European Journal of Operational Research 225 (2013) 36–43

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

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

Continuous Optimization

Optimal algorithms for the α-neighbor p-center problem

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

[a] IBM Research, Haifa University Campus, Mount Carmel, Haifa 31905, Israel
[b] Raymond and Beverly Sackler, School of Physics and Astronomy, Tel-Aviv University, Tel-Aviv 69978, Israel

ARTICLE INFO

Article history:
Received 4 December 2011
Accepted 22 September 2012
Available online 2 October 2012

Keywords:
α-Neighbor p-center
Continuous optimization
Discrete optimization
Relaxation

ABSTRACT

Assigning multiple service facilities to demand points is important when demand points are required to withstand service facility failures. Such failures may result from a multitude of causes, ranging from technical difficulties to natural disasters. The α-neighbor p-center problem deals with locating p service facilities. Each demand point is assigned to its nearest α service facilities, thus it is able to withstand up to $\alpha - 1$ service facility failures. The objective is to minimize the maximum distance between a demand point and its αth nearest service facility. We present two optimal algorithms for both the continuous and discrete α-neighbor p-center problem. We present experimental results comparing the performance of the two optimal algorithms for $\alpha = 2$. We also present experimental results showing the performance of the relaxation algorithm for $\alpha = 1, 2, 3$.

1. Introduction

The α-neighbor p-center problem, presented by Krumke [17], is a generalization of the p-center problem (see, for example, [12]). Given the locations of n demand points, the objective in the classic p-center problem is to locate p service facilities so as to minimize the maximum distance between a demand point and its nearest service facility. In the α-neighbor p-center generalization, each demand point is assigned α service facilities, so that each demand point could withstand the failure of $\alpha - 1$ service facilities; thus the objective is to minimize the maximum distance between a demand point and its αth nearest service facility. It is assumed that all service facilities perform the same kind of service, and that the number of demand points that can get service from a given center is unlimited.

There are two main variants to the classic p-center problem. In the *continuous* p-center problem the service facilities may be located anywhere on the plane. In the *discrete* p-center problem there is a finite set of potential service points to choose from. Research on the α-neighbor p-center problem has so far concentrated on approximation algorithms for the discrete cases, where the set of potential service points is the same as the set of demand points [1,16,17].

The α-neighbor p-center problem (like the classic p-center problem) is a problem of locating p service centers. It can also be viewed as a problem of locating p circles so that each demand point is covered α times, where the centers of the p circles are the locations of the p service facilities. The goal is to minimize the radius of the maximal circle. For convenience, in this paper we treat the α-neighbor p-center as a problem of locating p circles.

Following the work by Toregas et al. [25] which dealt with single coverage in location of emergency service facilities problems, several authors studied different versions of problems with multiple coverage of demand points. Daskin and Stern [8] considered a hierarchical objective set covering (HOSC) problem: finding the minimum number of vehicles needed to cover all the zones while simultaneously maximizing the extent of multiple coverage of zones. This was followed by a work by Daskin [5] who extended the maximum covering location model to account for the chance that when demand arrives at the system, it will not be covered since all facilities capable of covering the demand are engaged serving other demands. Daskin presented a heuristic solution algorithm dealing with the location problem on networks. Hogan and ReVelle [13] discussed the problem of backup coverage, namely the second coverage of a demand node on a network. More precisely, first coverage is traded off against backup coverage. Using an example problem, they showed that significant levels of backup coverage may be provided in a system without substantial loss of first coverage.

ReVelle and Hogan [21] introduced the maximum available location problem (MALP) which positions p servers in such a way as to maximize the population which will find an available server within a time standard with α reliability. The authors point out that the MALP builds on the probabilistic location set-covering problem in concept, and on backup covering and expected covering models in technical detail. Marianov and ReVelle [18] suggested a more realistic model for emergency systems, namely, the Queueing-MALP or Q-MALP. Whereas in MALP, it is assumed that the probabilities of different servers being busy are independent,

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

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.09.041

in Q-MALP, results from queueing theory are utilized to relax this assumption.

Khuller et al. [16] provided a polynomial time approximation algorithm for the α-neighbor p-center problem that achieved an approximation factor of 3 for any α, and an approximation factor of 2 for $\alpha < 4$. Guha et al. [11] considered a similar problem, the fault-tolerant location problem on networks; they discussed the factor approximation of the fault-tolerant location problem and provided a greedy local-search technique that yields an approximation ratio of 2.41. A further improvement by Swamy and Shmoys [24] showed that there is a 2.076-approximation of the problem. Church and Gerrard [4] considered the multi-level location set-covering model (ML-LSCP) as a search for the smallest number of facilities needed to cover each demand a preset number of times. In their version of the problem, one has a number of potential sites that can be used for the placement of a facility. These may be nodes of a network or discrete points on a continuous terrain. Eiselt and Marianov [9] presented a practical application of the solution of this problem, namely, a mobile phone tower location for survival after natural disasters. Their work was motivated by the aftermath of a magnitude 8.8 earthquake that had hit Chile in 2010. One outcome was that cellular phones stopped working in large areas for a long time, and the lack of communication had a distressing effect. Eiselt and Marianov assumed that in case of a disaster, there will be exactly one failure. The reason for the assumption is that radio base stations are usually separated by a significant distance so that natural disasters such as landslides are unlikely to affect more than one at a time. They also suggest that, according to past experience, terrorists always strike one target at a time. Thus, the double cover is a sufficient demand. However, Eiselt and Marianov [9] did not solve the same problem of a full double cover which is the subject of the present work, but rather concentrated on minimization of the worst-case loss using constant-radius circles.

Some works on assigning multiple service facilities have allowed locating several service points at the exact same location, while others have not. In Church and Gerrard [4], it is assumed that the facilities cannot be co-located and must be distributed at most one per site. This is the case, for example, in the problem discussed by Eiselt and Marianov [9] dealing with phone tower survival after natural disasters. On the other hand, the possibility of co-location is allowed in other cases. Hogan and ReVelle [13] demonstrated that when backup coverage is employed as one of the primary objectives in a problem, co-location of facilities may result. In an extended review report by Serra and Marianov [22], different aspects of co-location are discussed, along with several references. In the present paper we address both problems, with and without co-location. We concentrate mainly on the α-neighbor p-center problem where co-location of multiple service points is allowed. In another review paper, Goldberg [10] discusses models for the deployment of emergency service vehicles. Among other subjects, he reviews the literature concerning models that incorporate backup coverage.

In this paper we present an optimal algorithm for both the continuous and discrete α-neighbor p-center problem. To the best of our knowledge, no optimal algorithm has previously been suggested for the α-neighbor p-center problem, and the continuous variant of the problem has never been considered.

2. Optimal algorithm

2.1. Minieka's algorithm

Minieka [19] provided an optimal algorithm for the p-center problem on a graph. In this variant of the p-center problem, the service points could be placed anywhere on the graph edges and vertices. Minieka's algorithm has also been modified to solve the discrete and continuous p-center problem.

Algorithm 1. Skeleton of Minieka's Algorithm

```
Feasible ← RandomlyChoosePServicePoints()
Best_Candidate ← Feasible
Upper_Bound ← GetValue(Feasible)
while (true)
  Feasible ← FindBetterSolution(Upper_Bound)
  if (no feasible solution found)
    halt and return Best_Candidate
  else
    Best_Candidate ← Feasible
    Upper_Bound ← GetValue(Feasible)
```

The basic idea for Minieka's algorithm is very simple, and is represented in Algorithm 1. One begins with an arbitrary set of p service points, and at each iteration solves a subroutine which attempts to find a better solution to the p-center problem. The algorithm halts when no better solution is found.

Function GetValue returns the value of a feasible solution, which in the p-center case is the maximum distance between a demand point and its nearest service facility.

The heart of Minieka's algorithm is subroutine FindBetterSolution, which, given an upper bound, is able to determine whether there exists a solution to the p-center problem with value better than that upper bound. The subroutine also finds a better solution if one exists. Subroutine FindBetterSolution, which we will describe in further detail, solves a set-covering problem. The ability to translate the problem of finding a better p-center solution to a set-covering problem depends on there being a finite number of circles. Minieka [19] shows that is the case for the p-center problem on graphs.[1]

We now describe how subroutine FindBetterSolution is implemented:

- Input: n demand points, a value p, and a value *Upper_Bound*
- Output: a set of p circles with radii less than *Upper_Bound* that cover each demand point (if such a set exists).
  - (1) Construct C, the finite set of circles induced by the n demand points. Note: C need not be constructed each time subroutine FindBetterSolution is called, for C is constant for any set of n demand points.
  - (2) Construct C', the set of all circles in C with radii less than *Upper_Bound*. C' is constructed by discarding from C all circles with radii greater or equal to the current upper bound; these circles cannot be used in a solution with value less than *Upper_Bound*.
  - (3) Determine whether there is a subset of at most p circles that cover all demand points. To do this, we:
    - (a) Assign one size-n vector, v_c, to each circle c in C'. For each circle c, the value of v_c in the position corresponding to demand point i is 1 if and only if point i is covered by circle c:

$$v_c(i) = \begin{cases} 1 & \text{circle } c \text{ covers point } i \\ 0 & \text{otherwise} \end{cases}.$$

    - (b) Run a set-covering algorithm on this set of vectors to obtain $V_{\min}$, a minimal set of vectors whose sum is

[1] More precisely, Minieka shows that there is a finite number of service points to consider, but the transition to circles is immediate.

greater than the 1-vector (the size-n vector which consists only of 1's). If $V_{\min}$ contains no more than p vectors, return the corresponding set of circles.

In step 3b, determining whether there are at most p vectors whose sum is greater than the 1-vector is equivalent to determining whether there are at most p circles that cover all demand points.

Minieka's algorithm [19] is a sequence of set-covering problems; each step solves a set-covering problem to find $V_{\min}$. If $V_{\min}$ contains no more than p vectors, then the corresponding set of circles is a better solution to the p-center problem, as they cover all demand points, and their radii are smaller than the current upper bound. If $V_{\min}$ contains more than p vectors, then there can be no better solution to the p-center problem, and we can halt.

Note that solving the set-covering problem to optimality does not guarantee that we solved the p-center problem to optimality. Though we only consider circles with radii smaller than the current upper bound, when solving the set-covering problem we completely ignore the radii of the candidate circles.

Step 3b in FindBetterSolution can be replaced by a feasibility problem, as suggested by Ilhan et al. [15]. Instead of solving "find the minimal number of vectors whose sum is greater than the 1-vector", Ilhan et al. suggest solving a slightly easier problem, "find a subset of at most p vectors whose sum is greater than the 1-vector (if such a subset exists)". This *feasibility sub-problem*, like the set-covering problem, is NP-hard, but it has been experimentally shown that replacing the set-covering algorithm with the feasibility sub-problem algorithm improves the performance of the p-center algorithm [2].

Both the set-covering problem and the feasibility sub-problem can easily be represented as Integer Programming problems, the former an optimization Integer Programming problem, the latter a feasibility Integer Programming problem. Both problems can be solved by any integer programming software. The difference between the set-covering problem and the feasibility sub-problem is that in the set-covering problem one tries to minimize the number of vectors; in the feasibility sub-problem there is no function to minimize, but there is one extra constraint which limits the number of vectors to p.

Although Minieka's algorithm was written for a variant of the p-center problem where the service points could be placed anywhere on the graph edges and vertices, it also applies to the discrete and continuous p-center problems. In order to apply Minieka's algorithm to the discrete and continuous p-center problems, one needs there to be a finite set of circles to consider. As in the version of the p-center problem solved by Minieka, the set of circles to consider for the discrete and continuous p-center problems is the set of *critical* circles. Intuitively, a circle is critical if it cannot be shrinked, even slightly, without hurting the coverage of at least one demand point. For the discrete p-center problem, one needs only to consider circles whose centers reside at the potential service points and whose radii are determined by one of the demand points. Chen and Handler [3] show that in the case of the continuous p-center problem on a two-dimensional plane, given n demand points, there are $\binom{n}{3}+\binom{n}{2}+n$ critical circles to consider. The circles to consider are $\binom{n}{3}$ circles determined by three points on their circumference, $\binom{n}{2}$ circles defined by two points determining the diameter and n null circles located at the demand points. Chen and Handler [3] also show that of the circles determined by three demand points, the circles determined by three points forming an obtuse triangle may be disregarded.

2.2. Adjustment for the α-neighbor p-center problem

We explain how to adjust Minieka's algorithm to solve both the continuous and discrete α-neighbor p-center problems. Algorithm 1 clearly also applies to the α-neighbor p-center problems, but the implementations of FindBetterSolution and GetValue need to change. We concentrate on the α-neighbor p-center problem where co-location of multiple service points is allowed, but we also address the other problem, where co-location is forbidden.

In subroutine FindBetterSolution, the set of circles to consider for the α-neighbor p-center problem is the same set of *critical* circles as in the classic p-center problem. Since we work with the same set of circles, we also use the same set of vectors as in the p-center problem. However, rather than finding the minimal number of vectors whose sum is greater than the 1-vector, we now attempt to find the minimal number of vectors whose sum is greater than the α-vector (the size-n vector which consists only of α's). This is equivalent to finding the minimal set of circles that cover each demand point at least α times. This is exactly the redundant set-covering problem, presented and solved by Van Slyke [23], with the redundancy level set to α. Note that for the redundant set-covering problem, several service facilities could be located at the exact same position, which was not possible for the classic p-center problem. This means that if, for the p-center problem, we programmed our integer programming software to limit the integer values of our variables to be either 0 or 1, we must remove that constraint for the α-neighbor p-center problem. (If co-location of service facilities is not allowed, then the binary constraint should remain.)

Inspired by the improvement suggested by Ilhan et al. [15] for the original p-center problem, we propose implementing FindBetterSolution as the feasibility Integer Programming problem described in Section 2.1, where the 1-vector is replaced by the α-vector.

Function GetValue returns the value of a feasible solution. When adapted to the α-neighbor p-center problem, it computes the maximum distance between a demand point and its αth nearest service facility, rather than the maximum distance between a demand point and its nearest service facility.

3. Relaxation algorithms

3.1. Relaxation algorithm for the p-center problem

Relaxation (in the context of this paper) [3,12] is a method for *optimally* solving a large location problem by solving a succession of small sub-problems. The classic relaxation algorithm [3,12] starts with an upper bound of infinity, and keeps updating it until the optimal value is reached, similarly to Minieka's algorithm [19] (though Minieka's algorithm does not involve relaxation). 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 many heuristic methods which usually yield local minima.

Relaxation algorithms are 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. Like Minieka's algorithm, we keep searching for a feasible solution which improves upon the current candidate solution until we prove that none exists.

Relaxation applies to a location problem if it has the following properties:

- (Property 1) There must be an algorithm capable of answering the question: "is there a solution to the problem with value better than x", and of finding such a solution if one exists
- (Property 2) The problem must be such that if there is no solution to a sub-problem with value better than x, then there can be no solution to the full problem with value better than x.

Lemma 1. *The p-center problem satisfies Properties 1 and 2.*

Proof. Property 1 is satisfied, since FindFeasibleSolution, as described in Section 2.1, identifies a better solution if one exists. Property 2 is also satisfied, as any set of circles that covers all demand points also covers any subset of the demand points. Therefore any feasible solution to the full problem is also a feasible solution to any sub-problem. As a result, the value of the optimal solution to the full problem must be greater or equal to the value of the optimal solution to any sub-problem. Suppose that there is no solution to a sub-problem with value better than x. Assume to the contrary that there exists a feasible solution to the full problem with value better than x. Then this must also be a feasible solution to the sub-problem. Therefore, there exists a solution to the sub-problem with value better than x, a contradiction. □

We now describe the relaxation algorithm in greater detail, and show why it halts with the optimal solution for any location problem with the two above properties.

Algorithm 2. Skeleton of Relaxation Algorithms.

```
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
      AddDemandPoints(Sub)
```

Algorithm 2 describes the skeleton of a relaxation algorithm. We begin with an upper bound of infinity. ChooseRandomSubset chooses an initial random subset of the demand points. The algorithm does not state how to choose the initial random subset, nor its initial size, but the algorithm's correctness does not rely on that choice.

At each iteration we work with a subset of the demand points, and run FindFeasibleSolution. Its inputs are *Sub*, the current demand-point subset, and a value *Upper_Bound*, the current upper bound. The subroutine answers the question: "Is there a solution to the sub-problem with value less than *Upper_Bound*?". In other words, it finds a feasible (not necessarily optimal) solution to the relaxed (smaller) p-center problem, with value less than *Upper_Bound* (if such a solution exists).

If FindFeasibleSolution finds a better solution to the sub-problem, we check whether this solution also happens to be a solution to the full problem. For the p-center problem, for instance, we check whether the set of circles which cover the subset of demand points, also happens to cover all demand points. If the found solution is also a feasible solution to the full problem, then it is better than the best solution we have found so far; we update *Best_Candidate* to be the set of service points returned by the subroutine, and we update *Upper_Bound* to be the value of this new solution.

If, on the other hand, the solution returned by FindFeasibleSolution is not a feasible solution to the original problem, then we have not learned much from this iteration. We call such steps "uninformative", since they do not help us improve our bound on the solution. When this happens, we simply add more demand points to our subset, and try again. The algorithm does not state how to add a demand point to the subset in the uninformative steps, and the correctness of the algorithm does not rely on that choice. The algorithm remains correct even if we add more than one demand point following each uninformative step [2]; in fact, this may help reduce the overall number of uninformative steps. A good heuristic for adding k demand points, for the classic p-center problem, is to add the k farthest demand points from the current service points.

If FindFeasibleSolution proves that there is no solution to the sub-problem with value less than *Upper_Bound*, then the algorithm halts and returns the best feasible solution we have found so far, *Best_Candidate*.

Lemma 2. *For any problem satisfying Property 2, if Algorithm 2 halts, it return the optimal solution.*

Proof. At the point where Algorithm 2 halts, we have already found a feasible solution to the full problem with value *Upper_Bound*. We halt when we find no solution to a sub-problem with value less than *Upper_Bound*. From Property 2 it follows that there can be no solution to the full problem with value less than *Upper_Bound*. Therefore, the feasible solution we have found, with value *Upper_Bound*, is optimal. □

We have shown that if Algorithm 2 halts, then it must return the optimal solution. Is it possible that the algorithm will never halt? The algorithm must always halt as there is a limit to the number of uninformative steps. Whenever there is an uninformative step, we increase the size of the sub-problem. In the worst case, at a certain point the sub-problem will reach the size of the full problem, i.e. *Sub* will become the set of all demand points. When that happens, there can be no more uninformative steps. There can be no set of circles which are a feasible solution to the sub-problem but not a feasible solution to the full problem, when the sub-problem and full problem are the same.

3.2. Relaxation algorithm for the α-neighbor p-center problem

In this section we show that the relaxation algorithm also applies to the α-neighbor p-center problem.

Lemma 3. *The α-neighbor p-center problem satisfies Properties 1 and 2.*

Proof. Property 1 is satisfied, since FindFeasibleSolution, as described in Section 2.2, identifies a better solution if one exists. Property 2 is also satisfied; given a set of circles, if all demand points are covered by at least α circles, then this must also be true for any subset of the demand points. It follows that the value of the optimal solution to the full problem must be greater or equal to the value of the optimal solution to any sub-problem. Suppose that there is no solution to a sub-problem with value better than x. As before, we assume to the contrary that there exists a feasible solution to the full problem with value better than x. Then this must also be a feasible solution to the sub-problem. Therefore, there exists a solution to the vsub-problem with value better than x, a contradiction. □

Satisfying the two properties ensures that the relaxation algorithm halts and returns the optimal solution.

Here is a list of changes to the relaxation algorithm for the classic p-center problem, so as to adjust it to the α-neighbor p-center problem:

(1) In the implementation of FindFeasibleSolution, we replace an algorithm for the set-covering problem with an algorithm for the Van Slyke's redundant set-covering problem [23] (or rather its feasibility sub-problem counterpart).
(2) We change the implementation of AddDemandPoints. As we mentioned in Section 3.1, the relaxation algorithm does not state how to add demand points to the subset in the uninformative steps, and the correctness of the algorithm does not rely on the choice. For the classic p-center problem, a good heuristic is to add the k farthest demand points from the current service points. The equivalent for the α-neighbor p-center problem is to choose the k demand points such that their αth nearest service points are the farthest.
(3) In function GetValue, which returns the value of a feasible solution, we compute the maximum distance between a demand point and its αth nearest service facility, rather than the maximum distance between a demand point and its nearest service facility.
(4) We replace the algorithm for checking whether a feasible solution to the sub-problem is also a feasible solution to the original full problem. For the α-neighbor p-center problem, we need to take into account the possibility that several service facilities are located at the exact same position, which was not possible for the classic p-center problem (this is true only if co-location of service facilities is allowed).

3.3. Is relaxation suitable for the α-neighbor p-center problem?

Just because the relaxation algorithm yields an optimal solution, does not guarantee that it does so efficiently. Fortunately, the α-neighbor p-center, like the classic p-center problem, has a property that suggests that relaxation may be suitable for it: as the upper bound improves, the sub-problems become easier to solve. As the relaxation algorithm runs, the size of the sub-problems, in terms of number of demand points, increases. At the same time, as the upper bound improves, one can discard all of the circles with radius above the upper bound. As we have mentioned, there is one vector for each circle, which means that discarding many circles significantly reduces the number of vectors, and could significantly reduce the amount of time it takes to solve the set-covering problem or feasibility sub-problem.

It is important to note that the algorithm we presented in Section 2.2, like Minieka's algorithm for the classic p-center problem, also has the property that as the upper bound improves, the number of vectors significantly decreases. The advantage of relaxation is that one can typically get a good upper bound quickly, while the sub-problems solved are still relatively small (in terms of the number of demand points).

4. Experimental results

4.1. Methodology

We compare the optimal algorithm presented in Section 2 with the relaxation algorithm presented in Section 3. We also present experimental results of the relaxation algorithms for $\alpha = 1, 2, 3$. We run the *continuous* version of our algorithms, on problems taken from TSP-Lib [20].

4.2. Experimental setup

The experiments were conducted on a computer with an Intel Core i7-2600 4-core 3.4 giga hertz CPU. The computer runs Ubuntu 10.04 LTS. The code is written in C and compiled using gcc 4.4.3. We used CPLEX version 12.1 [14] for the solution of the set-covering problems and the feasibility sub-problems. CPLEX implements optimizers based on the simplex algorithms.

4.3. Relaxation vs. non-relaxation algorithms

Non-relaxation algorithms have an advantage over relaxation algorithms in that they typically complete in fewer iterations, since they have no "uninformative" steps (steps which do not improve the bound on the solution). However, relaxation algorithms have a significant advantage over non-relaxation algorithms in that they typically reach a good bound on the solution quickly, while still solving relatively small sub-problems. Once a tight bound on the solution is reached, solving both relaxation and non-relaxation algorithms typically become much easier. This is certainly true for the classic p-center and the α-neighbor p-center problems, as explained in Section 3.3.

We compared the performance of our algorithms for the 2-neighbor p-center problem on problems att48, eil101, and ch150, taken from TSP-Lib [20].

Each row in Table 1 contains the name of the problem (att48, eil101, or ch150), the size of the problem (number of demand points in the full problem), the p-value, and the value of the optimal solution (column *obj*). Each row also contains the amount of time the non-relaxation algorithm ran and amount of time the relaxation algorithm ran. We show the total number of sub-problems (subroutine FindBetterSolution for the non-relaxation case and subroutine FindFeasibleSolution for the relaxation case) that each algorithm ran until reaching the optimal solution (columns *non-relaxation num steps* and *relaxation num steps*). For the relaxation algorithm we also present the maximal size of a sub-problem, in terms of number of demand points, that the relaxation algorithm had to solve (column *relaxation max prob*).

As Table 1 shows, the relaxation algorithm performs much better than the non-relaxation algorithm, and the advantage of relaxation becomes more pronounced as the number of demand points increases.

The relaxation algorithm accepts a parameter k, which is the maximal number of demand points that we add to the sub-problem following an uninformative step. We present the performance of the relaxation algorithm for $k = 2$, which for these problems produced the best results.

For problem d493 from TSP-Lib, the relaxation algorithm reached the optimal result after 0.37 second for $p = 10$, while the non-relaxation algorithm failed to complete even a single iteration in well over an hour.

4.4. Classic p-center vs. 2-neighbor p-center problems

We compare the performance of the relaxation algorithm for the classic p-center problem to the performance of the relaxation algorithm for the 2-neighbor p-center problem. Table 2 shows the performance of the relaxation algorithms for the classic continuous p-center problem and the continuous α-neighbor p-center problem on problem pr439 taken from TSP-Lib [20]. We present the performance of the relaxation algorithms with parameter k set to 1, which for these problems produced the best results.

For p values 10, 20, 30 and 40, the relaxation algorithm for the 2-neighbor p-center problem took less time to run than the relaxation algorithm for the classic p-center. For p values 50, 60, 70, 80, 90 and 100, the algorithm for the classic p-center problem was faster. For $p = 90$, the relaxation algorithm for the 2-neighbor p-center problem took over 18 times more time than the classic p-center equivalent. The differences in the performance of the relaxation algorithms have a lot to do with the solution value. For the same input and the same p

Table 1
The performance of the non-relaxation and relaxation algorithms for the 2-neighbor p-center on continuous problems with different values of p. For both algorithms we present the total number of relaxation sub-problems solved. For the relaxation algorithm we also present the maximal size of a relaxation sub-problem.

Input	n	p	Obj.	Non-relaxation (seconds)	Non-relaxation num. steps	Relaxation (seconds)	Relaxation max. prob.	Relaxation num. steps
att48	48	10	1377.534392	0.54	31	0.02	22	42
att48	48	20	820.451857	0.37	52	0.08	40	77
att48	48	30	601.592054	0.29	58	0.10	43	98
att48	48	40	474.647237	0.27	65	0.08	45	92
eil101	101	10	19.455076	28.90	34	0.13	38	55
eil101	101	20	12.138497	32.43	64	16.76	84	138
eil101	101	30	9.219544	26.54	87	10.51	97	143
eil101	101	40	7.532275	21.25	98	9.29	99	189
eil101	101	50	6.363961	10.41	130	5.22	101	206
eil101	101	60	5.758756	7.30	109	3.09	101	201
eil101	101	70	5.000000	5.84	108	1.02	99	188
eil101	101	80	4.527693	5.73	125	0.76	101	198
eil101	101	90	4.031129	5.60	118	0.69	101	196
eil101	101	100	3.640055	5.39	107	0.55	101	176
ch150	150	10	184.060380	94.52	38	0.17	46	59
ch150	150	20	122.551815	196.38	70	29.29	107	150
ch150	150	30	93.976758	84.19	96	27.98	128	170
ch150	150	40	76.384585	93.95	116	30.40	139	205
ch150	150	50	65.424245	57.72	136	23.77	146	263
ch150	150	60	55.716472	38.93	170	4.52	144	269
ch150	150	70	50.776950	31.71	153	3.35	146	294
ch150	150	80	47.410802	31.17	175	3.58	150	303
ch150	150	90	41.276098	30.92	187	2.97	148	324
ch150	150	100	38.005058	30.66	204	2.60	149	306
ch150	150	110	33.236117	30.56	215	2.49	148	332
ch150	150	120	31.984077	30.48	217	2.63	147	359
ch150	150	130	29.554260	31.59	207	2.40	148	343
ch150	150	140	27.063553	30.05	203	2.51	147	353

Table 2
The performance of the relaxation algorithm for the classic continuous p-center problem (a), and the relaxation algorithm for the continuous 2-neighbor p-center problem (b) on the pr439 problem from TSP-Lib, with different values of p. The two rightmost columns present the maximal size of a relaxation sub-problem and the total number of relaxation sub-problems solved.

Input	n	p	Obj.	Relaxation (seconds)	Relaxation max. prob.	Relaxation num. steps
(a)						
pr439	439	10	1716.509904	6.30	120	177
pr439	439	20	1029.714766	28.73	187	305
pr439	439	30	739.192972	101.37	254	393
pr439	439	40	580.005388	133.26	316	510
pr439	439	50	468.541620	294.62	353	583
pr439	439	60	400.195265	192.85	376	664
pr439	439	70	357.945527	116.58	382	658
pr439	439	80	312.500000	120.87	401	696
pr439	439	90	280.902563	95.83	403	724
pr439	439	100	256.680194	101.44	412	778
(b)						
pr439	439	10	2752.638635	0.37	52	82
pr439	439	20	1716.509904	3.04	108	162
pr439	439	30	1271.829545	18.32	158	237
pr439	439	40	1008.169753	27.20	196	314
pr439	439	50	874.270557	605.65	250	389
pr439	439	60	739.192972	978.71	260	404
pr439	439	70	621.741506	1888.61	306	493
pr439	439	80	580.005388	1576.88	322	515
pr439	439	90	530.477379	1737.54	341	565
pr439	439	100	463.175183	1443.72	352	587

value, the solution to the 2-neighbor p-center problem must be greater or equal to the solution to classic p-center problem (since any feasible solution to the 2-neighbor p-center problem is also a feasible solution to the classic p-center problem). On the one hand a higher optimal solution value may result in fewer iterations, which helps reduce running time. Fewer steps also suggest fewer uninformative steps, which means that there are fewer times in which we need to add demand points to the sub-problem; this leads to smaller sub-problems, which also helps reduce running time. On the other hand, a higher optimal solution means that as the algorithm runs, the upper bounds are typically greater, and there are less circles to discard. This leads to more vectors in the set-covering or the feasibility sub-problems, and to poorer performance. For the more difficult problems, the cost of having a higher solution value far outweighs the benefit.

Note that the solutions to the classic p-center problem with p values 10, 30, and 40, are the same as the solutions to the 2-neighbor p-center problem with p values 20, 60, and 80 (respectively). In these cases, the solution to the 2-neighbor p-center problem is the same as the solution to the respective classic p-center problem, where each service point is replaced by two service points at the

exact same location. This is possible since we ran the version of the algorithm which allowed co-location of service points.

Note also that solving the classic p-center problem for some value $p = p'$, often takes much less time than solving the 2-neighbor p-center problem for $p = 2p'$. This suggests a relatively quick way to obtain a good upper bound on a 2-neighbor p-center problem for $p = 2p'$. We simply run the classic p-center problem with $p = p'$, and replace each service point with two service points at the same location. This is clearly a feasible (and as we have shown, occasionally the optimal) solution to the 2-neighbor p-center problem. Good upper bounds on the optimal solution are extremely useful in speeding up iterative algorithms such as the ones we present in Sections 2.2 and 3.1.

4.5. Different values of k

One of the parameters for the relaxation algorithm is k, the maximal number of demand points that we add to the sub-problem following an uninformative step.

Given a feasible solution to the current sub-problem, the heuristic we use for adding demand points to the sub-problem does the following:

(1) Checks whether all demand points of the full problem are covered by the feasible solution.
(2) If there are at most k demand points not covered, adds them all.
(3) If there are more than k demand points, adds the k demand points such that their αth nearest service points are the farthest.

In choosing k there is a tradeoff between the number of iterations, and the difficulty of the iterations. If the value chosen for k is low, then one may have many uninformative steps; this will increase the overall number of iterations. If the value chosen for k is high, then the smaller sub-problems would quickly (after only a few uninformative steps) cease to be particularly small. The "right" value for k depends on which is more expensive: solving many smaller problems, or solving fewer larger problems. This computational price depends greatly on the problem one is trying to solve. It is our experience that for the harder problems, choosing a low value for k is better. Problems with many demand points tend to be harder than problems with fewer demand points. Continuous problems are often much harder than discrete problems.

Fig. 1 shows the performance of relaxation algorithms for the α-neighbor p-center problem for different values of $k (k = 1, 2, \ldots, 8)$ and for $\alpha = 1, 2, 3$. We ran the relaxation algorithms on problem pr439 taken from TSP-Lib with $p = 10, 20, \ldots, 100$. For each value of α and k, we averaged the results of the 10 runs (for $p = 10, 20, \ldots, 100$). Fig. 1 shows that for the pr439 problem, the best performance is achieved for $k = 1$ (for $\alpha = 1, 2, 3$). It also shows that as the k value increases, the number of iterations tends to go down, while the average size (in terms of demand points) of the maximal sub-problem tends to go up.

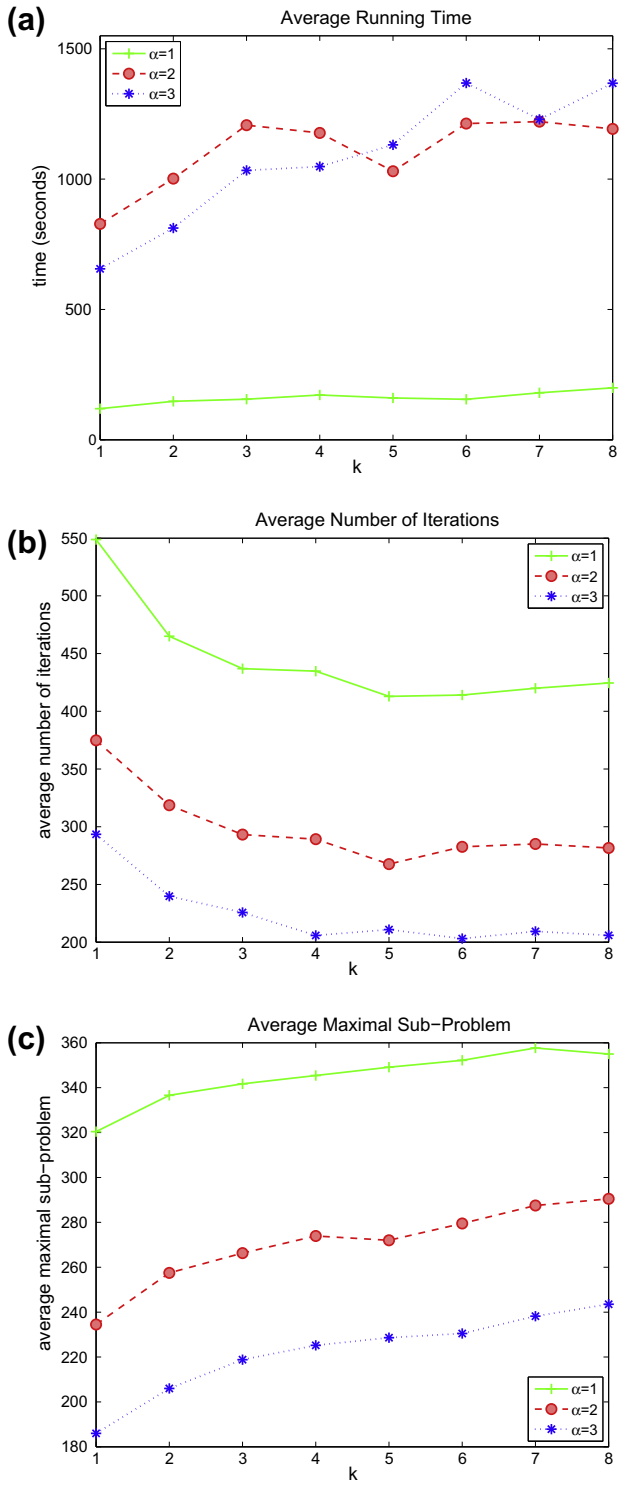

Fig. 1. Performance of relaxation algorithms for the α-neighbor p-center problem for different values of k and for $\alpha = 1, 2, 3$. We ran the relaxation algorithms on problem pr439 taken from TSP-Lib with $p = 10, 20, \ldots, 100$. We show (a) the average running time, (b) the average number of iterations, and (c) the average size (in terms of demand points) of the maximal sub-problem.

5. Conclusions and future work

We present two new algorithms for the α-neighbor p-center problem, one of which is a relaxation algorithm. We experimentally show that the relaxation algorithm is more efficient, and that the advantage of relaxation becomes more pronounced as the number of demand points increases.

The algorithms we present are variations of algorithms for the classic p-center problem, customized for the α-neighbor p-center problem. The two algorithms are Minieka's algorithm [19] and a relaxation algorithm [3,12]. It would be interesting to test the performance of variations of other algorithms for the p-center problem, such as Daskin's algorithm [6,7], which performs a binary search for the optimal solution, and other relaxation algorithms [2].

Acknowledgements

The authors thank the two anonymous referees for their valuable comments.

References

[1] Shiva Chaudhuri, Naveen Garg, R. Ravi, The *p*-neighbor *k*-center problem, Information Processing Letters 65 (February) (1998) 131–134.
[2] Doron Chen, Reuven Chen, New relaxation-based optimal algorithms for the solution of the continuous and discrete *p*-center problems, Computers & Operations Research 36 (2009) 1646–1655.
[3] Reuven Chen, Gabriel Y. Handler, Relaxation method for the solution of the minimax location–allocation problem in Euclidean space, Naval Research Logistics 34 (1987) 775–787.
[4] Richard L. Church, Ross A. Gerrard, The multi-level location set covering model, Geographical Analysis 35 (2003) 277–289.
[5] Mark S. Daskin, A maximum expected covering location model, Transportation Science 17 (1983) 48–70.
[6] Mark S. Daskin, Network and Discrete Location, Models: Algorithms and Applications, Wiley Interscience Pub., John Wiley and Sons Inc., New-York, 1995.
[7] Mark S. Daskin, A new approach to solving the vertex *p*-center problem to optimality: algorithm and computational results, Communications of the Operations Research Society of Japan 45 (9) (2000) 428–436.
[8] Mark S. Daskin, Edmund H. Stern, A hierarchical objective set covering model for emergency medical service vehicle deployment, Transportation Science 15 (1981) 137–152.
[9] H.A. Eiselt, Vladimir Marianov, Mobile phone tower location for survival after natural disasters, European Journal of Operational Research 216 (2012) 563–572.
[10] Jeffrey B. Goldberg, Operations research models for the deployment of emergency services vehicles, EMS Management Journal 1 (1) (2004) 20–39.
[11] Supidto Guha, Adam Meyerson, Kamesh Munagala, A constant factor approximation algorithm for the fault-tolerant facility location problem, Journal of Algorithms 48 (2003) 429–440.
[12] Gabriel Y. Handler, Pitu B. Mirchandani, Location on Networks: Theory and Algorithms, MIT Press, Cambridge, MA, 1979.
[13] Kathleen Hogan, Charles ReVelle, Concepts and applications of backup coverage, Management Science 32 (1986) 1434–1444.
[14] IBM. IBM ILOG CPLEX V12.1 – User's Manual for CPLEX, 2009.
[15] T. Ilhan, F.A. Özsoy, M.C. Pinar, An Efficient Exact Algorithm for the Vertex *p*-Center Problem and Computational Experiments for Different Set Covering Subproblems, Technical report, 2002 <http://www.optimization-online.org/DB_HTML/2002/12/588.html>.
[16] Samir Khuller, Robert Pless, Yoram J. Sussmann, Fault tolerant *k*-center problems, Theoretical Computer Science 242 (July) (2000) 237–245.
[17] S.O. Krumke, On a generalization of the *p*-center problem, Information Processing Letters 56 (October) (1995) 67–71.
[18] Vladimir Marianov, Charles ReVelle, The queueing maximal availability location problem: a model for the siting of emergency vehicles, European Journal of Operational Research 93 (1996) 110–120.
[19] Edward Minieka, The *m*-center problem, SIAM Review 12 (1970) 139–140.
[20] Gerhard Reinelt, TSP-Lib <http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html>.
[21] Charles ReVelle, Kathleen Hogan, The maximum availability location problem, Transportation Science 23 (1989) 192–200.
[22] Daniel Serra, Vladimir Marianov, New Trends in Public Facility Location Modeling, Technical report, May 2004. UPF Economics and Business Working Paper No. 755 <http://ssrn.com/abstract=563843> or http://dx.doi.org/10.2139/ssrn.563843.
[23] Robert Van Slyke, Redundant set covering in telecommunications networks, in: Proceedings IEEE Large Scale Systems Symposium, Virginia Beach, Va, October 1982, pp. 217–222.
[24] Chaitanya Swamy, David B. Shmoys, Fault-tolerant facility location, Combinatorical Optimization 21 (2006) 735–736.
[25] Constantine Toregas, Ralph Swain, Charles ReVelle, Lawrence Bergman, The location of emergency facilities, Operations Research 19 (1971) 1363–1373.