear programming relaxation P is, asymptotically in the number of points, about 0.998% of optimum.

For each set of points, we set the fixed costs to be equal to $\sqrt{n}/10$ (type I), $\sqrt{n}/100$ (type II) and $\sqrt{n}/1000$ (type III). These different values provided instances with different properties.

Finally, to prevent numerical problems, we rounded the data to 4 significant digits and made all the data entries integer (this seemed to benefit DUALOC more than any other heuristics tested).

In Table 1, we report typical outputs of our experiments. The first column corresponds to the value of n, that is, the number of clients or facility locations. The second column corresponds to the value of each fixed cost f_i (before rounding up). The following two columns correspond to the values of the dual ascent and dual adjustment procedures of [9]. The next column corresponds to the lower bound provided by the volume algorithm. The following next two columns correspond to the simple randomized rounding algorithm and to the more sophisticated randomized algorithm of Section 3. Finally, the last column corresponds to our new heuristic of Section 4.

In Table 2, we report the relative errors of the heuristics tested (in average over 5 runs of each size/fixed cost). For the different types of instances, Figures 1, 2 and 3 give a more visual comparison of the performance of the algorithms. Notice that the difficulty of the problems changes from very hard for instances with large facility costs (type I), in which only few facilities can be opened, to fairly easy for instances with small facility costs (type III), in which almost all facilities can be opened.

A comparison of running times between the dual adjustment and the heuristic of Section 4 is given in Table 3 (time is measured in seconds).

In Table 4, we show how the number of iterations of the volume algorithm is reduced by the heuristic of Section 4. Originally, the stopping criteria for the volume algorithm is when the violation of equations (2) is less that 2% and the difference between the lower bound and the primal value is less than 1%. When we run V&RRWC, we stop when the difference between the lower bound and the value of the heuristic solution is less than 1%.

number of clients n	fixed costs	ASCENT	ADJUST	VOL	RR	RRWC	V&RRWC
500	$\sqrt{n}/10$	867381	836869	794686	794686	794686	796439
	$\sqrt{n}/100$	340219	325476	325270	328946	325618	326371
	$\sqrt{n}/1000$	99284	99064	99045	102089	102494	100410
1000	$\sqrt{n}/10$	1681731	1464966	1426160	1433870	1426820	1429330
	$\sqrt{n}/100$	631893	607111	603820	606043	607357	607372
	$\sqrt{n}/1000$	223019	221039	221004	228184	227572	224450
1500	$\sqrt{n}/10$	2476105	2126320	2009890	2095780	2060880	2022070
	$\sqrt{n}/100$	944951	888220	873405	890320	879189	880090
	$\sqrt{n}/1000$	341528	338282	337018	341672	342187	339323
2000	$\sqrt{n}/10$	3174067	2687037	2564950	2673720	2616440	2575990
	$\sqrt{n}/100$	1230267	1135976	1117760	1140000	1123860	1125150
	$\sqrt{n}/1000$	447213	440799	439569	444972	443732	443115
2500	$\sqrt{n}/10$	3790883	3326473	3087610	3201380	3165270	3098390
	$\sqrt{n}/100$	1511424	1375410	1352490	1388520	1368280	1364750
	$\sqrt{n}/1000$	550647	541157	539463	548989	547659	543897
3000	$\sqrt{n}/10$	4568080	3785751	3610970	3732960	3636350	3628860
	$\sqrt{n}/100$	1766589	1635448	1589360	1657590	1635740	1603490
	$\sqrt{n}/1000$	660188	644297	640693	655407	652844	645869

Table 1: Typical values of our experiments.

number of clients n	fixed costs	ASCENT	ADJUST	RR	RRWC	V&RRWC
500	$\sqrt{n}/10$	9.27%	4.01%	0.28%	0.06%	0.73%
	$\sqrt{n}/100$	6.28%	0.18%	1.37%	0.25%	0.52%
	$\sqrt{n}/1000$	0.10%	0.00%	3.26%	3.64%	1.77%
1000	$\sqrt{n}/10$	15.54%	4.18%	1.11%	0.36%	0.54%
	$\sqrt{n}/100$	6.51%	0.99%	0.96%	0.57%	0.56%
	$\sqrt{n}/1000$	1.05%	0.08%	2.92%	1.03%	1.47%
1500	$\sqrt{n}/10$	20.85%	5.27%	3.44%	1.81%	0.66%
	$\sqrt{n}/100$	9.87%	1.68%	2.38%	1.11%	0.93%
	$\sqrt{n}/1000$	1.82%	0.35%	1.68%	1.71%	0.79%
2000	$\sqrt{n}/10$	23.45%	4.85%	3.74%	2.09%	0.75%
	$\sqrt{n}/100$	10.62%	2.43%	1.74%	0.78%	0.76%
	$\sqrt{n}/1000$	02.34%	1.21%	1.68%	1.63%	0.90%
2500	$\sqrt{n}/10$	22.78%	7.12%	5.47%	3.99%	0.71%
	$\sqrt{n}/100$	12.40%	2.20%	3.64%	1.69%	1.06%
	$\sqrt{n}/1000$	2.79%	0.36%	1.58%	1.53%	0.79%
3000	$\sqrt{n}/10$	25.73%	16.79%	3.12%	1.74%	0.71%
	$\sqrt{n}/100$	12.79%	4.27%	3.42%	2.28%	0.93%
	$\sqrt{n}/1000$	3.62%	0.62%	1.79%	1.35%	0.85%

Table 2: Performance of the algorithms.

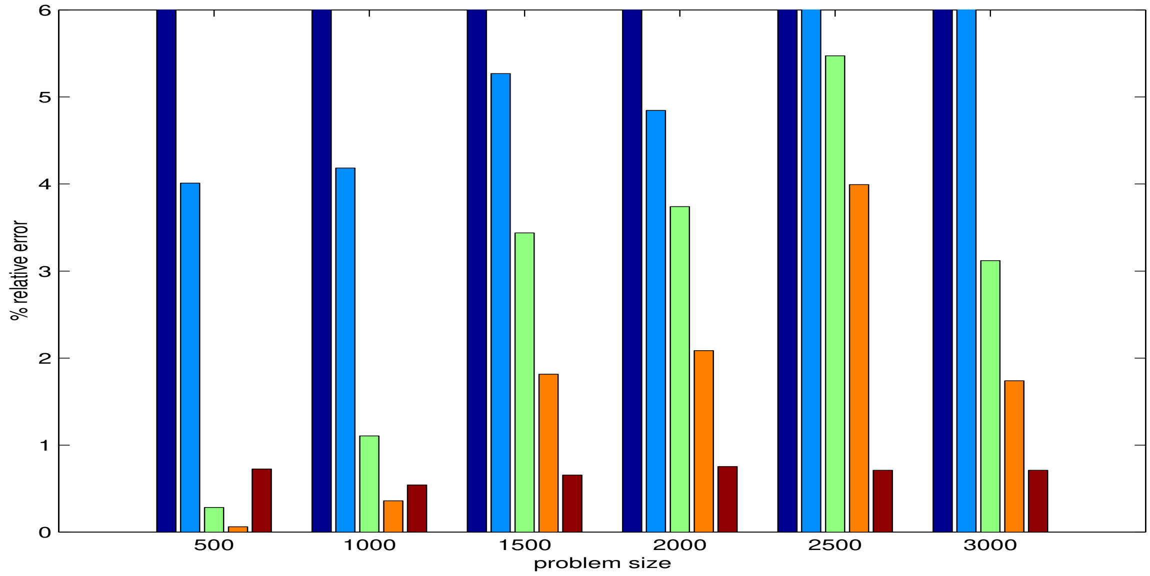

Figure 1: Relative errors for instances of type I. The columns correspond, from left to right, to ASCENT, ADJUST, RR, RRWC, V&RRWC.

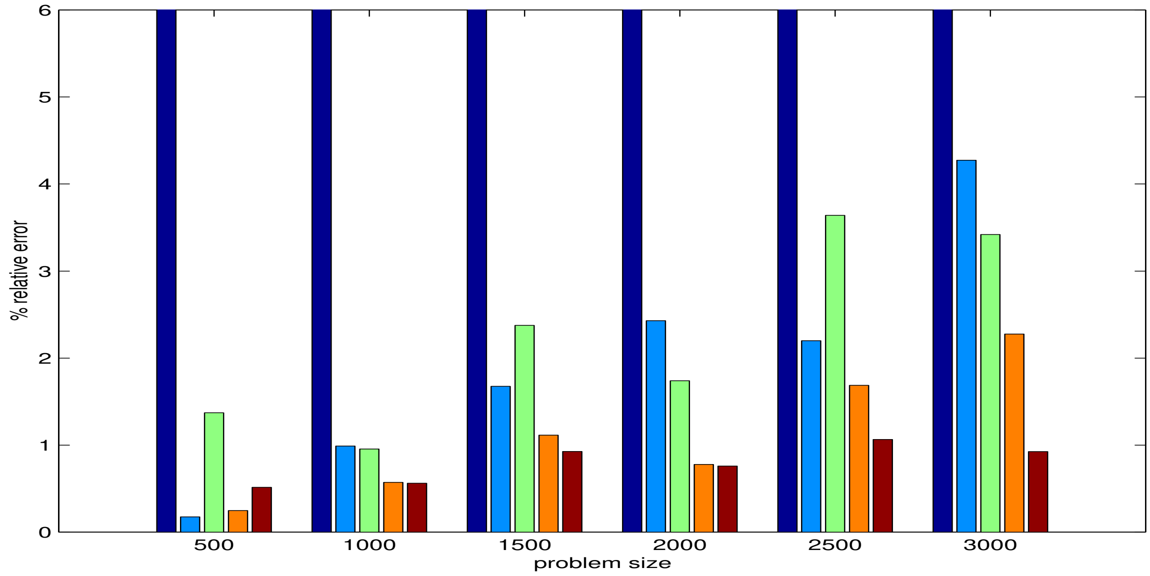

Figure 2: Relative errors for instances of type II. The columns correspond, from left to right, to ASCENT, ADJUST, RR, RRWC, V&RRWC.

number of clients n	fixed costs	ADJUST	V&RRWC
500	$\sqrt{n}/10$	13s	91s
	$\sqrt{n}/100$	4s	32s
	$\sqrt{n}/1000$	13s	27s
1000	$\sqrt{n}/10$	42s	480s
	$\sqrt{n}/100$	27s	145s
	$\sqrt{n}/1000$	57s	142s
1500	$\sqrt{n}/10$	151s	1221s
	$\sqrt{n}/100$	37s	364s
	$\sqrt{n}/1000$	129s	308s
2000	$\sqrt{n}/10$	481s	2513s
	$\sqrt{n}/100$	719s	2827s
	$\sqrt{n}/1000$	721s	1748s
2500	$\sqrt{n}/10$	1281s	4784s
	$\sqrt{n}/100$	1376s	3528s
	$\sqrt{n}/1000$	1318s	2604s
3000	$\sqrt{n}/10$	6911s	7512s
	$\sqrt{n}/100$	4074s	5510s
	$\sqrt{n}/1000$	1962s	3922s

Table 3: Comparing running times.

number of clients n	fixed costs	VOL	V&RRWC
500	$\sqrt{n}/10$	750	350/7
	$\sqrt{n}/100$	603	305/18
	$\sqrt{n}/1000$	535	535/45
1000	$\sqrt{n}/10$	990	490/10
	$\sqrt{n}/100$	610	390/19
	$\sqrt{n}/1000$	525	545/46
1500	$\sqrt{n}/10$	1102	590/11
	$\sqrt{n}/100$	702	465 /24
	$\sqrt{n}/1000$	550	340/25
2000	$\sqrt{n}/10$	1110	692/16
	$\sqrt{n}/100$	753	480/21
	$\sqrt{n}/1000$	549	320/22
2500	$\sqrt{n}/10$	1200	850/17
	$\sqrt{n}/100$	870	782/25
	$\sqrt{n}/1000$	555	330/23
3000	$\sqrt{n}/10$	1200	710/12
	$\sqrt{n}/100$	1200	590/24
	$\sqrt{n}/1000$	1200	330/23

Table 4: Reduction on the number of iterations.

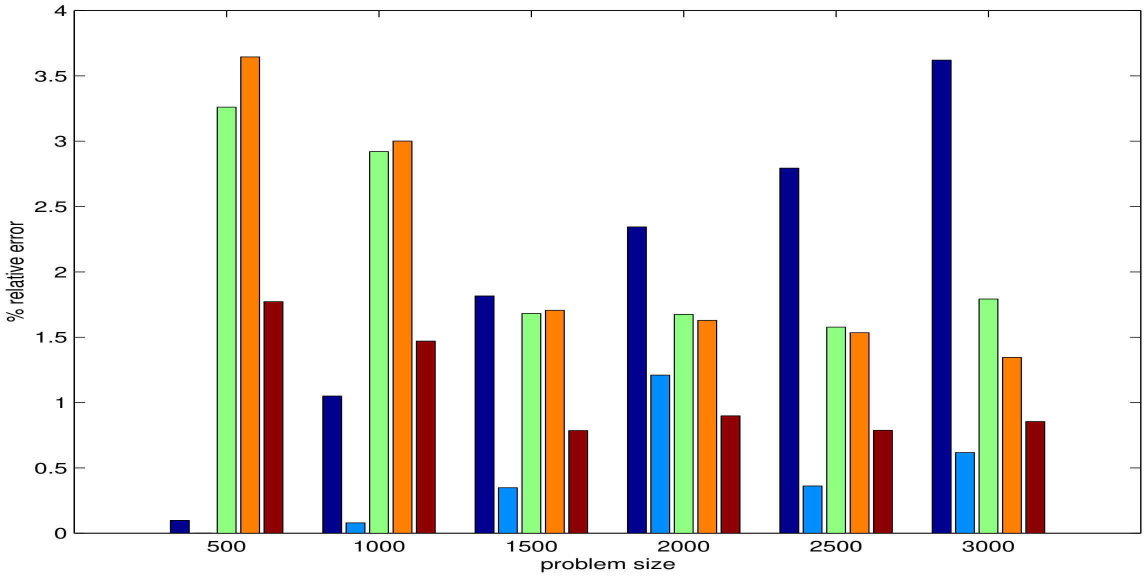

Figure 3: Relative errors for instances of type III. The columns correspond, from left to right, to ASCENT, ADJUST, RR, RRWC, V&RRWC.

As a final remark we point out that our experiments seem to suggest that large instances of the UFLP (say $n \geq 1000$) are inherently harder to solve, and that heuristics that work very well for medium-sized problems might do poorly in them.

References

[1] S. Ahn, C. Cooper, G. Cornuéjols, and A.M. Frieze. Probabilistic analysis of a relaxation for the k-median problem. *Mathematics of Operations Research*, 13:1–31, 1988.

[2] M.L. Balinski. Integer programming: methods, uses, computation. *Management Science*, 12(3):253–313, 1965.

[3] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with the subgradient method. Technical report, IBM Watson Research Center, 1998.

[4] F.A. Chudak. *Improved approximation algorithms for the uncapacitated facility location problem.* PhD thesis, Cornell University, 1998.

[5] F.A. Chudak. Improved approximation algorithms for uncapacitated facility location. In *Proceedings of the 6th IPCO Conference*, pages 180–194, 1998.

[6] F.A. Chudak and D.B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. In preparation, 1999.

[7] G. Cornuéjols, G.L. Nemhauser, and L.A. Wolsey. The uncapacitated facility location problem. In P. Mirchandani and R. Francis, editors, *Discrete Location Theory*, pages 119–171. John Wiley and Sons, Inc., New York, 1990.

[8] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.

[9] D. Erlenkotter, 1991. Program DUALOC – Version II. Distributed on request.

[10] S. Guha and S. Khuller. Greedy strikes back: improved facility location algorithms. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.

[11] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 49:62–88, 1991.

[12] C. Lemaréchal. Nondifferential optimization. In G.L. Nemhauser, A.H.G. Rinnoy Kan, and M.J. Todd, editors, *Optimization*, Handbooks in Operations Research, pages 529–572. North Holland, 1989.

[13] P. Mirchandani and R. Francis, eds. *Discrete Location Theory*. John Wiley and Sons, Inc., New York, 1990.

[14] D.B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[15] M.I. Sviridenko, July, 1998. Personal communication.

[16] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:145–173, 1975.