Computers & Operations Research 36 (2009) 1646 – 1655

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems

Doron Chen[a,*], Reuven Chen[b]

[a]IBM Haifa Research Lab, Tel-Aviv Site, Israel
[b]Raymond and Beverly Sackler School of Physics and Astronomy, Tel-Aviv University, Tel-Aviv 69978, Israel

ARTICLE INFO

Available online 3 April 2008

Keywords:
p-Center
Relaxation

ABSTRACT

We present new relaxation algorithms for the uncapacitated continuous and discrete p-center problems. We have conducted an experimental study that demonstrated that these new algorithms are extremely efficient.

1. Introduction

The p-center problem (see, for example, [1]), also known as the minimax location–allocation problem, deals with the optimal location of emergency facilities. The locations of n demand points are given, and we need to locate p service facilities. The *value* of a candidate solution to the p-center problem is the maximum distance between a demand point to its nearest service facility. Our objective is to find the solution with the minimal value; we want to locate the service facilities so as to minimize the maximum distance between a demand point to its nearest service facility. It is assumed that all the facilities perform the same kind of service, and that the number of demand points that can get service from a given center is unlimited.

Relaxation (in the context of this paper) [1,2] is a method to *optimally* solve a large location problem by solving a succession of small sub-problems. It is an iterative algorithm which updates, at each step, bounds on the optimal solution, until the optimal solution is reached. This paper presents new relaxation algorithms for the p-center problem.

Every step of a relaxation algorithm involves solving a p-center-like problem on a subset of the demand points. Our input is the subset and a value r, which is called the *coverage distance*. We need to answer: "Is there a solution to the sub-problem with value less than r?".

The new relaxation algorithms we describe try to reduce the number of iterations, or reduce the sizes of sub-problems, or reduce the values of the coverage distances (or a combination of these factors), so that we can improve performance, and therefore solve larger problems to optimality.

There are two main variants of the p-center problem in the literature; they differ by the possible location of the service points. Many authors deal with the *continuous* problem in which the points to be located optimally can be anywhere in the plane, but another interesting problem is the *discrete* case where there is a finite set of potential points (x_j, y_j) out of which one wishes to find the points which fulfill the minimax condition. In some cases, weights w_i are associated with the service points (a_i, b_i). Another classification of the problems is associated with the relevant metrics. In many cases, the distances between demand and service points are Euclidean (e.g., [3]). Also considered are problems where the distances are defined by minimal distances on a graph; this variant was first solved by Minieka [4].

The formulation of the Euclidean unweighted p-center problem is

$$\min_{X_1,\ldots,X_p}\left\{\max_{1\leqslant i\leqslant n}\left[\min_{1\leqslant j\leqslant p} r_{ij}\right]\right\}$$

where $X_j = (x_j, y_j)$ for $j = 1, \ldots, p$ is the location of the new facility and $r_{ij} = [(a_i - x_j)^2 + (b_i - y_j)^2]^{1/2}$. Megiddo and Supowit [5] have shown that both the p-center and p-median problems are NP-hard and that it is NP-hard even to approximate the p-center problems sufficiently closely. On the other hand, Hochbaum [6] has shown that given certain assumptions on the input distribution, there are polynomial algorithms that deliver a solution asymptotically close to the optimum with probability that is asymptotically one.

Most of the methods developed for solving the continuous Euclidean problem are geometrical in nature. When we are looking for a single service point ($p = 1$), the solution of the problem will be the center of the smallest circle enclosing n given points in the plane (see e.g., [7]). This can occur in one of two ways. The smallest circle

* Corresponding author. Tel.: +972 3 7689401; fax: +972 3 7689545.
E-mail addresses: cdoron@il.ibm.com (D. Chen), chenr@tau.ac.il (R. Chen).

can be determined by three demand points on its circumference or, alternatively, by two points on the two ends of a diameter. In the former case, the three points are the edges of an acute triangle [8]. The geometrical methods are based on a sophisticated search for the smallest enclosing circle among the circles built on subsets of two and three demand points. This includes the repeated solution of relaxed, smaller sub-problems as described below in the broader context of the *p*-center problem.

Chen [9] suggested a method that enables both the solution of the minisum and minimax location–allocation continuous problems by using a differentiable approximation to the objective function and solving it by using nonlinear programming. This enabled the solution of relatively large problems, but the result was not necessarily optimal since local minima may have been reached. Drezner [3,10] presented heuristic and optimal algorithms for the *p*-center problem in the plane. The heuristic method yielded results for problems with up to $n = 2000$ and $p = 10$ whereas the optimal method solved problems with up to $n = 30$, $p = 5$ or $n = 40$, $p = 4$. Watson-Gandy [11] suggested an algorithm that can optimally solve problems with up to about 50 demand points and 3 centers in reasonable time. The *p*-center problem on networks has been solved by Minieka [4] and by Toregas et al. [12]. A finite method, which is rather inefficient for large problems was suggested. An improvement based on the use of relaxations was offered by Handler and Mirchandani [1]. In Section 2 we elaborate on relaxation methods.

Some other papers dealing with the continuous *p*-center problem include the following. Hwang et al. [13] describe a slab-dividing approach, which is expected to efficiently solve the Euclidean *p*-center problem. These authors show that their algorithm has time complexity of $O(n^{O(\sqrt{p})})$. Suzuki and Drezner [14] propose heuristic procedures and upper bounds on the optimal solution where the demand points are distributed on a square. One of the methods they use employs the Voronoi heuristic. The same method has been recently used by Wei et al. [15]; the authors explore the complexity of solving the continuous space *p*-center problem in location planning. Agarwal and Sharir [16] discuss efficient approximate algorithms for problems in geometric optimization, which include the Euclidean *p*-center in *d* dimensions. Hale and Moberg [17] give a broad review on location problems, which includes the Euclidean *p*-center problem.

The discrete *p*-center problem is also known to be NP-hard [18]. For a review on discrete network location models see Current et al. [19]. Daskin [20] presents an optimal algorithm which solves the discrete *p*-center problem by performing a binary search over possible solution values. This algorithm solves maximal covering sub-problems, rather than the set-covering sub-problems solved by Minieka [4]. Mladenović et al. [21] present a basic Variable Neighborhood Search and two Tabu Search heuristics for the *p*-center problem without the triangle equality. Elloumi et al. [22] present a new integer linear programming formulation for the discrete *p*-center problem and show how to use this new formulation to obtain tight bounds on the optimal solution. They use these bounds in an exact solution method and report very good computational results. Recent works on the discrete problem include algorithms given by Caruso et al. [23] and by Ilhan et al. [24]. The latter authors describe an efficient exact method for the discrete *p*-center problem. A tight lower bound to the optimal value is found in an initial phase of the algorithm, which consists of solving linear programming sub-problems. Good computational results are reported for each of an extensive list of test problems derived from OR-Lib and TSP-Lib problems with up to 900 data points.

We present new relaxation algorithms. We report excellent computational results for both the continuous and discrete cases. We solved problems taken from OR-Lib [25] and TSP-Lib [26].

The rest of the paper is structured as follows. Section 2 explains the principles of relaxation and present new relaxation algorithms. In Section 3 we present the results of our experimental study. Section 4 contains conclusions and open problems.

2. Relaxation algorithms for the *p*-center problem

2.1. The *p*-center problem

The *p*-center problem, also known as the minimax location–allocation problem, deals with the optimal location of emergency facilities. We are given the locations of *n* demand points, and the objective is to locate *p* service facilities so as to minimize the maximum distance between a demand point to its nearest service facility.

Here is an equivalent way of looking at the same problem: we are given the locations of *n* demand points. We need to locate *p* circles that will cover all of the demand points. Our objective is to minimize the radius of the maximal circle. Clearly, this is exactly the *p*-center problem, where the centers of the *p* circles are the locations of the *p* service facilities.

When we consider the second interpretation of the *p*-center problem, we say that a set of *p* circles is a *feasible* solution to the problem, if the circles cover all of the demand points. When we consider the first interpretation, then any set of *p* points is considered a *feasible* solution to the *p*-center problem. Whenever convenient, we will alternate between the two equivalent definitions of the *p*-center problem.

2.2. Theory

Relaxation is a simple method to *optimally* solve a large location problem by solving a succession of small sub-problems. Although one cannot know in advance how many sub-problems need to be solved, once the global optimum is reached, it is identified as such. This as opposed to some heuristic methods which usually yield local minima. Though in the worst case relaxation may be very slow, it is usually very efficient.

Chen and Handler [2] adapted the relaxation method, previously suggested for the solution of location problems on networks [1], to the problem in continuous Euclidean two-dimensional space. In the solution of the *p*-center problem there is usually only one circle which is critical in the sense that two or three demand points are on its circumference. There is much freedom in the exact position of the other circles and therefore, in the location of all but one of the centers. The value of the solution is determined by the radius of this critical circle, whereas the radii of the other circles may vary in size below this critical value. Thus, the number of possible optimal solutions is usually infinite. Chen and Handler [2] proved a theorem stating that among all the optimal solutions to the minimax problem of serving *n* demand points in Euclidean space by *p* service points, there is at least one in which all demand points are covered by critical circles, the largest of which has a radius r_p, which is the value of the solution. With the aid of this theorem, the search can be reduced to a finite number of critical circles.

The number of critical circles to be considered is $\binom{n}{3} + \binom{n}{2} + n$, where $\binom{n}{3}$ is the number of circles determined by three points on their circumference, $\binom{n}{2}$ is the number of circles defined by two points determining the diameter and *n* is the number of null circles; a null circle is a service point located at a demand point, the former serving only the latter. The number of possible combinations to cover *n* points by *p* critical circles becomes very large when *n* is large. However, geometrical considerations associated with a known upper bound and with the properties of relevant triangles defined by demand points, significantly reduce the size of the sub-problem to be solved.

The discrete case is slightly simpler. Each critical circle is defined by a single potential service point and a single demand point. Here too, we can significantly reduce the size of the sub-problem. We only consider a single circle for each potential service point: a circle centered at the service point and whose radius is the distance to the furthest demand point still within the upper bound.

2.3. Relaxation

Before we present new improvements of relaxation algorithms, and new relaxation-based algorithms, we will explain, in further detail, the classic relaxation algorithm, suggested by Handler and Mirchandani [1].

Relaxation is a simple method to *optimally* solve a large location problem by solving a succession of small sub-problems. Fortunately, optimality can be achieved even though each sub-problem need not be solved to optimality. The relaxation algorithm is iterative; at each step a candidate solution to the original full problem is considered, which provides us with an upper bound on the optimal solution. We search, at each step, for a solution which improves upon the current candidate solution, until we prove that none exists.

Algorithm 1 describes the skeleton of a relaxation algorithm.

Alogorithm 1. Skeleton of relaxation algorithm

```
Upper_Bound ← ∞
Sub ← CHOOSERANDOMSUBSET()
while (solution not found)
  Feasible ← FINDFEASIBLESOLUTION(Sub, Upper_Bound)
  if (no feasible solution found for sub-problem)
    halt and return Best_Candidate
  else
    if (Feasible is a feasible solution to the original full problem)
      Best_Candidate ← Feasible
      Upper_Bound ← GETVALUE(Feasible)
    else
      ADDDEMANDPOINT(Sub)
```

When Algorithm 1 halts, *Best_Candidate* contains a solution to the original (full) location problem with value *Upper_Bound*. Furthermore, the algorithm halts after FINDFEASIBLESOLUTION fails to find a feasible solution with value better than *Upper_Bound*. Since there exists no solution to the sub-problem with value better than *Upper_Bound*, there could be no solution to the original problem with value better than *Upper_Bound*. Therefore the returned solution is optimal.

At each iteration we solve FINDFEASIBLESOLUTION(*Sub*, *Upper_Bound*). Our input is a subset, *Sub*, and a value *Upper_Bound*, which we call the *coverage distance*. We need to answer: "Is there a solution to the sub-problem with value less than *Upper_Bound*?''. In other words, we need to find a feasible (not necessarily optimal) solution to the relaxed (smaller) p-center problem, with value less than *Upper_Bound*. As we have explained earlier, we need only consider a finite number of critical circles in order to solve the problem.[1] Furthermore, since we are looking for a feasible solution with value less than *Upper_Bound*, we can eliminate, from our finite set of circles, all circles with radii greater or equal to *Upper_Bound*. As a result, the lower the coverage distance, the smaller the relaxed problem that we need to solve.

Three important factors that influence the running times of relaxation algorithms are:

(1) The sizes of the sub-problems, in term of the number of demand points.
(2) The number of sub-problems we need to solve, until we identify the optimal solution.
(3) The values of the coverage distances. The lower the coverage distances, the faster the algorithm.

In creating new relaxation algorithms, we are guided by these three factors. We try to design algorithms that will either reduce the number of sub-problems we need to solve, or reduce the sizes of sub-problems, or reduce the values of the coverage distances (or a combination of these factors).

2.4. Improvement: efficient updating of the upper bound

A simple change to Handler and Mirchandani's relaxation algorithm [1,2] has a considerable effect on its performance. Recall that whenever we find a feasible solution to the relaxed problem, we check to see if it is also feasible for the original full problem (see Algorithm 1). But what is a feasible solution? it depends on which of the two equivalent interpretations of the p-center problem we are considering (see Section 2.1). In [1,2], we look at the set of (at most) p circles which cover all of the demand points in the sub-problem, and check whether they also cover all of the demand points in the original full problem. We suggest a better alternative; if we view the p-center problem as a problem of locating a set of (at most) p service points, rather than viewing it as a problem of locating a set of (at most) p circles, then any set of p service points is a feasible (not necessarily optimal) solution to the full problem. Therefore, we consider the feasible solution which consists of the *centers* of the circles which cover the demand points in the sub-problem. This is a feasible solution to the full problem. We check to see if its value is less than the current *Upper_Bound*. If so, we update *Best_Candidate* and *Upper_Bound*, and continue.

As a result of this change, the upper bounds are updated more often. This has a very positive effect on performance; it allows us to make the most of the smaller sub-problems. By the time we get to the larger sub-problems, our upper bound is much better, and we solve problems with smaller coverage distances.

2.5. Improvement: adding more than one point

In the classic relaxation algorithm, we add a single demand point to the sub-problem whenever the feasible solution we have found for the sub-problem is not feasible for the complete full problem. But why add just one point?

Many of the sub-problems that relaxation solves are not "informative'' in the sense that they do not help us improve our bound on the solution, but simply serve as an indication that we need to add more points to the sub-problem. Adding more than one point may serve to reduce the number of "uninformative'' steps, and thus decrease the number of sub-problems we need to solve before we reach the optimal solution.

On the other hand, we do not want to add too many points. If we added, for instance, $k = 100$ demand points at a time, then our smaller sub-problems would quickly cease to be particularly small. We have shown experimentally (see Sections 3.3 and 3.4) that adding $k > 1$ points at a time often has very positive effect on performance.

The correctness of the relaxation algorithm does not depend on the points we choose to add. However, in order to improve performance, we add the k demand points farthest from the service points of the current feasible solution of the sub-problem. The distance of a demand point to a feasible solution is the distance of the demand point to the nearest service point. We add the most distant demand point, the second most distant, and so on up to the k-th most distant. If there are less than k points not

[1] This is true for both the discrete and the continuous cases.

covered by the current solution, we only add the demand points not covered.

Although we have not tested it experimentally, it appears that the best approach is a varying value of k. It appears to be a good idea to start with $k = 1$, and as the sub-problems grow larger, to increase k. The motivation behind this idea is that solving very small sub-problems, though adding more iterations, is very cheap computationally.

2.6. New algorithm: reverse relaxation

As we have mentioned earlier, solving sub-problems with smaller coverage distances has a profound effect on performance. The classic relaxation algorithm starts with an upper bound of infinity, and keeps updating it until the optimal value is reached (this is similar to Minieka's algorithm [4], which does not involve relaxation). In this section we describe an algorithm which starts with a lower bound of 0, and constantly updates it (upwards), until the optimal value is reached. Intuitively, it is better to solve sub-problems which are "too small'' (in terms of coverage distance), than to solve problems which are "too large''.

This idea is similar to that suggested by Ilhan et al. [24] for the discrete problem. Ilhan et al.'s algorithm has two phases. In the first stage they compute a tight lower bound on the optimal solution. At the second stage they gradually increase the lower bound, until the optimal value is reached. Our algorithm, which we call *Reverse Relaxation*, combines this approach with relaxation.

Reverse algorithm is based on two facts. First of all, the optimal solution of the p-center problem on a subset of the demand points is a lower bound on the optimal solution of the full p-center problem. Secondly, for any p-center problem we can limit ourselves to a finite set of possible values for the optimal solution (we use the same geometrical considerations which allow us only to consider a finite set of circles). The first fact helps us find tight lower bounds on the optimal solution (for instance, the solution to a size-50 sub-problem may be an extremely tight bound on the full size-500 problem). It also tells us that if an optimal solution to a sub-problem covers all of the demand points of the original full problem, then it is also the optimal solution to the full problem. The second fact helps us find the optimal solution to a sub-problem, by going over all the finite possible values for the solution, from the current lower-bound upwards, until a feasible solution is found.

Algorithm 2 describes the skeleton of a reverse relaxation algorithm.

Algorithm 2. Skeleton of a reverse relaxation algorithm

```
Lower_Bound ← 0
Sub ← CHOOSERANDOMSUBSET()
while (solution not found)
  Feasible ←FINDFEASIBLESOLUTION(Sub, Lower_Bound)
  if (feasible solution found for sub-problem)
    if (Feasible is a feasible solution to the original full problem)
      halt and return Feasible
    else
      ADDDEMANDPOINTS(Sub, k)
  else
    Lower_Bound ←GETNEXTBOUND(Sub, Lower_Bound)
```

Function GETNEXTBOUND returns the smallest value, among the possible values for the optimal solution, which is greater than *Lower_Bound*.

The main disadvantage of reverse relaxation is that we often need to solve many sub-problems before the optimal solution is reached. This is due to the fact that the finite set of possible solutions to a p-center problem is often very large, particularly for the continuous problem. However, this is usually not the case for discrete problems with integer distances. In Section 3.3 we will show that, indeed, reverse relaxation is extremely efficient for discrete problems with integer distances.

2.7. New algorithm: binary relaxation

The classic relaxation algorithm is similar to Minieka's, in the sense that the bound on the optimal solution is constantly updated downwards, until the optimal value is reached. The final relaxation algorithm that we suggest is similar to Daskin's [20,27], in the sense that it carries out a binary search on the optimal solution; it updates, at each step, either an upper or a lower bound on the optimal solution, until the optimal value is reached.

Binary relaxation attempts to enjoy the best of all worlds. It solves relatively few sub-problems, and it solves these sub-problems with typically small coverage-distance values.

At each step in binary search, we check whether there is a solution to the sub-problem with value less than *Coverage_Distance*. If there is none, then we update our lower bound to be *Coverage_Distance*, as there can be no solution to the full problem with value less than *Coverage_Distance*. If there is a solution to the sub-problem with value less than *Coverage_Distance*, then we check whether it is also a feasible solution to the full problem. If it is a solution to the full problem, we update *Upper_Bound* to be the value of this solution; if not, we add demand points to the sub-problem. In order to identify the optimal solution, whenever we update *Upper_Bound*, we check whether there is a solution to the sub-problem with value less than *Upper_Bound*. If there is none, we halt.

The same arguments that prove the correctness of the classic relaxation algorithm, also apply here. The algorithm halts when *Best_Candidate* contains a solution to the original (full) location problem with value *Upper_Bound*. Furthermore, the algorithm halts after we fail to find a feasible solution for the sub-problem with value better than *Upper_Bound*. Since there exists no solution to the sub-problem with value better than *Upper_Bound*, there could be no solution to the original problem with value better than *Upper_Bound*. Therefore, if the algorithm halts, then the returned solution is optimal.

But will the algorithm halt? A problematic scenario is when *Upper_Bound* is the optimal solution for the original full problem, but is not identified as such. This happens when we find a feasible solution for the sub-problem with value better than *Coverage_Distance*, though no such solution exists for the full problem. The problem arises from the fact that we perform a binary search to look for the optimal solution between *Lower_Bound* and *Upper_Bound*. If the optimal solution is exactly *Upper_Bound* then we get closer and closer to it, but we never quite reach it.

We solve this problem by checking, at each step, whether *Lower_Bound* is sufficiently close to *Upper_Bound*. If it is sufficiently close, then we halt. The problem we described does not happen often, particularly for p-center problems with a large number of demand points. As a result, performance is typically very good.

In our experiments (see Section 3), we ran a variant of Algorithm 3 which attempts to make the most of the smaller sub-problems before adding more demand points. In the variant we ran, whenever we find that the solution to the sub-problem is a solution to the original full problem, we enter a loop: we keep updating the upper bound as long as the feasible solutions we find for the sub-problem are feasible solutions for the original full problem.

If it is possible to create an array of all the possible solution values, then this algorithm can be improved by performing the binary search on the indices of this array.

Algorithm 3. Skeleton of a binary relaxation algorithm

```
$Lower\_Bound \leftarrow 0$
$Upper\_Bound \leftarrow \infty$
$Sub \leftarrow$ CHOOSERANDOMSUBSET()
while (solution not found)
  $Coverage\_Distance \leftarrow (Lower\_Bound + Upper\_Bound)/2$
  $Feasible \leftarrow$ FINDFEASIBLESOLUTION($Sub$, $Coverage\_Distance$)
  if (no feasible solution found for sub-problem)
    $Lower\_Bound \leftarrow Coverage\_Distance$
  else
    if ($Feasible$ is a feasible solution to the original full problem)
      $Best\_Candidate \leftarrow Feasible$
      $Upper\_Bound \leftarrow$ GETVALUE($Feasible$)
      if there is no feasible solution for $Sub$ with value
      less than $Coverage\_Distance$
        halt and return $Best\_Candidate$
    else
      ADDDEMANDPOINTS($Sub$, $k$)
```

2.8. Relaxation's weakness

Relaxation-based iterative algorithms represent a different approach to previous optimal iterative algorithms for the discrete p-center problem, such as those suggested by Ilhan et al. [24], Daskin [20,27] and Elloumi et al. [22]. The traditional iterative algorithms update, at each step, an upper or lower bound on the optimal solution, until the optimal solution is reached.

Roughly speaking, while traditional algorithms need to solve few large problems, relaxation algorithms need to solve many small problems. Relaxation needs to solve more problems, since many of its steps are not "informative" in the sense that they do not help us improve our bound on the solution, but simply indicate that we need to add more points to the sub-problem.

Relaxation performs particularly well for problems with relatively small values of p (see Table 2). As the value of p grows, the sizes of the problems that need to be solved grow larger too. Consequently, there exist problems for which relaxation needs to solve many sub-problems, where many of these sub-problems are not particularly small. For such problems one would be better off to use a non-relaxation algorithm which would solve much fewer sub-problems.

However, relaxation is not useless even for such "difficult" problems. Relaxation can be used to efficiently compute upper and lower bounds on the optimal solution (see Section 3.3.2). We can halt the relaxation algorithm once the sizes of the sub-problems become too large, and feed the bounds we have obtained to non-relaxation algorithms such as Daskin's [20,27]. Analyzing the performance of such semi-relaxation algorithms is outside the scope of this paper.

3. Experimental results

3.1. Methodology

For the discrete p-center problem, we compare the performance of our relaxation code to that of Ilhan et al.'s [24], run on the same computer. Ilhan et al.'s algorithm is a non-relaxation iterative algorithm which is a variant of Minieka's algorithm [4]. Unless we state otherwise, we solve at each iteration a feasibility sub-problem rather than a set-covering sub-problem [24].

Comparing our algorithms to that of Ilhan et al. gives us insight to the strengths and weaknesses of relaxation, although it should be noted that we made little effort to optimize either code.

Our relaxation algorithms accept a parameter k, which is the number of demand points we add to the sub-problem following an "uninformative" step (see Section 2.5). Unless stated otherwise, we tested the performance of the relaxation algorithms for different values of k, and report the results for the k values which yielded the best results.

3.2. Experimental setup

The experiments were conducted on a 3.2 GHz Pentium 4 computer with 2 GB of main memory. The computer runs the Linux 2.6.17. The code is written in C and compiled using gcc-4.0.

We formulated both the set-covering and the feasibility sub-problems as Integer Programming problems. We used CPLEX version 7.5 [28] for the solution of these Integer Programming problems. CPLEX implements optimizers based on the simplex algorithms.

3.3. Experimental analysis—discrete problems

3.3.1. Discrete p-center

Like Ilhan et al. [24] and Daskin [20], we tested the performance of our relaxation algorithm on the p-median (pmed) inputs[2] from OR-Lib [25]. We have found that on these problems improved relaxation, reverse relaxation and binary relaxation work best with parameters $k = 10$, 19 and 36, respectively.

Surprisingly, we have found that on these problems relaxation is not particularly sensitive to the values of k. It takes improved relaxation to solve the entire set of 40 problem between 25 and 28 s for any value of k between 5 and 50. It takes reverse relaxation between 10 and 15 s for any value of k between 5 and 50. It takes binary relaxation between 10 and 17 s for any value of k between 5 and 50.

It takes the original relaxation algorithm 55.02 s to solve all 40 problems. On the same computer, it took Ilhan et al.'s algorithm (the Minieka sub-problem variant) 83.58 s to solve all problems.

We divided the pmed problems to three: problems with "small" p values ($p = 5, 10$), "medium" p values ($20 \leqslant p \leqslant 67$), and "large" p values ($70 \leqslant p \leqslant 200$). Table 1 sums up the total running times for the "small", "medium", and "large" sets of problems. For complete information on the performance of relaxation algorithms versus Ilhan et al.'s algorithm on the pmed problems, see the Appendix.

We now demonstrate that our algorithms can handle larger p-center problems. Like Elloumi et al. [22], we tested the performance of our relaxation algorithms on problems rl1323 and u1817 from TSP-Lib [26]. Given the coordinates of the demand points in the plane, the Euclidean distance is computed and rounded to the nearest integer for every pair of demand points.

Surprisingly, we have found that for problems of this magnitude, it is better to solve the set-covering sub-problem at each step, rather than Ilhan et al.'s feasibility sub-problem, although for smaller problems the feasibility sub-problem was often much more efficient.

Table 2 shows the performance of reverse relaxation on problem u1817. We made no attempt to find the optimal parameter k for this problem. We chose $k = 16$, which seemed to work well. Besides running times, Table 2 shows, for each problem, the size of the maximal sub-problem solved by the relaxation algorithm, and the total number of sub-problems solved by the relaxation algorithm.

Table 3 shows the performance of the reverse and binary relaxation algorithms on problem rl1323 with $k = 16$.

[2] In these problems, the set of demand points and the set of potential service points are the same.

Table 1
Total running times (in seconds) of Ilhan et al.'s algorithm and different relaxation algorithms for discrete pmed problems.

	Ilhan et al. (s)	Relaxation (s)	Improved relaxation $k=1$ (s)	Improved relaxation $k=10$ (s)	Reverse relaxation $k=1$ (s)	Reverse relaxation $k=19$ (s)	Binary relaxation $k=1$ (s)	Binary relaxation $k=36$ (s)
Small	58.73	7.16	6.21	4.9	5.43	3.97	6.11	**3.07**
Medium	5.44	9.46	9.15	5.6	6.24	**2.02**	6.95	2.07
Large	19.41	38.4	39.36	15.39	30.31	**4.6**	31.61	5.5

In bold font we mark the best results for the set of problems.

Table 2
The performance of our reverse relaxation with $k=16$ for u1817, a large discrete problem from TSP-Lib.

Input	n	p	Obj.	Reverse relaxation (s)	Max. prob.	No. sub.
u1817	1817	5	715	13.23	264	488
u1817	1817	10	458	112.62	412	292
u1817	1817	15	359	1728.53	626	223
u1817	1817	20	309	9061.35	771	203
u1817	1817	25	272	7919.15	742	180
u1817	1817	30	241	19556.36	789	171
u1817	1817	35	227	19109.32	903	170

The two rightmost columns are for the maximal size of a relaxation sub-problem and the total number of relaxation sub-problems.

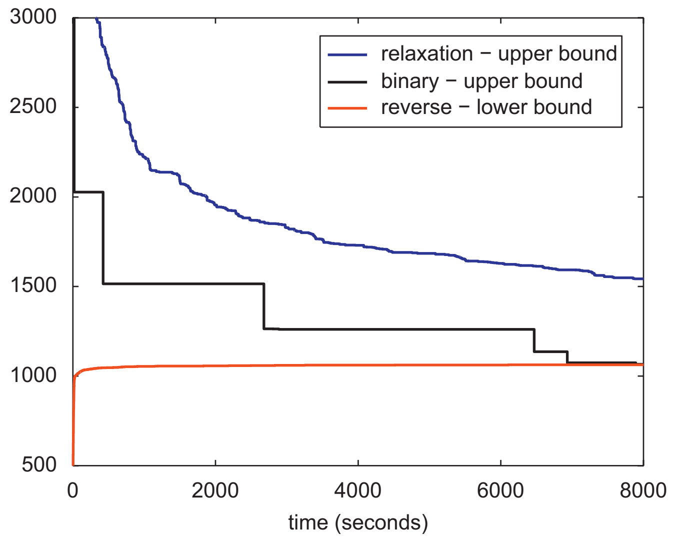

Fig. 1. The progression of relaxation bounds for discrete problem rl1323 with 1323 demand points for $p=60$. The figure shows the upper bounds obtained by improved relaxation and binary relaxation as well as the lower bounds obtained by reverse relaxation. All these algorithms were run with parameter $k=16$.

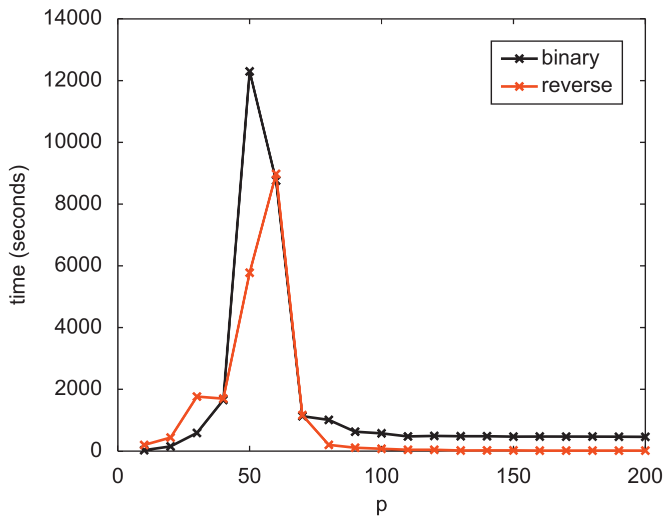

Fig. 2. The running times (in seconds) of binary relaxation and reverse relaxation for discrete problem rl1323 with 1323 demand points for different values of p. Both algorithms were run with parameter $k=16$.

3.3.2. The progression of relaxation bounds

We now demonstrate the progression of relaxation bounds on a large discrete p-center problem. Fig. 1 shows the upper and lower bounds obtained by relaxation algorithms on the solution of discrete problem rl1323 with $p=60$. Note that the progression of improved relaxation and reverse relaxation are much more gradual than that of the binary relaxation. Also note that although it takes about 2 h and 26 min for reverse relaxation to solve the problem, it reaches the lower bound of 1011 after 1 min, and 1042 after 5 min. The exact solution is 1063.

3.3.3. "Difficult" p-center problems

Fig. 2 shows the running times of binary relaxation and reverse relaxation on discrete problem rl1323 for different values of p. Although the behavior of the two algorithms is different, note that both algorithms find problems with small values of p and problems with large values of p to be relatively easy. Intermediate values are difficult for both algorithms. The reason that problems with small values of p are easy is that they can be solved before the relaxed sub-problems become too large (in terms of the number of demand points). Problems with large values of p are also easy, although the sizes of the maximal relaxation sub-problems we need to solve are quite large. Relaxation works well in these cases because the earlier stages, when the sub-problems are small, yield tight bounds on the solution, which simplify the problems we need to solve as the relaxation subsets become bigger. Another way of looking at it is that problems with large values of p are easy because the value of the solution is small, and we need to solve sub-problems with small coverage distances. As we have mentioned earlier, smaller coverage distances tend to lead to easier sub-problems.

3.3.4. Discrete p-center problems with Euclidean distances

Like Ilhan et al. [24], we tested the performance of our relaxation algorithm on problems from TSP-Lib [26]. Table 4 sums up the total running times for $p=5, 10, 20, 40$. For more details on the performance of relaxation versus Ilhan et al.'s algorithm on the TSP-Lib problems, see the Appendix.

Table 3
The performance of reverse and binary relaxation with $k=16$ for rl1323, a large discrete problem from TSP-Lib.

Input	n	p	Obj.	Reverse relaxation (s)	Reverse max. prob.	Reverse no. sub.	Binary relaxation (s)	Binary max. prob.	Binary no. sub.
rl1323	1323	10	3077	198.38	385	2220	**27.03**	490	96
rl1323	1323	20	2016	437.65	512	1409	**152.36**	614	100
rl1323	1323	30	1632	1764.54	630	1103	**585.97**	808	123
rl1323	1323	40	1352	1696.81	686	892	**1655.82**	907	135
rl1323	1323	50	1187	**5776.33**	798	778	12 298.17	959	140
rl1323	1323	60	1063	8976.75	782	655	**8756.93**	1014	148
rl1323	1323	70	972	1162.37	811	583	**1128.03**	911	136
rl1323	1323	80	895	**202.59**	769	518	1010.09	957	142
rl1323	1323	90	832	**106.78**	782	464	626.40	992	144
rl1323	1323	100	787	**74.56**	791	442	574.26	1013	149
rl1323	1323	110	741	**42.85**	814	410	475.27	972	137
rl1323	1323	120	697	**41.98**	780	373	490.38	1012	152
rl1323	1323	130	664	**19.18**	775	347	479.65	1050	156
rl1323	1323	140	651	**22.21**	767	345	479.42	1058	162
rl1323	1323	150	617	**20.80**	799	325	468.11	1030	153

In bold font we mark the best result for the problem.

Table 4
Total running times (in seconds) of Ilhan et al.'s algorithm and different relaxation algorithms for discrete problems with Euclidean distances from TSP-Lib.

	Ilhan et al. (s)	Relaxation (s)	Improved relaxation $k=1$ (s)	Improved relaxation $k=13$ (s)	Reverse relaxation (SC) $k=1$ (s)	Reverse relaxation (SC) $k=10$ (s)	Binary relaxation $k=1$ (s)	Binary relaxation $k=11$ (s)
$p=5$	89.82	8.17	6.42	6.33	15.79	89.05	3.35	**3.27**
$p=10$	64.92	48.73	29.68	20.4	42.77	119.57	10.65	**9.56**
$p=20$	120.77	248.9	124.77	106.59	274.69	155.13	64.84	**33.43**
$p=40$	457.61	808.61	364.46	**112.8**	1390.61	434.4	1779.07	132.01

In bold font we mark the best result for the set of problems.

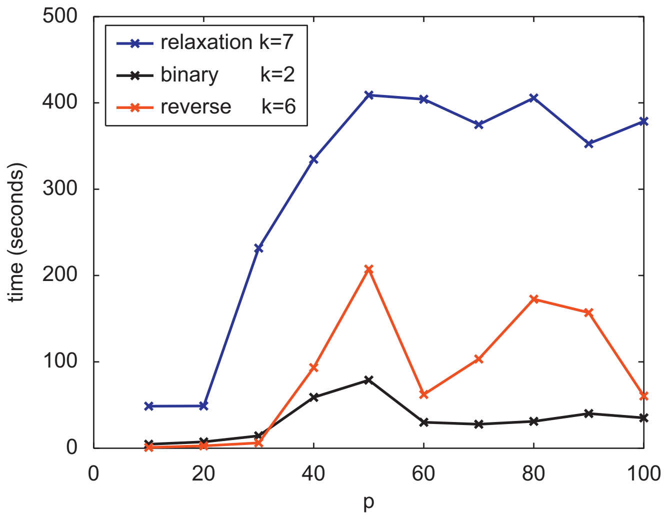

Fig. 3. The running times (in seconds) of the new relaxation algorithms for continuous problem pr439 with 439 demand points for different values of p.

Not surprisingly, we see that reverse relaxation does not perform well on these problems. For these problems we have found that reverse relaxation performs better with the set-covering sub-problem rather than Ilhan et al.'s feasibility sub-problem.

3.4. *Experimental analysis—continuous problems*

3.4.1. *Continuous p-center problems with Euclidean distances*

Fig. 3 shows the performance of the new relaxation algorithms on problem pr439 for TSP-Lib [26] with 439 demand points. We solve the problem for $p=10, 20, \ldots, 100$ service points. To the best of our knowledge, no one has ever attempted to solve *continuous* problems of this magnitude. We got the best results for improved relaxation with $k=7$, for binary relaxation with $k=2$ and for reverse relaxation with $k=6$. We have found that reverse relaxation performs better with the set-covering sub-problem rather than Ilhan et al.'s feasibility sub-problem. For complete information on the performance of the relaxation algorithms on this problem, see the Appendix.

4. Conclusions and open problems

4.1. *Open problems*

We list a number of ideas how to improve the performance of relaxation algorithms.

Chen and Handler [2] proposed to solve the p-center problem by first solving, optimally, the 1-center, then the 2-center, up to the $(p-1)$-center problem. The motivation is to start solving the p-center problem when you already have a reasonably tight upper bound on the solution. We have generally found that this approach hinders, rather than improves, performance. However, it might be beneficial to use other "step sizes"; if, for instance, we want to solve a 40-center problem, we might do well to first pre-solve the 10-, 20- and 30-center problems. If these "pre" problems are also difficult to solve, we can begin solving them and stop before we reach optimality. Any upper bound on the 30-center problem is also a bound on the 40-center problem.

In Section 2.5 we suggested to use a varying value of k (the number of demand points we add to a sub-problem if we cannot improve the bounds). It appears to be a good idea to start with $k=1$, and as the sub-problems grow larger, to increase k. We have not tested this idea experimentally.

Since discrete p-center problems are easier to solve than their continuous counterparts, we could attempt to improve the performance of relaxation on continuous problems by using the solution of a discrete problem as an initial upper bound. We could solve the discrete problem where the set of potential service points is identical to the set of demand points.

Table 5
The performance of Ilhan et al.'s algorithm and different relaxation algorithms for discrete problems with "small" values of p.

Input	n	p	Obj.	Ilhan et al. (s)	Relaxation (s)	Binary relaxation $k=34$ (s)	Reverse relaxation $k=19$ (s)	Max. prob. reverse relaxation	No. sub. reverse relaxation
pmed1	100	5	127	0.27	0.12	**0.09**	0.27	45	95
pmed6	200	5	84	0.38	0.19	**0.05**	0.22	44	79
pmed11	300	5	59	0.81	0.19	**0.1**	0.16	29	51
pmed16	400	5	47	1.15	0.19	**0.04**	0.2	27	42
pmed21	500	5	40	1.82	0.45	**0.19**	0.22	34	43
pmed26	600	5	38	4.35	0.23	**0.19**	0.21	29	39
pmed31	700	5	30	3.86	0.27	**0.08**	0.15	41	34
pmed35	800	5	30	4.88	0.17	**0.11**	0.2	23	33
pmed38	900	5	29	7.75	0.25	**0.08**	0.16	28	27
pmed2	100	10	98	0.11	0.27	**0.01**	0.13	41	63
pmed3	100	10	93	0.09	0.3	**0.04**	0.16	45	53
pmed7	200	10	64	0.25	0.24	0.19	**0.11**	42	46
pmed12	300	10	51	0.68	0.28	**0.13**	0.15	39	43
pmed17	400	10	39	1.18	0.5	0.23	**0.21**	54	37
pmed22	500	10	38	2.54	1.36	0.28	**0.26**	69	38
pmed27	600	10	32	3.42	0.34	0.2	**0.17**	47	35
pmed32	700	10	29	11.01	0.59	0.5	**0.3**	59	32
pmed36	800	10	27	5.33	0.78	**0.18**	0.4	52	32
pmed39	900	10	23	8.85	0.44	0.38	**0.29**	54	27

The two rightmost columns are for the size of the maximal sub-problem solved by the reverse relaxation algorithm and for the total number of sub-problems solved by the reverse relaxation algorithm. In bold font we mark the best result for the problem.

Table 6
The performance of Ilhan et al.'s algorithm and different relaxation algorithms for discrete problems with "medium" values of p.

Input	n	p	Obj.	Ilhan et al. (s)	Relaxation (s)	Binary relaxation $k=34$ (s)	Reverse relaxation $k=19$ (s)	Max. prob. reverse relaxation	No. sub. reverse relaxation
pmed4	100	20	74	**0.08**	0.32	0.1	0.19	62	60
pmed8	200	20	55	0.18	0.37	**0.1**	0.18	78	57
pmed13	300	30	36	0.4	0.7	**0.11**	0.21	85	43
pmed5	100	33	48	0.08	0.37	0.13	**0.07**	59	41
pmed9	200	40	37	0.17	0.57	**0.11**	0.12	90	41
pmed18	400	40	28	0.74	1.21	0.4	**0.37**	141	41
pmed23	500	50	22	0.89	2.4	0.51	**0.3**	151	33
pmed14	300	60	26	0.3	1.11	**0.25**	**0.25**	143	42
pmed28	600	60	18	2.49	1.77	0.3	**0.27**	170	29
pmed10	200	67	20	0.11	0.64	**0.06**	**0.06**	115	27

Table 7
The performance of Ilhan et al.'s algorithm and different relaxation algorithms for discrete problems with "large" values of p.

Input	n	p	Obj.	Ilhan et al. (s)	Relaxation (s)	Binary relaxation $k=34$ (s)	Reverse relaxation $k=19$ (s)	Max. prob. reverse relaxation	No. sub. reverse relaxation
pmed33	700	70	15	3.92	3.42	0.64	**0.48**	216	32
pmed19	400	80	18	0.38	2.06	0.32	**0.29**	200	33
pmed37	800	80	15	3.49	3.81	0.76	**0.61**	237	35
pmed40	900	90	13	7.11	7.2	**0.81**	0.95	310	36
pmed15	300	100	18	0.23	1.16	0.18	**0.14**	166	28
pmed24	500	100	15	0.61	2.77	0.46	**0.31**	232	31
pmed29	600	120	13	0.8	3.01	0.44	**0.37**	252	31
pmed20	400	133	13	0.35	2.68	0.34	**0.22**	226	30
pmed34	700	140	11	1.24	3.76	0.55	**0.51**	306	34
pmed25	500	167	11	0.55	3.52	0.45	**0.34**	279	31
pmed30	600	200	9	0.73	5.01	0.55	**0.38**	331	29

Another promising approach is to "restart" when the sub-problems get too large. At any point in the relaxation algorithms, we can start over with a small new sub-problem, and proceed while using the upper and/or lower bounds obtained so far. Restarts could be combined with randomization. Whenever we restart, we could choose a random subset of the demand points. Also, whenever we need to add demand points to the sub-problem we can use randomization. Currently, we always add the k farthest demand points from the current service points, which is a very good heuristic. However, if solving the larger sub-problem takes too long, we can replace the k farthest demand points with k random points. The correctness of the relaxation algorithm does not depend on the points we choose to add.

It might be interesting to investigate whether we can employ Vijay's method [29] to solve the reduced problems and improve our relaxation algorithm for the continuous case.

Another question is whether our relaxation algorithms can be improved by applying heuristics to efficiently provide us with tight initial upper and lower bounds on the solution.

We have witnessed that the choice of algorithm for the sub-problems had a profound effect on performance. In some cases the set-covering algorithm was several times more efficient than the

Table 8
The performance of Ilhan et al.'s algorithm and relaxation algorithms for discrete p-center problems with Euclidean distances.

Input	n	p	Obj.	Ilhan et al. (s)	Relaxation (s)	Improved relaxation $k = 13$ (s)	Binary relaxation $k = 11$ (s)	Max. prob. binary relaxation	No. sub. binary relaxation
kroA200.tsp	200	5	911.412091	0.29	0.36	0.26	**0.13**	73	22
kroB200.tsp	200	5	897.669204	0.39	0.24	0.17	**0.06**	46	15
gr202.tsp	202	5	19.384514	12.23	0.1	0.1	**0.05**	18	15
pr226.tsp	226	5	3720.551034	0.7	0.41	0.39	**0.1**	74	19
pr264.tsp	264	5	1610.124219	0.42	0.16	0.24	**0.05**	38	11
pr299.tsp	299	5	1336.272801	0.71	0.95	0.62	**0.25**	82	32
lin318.tsp	318	5	1101.339639	0.66	0.58	**0.57**	0.7	100	64
pr439.tsp	439	5	3196.580204	1.71	0.71	0.75	**0.32**	93	26
pcb442.tsp	442	5	1024.74387	2.32	1.79	1.39	**0.44**	116	29
d493.tsp	493	5	752.908474	14.75	0.77	**0.51**	0.76	62	51
d657.tsp	657	5	880.908537	55.64	2.1	1.33	**0.41**	114	26
kroA200.tsp	200	10	598.819672	0.37	1.19	0.56	**0.29**	104	29
kroB200.tsp	200	10	582.103943	0.35	1.69	0.67	**0.25**	110	27
gr202.tsp	202	10	9.334002	10.67	0.39	0.29	**0.14**	77	28
pr226.tsp	226	10	2326.478025	0.21	0.5	0.39	**0.12**	98	25
pr264.tsp	264	10	850	0.31	0.46	0.47	**0.16**	78	29
pr299.tsp	299	10	888.835755	0.84	2.05	1.43	**0.81**	128	73
lin318.tsp	318	10	743.210603	0.52	2.32	1.55	**0.5**	179	43
pr439.tsp	439	10	1971.832904	1.39	1.49	0.79	**0.45**	126	37
pcb442.tsp	442	10	670.820393	3.57	9.5	4.23	**1.49**	197	38
d493.tsp	493	10	458.304571	13.16	2.97	2.23	**0.69**	155	38
d657.tsp	657	10	574.744682	33.53	26.17	7.79	**4.66**	245	46
kroA200.tsp	200	20	389.307077	**0.21**	1.9	0.96	0.26	141	39
kroB200.tsp	200	20	382.280002	0.79	2.23	0.86	**0.2**	141	36
gr202.tsp	202	20	5.56569	9.08	1.29	0.63	**0.21**	121	35
pr226.tsp	226	20	1365.650028	0.27	0.68	0.56	**0.13**	125	26
pr264.tsp	264	20	514.781507	0.28	0.87	0.84	**0.22**	122	37
pr299.tsp	299	20	559.016994	**0.43**	3.58	1.94	0.84	181	73
lin318.tsp	318	20	496.452415	**0.4**	2.65	1.45	0.8	172	75
pr439.tsp	439	20	1185.59057	0.99	2.65	1.91	**0.61**	170	46
pcb442.tsp	442	20	447.213595	**1.73**	16.65	11.32	10.52	315	61
d493.tsp	493	20	312.744896	55.73	12.47	6.13	**2.04**	225	51
d657.tsp	657	20	374.7	50.86	203.93	79.99	**17.6**	421	89
kroA200.tsp	200	40	258.259559	0.2	2.68	1.03	**0.18**	149	40
kroB200.tsp	200	40	253.237043	**0.19**	2.89	1.01	0.22	163	42
gr202.tsp	202	40	2.971363	7.01	2.93	1.2	**0.22**	148	44
pr226.tsp	226	40	650	0.21	1.13	0.8	**0.18**	156	34
pr264.tsp	264	40	316.227766	**20.64**	443.51	29.14	32.75	202	46
pr299.tsp	299	40	355.31676	**0.35**	4.2	2.36	0.67	198	76
lin318.tsp	318	40	315.919293	**0.66**	5.66	2.06	0.71	211	47
pr439.tsp	439	40	671.751442	0.99	6.92	4.5	**0.84**	244	57
pcb442.tsp	442	40	316.227766	140.48	13.1	**4.82**	27.45	322	77
d493.tsp	493	40	206.015655	**2.03**	17.94	8.54	2.53	321	68
d657.tsp	657	40	249.51541	284.85	307.65	**57.34**	66.26	491	82

In bold font we mark the best result for the problem.

Table 9
The performance of relaxation algorithms for a continuous p-center problems with Euclidean distances.

Input	n	p	Obj.	Improved relaxation $k = 7$ (s)	Reverse relaxation (SC) $k = 2$ (s)	Binary relaxation $k = 6$ (s)	Max. prob. binary relaxation	No. sub. binary relaxation
pr439.tsp	439	10	1716.509904	48.67	**0.84**	4.53	147	51
pr439.tsp	439	20	1029.714766	48.88	**2.63**	7.28	176	60
pr439.tsp	439	30	739.192972	231.76	**6.17**	14.33	257	71
pr439.tsp	439	40	580.005388	334.66	93.38	**58.92**	314	90
pr439.tsp	439	50	468.54162	408.9	207.45	**78.89**	345	95
pr439.tsp	439	60	400.195265	404.17	62.19	**29.98**	341	82
pr439.tsp	439	70	357.945527	374.86	103.28	**27.72**	360	83
pr439.tsp	439	80	312.5	405.57	172.59	**31.04**	375	89
pr439.tsp	439	90	280.902563	352.71	157.07	**40.05**	408	101
pr439.tsp	439	100	256.680194	378.59	60.4	**35.07**	401	105

In bold font we mark the best result for the problem.

feasibility sub-problem. In other cases it was the other way around. This suggests that there could be other algorithms, or other implementations of the same algorithms, which might yield better results.

4.2. Conclusions

Relaxation is a simple method to *optimally* solve a large location problem by solving a succession of small sub-problems.

We presented new variants of relaxation, for the discrete and continuous p-center problems. We have conducted an experimental study that demonstrated that these new variants are very efficient. For the discrete case, our algorithm often outperforms other optimal algorithms; for the continuous case, the algorithm solves problems which were previously considered too large.

Our experiments show that relaxation algorithms work particularly well for problems with either small or large value of p. For small values of p relaxation needs to solve relatively small sub-problems in terms of the number of demand points. For large values of p relaxation need to solve relatively small problems in terms of the coverage distance.

Acknowledgments

Thanks to Taylan Ilhan for sending us his code for the discrete p-center problem. Thanks to Gil Shklarski, Sivan Toledo and Leon Ankonina for their help with the utilization of CPLEX.

Appendix A. Tables

A.1. Discrete p-center problems

Tables 5–7 show the performance of Ilhan et al.'s algorithm, the original relaxation algorithm, the binary relaxation algorithm with $k = 34$ and the reverse relaxation algorithm with $k = 19$. Tables 5–7 show the performance of these algorithms for problems with "small", "medium", and "large" values of p, respectively. The input files are taken from OR-Lib [25]. Besides running times, the tables show, for each problem, the size of the maximal sub-problem solved by the reverse relaxation algorithm, and the total number of sub-problems solved by the reverse relaxation algorithm.

A.2. Discrete p-center problems with Euclidean distances

Table 8 shows the performance of Ilhan et al.'s algorithm, the original relaxation algorithm, the improved relaxation algorithm with $k = 13$ and the binary relaxation algorithm with $k = 11$ on discrete p-center problems with Euclidean distances. The input files are taken from TSP-Lib [26]. Besides running times, the tables show, for each problem, the size of the maximal sub-problem solved by the binary relaxation algorithm, and the total number of sub-problems solved by the binary relaxation algorithm.

A.3. Continuous p-center problems with Euclidean distances

Table 9 shows the performance of the new relaxation algorithms for a continuous p-center problem with 439 demand points. The problem is pr439, taken from TSP-Lib [26].

References

[1] Handler GY, Mirchandani PB. Location on networks: theory and algorithms. Cambridge, MA: MIT Press; 1979.
[2] Chen R, Handler GY. Relaxation method for the solution of the minimax location–allocation problem in Euclidean space. Naval Research Logistics 1987;34:775–87.
[3] Drezner Z. The p-center problem—heuristic and optimal algorithms. Journal of Operation Research 1984;8:741–8.
[4] Minieka E. The m-center problem. SIAM Reviews 1970;12:139–40.
[5] Megiddo N, Supowit KJ. On the complexity of some common geometric location problems. SIAM Journal on Computation 1984;13:182–96.
[6] Hochbaum DS. When are NP-hard problems easy? Annals of Operations Research 1984;1:201–14.
[7] Chrystal G. On the problem to construct the minimum circle enclosing n given points in the plane. Proceedings of the Edinburgh Mathematical Society 1885;3:30–3.
[8] Rademacher H, Toeplitz O. The spanning circle of a finite set of points. In: the enjoyment of mathematics. Princeton, NJ: Princeton University Press; 1957. p. 103–10.
[9] Chen R. Solution of minisum and minimax location–allocation problems with Euclidean distances. Naval Research Logistics Quarterly 1983;30:449–59.
[10] Drezner Z. The planar two center and two median problems. Transportation Science 1984;18:351–61.
[11] Watson-Gandy CDT. The multi-facility min–max Weber problem. European Journal of Operational Research 1984;18:44–50.
[12] Toregas C, Swain R, ReVelle C, Bergmann L. The location of emergency service facilities. Operations Research 1971;19:1363–73.
[13] Hwang RZ, Lee RCT, Cheng RC. The slab dividing approach to solve the Euclidean p-center problem. Algorithmica 1993;9:1–22.
[14] Suzuki A, Drezner Z. The p-center location problem in the area. Location Science 1996;4:69–82.
[15] Wei H, Murray AT, Xiao N. Solving the continuous space p-center problem: planning applications issues. IMA Journal of Management and Mathematics 2006;17:413–25.
[16] Agarwal PK, Sharir M. Efficient algorithms for geometric optimization. ACM Computing Surveys 1998;30:428–58.
[17] Hale SH, Moberg CR. Location science research: a review. Annals of Operations Research 2004;123:21–35.
[18] Kariv O, Hakimi SL. An algorithmic approach to network location problems. Part 1: p-centers. SIAM Journal of Applied Mathematics 1971;37:513–38.
[19] Current J, Daskin M, Schilling D. Facility location: applications and theory. In: Drezner Z, Hamacher HW, editors. Discrete network location models. Berlin: Springer; 2001. p. 83–120.
[20] Daskin MS. A new approach to solving the vertex p-center problem to optimality: algorithm and computational results. Communications of the Operations Research Society of Japan 2000;45(9):428–36.
[21] Mladenović N, Labbé M, Hansen P. Solving the p-center problem with tabu search and variable neighborhood search. Networks 2003;42:48–64.
[22] Elloumi S, Labbé M, Pochet Y. A new formulation and resolution method for the p-center problem. INFORMS Journal of Computing 2004;16:84–94.
[23] Caruso C, Colorni A, Aloi L. Dominant, an algorithm for the p-center problem. European Journal of Operational Research 2003;149:53–64.
[24] Ilhan T, Özsoy FA, Pinar MC. An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems. Technical Report; 2002. Available at URL: ⟨http://www.optimization-online.org/DB_HTML/2002/12/588.html⟩.
[25] Beasley JE. A note on solving large p-median problems. European Journal of Operational Research 1985;21:270–3 ⟨http://people.brunel.ac.uk/mastjjb/jeb/orlib/pmedinfo.html⟩.
[26] Reinelt G. TSP-Lib. URL: ⟨http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html⟩.
[27] Daskin MS. Network and discrete location, models: algorithms and applications. New York: Wiley Interscience Publications, Wiley; 1995.
[28] ILOG, Inc. Gentilly, France. ILOG CPLEX 7.5 User's Manual; November 2001.
[29] Vijay J. An algorithm for the p-center problem in the plane. Transportation Science 1985;19:235–45.