IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 24, 2019, accepted August 1, 2019, date of publication August 8, 2019, date of current version August 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2933875

Approximation Algorithms for the Vertex K-Center Problem: Survey and Experimental Evaluation

JESUS GARCIA-DIAZ[1,2], ROLANDO MENCHACA-MENDEZ[3], RICARDO MENCHACA-MENDEZ[3], SAÚL POMARES HERNÁNDEZ[2,4], JULIO CÉSAR PÉREZ-SANSALVADOR[1,2], AND NOUREDDINE LAKOUARI[1,2]

[1]Consejo Nacional de Ciencia y Tecnología, CDMX 03940, Mexico
[2]Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla 72840, Mexico
[3]Centro de Investigación en Computación, CDMX, Mexico 07738, Mexico
[4]CNRS, LAAS, 31400 Toulouse, France

Corresponding author: Jesus Garcia-Diaz (jesgadiaz@inaoep.mx)

ABSTRACT The vertex k-center problem is a classical NP-Hard optimization problem with application to Facility Location and Clustering among others. This problem consists in finding a subset $C \subseteq V$ of an input graph $G = (V, E)$, such that the distance from the farthest vertex in V to its nearest center in C is minimized, where $|C| \leq k$, with $k \in Z^+$ as part of the input. Many heuristics, metaheuristics, approximation algorithms, and exact algorithms have been developed for this problem. This paper presents an analytical study and experimental evaluation of the most representative approximation algorithms for the vertex k-center problem. For each of the algorithms under consideration and using a common notation, we present proofs of their corresponding approximation guarantees as well as examples of tight instances of such approximation bounds, including a novel tight example for a 3-approximation algorithm. Lastly, we present the results of extensive experiments performed over *de facto* benchmark data sets for the problem which includes instances of up to 71009 vertices.

INDEX TERMS Approximation algorithms, k-center problem, polynomial time heuristics.

I. INTRODUCTION

Perhaps one of the first center selection problems for which there is historical register is the following: "given three points in the plane, find a fourth point such that the sum of its distances to the three points is minimized" [1]. Given its simplicity, it is hard to establish who first stated this problem. However, this problem is usually associated to Pierre de Fermat, who asked this question around 1636, and its first registered solution is associated to Evangelista Torricelli [1]. An extension of this problem is known as the Weber's problem, where the points have an associated cost and the goal is to locate not 1 but k centers [1]. By adding new properties and restrictions to a basic k-center problem, the collection of k-center problems have become larger over the years.

One of the basic center selection problems that more directly gave rise to many other center problems is known as the absolute 1-center problem. This problem was formally introduced by Hakimi when he faced the problem of finding the best location for a *police station* in a *highway system* [2]. The goal of this problem is to minimize the distance from the farthest *community* to the *police station*. In a graph, an absolute 1-center is a location along any edge that minimizes the distance from the farthest vertex to such location. The k-center problem is a more general version of the absolute 1-center problem, where the goal is to locate $k > 1$ centers in the graph, such that the distance from the farthest vertex to its nearest center is minimized. These centers may be located along any edge (absolute k-center) or may be restricted to vertices (vertex k-center). Besides, there can be a cost associated with the selection of every vertex (weighted vertex k-center problem) [3].

There are many other restrictions that yield to different k-center problems. As instance, the following problems aims at minimizing the maximum distance from vertices to its nearest centers under different assumptions and/or restrictions:

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

- The asymmetric k-center problem, where the input is an asymmetric directed graph that satisfies the directed triangle inequality. The best possible polynomial algorithm for this problem delivers a $O(\log^* n)$ approximation [4], [5].
- The capacitated k-center problem, where each center can *attend* only a fixed number of vertices [6].
- The heterogeneous capacitated k-center problem, which is similar to the capacitated k-center problem, except that the capacities of each center are also assigned [7].
- The aligned k-center problem, where the centers must be selected from a previously defined line or polygon [8].
- The edge-dilation k-center problem, where the goal is to minimize the maximum ratio of the distance between any two nodes via their respective centers to their true graph distance [9].
- The fault tolerant k-center problem, where each selected center must have a set of $\alpha \leq k$ centers close to it [10].
- The fault tolerant capacitated k-center problem, where each center can *attend* only a fixed number of vertices, and after the *failure* of some centers, the vertices can be reassigned to another centers [11].
- The p-neighbor k-center problem, where given an integer p the goal is to minimize the maximum distance of any non-center vertex to its p^{th} closest center [12].
- The k-center problem with minimum coverage, where centers are required to attend a minimum of vertices [13].
- The mixed k-center problem, where m centers must be in the set of vertices, and the rest can be anywhere ($m < k$) [14].
- The p-next center problem, where the objective is to minimize the distance from the farthest vertex to its closest center plus the distance between this center to its closest alternative center [15].

This paper focuses on the "uncapacitated unweighted vertex k-center problem" and its known approximation algorithms for graphs in a metric space. We think this reduction is necessary because the approximation algorithms for this problem are conceptually so close that they should be stated as clear as possible as similar expressions of the same basic ideas. For the rest of the paper, we refer to the "uncapacitated unweighted vertex k-center problem" just as the "vertex k-center problem".

The remaining of the paper is organized as follows. Section II presents a brief summary on the vertex k-center problem and its related algorithmic solving techniques. Section III presents an analysis of the known approximation algorithms for the vertex k-center problem which includes proofs of their approximation guarantees, and examples of tight instances of the approximation bounds. The main goal of this section is to present all the known approximation algorithms for the vertex k-center problem as similar expressions of the same basic ideas by using a common notation. Section IV shows the result of applying all the described algorithms over a set of the most well known benchmark data sets for the vertex k-center problem. Finally, Section V presents the concluding remarks.

II. THE VERTEX K-CENTER PROBLEM

Formally, the vertex k-center problem is defined as follows. Given a complete undirected graph $G = (V, E)$ with edge costs satisfying the triangle inequality, and a positive integer k; find a subset $C \subseteq V$ of centers, with $|C| \leq k$, such that the distance from the farthest vertex $v \in V$ to its closest center in C is minimized [16]–[20]. More specifically, the goal is to find a subset C of centers of cardinality at most k that minimizes $r(C) = \max_v\{d(v, C)\}$, where $d(v, C)$ is the distance between a vertex v and the set C, which is defined as the cost of the cheapest edge from v to any of the vertices in C. The solution size $r(C)$ is usually known as the *covering radius* because on a two dimensional euclidean space the centers define disks that cover its nearest vertices and the radius of the larger disk is the solution size. The vertex k-center problem is perhaps the most fundamental among the k-center problems. In addition, it has potential application to different areas, such as Facility Location [20]–[22], Clustering [23], emergency services, computer network services, distribution, and more [1].

Due to its importance, the vertex k-center problem has been addressed through different algorithmic approaches, such as heuristics, metaheuristics, exact algorithms, and of course approximation algorithms. Among these approaches, the approximation algorithms stand as the most efficient and reliable, because the vertex k-center problem can not be solved in polynomial time within an approximation factor of $\rho < 2$, unless $P = NP$ [3], [17]–[19], [24], [25]. Therefore, under $P \neq NP$, the best possible polynomial time algorithms for this problem must deliver 2-approximated solutions. To date, there are at least three known approximation algorithms that achieve this approximation factor: the Sh algorithm [19], [26], the Gon algorithm [3], [17], and the HS algorithm [18], [19]. Section III describes these algorithms using a common notation, which helps to understand how these algorithms relate to each other.

Even though the Sh, Gon and HS algorithms achieve the best possible approximation factor (if $P \neq NP$), they tend to perform poorly on most benchkmark data sets [27]–[29]. For this reason, many heuristic, metaheuristic and exact algorithms have been designed through the years. Even though these algorithms do not guarantee to converge quickly or to find optimal solutions, they find the best-known solutions (or even optimal solutions) on most benchmark data sets [21], [22], [28]–[33]. So, in one hand there are 2-approximation algorithms, which are theoretically the best possible, but practically useless on many specific instances. On the other hand, there are heuristics that may run in polynomial time, but do not give any guarantee about the quality of the generated solutions, and yet may find *near optimal* solutions on benchmark data sets. There are metaheuristics that do not run in polynomial time, give no guarantee on the quality of the generated solution, and yet may deliver *near* *optimal* solutions on benchmark datasets. And finally, there are exact algorithms, which deliver the optimal solution, but give no guarantee about the termination time.

Among the polynomial heuristic algorithms are the Gr greedy pure algorithm [27], [28], the Scr algorithm [28], and the CDSh algorithm [34]; the last two being considered as the polynomial time algorithms for the vertex k-center problem with best empirical performance [28], [29], [34]. Among the exact algorithms are those proposed by Daskin [35], Ilhan *et al.* [36], Elloumi *et al.* [37], Al-Khedhairi and Salhi [38], Chen and Chen [39], Calik and Tansel [40], and Contardo *et al.* [23]. All these exact algorithms are based on Integer Programming or Mixed Integer Programming formulations. Through empirical results, these exact algorithms have shown to be relatively efficient on most instances from benchmark data sets. Among the metaheuristic algorithms are proposals based on Tabu Search [30], Variable Neighborhood Search [30], [31], Scatter Search [21], GRASP [21], Memetic Genetic Algorithms [32], Harmony Search [22], and Bee Colony Optimization [33]. Although these methods give no guarantee either the quality of the solutions they find nor the execution time, all of them perform better than the 2-approximated Sh, Gon, and HS algorithms on most benchmark data sets reported in the literature. Because of this, we also describe a fourth approximation algorithm, the CDS algorithm [34], which despite having a sub-optimal approximation factor of 3, significantly outperforms the known 2-approximation algorithms over the *de facto* benchmark data sets from the literature. To some extent, the CDS algorithm achieve a balance between the advantages of the approximation algorithms and the other algorithmic approaches, because it runs in polynomial time, it guarantees that the quality of the solution is not arbitrary, and the empirical results show that the generated solutions are not only theoretically *near optimal*, but also *competitive* in practical terms.

III. ANALYTICAL STUDY OF APPROXIMATION ALGORITHMS FOR THE VERTEX K-CENTER PROBLEM

In the context of combinatorial optimization problems, approximation algorithms exploit relevant structural properties of each particular problem in a *natural* and strict way [41], sacrificing optimality in order to preserve a polynomial execution time. The approximation algorithms described in this paper exploit a common relevant structural property of the vertex k-center problem, which is described through the specification of the 2-approximated Sh algorithm (Subsection III-A). In a nutshell, to get 2-approximated solutions it suffices to select centers sufficiently far apart. Subsections III-B and III-C describe the 2-approximated Gon and HS algorithms respectively. Subsection III-D describes the 3-approximated CDS algorithm. We begin this section by presenting Theorem 1, which shows that the best possible polynomial algorithm for the vertex k-center problem delivers 2-approximated solutions [3], [17]–[19], [41]. The proof consists in showing that the problem of computing

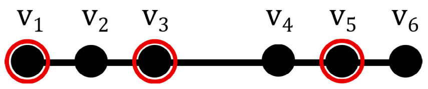


FIGURE 1. A dominating set $\{v_1, v_3, v_5\}$.

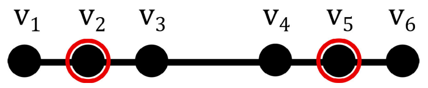


FIGURE 2. Minimum dominating set $\{v_2, v_5\}$.

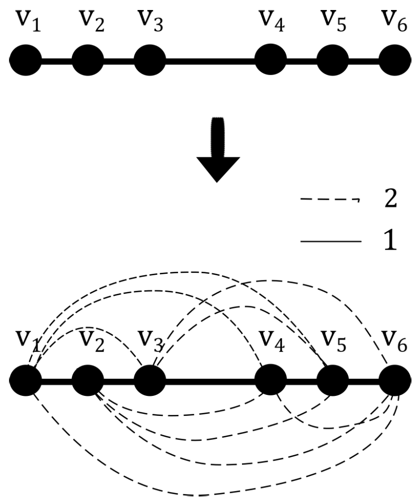


FIGURE 3. Polynomial transformation from minimum dominating set to vertex k-center problem. The original graph must be completed. The old edges have cost 1, and new edges cost 2.

2-approximated solutions in polynomial time is as hard as solving a well known NP-Hard problem, also in polynomial time. For the proof of Theorem 1 we will use the NP-Hard minimum dominating set problem [42]–[44], where a dominating set and a minimum dominating set are defined as follows.

Definition 1: Given an input graph $G = (V, E)$, a dominating set is a subset $D \subseteq V$ such that for every vertex $v \in V$, an edge $(v, u) \in E$ with either u or v in D exists. Figure 1 shows a dominating set.

Definition 2: A minimum dominating set is a set of minimum cardinality among all the dominating sets. Figure 2 shows a minimum dominating set.

Theorem 1: Under $P \neq NP$, it is not possible to solve the vertex k-center problem with an approximation factor of $\rho < 2$.

Proof: The proof is by a polynomial time reduction from the NP-Hard minimum dominating set problem to the vertex k-center problem. Given an input graph $G = (V, E)$ for the minimum dominating set problem, create a new input graph $G' = (V, E')$ for the vertex k-center problem. The new graph G' is a complete graph, where $(u, v) \in E'$ has a weight of 1 if $(u, v) \in E$, and a weight of 2 if $(u, v) \notin E$. Notice that the edge costs in G' satisfy the triangle inequality and that this transformation takes polynomial time (Fig. 3). Assuming that a $(2-\epsilon)$-approximation algorithm for the vertex k-center

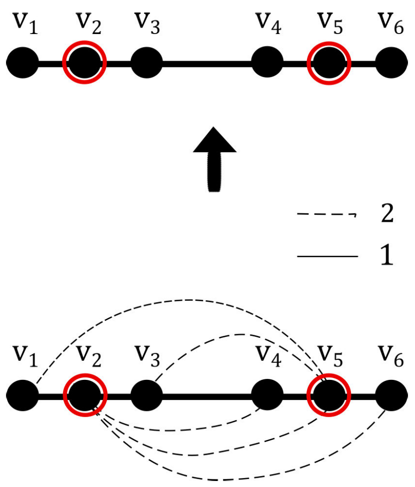


FIGURE 4. **The optimal solution C^* for the vertex k-center problem with $k = 2$ has size $r(C^*) = 1$. Implying that the solution to the vertex k-center problem is a dominating set for the original graph. Besides, this solution is a dominating set of minimum size.**

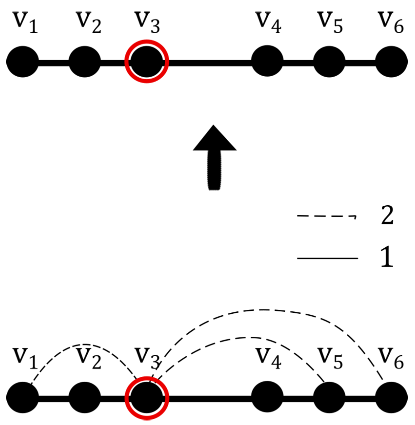


FIGURE 5. **The optimal solution C^* for the vertex k-center problem with $k = 1$ has size $r(C^*) = 2$. Implying that the minimum dominating set for the original graph must have cardinality greater than $k = 1$.**

problem exists, it will always returns optimal solutions for the graph G'. This is because the only possible values for the optimal covering radius are either 1 or 2.

When this algorithm returns a solution of size 1 for a given value of k, this solution defines a dominating set of size k in the original graph G because all the vertices are at distance 1 from at least one of the centers (Fig. 4). Similarly, any solution of size 2 does not define a dominating set in the original graph because at least one vertex is not connected to a center by an edge in the original graph (Fig. 5).

Now, the size of the minimum dominating set is unknown. This issue can be removed by solving the vertex k-center problem over G' with every possible size of the minimum dominating set, i.e., $k = 1, 2, ..., n$, where $n = |V|$. Among all the dominating sets that are generated, the one with minimum size is the minimum dominating set (Fig. 5). Therefore, if a $(2 - \epsilon)$-approximation algorithm for the vertex k-center problem exists, then the NP-Hard minimum dominating set problem can be solved in polynomial time, implying that $P = NP$. The contrapositive of this assertion is Theorem 1. □

Theorem 1 shows two things: there is a polynomial time reduction from the minimum dominating set problem to the vertex k-center problem, and solving the vertex k-center problem with an approximation factor $\rho < 2$ is NP-Hard. Besides these two points, Theorem 1 also gives an idea on how to solve the vertex k-center problem with a reduction to the minimum dominating set problem when the size of the optimal solution for the vertex k-center problem is known ahead of time (Lemma 2).

Definition 3: Given a graph $G = (V, E)$, a pruned graph (or bottleneck graph) $G_r = (V, E_r)$ is a graph such that E_r consists of all the edges in E with cost less than or equal to r.

Lemma 2: Given an instance $G = (V, E)$ for the vertex k-center problem, and the size r^* of its optimal solution C^*. The minimum dominating set for the pruned graph $G_{r^*} = (V, E_{r^*})$ have at most k vertices.

Proof: The proof is by contradiction. If the minimum dominating set for the pruned graph $G_{r^*} = (V, E_{r^*})$ have more than k vertices, then for any set C_k of k vertices there will always be a vertex e such that for all edges $(e, u) \in E_{r^*}$ neither e nor u are in C_k. If no edge $(e, u) \in E : e, u \notin C_k$ is present in E_{r^*} then its cost in E must be greater than r^*, implying that any set of k centers in the original graph G define a solution with size greater than r^*. Therefore, the size r^* of the optimal solution was not optimal. The contrapositive of this assertion is Lemma 2. □

The relationship between the vertex k-center problem and the minimum dominating set problem has been exploited extensively in the past. Most exact algorithms for the vertex k-center problem take advantage of this relationship, which was first noticed by Minieka [45]. It is important to remark that the authors of most of these exact algorithms refer to the minimum dominating set problem as a set cover problem [20], [45]. Since both problems (minimum set cover and minimum dominating set) are mutually reducible in polynomial time, there is no harm in calling them by the same name. Yet, from a purist point of view and in order to avoid confusion we refer to the minimum dominating set just by this name. The main issue with the approach of reducing the vertex k-center problem to the minimum dominating set is that even if the size of the optimal solution is known ahead of time, the resulting problem is still NP-Hard. Thus, under $P \neq NP$, and on general instances, any exact algorithm for the vertex k-center have exponential running time. Eq. 1 and Eq. 2 show the Integer Programming formulation for the vertex k-center problem as a minimum dominating set problem when the size r^* of the optimal solution C^* for the vertex k-center problem is known ahead of time [1], [20], [23].

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq 1, \quad \forall j \in \{1, 2, ..., n\} \end{aligned} \tag{1}$$

$$\text{where } a_{ij} = \begin{cases} 1, & \text{if } d(v_i, v_j) \leq r^* \\ 0, & \text{otherwise} \end{cases}$$
$$\text{and } x_i \in \{0, 1\}$$
$$r^* = r(C^*) \qquad (2)$$

In the previous formulation, for every vertex v_i there is a binary variable x_i ($i \in \{1, 2, ..., |V|\}$). If the vertex v_i is in the minimum dominating set, then $x_i = 1$; otherwise, $x_i = 0$. Thus, the objective function (Eq. 1) aims at minimizing the sum of all x_i. For every pair of vertices there is a variable a_{ij}, which equals 1 if the edge that connects them has cost less than or equal to r^* ($d(v_i, v_j) \leq r^*$); otherwise, it equals 0 (Eq. 2). The whole matrix of a_{ij} values represents the pruned input graph, where all the vertices with weight greater than r^* are removed. There are $n = |V|$ constraints (Eq. 1), one for each vertex. In order to guarantee that each vertex is dominated by at least one element of the minimum dominating set, these constraints must be satisfied.

So, the vertex k-center problem can be reduced to a single minimum dominating set problem, but only when the size of the solution is known ahead of time. However, this issue can be addressed by the fact that the solution size must be equal to the weight of some edge in the input graph. Since there are $O(n^2)$ edges, the vertex k-center problem is reduced to solving $O(n^2)$ minimum dominating set problems, with r^* from Eq. 2 replaced by $r = w(e)$, where $e \in E$. From the set of obtained solutions, the optimal one for the vertex k-center problem is the one with no more than k vertices that has the minimum r. The number of minimum dominating set problems to be solved can be reduced from $O(n^2)$ to $O(\log n)$ by performing a binary search over the ordered set of $O(n^2)$ possible values of r. Algorithm 1 shows the basic reduction from vertex k-center problem to the minimum dominating set with binary search. This algorithm follows from Minieka's observations [45] and Daskin's algorithm [35]. Most exact algorithms for the vertex k-center problem are Integer or Mixed Integer Programming formulations that are inspired by this basic reduction. As instance, Daskin's algorithm follows the same sketch, but instead of performing a binary search it performs a bisection search, which consists in setting $high = w(e_m)$ and $low = w(e_1)$ [35]. The algorithm of Ilhan *et al.* [36] adds a feasibility phase, so that their Integer Programming formulation is solved only if the minimum dominating set of the pruned graph does not have more than k centers. The algorithm of Al-Khedhairi and Salhi [38], and Chen and Chen [39] works primarily on setting up tighter upper and lower bounds, so that the vertex k-center problem is reduced to less subproblems [38], [39].

Lemma 3: Algorithm 1 (BEA) returns the optimal solution to the vertex k-center problem.

Proof: For a given value of $r \in \{w(e_1), w(e_2), ..., w(e_m)\}$, the minimum dominating set C of the pruned graph G_r has cardinality $|C| \leq k$ or $|C| > k$. If $|C| \leq k$, then the optimal solution C^* of the vertex k-center problem on the original graph G has size $r^* \leq r$; thus, *high* is set to *mid*. If $|C| > k$, then the minimum dominating set of every pruned graph $G_{r'}$, with $r' \leq r$, has more than k elements; thus, *low* is set to *mid*. Therefore, at the last iteration, Algorithm 1 is executed with a value $w(e_{mid})$ that equals r^*. So, by Lemma 2 Algorithm 1 (BEA) returns an optimal solution. □

Algorithm 1 A Basic Exact Algorithm for the Vertex K-Center Problem (BEA)

```
Input: An undirected graph $G = (V, E)$, an integer $k$,
       and an ordered list of the $m$ edge weights of $G$:
       $w(e_1), w(e_2), ..., w(e_m)$ where $w(e_i) \leq w(e_{i+1})$
Output: A set of vertices $C \subseteq V$, $|C| \leq k$
1  $high = m$ ;
2  $low = 1$ ;
3  while $high - low > 1$ do
4      $mid = \lceil (high + low)/2 \rceil$ ;
5      $C = minimumDominatingSet(G, w(e_{mid}))$ ;
6      if $|C| \leq k$ then
7          $high = mid$ ;
8      else
9          $low = mid$ ;
10     end
11 end
12 return $C$ ;
```

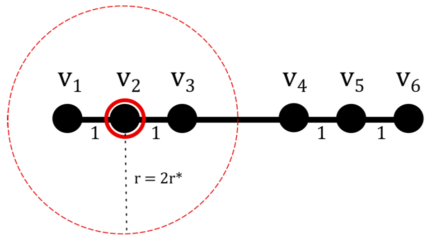
A. Sh ALGORITHM

Independently introduced by Shmoys in 1995, and by Plesník in 1987, the Sh algorithm is an $O(kn)$ 2-approximated algorithm that reveals a relevant structural property of the vertex k-center problem [19], [26]. This property is that the iterative selection of centers from a set of vertices at an *appropriate* distance, suffices to guarantee that every center from the optimal solution is close to one of the selected centers; therefore, all the vertices assigned to each center in the constructed solution are also close to one of the centers from the optimal solution. This structural property is exploited by all the approximation algorithms reviewed in this paper. However, unlike other approximation algorithms, the Sh algorithm requires a guess r on the size of the optimal solution, which represents a disadvantage. Fortunately, as will be explained in Section III-B and III-C, the Gon and HS algorithms are clever implementations of the Sh algorithm that eliminate this disadvantage.

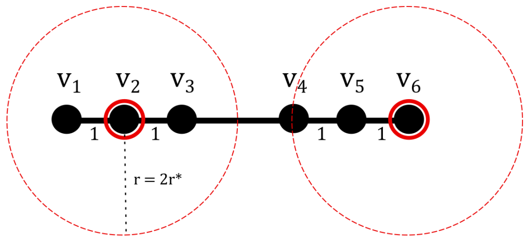
Algorithm 2 shows the pseudocode of the Sh' algorithm, which is equivalent to the original Sh algorithm. For convenience in the argument, we will use the Sh' algorithm instead of the original Sh algorithm (the difference between Sh' and Sh is subtle and is explained at the last paragraph of this section). The Sh' algorithm receives as input a guess r on the size (covering radius) of the optimal solution which is denoted by C^*. At the first step, the algorithm selects any vertex $v \in V$ as initial center c_1. At the next $k - 1$ iterations, it selects as center c_i any vertex at distance larger than $2r$ from the already selected centers. Using Lemma 4, Theorem 6 shows that the Sh' algorithm returns a 2-approximated solution when $r = r^*$, where r^* is the size $r(C^*)$ of the

Algorithm 2 Sh' Algorithm

```
Input: An undirected graph $G = (V, E)$, an integer $k$,
       a covering radius $r$
Output: A set of vertices $C \subseteq V$, $|C| = k$
1  $c_1$ = any vertex on $V$ ;
2  $C = \{c_1\}$ ;
3  for $i = 2$ to $k$ do
4      $c_i$ = A vertex $v \in V$ such that $d(v, C) > 2r$ ;
5      if $c_i$ exists then
6          $C = C \cup \{c_i\}$ ;
7      else
8          $C = C \cup$ any vertex $v \in V$ ;
9      end
10 end
11 return $C$ ;
```



(a) The first center is selected at random (v_2). At the next iterations, any sufficiently far vertex (at distance greater than $2r^*$ from the current partial solution) is selected.



(b) One of the sufficiently far vertices is v_6. The resulting solution has size $2r^*$. This is a tight example for the Sh' algorithm.

FIGURE 6. Sh' algorithm execution with $k = 2$. (a) The first center is selected at random (v_2). At the next iterations, any sufficiently far vertex (at distance greater than $2r^*$ from the current partial solution) is selected. (b) One of the sufficiently far vertices is v_6. The resulting solution has size $2r^*$. This is a tight example for the Sh' algorithm.

optimal solution C^*. Fig. 6 shows a simple example of how the Sh' algorithm works.

Lemma 4: If $r \geq r^*$, the Sh' algorithm generates a solution C of size $r(C) \leq 2r$.

Proof: First, if during any of the k iterations of the Sh' algorithm the selected center c_i is at a distance less than or equal to $2r$, then all vertices are at a distance less than or equal to $2r$ from the current solution and hence, the algorithm returns a solution with covering radius $r(C) \leq 2r$. Now, for the rest of the proof we will assume that during all the iterations of the Sh' algorithm the selected center c_i is at a distance larger than $2r$ from the current partial solution. The overall description of the proof is as follows. We have to show that the center c_i, selected by the Sh' algorithm during the i-th iteration, covers within a radius of 2r all the vertices covered by its nearest vertex c_i^* in the optimal solution C^*. Then, we show that the centers c_i^* considered during the k iterations of the algorithm are different. Consequently, all the k centers of the optimal solution have a center in the generated solution that is responsible of covering their vertices within a $2r$ radius. In this way, the Sh' algorithm returns a solution with covering radius $r(C) \leq 2r$.

By definition, the distance from every selected vertex c_i to its nearest center $c_i^* \in C^*$ is less than or equal to r^*, which is less than or equal to r.

$$d(c_i, c_i^*) \leq r^* \leq r \tag{3}$$

Therefore, every selected center c_i covers within a $2r$ radius the vertices covered by $c_i^* \in C^*$.

$$\forall v \in V : d(v, c_i^*) \leq r, \quad d(c_i, v) \leq d(c_i, c_i^*) + d(c_i^*, v) \leq 2r \tag{4}$$

Now, we proceed by contradiction to show that all the centers c_i^* covered during the k iterations of the algorithm are different. Suppose that the center c_i^* was already covered at an iteration $j < i$, i.e., that $c_i^* = c_j^*$. Therefore, the distance from c_i to c_j is at most $2r$.

$$d(c_i, c_j) \leq d(c_i, c_i^*) + d(c_j^*, c_j) \leq 2r \tag{5}$$

However, by assumption we know that $d(c_j, c_i) > 2r$ at every iteration, contradicting Eq. 5. □

Corollary 5: If the Sh' algorithm returns a solution C of size $r(C) > 2r$, then $r < r^*$.

Proof: This is the contrapositive of Lemma 4. □

Theorem 6: If $r = r^*$, the Sh' algorithm returns a 2-approximated solution.

Proof: By Lemma 4. If $r \geq r^*$, the Sh' algorithm returns a solution C of size $r(C) \leq 2r$. Thus, if $r = r^*$, C has size of $r(C) \leq 2r^*$. □

As said at the beginning of this section, the algorithm Sh' (Algorithm 2) is equivalent to the original Sh algorithm [19], [26]. On one hand, the original Sh algorithm iterates until there are no more vertices at distance greater than $2r$ from the solution. So, for a given value of r it may returns a set of centers C with cardinality different from k. On the other hand, Algorithm 2 iterates exactly k times, and always returns a solution of cardinality k. Despite this difference, Lemma 4, Corollary 5, and Theorem 6 holds for the original Sh algorithm. By Lemma 4, if $r \geq r^*$, k centers are enough to *cover* all the vertices within a radius of $2r$. Thus, the original Sh algorithm does not need to iterate more than k times in order to find a solution of size less than or equal to $2r$. Theorem 6 immediately follows. Corollary 5 also holds, being a solution C "*of size* $r(C) > 2r$" (Sh' algorithm) equivalent to a solution C "*of cardinality greater than* k" (original Sh algorithm), in the sense that both cases imply that $r < r^*$.

Algorithm 3 Gon Algorithm

```
Input: An undirected graph $G = (V, E)$, and an integer $k$
Output: A set of vertices $C \subseteq V$, $|C| = k$
1 $c_1 =$ any vertex on $V$ ;
2 $C = \{c_1\}$ ;
3 for $i = 2$ to $k$ do
4     $c_i =$ A vertex $v \in V$ such that $distance(v, C)$ is maximal ;
5     $C = C \cup \{c_i\}$ ;
6 end
7 return $C$ ;
```

1) COMPLEXITY

The Sh' algorithm consists of k iterations. At every iteration the distance from every vertex $v \in V$ to the current solution C is evaluated. This evaluation can be done in $O(n)$ steps, by comparing $d(v, C \setminus \{c_i\})$ with $d(v, c_i)$, and keeping the smallest value for every $v \in V$. Thus, the overall complexity of the Sh' algorithm is $O(kn)$.

2) TIGHT EXAMPLE

Figure 6 shows a tight example for the Sh' algorithm. For $k = 2$, the size of the optimal solution $C^* = \{v_2, v_5\}$ for this instance is of 1 unit (Figure 4). Sh' selects the first vertex at random ($c_1 = v_2$) (Figure 6a), and then selects a vertex at distance larger than $2r = 2r^* = 2$. If there is a tie, any vertex can be selected ($c_2 = v_6$). The size of this solution is of 2 units (Figure 6b).

B. GON ALGORITHM

Independently introduced by Gonzalez, and by Dyer and Frieze in 1985, the Gon algorithm is an $O(kn)$ 2-approximation algorithm that cleverly implements the Sh algorithm by removing the need of guessing the size of the optimal solution [3], [17].

By Lemma 4, if $r \geq r^*$, the Sh algorithm returns a solution C of size $r(C) \leq 2r$. This is achieved by the Sh algorithm by adding a center at distance greater than $2r$ from the current solution at every iteration. However, instead of using a guess on r to find a sufficiently far vertex, the Gon algorithm just adds the farthest vertex to the solution at every iteration. The key idea is that if a non-empty set of vertices at distance larger than $2r$ from a partial solution exists, it must includes the farthest vertex. Therefore, Theorem 7 holds. Algorithm 3 shows the pseudocode of the Gon algorithm.

Theorem 7: The Gon algorithm returns 2-approximated solutions.

1) COMPLEXITY

Just like the Sh' algorithm, the Gon algorithm consists of k iterations. At every iteration, the distance from every vertex $v \in V$ to the current solution C is evaluated. This evaluation can be done in $O(n)$ steps, by comparing $d(v, C \setminus \{c_i\})$ with $d(v, c_i)$, and keeping the smallest value for every $v \in V$. Thus, the overall complexity of the Gon algorithm is $O(kn)$.

2) TIGHT EXAMPLE

Figure 6 shows a tight example for the Sh' algorithm. This same instance also works as tight example for the Gon algorithm. For $k = 2$, the size of the optimal solution $C^* = \{v_2, v_5\}$ for this instance is of 1 unit (Figure 4). Gon selects the first vertex at random ($c_1 = v_2$) (Figure 6a), and then selects the farthest vertex ($c_2 = v_6$). The size of this solution is of 2 units (Figure 6b).

C. HS ALGORITHM

Introduced independently by Hochbaum and Shmoys in 1985, and by Plesník in 1987, the HS algorithm is an $O(n^2 \log n)$ 2-approximated algorithm that, just like the Gon algorithm, cleverly implements the Sh algorithm [18]. However, while the Gon algorithm removes the need of guessing the value of r, the HS algorithm still needs such guess. Actually, the inner cycle of the HS algorithm is equivalent to the Sh algorithm, and receives different values of r through a binary search over all the possible values of r until a value $r \leq r^*$, sufficiently large to create a solution of size $2r$, is found. Notice that the size of any solution must be equal to the weight of some edge of the input graph. So, the HS algorithm uses values from the set $\{w(e_1), w(e_2), ..., w(e_m)\}$, where $m = |E|$.

In their original paper, Hochbaum and Shmoys use the relationship between the vertex k-center problem and the dominating set problem on square graphs for its formal characterization [18]. However, the formal characterization can also be established in terms of the Sh algorithm, getting an analysis more similar to the one presented by Plesník [19]. Algorithm 4 shows the pseudocode of the HS' algorithm and Theorem 8 establishes its correctness. For convenience in the argument, we will use the HS' instead of the original HS algorithm. Actually, the HS' algorithm (Algorithm 4) is equivalent to the original HS algorithm. This is because while the inner cycle of the HS algorithm is the Sh algorithm, the inner cycle of the HS' algorithm is the Sh' algorithm. And, as stated in Subsection III-A, the Sh' and Sh algorithms are equivalent.

Theorem 8: The HS' algorithm returns 2-approximated solutions.

Proof: Lets recall how the Sh' algorithm works. At each i-th iteration the Sh' algorithm selects as center c_i a vertex whose distance from the current solution is larger than $2r$. The inner cycle of the HS' algorithm works in the same way, selecting as center any vertex whose distance from the current solution is larger than $2r$, where r equals the cost of some edge from the input graph. If the inner cycle of HS' is executed with every possible value of r (from $w(e_1)$ to $w(e_m)$), at some point r equals r^*, and therefore the HS' algorithm equals the Sh' algorithm. However, this leads to an $O(kn^3)$ algorithm. Fortunately, this complexity can be improved by performing a binary search over the set of possible values of r.

Algorithm 4 HS' Algorithm

```
Input: An undirected graph $G = (V, E)$, an integer $k$,
       and a non-decreasing ordered list of edges'
       costs $w(e_1), w(e_2), \ldots, w(e_m)$
Output: A set of vertices $C \subseteq V$, $|C| = k$
1  $low = 1$ ;
2  $high = m$ ;
3  $C = \emptyset$ ;
4  repeat
5      $mid = \lceil (high + low)/2 \rceil$ ;
6      $r = w(e_{mid})$ ;
7      $c_1 =$ any vertex in $V$ ;
8      $C' = \{c_1\}$ ;
9      for $i = 2$ to $k$ do
10         $c_i =$ A vertex $v \in V : d(v, C) > 2r$ ;
11         if $c_i$ exists then
12             $C' = C' \cup \{c_i\}$ ;
13         else
14             $C' = C' \cup$ any vertex $v \in V$ ;
15         end
16     end
17     if $r(C') \leq 2r$ then
18         $high = mid$ ;
19         $C = C'$ ;
20     else
21         $low = mid$ ;
22     end
23 until $high = low + 1$;
24 return $C$ ;
```

By performing a binary search over the $O(n^2)$ possible sizes of the optimal solution (from $w(e_1)$ to $w(e_m)$), the complexity of the HS' algorithm is reduced from $O(kn^3)$ to $O(kn \log n)$, where the complexity of the inner cycle is $O(kn)$, and the complexity of the outer cycle is $O(\log n)$. At each iteration of the outer cycle, the inner cycle is executed with a specific value $r = w(e_{mid})$, and a solution C' is generated. By Corollary 5, if C' has size $r(C') > 2r$, then $r < r^*$; if this is the case, *low* is set to *mid*; otherwise, *high* is set to *mid*. So, every time *high* is set to *mid* implies that it is possible to generate a solution C' of size at most $2r$, where $r = w(e_{high})$. Similarly, every time *low* is set to *mid* implies that $r^* > w(e_{low})$. In other words, at every time, $r^* > w(e_{low})$, and the inner cycle of HS' is capable of producing a solution of size at most $2w(e_{high})$. This way, at the end of the algorithm (when $high = low+1$, and *mid* is set to *high*), $r = w(e_{mid}) \leq r^*$ and the HS' algorithm is capable of generating a solution C of size $r(C) \leq 2r$. Thus, the HS' algorithm returns a solution C with size $r(C) \leq 2r \leq 2r^*$. □

1) COMPLEXITY

The HS' algorithm consists of the execution of the Sh' algorithm with different values of r. Since the values of r are taken from a set of $O(n^2)$ values using binary search, the overall complexity of HS', as shown in Algorithm 4, is $O(kn \log n)$. However, since this algorithm requires as input an $O(n^2)$ ordered set of edge costs, and since this ordering can be performed in $O(n^2 \log n)$ steps, the overall complexity of the HS' algorithm is $O(n^2 \log n)$, as originally stated by Hochbaum and Shmoys [18], [19].

2) TIGHT EXAMPLE

The example of Figure 6 shows a tight example for the Sh' and Gon algorithms. This same instance also works as tight example for the HS' algorithm. For $k = 2$, the size of the optimal solution $C^* = \{v_2, v_5\}$ for this instance is of 1 unit (Figure 4). Given the cost $r \in \{1, 2, 3, 4, 5\}$ of any edge from the input complete graph, the HS' algorithm selects the first vertex at random ($c_1 = v_2$). If some vertex at distance larger than $2r$ exists, it is selected as a center ($c_2 = v_6$) (Figure 6a); otherwise, a center is selected at random ($c_2 = v_6$). The size of this solution is of 2 units (Figure 6b).

D. CDS ALGORITHM

Presented by Garcia-Diaz et-al in 2017, the CDS algorithm is an $O(n^4)$ 3-approximated algorithm for the vertex k-center problem [34]. Despite having a sub-optimal performance, it significantly outperforms the Gon and HS' algorithms over the *de facto* benchmark data sets from the literature. Some heuristic variants that can be derived from CDS are the CDSh and CDSh+ algorithms, which have a more practical complexity of $O(n^2 \log n)$ and $O(n^3 \log n)$, respectively. Algorithm 5 shows the Critical Dominating Set algorithm, which is the fundamental piece of the CDS, CDSh and CDSh+ algorithms. Notice that the CDS algorithm and the Critical Dominating Set algorithm are not the same.

The Critical Dominating Set procedure takes advantage of the fact that a minimum dominating set on the pruned graph G_{r^*} of the input graph G is actually a solution for the vertex k-center problem (Lemma 2). However, the minimum dominating set problem is also an NP-Hard problem. So, in order to preserve a polynomial execution time, this relationship is exploited heuristically, giving no guarantee of finding the minimum dominating set. Just like the Gon and HS' algorithms, the Critical Dominating Set algorithm exploits the structural property revealed by the Sh' algorithm, but in a relaxed way. While the Gon and HS' algorithms select as center a vertex that guarantees the construction of a 2-approximated solution, the Critical Dominating Set algorithm also tries to maximize the number of vertices *dominated* by the newly selected center. This decision hurts the approximation factor, but significantly improves the algorithm's performance compared to that of the Gon and HS' algorithms when tested on the *de facto* benchmark data sets.

The Critical Dominating Set procedure consists of the following steps. First, the *PrunedGraph* subroutine constructs the graph $G_r = (V, E_r)$, where $E_r \subseteq E$ is the set of edges with cost less than or equal to r (Definition 3). Secondly, the *GetInitialScore* subroutine evaluates the initial *Score* of each vertex in G_r, which measures the fitness of every vertex

Algorithm 5 *Critical Dominating Set* Procedure

```
Input: An undirected graph $G = (V, E)$, an integer $k$,
       and a covering radius $r$
Output: A set of vertices $C \subseteq V$, $|C| = k$
// $N(v)$ is the set of neighbors of $v$ in $G_r$ ;
$C = \emptyset$ ;
$G_r = PrunedGraph(G, r)$ ;
$D = \emptyset$ ;
foreach $v \in V$ do
    $Score[v] = GetInitialScore(v, G_r)$ ;
end
for $i = 1$ to $k$ do
    $f$ = A vertex $v \in V$ such that $distance(v, C)$ is
    maximal ;
    $c_i$ = A vertex $v \in N(f) \cup f$ of maximum $Score$ ;
    $S = (N(c_i) \cup c_i) \setminus D$ ;
    foreach $v \in S$ do
        foreach $u \in N(v)$ do
            $Score[u] = Score[u] - 1$
        end
    end
    $D = D \cup N(c_i) \cup c_i$ ;
    $C = C \cup \{c_i\}$ ;
end
return $C$ ;
```

to be selected as center. When evaluated for the first time, this *Score* is just the degree of each vertex; in other words, it is the number of vertices that are in the neighborhood of each vertex. During every one of the next k iterations, the algorithm selects the farthest vertex f_i from the current solution C. This selection is made at random in the first iteration, because at this point C is the empty set. Then, a vertex c_i of maximum *Score* from $N(f_i) \cup f_i$ is selected as part of the solution. Finally, the *Score* of each vertex is updated. At any time during the execution of the algorithm, the *Score* assigned to each vertex $v \in V$ is equal to the number of vertices in $N(v)$ such that none of them is connected to some previously selected center. More specifically, at any iteration i, $\forall v \in V$, $Score(v) = |\{u : u \in N(v) \wedge u \notin N(C \setminus c_i)\}|$. Lemma 9 establishes the correctness of the Critical Dominating Set procedure.

Lemma 9: If $r \leq r^*$, the Critical Dominating Set procedure returns a 3-approximated solution.

Proof: First, if during any of the k iterations of the Critical Dominating Set algorithm, the farthest vertex f_i is at a distance less than or equal to $3r^*$, then all vertices are at a distance less than or equal to $3r^*$ from the current solution and hence, the algorithm returns a solution with covering radius $r(C) \leq 3r^*$. Now, for the rest of the proof we will assume that during all the iterations of the Critical Dominating Set procedure, the farthest vertex f_i is at a distance larger than $3r^*$ from the current solution. The overall description of the proof is as follows. We have to show that the center c_i selected by the Critical Dominating Set procedure during the i-th iteration covers within a radius of $3r^*$, all the vertices covered by its nearest vertex $c^*_{f_i}$ in the optimal solution C^*, where $c^*_{f_i}$ is the closest center to f_i ($c^*_{f_i} \in C^*$). Then, we show that the centers $c^*_{f_i}$ considered during the k iterations of the algorithm are different. Consequently, all the k centers of the optimal solution have a vertex in the generated solution that is responsible of covering their vertices within a $3r^*$ radius. Therefore, the Critical Dominating Set procedure returns a solution with covering radius $r(C) \leq 3r^*$.

First, we show that the center c_i selected at iteration i, is at a distance less than or equal to $2r^*$ from the center $c^*_{f_i} \in C^*$ which is the closest to f_i. The distance $d(f_i, c^*_{f_i})$ is less than or equal to r^* because $c^*_{f_i}$ is the closest center in the optimal solution C^* to f_i, and the distance $d(f_i, c_i)$ is less than or equal to r^* because $c_i \in N(f_i) \cup f_i$. Therefore, by using the triangle inequality we get

$$d(c_i, c^*_{f_i}) \leq d(f_i, c_i) + d(f_i, c^*_{f_i}) \leq 2r^* \tag{6}$$

Using the triangle inequality again, we see that any vertex v at a distance less than or equal to r^* from the center $c^*_{f_i}$ is at a distance less than or equal to $3r^*$ from the center c_i selected by the algorithm.

$$\forall v \in V : d(v, c^*_{f_i}) \leq r^*, \quad d(c_i, v) \leq d(c^*_{f_i}, c_i) + d(v, c^*_{f_i}) \leq 3r^* \tag{7}$$

Now, we proceed by contradiction to show that all the centers $c^*_{f_i}$ covered during the k iterations of the algorithm are different. Suppose that the center $c^*_{f_i}$ was already covered at an iteration $j < i$, i.e., that $c^*_{f_i} = c^*_{f_j}$. From Eq. 6 we know that $d(c_j, c^*_{f_j}) \leq 2r^*$ and from the assumption that $c^*_{f_i}$ is the closest optimal center to f_i we have that $d(f_i, c^*_{f_i}) \leq r^*$. Using the triangle inequality and the previous equations we get

$$d(c_j, f_i) \leq d(c_j, c^*_{f_j}) + d(c^*_{f_i}, f_i) \leq 3r^* \tag{8}$$

However, by assumption we know that $d(c_j, f_i) > 3r^*$ at every iteration, contradicting Eq. 8. □

Algorithm 6 shows the pseudocode of the CDS algorithm, which consists in executing the Critical Dominating Set algorithm with every possible value of r, which are taken from the set of edge costs. This way, at some point it will be executed with the size of the optimal solution, and by Lemma 9 the following theorem holds.

Theorem 10: The CDS algorithm returns 3-approximated solutions.

1) COMPLEXITY

The CDS algorithm consists in executing the Critical Dominating Set procedure with different values of r, which are taken from a set of $O(n^2)$ values. The Critical Dominating Set procedure consists of 3 stages. First, a pruned graph G_r is constructed by removing the edges with cost larger than r from the input graph; this can be done in $O(n^2)$ steps. Secondly, the degree of every vertex is evaluated; this can

Algorithm 6 CDS Algorithm

```
Input: An undirected graph $G = (V, E)$ and an
       integer $k$
Output: A set of vertices $C \subseteq V$, $|C| = k$
1  $C = \emptyset$ ;
2  $r(C) = \infty$ ;
3  $m = |E|$ ;
4  for $i = 1$ to $m$ do
5      $C' = CriticalDominatingSet(G, k, w(e_i))$ ;
6      if $r(C') \leq r(C)$ then
7          $C = C'$ ;
8          $r(C) = r(C')$ ;
9      end
10 end
11 return $C$;
```

be done in $O(n^2)$ steps. Thirdly, the following steps are executed inside a cycle of k iterations. A vertex f_i of maximal distance is selected in $O(n)$ steps. A vertex c_i of maximum *Score* is selected from $N(f_i) \cup f_i$ in $O(n)$ steps. The *Score* of the neighbors of the vertices dominated for the first time is reduced by 1 unit for each neighbor of c_i that is connected to them. This can be done in $n_i \times n$ steps, where n_i is the number of neighbors of c_i (including c_i) minus the number of vertices in D. Since D contains the neighbors of the previously selected centers c_j, $j < i$, by the last iteration ($i = k$) $n_1 + n_2 + \cdots + n_k = n$. Therefore, the overall complexity of the whole k iterations is $O(n^2)$. Thus, the overall complexity of the CDS algorithm is $O(n^4)$.

2) TIGHT EXAMPLE

While the Sh', Gon, and HS' algorithms have a simple tight example (Figure 6), the tight example for the CDS algorithm was harder to find. Figure 7 shows the adjacency matrix of a tight example $G = (V, E)$ with $k = 4$ for the CDS algorithm that certificates that its 3 approximation factor is tight. The cost of the edges of G satisfies the triangle inequality. This can be confirmed by applying the Floyd-Warshall algorithm [46], [47] over G, which returns the exact same graph, implying that the shortest path between every pair of vertices u and v is given by the edge $(u, v) \in E$. The optimal solution for this instance is $C^* = \{v_2, v_5, v_8, v_{11}\}$, and the distance from the vertices to their nearest center is 1 (shadowed regions). Thus, the size of the optimal solution for this instance is $r(C^*) = 1$.

Since the Critical Dominating Set algorithm guarantees the generation of 3-approximated solutions only when $r \leq r^*$, Table 1 shows the state of the algorithm at each iteration with $r = r^* = 1$, which is the smallest edge cost in the input graph. Notice that if self-loops were considered, the CDS algorithm would be executed at some point with a value $r = 0$, and it would behave exactly as the Gon algorithm, implying that the CDS algorithm is actually a 2-approximated algorithm. However, according to our experiments, the real mechanism that allows the CDS algorithm to outperform the

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
v_1	0	1	2	3	4	5	6	6	6	6	6	6
v_2	1	0	1	2	3	4	5	5	5	5	5	5
v_3	2	1	0	1	2	3	4	4	4	4	4	4
v_4	3	2	1	0	1	2	3	3	3	3	3	3
v_5	4	3	2	1	0	1	2	3	4	2	3	4
v_6	5	4	3	2	1	0	1	2	3	1	2	3
v_7	6	5	4	3	2	1	0	1	2	2	3	4
v_8	6	5	4	3	3	2	1	0	1	3	4	5
v_9	6	5	4	3	4	3	2	1	0	4	5	6
v_{10}	6	5	4	3	2	1	2	3	4	0	1	2
v_{11}	6	5	4	3	3	2	3	4	5	1	0	1
v_{12}	6	5	4	3	4	3	4	5	6	2	1	0

FIGURE 7. Adjacency matrix of a tight example $G = (V, E)$ for the 3-approximated CDS algorithm. $|V| = 12$, $k = 4$, $OPT = C^* = \{v_2, v_5, v_8, v_{11}\}$, $r(C^*) = 1$.

TABLE 1. State of the variables of the Critical Dominating Set procedure at each iteration over the tight example from Fig. 7, where $k = 4$, and $r = 1$. The final row shows the state at the end of the algorithm.

i	farthest vertices	f_i	$N(f_i) \cup f_i$	maximum *Score* vertices	c_i
1	V	v_4	$\{v_3, v_4, v_5\}$	$\{v_3, v_4, v_5\}$	v_4
2	$\{v_1, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$	v_7	$\{v_6, v_7, v_8\}$	$\{v_6, v_7, v_8\}$	v_6
3	$\{v_1, v_9, v_{12}\}$	v_{12}	$\{v_{11}, v_{12}\}$	$\{v_{11}, v_{12}\}$	v_{12}
4	$\{v_1, v_9\}$	v_1	$\{v_1, v_2\}$	$\{v_1, v_2\}$	v_1
5	$\{v_9\}$	–	–	–	–

	$distance(v, C) \setminus Score$											
i	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
1	$\infty\setminus 1$	$\infty\setminus 2$	$\infty\setminus 2$	$\infty\setminus 2$	$\infty\setminus 2$	$\infty\setminus 3$	$\infty\setminus 2$	$\infty\setminus 2$	$\infty\setminus 1$	$\infty\setminus 2$	$\infty\setminus 2$	$\infty\setminus 1$
2	$3\setminus 1$	$2\setminus 1$	$1\setminus 1$	$0\setminus 0$	$1\setminus 1$	$2\setminus 2$	$3\setminus 2$	$3\setminus 2$	$3\setminus 1$	$3\setminus 2$	$3\setminus 2$	$3\setminus 1$
3	$3\setminus 1$	$2\setminus 1$	$1\setminus 1$	$0\setminus 0$	$1\setminus 0$	$0\setminus 0$	$1\setminus 1$	$2\setminus 1$	$3\setminus 1$	$1\setminus 1$	$2\setminus 1$	$3\setminus 1$
4	$3\setminus 1$	$2\setminus 1$	$1\setminus 1$	$0\setminus 0$	$1\setminus 0$	$0\setminus 0$	$1\setminus 1$	$2\setminus 1$	$3\setminus 1$	$1\setminus 0$	$1\setminus 0$	$0\setminus 0$
5	$0\setminus 0$	$1\setminus 0$	$1\setminus 0$	$0\setminus 0$	$1\setminus 0$	$0\setminus 0$	$1\setminus 1$	$2\setminus 1$	$3\setminus 1$	$1\setminus 0$	$1\setminus 0$	$0\setminus 0$

i	C	D	S
1	$\emptyset$	$\emptyset$	$\{v_3, v_4, v_5\}$
2	$\{v_4\}$	$\{v_3, v_4, v_5\}$	$\{v_6, v_7, v_{10}\}$
3	$\{v_4, v_6\}$	$\{v_3, v_4, v_5, v_6, v_7, v_{10}\}$	$\{v_{11}, v_{12}\}$
4	$\{v_4, v_6, v_{12}\}$	$\{v_3, v_4, v_5, v_6, v_7, v_{10}, v_{11}, v_{12}\}$	$\{v_1, v_2\}$
5	$\{v_4, v_6, v_{12}, v_1\}$	$\{v_3, v_4, v_5, v_6, v_7, v_{10}, v_{11}, v_{12}, v_1, v_2\}$	–

Gon and HS' algorithms relies on its ability for taking local decisions that hurt its approximation guarantee. Therefore, we prefer to consider the CDS algorithm as a theoretically worst option – a 3-approximation algorithm – that tends to yield a better practical performance. However, it is a good idea to implement the CDS algorithm considering self-loops, because it does not increase the complexity of the algorithm. In fact, in the experiments presented in Section IV, we considered self-loops. It is important to point out that, even though the CDS algorithm can generate a tight solution with $r = r^* = 1$, it may generate better solutions with the other possible values of r, i.e. $\{0, 2, 3, 4, 5, 6\}$. In order to check how the algorithm performs in general, we implemented and executed the algorithm with different seeds. This allowed us to verify that in many times it returns tight solutions of size $3r^*$.

Table 1 is divided into three parts. Each one of these parts have one row for each one of the k iterations of the algorithm. In this case, $k = 4$. The state of the variables of the algorithm is shown in each row. Finally, there is an extra row that shows the final state of the algorithm. Specifically, at iteration $i = 1$ all the vertices are at distance ∞ from the current solution

Algorithm 7 CDSh Algorithm

```
Input: An undirected graph $G = (V, E)$, an integer $k$,
       and an ordered list of the weight of the $m$ edges
       of $G$: $w(e_1)$, $w(e_2)$, ..., $w(e_m)$ where
       $w(e_i) \leq w(e_{i+1})$
Output: A set of vertices $C \subseteq V$, $|C| = k$
1  $high = m$ ;
2  $low = 1$ ;
3  $C = \emptyset$ ;
4  $r(C) = \infty$ ;
5  while $high - low > 1$ do
6      $mid = \lceil (high + low)/2 \rceil$ ;
7      $C' = CriticalDominatingSet(G, k, w(e_{mid}))$ ;
8      if $r(C') \leq r(C)$ then
9          $C = C'$ ;
10         $r(C) = r(C')$ ;
11     end
12     if $r(C) \leq w(e_{mid})$ then
13         $high = mid$ ;
14     else
15         $low = mid$ ;
16     end
17 end
18 return $C$;
```

$C = \emptyset$, and their *Score* is equal to their degree in the input pruned graph G_{r^*}, where only the edges with cost less than or equal to $r^* = 1$ are included. Thus, the set of farthest vertices equals V. A vertex is selected at random from this set ($f_i = v_4$). Then, the vertex c_i of maximum *Score* in its neighborhood ($\{v_3, v_4, v_5\}$) is added to the current solution, $C = \{v_4\}$. Finally, the *Score* of every vertex is efficiently updated by setting $S = (N(v_4) \cup v_4) \setminus D$, where initially $D = \emptyset$. Finally D is set to $D \cup S = \{v_3, v_4, v_5\}$. The same process is repeated up to the last iteration. The final state of the algorithm is depicted in line 5 of each table, where it can be observed that the distance from v_9 to the generated solution is exactly 3, and the generated solution is $C = \{v_1, v_4, v_6, v_{12}\}$.

3) A MORE EFFICIENT HEURISTIC

The $O(n^4)$ complexity of the CDS algorithm becomes unpractical as the input grows. Algorithm 7 shows the CDSh algorithm, which has a more practical complexity of $O(n^2 \log n)$ by performing a binary search over the set of feasible covering radius. This binary search assumes that the solutions returned by the Critical Dominating Set procedure are optimal, which is not always true; otherwise it would solve the NP-Hard minimum dominating set in polynomial time. So, the approximation guarantees of the CDS algorithm are not present in the CDSh algorithm.

IV. EXPERIMENTAL PERFORMANCE EVALUATION

Table 2 shows the complexity of the approximation algorithms described in this paper, where Gon+ is the Gon

TABLE 2. Complexity and approximation factor (ρ) of some polynomial algorithms for the vertex k-center problem.

Algorithm	Complexity	ρ
Gon	$O(kn)$	2
HS	$O(n^2 \log n)$	2
CDSh	$O(n^2 \log n)$	-
Gon+	$O(kn^2)$	2
HS+	$O(n^3 \log n)$	2
CDSh+	$O(n^3 \log n)$	-
CDS	$O(n^4)$	3

TABLE 3. Mean (μ), standard deviation (σ), and average execution time reported by the tested algorithms over the *pmed* instances from OR-Lib.

40 *pmed* instances from OR-Lib			
Algorithm	μ	σ	average time (s)
Gon	1.527	0.095	8E-4
HS	1.392	0.100	8E-4
Gon+	1.304	0.120	0.070
HS+	1.258	0.110	0.625
CDSh	1.047	0.039	0.014
CDS	1.043	0.035	0.169
CDSh+	1.017	0.025	9.169
BEA	1.000	0.000	1.791

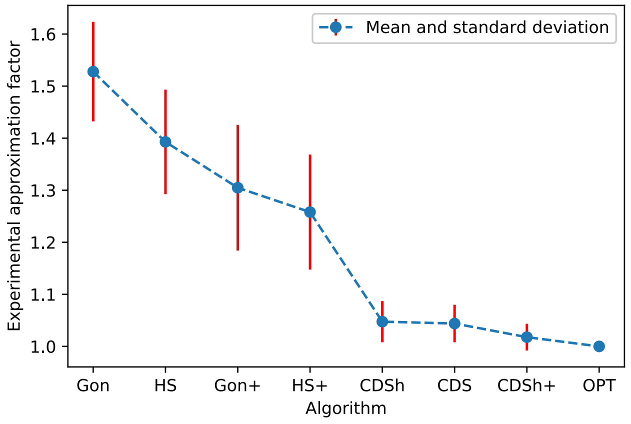


FIGURE 8. Mean and standard deviation reported by the tested algorithms over the *pmed* instances from OR-Lib.

algorithm repeated n times with a different initial center. The same idea applies to the HS+ and CDSh+ algorithms, which are the HS and CDSh algorithms repeated n times with a different initial center, respectively. All these algorithms were tested over optimal and best known solutions of *de facto* benchmark data sets from the literature. The optimal solutions were computed by the BEA exact algorithm (Alg. 1) using a commercial optimization software (Gurobi) with its default tuning parameters [48]. Tables 3 to 7, and Figures 8 to 12 show the experimental average approximation factor, experimental standard deviation, and average execution time of each algorithm over each benchmark data set. Tables 8 to 12 show the solution size found by each algorithm over each instance.

In all the experiments, we did not consider the time required for processing the input graph. In the case of the HS, CDSh, and CDSh+ algorithms, we also did not consider the time required for the ordering of the edge costs. Because of this, the execution time of the HS and HS+ algorithms have a complexity of $O(kn \log n)$ and $O(kn^2 \log n)$, respectively. All the polynomial time algorithms were implemented in C. The C code implementations can be downloaded from https://github.com/jesgadiaz/k-center-in-C. The script for the BEA exact algorithm was implemented in Python and each minimum dominating set problem was solved with Gurobi 8.1.0 [48]. All the implementations were executed on an Asus laptop with an Intel Core i5 processor of 2.3 GHz, and 24 Gb of RAM memory. At this point it is important to remark that memory is a limitation for any algorithm that requires to store the whole adjacency matrix of the input graph at every moment. To address this problem, only the relevant chunks of the adjacency matrix can be stored [23]. However, in general there will always be instances that require the whole adjacency matrix to be stored. For this reason, the experiments where the whole adjacency matrix is stored were limited to graphs with no more than 4663 vertices. For larger instances we did not store the adjacency matrix at all. Instead, we computed the cost of the edges every time needed. This is explained in more detail in the paragraph that corresponds to *large* instances.

Table 3 and Figure 8 show the mean and standard deviation of the experimental approximation factors obtained by the tested algorithms when executed over the *pmed* instances from OR-Lib [49]. This set consists of 40 instances with 100 to 900 vertices, and values of k from 5 to 200. For this specific set of instances, the edge costs are very often repeated. So, the number of edge costs goes down from $O(n^2)$ to just a few values. For this reason, the computationally more expensive algorithms CDS, CDSh+, and BEA had an acceptable average execution time. However, if the edge costs are not often repeated, the execution time of these algorithms increases a lot, which is the case with the other sets of instances. As expected, the Gon, HS, Gon+, and HS+ algorithms are among the most efficient, with an average execution time of 8E-4, 8E-4, 0.070, and 0.625 seconds, respectively. The CDS algorithm had an average execution time of 0.169 seconds. The CDSh and CDSh+ algorithms had an average execution time of 0.014 and 9.169 seconds, respectively. The reason why CDSh+ had a worst average execution time than CDS is that the set of edge costs is reduced from $O(n^2)$ to just a few values. So, while CDS consists of the *Critical Dominating Set* repeated a few times, the CDSh+ algorithm consists of the *Critical Dominating Set* repeated more than n times. With respect to the quality of the generated solutions, the HS+ algorithm had the best performance among the 2-approximated algorithms, with an experimental average approximation factor of 1.258. From all the tested polynomial time algorithms, the CDSh+ algorithm had the best performance, with an experimental average approximation factor of 1.017. Besides, the BEA basic exact algorithm shows a good performance, generating all the exact solutions in an average execution time of 1.791 seconds. Table 8 shows the detailed results obtained by the tested algorithms over the *pmed* instances.

Table 4 and Figure 9 show the mean and standard deviation of the experimental approximation factors obtained by the approximation algorithms when executed over a set of *small-size* instances from TSPLib [50]. This set consists of 40 instances with 200 to 657 vertices, and values of k from 5 to 40. As expected, the Gon, HS, Gon+, and HS+ algorithms are among the most efficient, with an average execution time of 4E-4, 6E-4, 0.014, and 0.269 seconds, respectively. The CDS algorithm had a total execution time of 283.774 seconds, which shows how inefficient this algorithm can be, and why its heuristic versions are more practical options. The CDSh and CDSh+ algorithms had a total execution time of 0.013 and 6.573 seconds, respectively. With respect to the quality of the generated solutions, the Gon+ algorithm had the best performance among the 2-approximated algorithms, with an experimental average approximation factor of 1.223. From all the tested polynomial time algorithms, the CDSh+ algorithm had the best performance, with an experimental average approximation factor of 1.040. Besides, the BEA basic exact algorithm shows a

TABLE 4. Mean (μ), standard deviation (σ), and average execution time reported by the tested algorithms over 40 *small* instances from TSPLib.

40 *small* instances from TSPLib			
Algorithm	μ	σ	average time (s)
Gon	1.401	0.112	4E-4
HS	1.351	0.115	6E-4
HS+	1.256	0.090	0.269
Gon+	1.223	0.068	0.014
CDSh	1.117	0.054	0.013
CDS	1.046	0.038	283.774
CDSh+	1.040	0.037	6.573
BEA	1.000	0.000	1.085

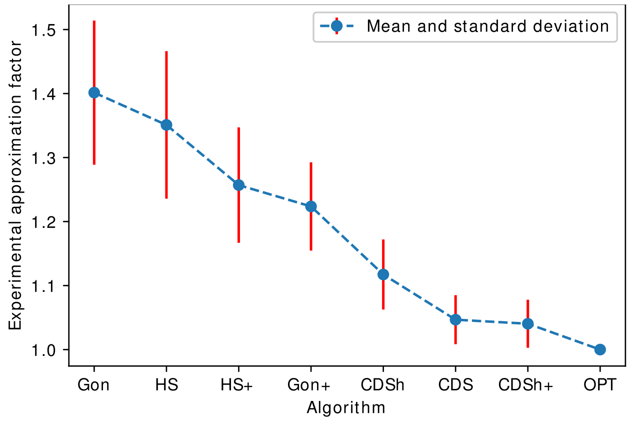


FIGURE 9. Mean and standard deviation reported by the tested algorithms over 40 *small* instances from TSPLib.

TABLE 5. Mean (μ), standard deviation (σ), and average execution time reported by the tested algorithms over the u1060, u1817 and mu1979 instances from TSPLib.

u1060, u1817 and mu1979 instances from TSPLib			
Algorithm	μ	σ	average time (s)
HS	1.423	0.070	0.034
HS+	1.364	0.069	60.160
Gon	1.337	0.056	0.001
Gon+	1.233	0.043	2.634
CDSh	1.159	0.052	0.314
CDSh+	1.100	0.045	613.999
BEA	1.000	0.000	1060.118

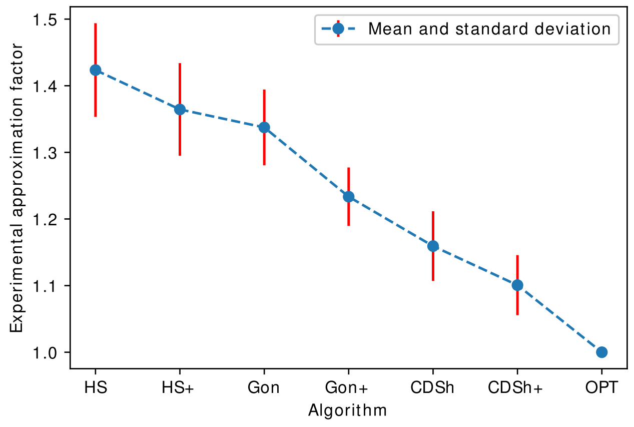


FIGURE 10. Mean and standard deviation reported by the approximation algorithms over the u1060, u1817 and mu1979 instances from TSPLib.

good performance, generating all the exact solutions in an average execution time of 1.085 seconds. Table 9 shows the detailed results obtained by the tested algorithms over the *small-size* TSPLib instances.

Table 5 and Figure 10 show the mean and standard deviation of the experimental approximation factors obtained by the approximation algorithms when executed over a set of *medium-size* instances from TSPLib [50]. This set consists of 3 instances with 1060 to 1979 vertices, and values of k from 10 to 150. As in the previous experiments, the Gon, HS, Gon+, and HS+ algorithms are among the most efficient, with an average execution time of 0.001, 0.034, 2.634, and 60.160 seconds, respectively. The CDS algorithm was not executed over this set of intances because it becomes very inefficient with instances of this size. The CDSh and CDSh+ algorithms had a total execution time of 0.314 and 613.999 seconds, respectively. With respect to the quality of the generated solutions, the Gon+ algorithm had the best performance among the 2-approximated algorithms, with an experimental average approximation factor of 1.233. From all the tested polynomial time algorithms, the CDSh+ algorithm had the best performance, with an experimental average approximation factor of 1.100. Finally, the BEA exact algorithm had the worst execution time performance, generating all the exact solutions in an average

TABLE 6. Mean (μ), standard deviation (σ), and average execution time reported by the tested algorithms over the pcb3038, nu3495 and ca4663 instances from TSPLib.

pcb3038, nu3496 and ca4663 instances from TSPLib			
Algorithm	μ	σ	average time (s)
HS	1.374	0.058	0.127
Gon	1.329	0.062	0.006
HS+	1.318	0.061	458.953
Gon+	1.221	0.029	19.172
CDSh	1.170	0.039	2.013
BEA	1.000	0.000	>60000

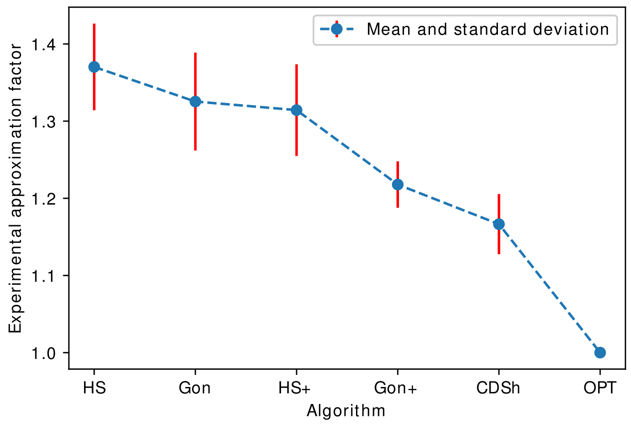


FIGURE 11. Mean and standard deviation reported by the approximation algorithms over the pcb3038, nu3496 and ca4663 instances from TSPLib.

TABLE 7. Mean (μ), standard deviation (σ), and average execution time reported by the tested algorithms over the sw24978, bm33708 and ch71009 instances from TSPLib.

sw24978, bm33708 and ch71009 instances from TSPLib			
Algorithm	μ	σ	average time (s)
HS	1.355	0.071	0.584
Gon	1.262	0.072	0.019
CDSh	1.193	0.051	689.723
Gon+	1.158	0.046	914.406

execution time of 1060.118 seconds. Table 10 shows the detailed results obtained by the tested algorithms over the *medium-size* TSPLib instances.

Table 6 and Figure 11 show the mean and standard deviation of the experimental approximation factors obtained by the approximation algorithms when executed over a set of more difficult *medium-size* instances from TSPLib [50]. This set consists of 3 instances with 3038 to 4663 vertices, and values of k from 10 to 150. From these instances only the optimal solutions from nu3496 and ca4663 could be computed by the exact algorithm BEA. From pcb3038, only the optimal solutions with $k \in \{10, 20, 30\}$ were computed. For values of k from 40 to 150, the solutions are much more difficult to find. Namely, the BEA algorithm had to run a couple of weeks just

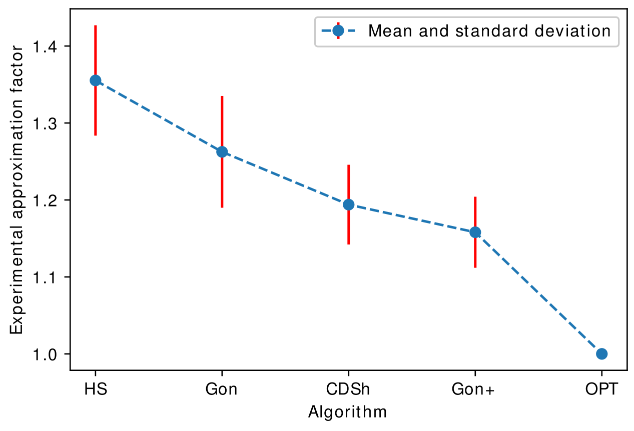


FIGURE 12. Mean and standard deviation reported by the approximation algorithms over the sw24978, bm33708 and ch71009 instances from TSPLib.

TABLE 8. Results obtained by each tested algorithm over the *pmed* instances from OR-Lib. The best found solutions are highlighted, and the optimal solutions are underlined.

				solution size						
instance	n	k	OPT	Gon	HS	Gon+	HS+	CDS	CDSh	CDSh+
pmed1	100	5	127	191	167	155	143	**127**	133	**127**
pmed2	100	10	98	149	118	117	118	**102**	**102**	**102**
pmed3	100	10	93	142	141	125	117	96	102	**96**
pmed4	100	20	74	112	100	92	94	**79**	**79**	**79**
pmed5	100	33	48	75	61	62	53	**48**	**48**	**48**
pmed6	200	5	84	118	104	98	96	86	87	**84**
pmed7	200	10	64	96	89	85	83	**66**	68	**64**
pmed8	200	20	55	78	75	71	68	**57**	61	**57**
pmed9	200	40	37	53	50	49	48	**37**	38	**37**
pmed10	200	67	20	29	28	29	28	**20**	**20**	**20**
pmed11	300	5	59	91	72	68	68	60	60	**59**
pmed12	300	10	51	84	72	66	61	**52**	53	**52**
pmed13	300	30	36	59	52	49	46	**37**	**37**	**37**
pmed14	300	60	26	39	36	36	34	**26**	**26**	**26**
pmed15	300	100	18	25	24	23	22	**18**	**18**	**18**
pmed16	400	5	47	66	56	52	52	**47**	**47**	**47**
pmed17	400	10	39	59	56	50	47	**39**	40	**39**
pmed18	400	40	28	42	41	39	37	30	30	**29**
pmed19	400	80	18	29	24	27	24	**19**	20	**19**
pmed20	400	133	13	19	18	17	18	**14**	**14**	**14**
pmed21	500	5	40	60	51	46	45	**40**	**40**	**40**
pmed22	500	10	38	61	49	47	46	**39**	**39**	**39**
pmed23	500	50	22	34	32	32	30	**23**	**23**	**23**
pmed24	500	100	15	23	22	21	20	**15**	16	**15**
pmed25	500	167	11	15	16	15	14	**11**	12	**11**
pmed26	600	5	38	59	50	43	43	39	39	**38**
pmed27	600	10	32	54	38	38	37	**32**	**32**	**32**
pmed28	600	60	18	28	26	25	26	19	19	**18**
pmed29	600	120	13	20	18	18	18	**13**	14	**13**
pmed30	600	200	9	14	12	13	12	10	10	**9**
pmed31	700	5	30	50	38	36	34	**30**	**30**	**30**
pmed32	700	10	29	42	38	37	35	30	30	**29**
pmed33	700	70	15	25	22	23	22	**16**	**16**	**16**
pmed34	700	140	11	17	16	16	16	**11**	**11**	**11**
pmed35	800	5	30	38	37	34	34	**30**	**30**	**30**
pmed36	800	10	27	39	37	34	33	**28**	**28**	**28**
pmed37	800	80	15	25	22	23	22	**16**	**16**	**16**
pmed38	900	5	29	47	37	31	31	**29**	**29**	**29**
pmed39	900	10	23	39	30	28	27	24	24	**23**
pmed40	900	90	13	21	18	19	18	**14**	**14**	**14**

to get an upper bound on the optimal solution size. To get best known solutions for these instances, we runned the TS1, TS2, VNS, IM-10, IM-100 and RI local search algorithms from the

TABLE 9. Results obtained by each tested algorithm over some *small* instances from TSPLib. The best found solutions are highlighted, and the optimal solutions are underlined.

				solution size						
instance	n	k	OPT	Gon	HS	Gon+	HS+	CDS	CDSh	CDSh+
kroA200	200	5	911.41	1227.91	1247.46	1108.84	1005.56	**911.41**	1066.82	**911.41**
		10	598.81	789.51	826.00	710.29	713.31	**599.47**	650.49	**599.47**
		20	389.30	533.02	454.58	466.00	452.92	414.01	431.61	**413.05**
		40	258.25	336.36	316.07	305.63	308.84	267.55	280.58	**261.19**
gr202	202	5	19.38	26.93	20.96	20.96	20.96	**19.38**	20.96	**19.38**
		10	9.33	14.98	12.08	11.17	11.08	**10.02**	10.64	**10.02**
		20	5.56	6.91	6.65	6.44	6.61	**5.66**	6.08	**5.66**
		40	2.97	3.98	3.70	3.53	3.70	**3.15**	3.30	**3.15**
pr226	226	5	3720.55	5629.38	4591.84	4601.08	4850.00	**3720.55**	4103.65	**3720.55**
		10	2326.47	3224.90	3324.15	3070.01	2761.34	**2439.77**	2750.45	**2439.77**
		20	1365.65	1952.56	2000.00	1703.85	1900.65	**1365.65**	1500.00	**1365.65**
		40	650.00	850.00	960.46	707.10	850.00	672.68	700.00	**670.82**
pr264	264	5	1610.12	2802.23	1947.43	1802.77	1802.77	**1610.12**	**1610.12**	**1610.12**
		10	850.00	1350.92	1350.92	1217.57	1274.75	**884.59**	**884.59**	**884.59**
		20	514.78	738.24	761.57	707.10	715.89	**538.51**	**538.51**	**538.51**
		40	316.22	400.00	500.00	360.55	424.26	350.00	350.00	**335.41**
pr299	299	5	1336.27	2055.29	1858.17	1700.18	1613.42	1382.93	1456.23	**1364.73**
		10	888.83	1188.64	1097.92	1033.46	1070.04	**901.38**	955.24	915.76
		20	559.01	806.22	804.39	691.46	731.55	**602.07**	651.92	607.30
		40	355.31	480.23	527.37	436.60	477.62	**400.00**	425.73	**400.00**
lin318	318	5	1101.33	1653.88	1412.95	1371.83	1351.76	1104.42	1225.55	**1101.33**
		10	743.21	1111.11	1141.32	906.53	922.00	**755.17**	800.56	**755.17**
		20	496.45	645.60	618.93	595.62	605.70	528.44	537.33	**501.81**
		40	315.91	394.00	440.81	388.49	414.73	330.05	341.23	**326.79**
pr439	439	5	3196.58	4445.01	3926.98	3610.57	3883.05	**3222.18**	3250.00	**3222.18**
		10	1971.83	2537.34	2631.53	2432.33	2452.04	**1971.83**	2081.46	**1971.83**
		20	1185.59	1869.82	1667.70	1530.52	1654.72	1211.66	1375.00	**1200.00**
		40	671.75	936.08	980.43	850.00	950.00	725.00	766.89	**707.99**
pcb442	442	5	1024.74	1386.54	1336.00	1252.99	1272.79	**1063.01**	1140.17	1077.03
		10	670.82	982.87	940.69	894.42	822.80	**707.10**	781.02	**707.10**
		20	447.21	623.61	599.08	565.68	570.08	**500.00**	**500.00**	**500.00**
		40	316.22	400.00	424.26	372.02	412.31	344.81	360.55	**325.72**
d493	493	5	752.90	1173.23	968.75	931.56	852.10	783.26	882.70	**755.92**
		10	458.30	639.18	608.08	580.47	572.06	**489.60**	544.94	**489.60**
		20	312.74	424.51	397.46	367.93	375.24	**336.15**	359.32	337.44
		40	206.01	259.43	255.93	243.03	273.09	217.01	229.39	**215.90**
d657	657	5	880.90	1352.08	1214.30	1110.72	1086.09	890.17	1124.09	**880.90**
		10	574.74	769.27	786.05	725.72	717.85	**601.74**	624.56	609.86
		20	374.70	508.85	546.35	464.27	458.66	426.23	449.81	**423.91**
		40	249.51	333.35	364.34	309.58	346.42	**278.35**	295.20	**278.35**

literature during a combined total time of 986,000 seconds using different seeds [30], [51]. As in the previous experiments, the Gon, HS, Gon+, and HS+ algorithms are among the most efficient, with an average execution time of 0.006, 0.127, 19.172, and 458.953 seconds, respectively. The CDS and CDSh+ algorithms were not executed over this set of intances because they become very inefficient with instances of this size. The CDSh algorithm had an average execution time of 2.013 seconds. With respect to the quality of the generated solutions, the HS+ algorithm had the best performance among the 2-approximated algorithms, with an experimental average approximation factor of 1.318. From all the tested polynomial time algorithms, the CDSh+ algorithm had the best performance, with an experimental average approximation factor of 1.170. Finally, the BEA exact algorithm had the worst execution time performance, generating exact solutions and upper bounds in an average execution time greater than 60,000 seconds. Table 11 shows the detailed results obtained by the tested algorithms over the more difficult *medium-size* TSPLib instances.

Table 7 and Figure 12 show the mean and standard deviation of the experimental approximation factors obtained by the approximation algorithms when executed over a set of *large-size* instances from TSPLib [50]. This set consists of three instances with 24,978 to 71,009 vertices, and values of k from 25 to 100. The optimal solutions of these instances are unknown. Thus, we used the best known solutions found

TABLE 10. Results obtained by each tested algorithm over the u1060, u1817, and mu1979 instances from TSPLib. The best found solutions are highlighted, and the optimal solutions are underlined.

				solution size					
instance	n	k	OPT	Gon	HS	Gon+	HS+	CDSh	CDSh+
u1060	1060	10	2273.08	3274.69	3335.97	3046.01	2864.33	2860.83	**2475.60**
		20	1580.79	2192.22	2139.81	1886.27	1963.47	1780.21	**1698.71**
		30	1207.77	1639.88	1700.79	1511.28	1604.41	1522.45	**1299.08**
		40	1020.56	1388.80	1359.94	1251.99	1299.41	1217.51	**1139.49**
		50	904.92	1166.66	1237.85	1070.32	1273.94	1050.75	**1000.70**
		60	781.16	1076.50	1151.99	985.41	1100.90	943.74	**906.22**
		70	710.74	999.84	1004.30	900.72	943.14	854.93	**790.13**
		80	652.16	906.22	961.42	806.57	921.36	752.26	**721.37**
		90	607.86	806.56	886.61	751.13	874.99	720.92	**671.17**
		100	570.00	720.92	828.09	706.62	791.08	652.43	**632.88**
		110	538.83	671.17	764.36	651.47	761.22	607.86	**583.32**
		120	510.27	651.74	710.59	632.12	699.52	582.91	**565.71**
		130	499.65	632.86	696.84	582.91	670.52	552.70	**538.22**
		140	452.46	600.50	671.17	565.77	667.56	514.59	**500.19**
		150	447.00	570.01	699.52	538.83	632.12	500.19	**495.01**
u1817	1817	10	457.90	632.68	614.33	548.31	578.38	505.78	**475.54**
		20	309.01	478.05	437.00	359.20	381.00	393.90	**338.89**
		30	240.98	323.77	349.20	296.22	330.20	296.21	**283.98**
		40	209.44	281.98	297.57	255.27	287.35	254.00	**236.22**
		50	184.90	255.24	273.54	228.61	253.99	218.49	**209.43**
		60	162.63	227.18	228.61	204.76	218.49	203.20	**193.42**
		70	148.10	203.20	215.89	184.90	203.20	183.16	**179.59**
		80	136.77	184.92	198.37	163.32	193.84	165.59	**152.41**
		90	129.50	170.39	190.41	160.65	179.59	152.39	**148.09**
		100	126.99	160.64	170.39	152.38	162.65	143.66	**136.77**
		110	109.24	148.12	170.28	148.09	160.64	130.74	**129.50**
		120	107.76	148.09	160.64	136.77	152.38	127.00	**126.99**
		130	104.72	136.79	148.10	129.50	148.09	119.78	**113.59**
		140	101.60	129.52	148.09	127.01	136.79	107.77	**107.76**
		150	91.60	127.01	136.79	126.99	129.50	107.76	**107.75**
mu1979	1979	10	1160.69	1606.55	1327.48	1364.01	1327.48	1237.49	**1160.69**
		20	750.52	1013.93	970.82	903.84	948.97	806.67	**768.29**
		30	552.01	741.09	769.01	683.94	721.84	638.55	**598.33**
		40	448.45	600.92	637.65	559.89	600.92	539.80	**467.85**
		50	380.89	511.26	560.25	471.61	539.80	430.71	**400.34**
		60	337.06	429.29	440.01	402.61	448.45	374.53	**371.08**
		70	305.60	393.65	431.21	363.73	401.38	336.66	**320.20**
		80	265.59	364.00	372.67	331.32	366.90	301.85	**284.80**
		90	238.16	320.59	348.44	300.31	343.59	267.45	**257.97**
		100	220.32	295.25	320.15	268.84	320.15	241.53	**235.08**
		110	203.44	258.73	301.84	243.16	290.28	225.75	**218.82**
		120	188.56	238.62	271.31	226.06	263.52	212.13	**203.44**
		130	173.69	221.30	247.16	210.41	242.04	200.70	**186.33**
		140	160.88	210.29	233.93	200.70	227.76	186.33	**179.50**
		150	152.02	197.22	212.52	189.20	212.52	171.47	**164.56**

by an hybrid VNS algorithm from the literature [31]. For this set of instances we only tested the Gon, Gon+, HS, and CDSh algorithms. The CDS, HS+, CDSh+, and BEA algorithms were not executed over this set of intances because they become very inefficient with instances of this size. Besides, since the adjacency matrix for these instances is very large, we did not stored such matrix. Instead, we computed every edge cost every time needed by the algorithm. Because of this, we did not executed a binary search over the set of edge costs for the HS and CDSh algorithms. Instead, we executed a bisection search between 0 and the most expensive edge cost. As in the previous experiments, the Gon and HS algorithms are among the most efficient, with an average execution time of 0.019 and 0.584, respectively. The CDSh and Gon+ algorithms had an average execution time of 689.723 and 914.406 seconds, respectively. With respect to the quality of the generated solutions, the Gon+ algorithm had the best performance among all the algorithms, with an experimental average approximation factor of 1.158. Table 12 shows the detailed results obtained by the tested algorithms over the *large-size* TSPLib instances.

TABLE 11. Results obtained by each tested algorithm over the pcb3038, nu3496, and ca4663 instances from TSPLib. The best found solutions are highlighted, optimal solutions are underlined, and best known solutions have an asterisk at the end.

				solution size				
instance	n	k	OPT	Gon	HS	Gon+	HS+	CDSh
pcb3038	3038	10	728.54	1101.18	970.18	890.99	891.98	**806.98**
		20	493.03	731.60	692.97	594.75	608.88	**582.60**
		30	393.49	542.27	529.07	**502.31**	536.15	508.83
		40	336.42*	464.48	483.00	426.00	451.11	**411.91**
		50	298.20*	406.28	421.75	366.00	394.03	**364.11**
		60	280.79*	363.74	365.85	**327.39**	358.51	343.11
		70	256.70*	324.22	358.51	309.07	340.01	**306.91**
		80	239.20*	306.16	338.24	292.15	314.24	**280.14**
		90	226.22*	291.24	309.00	273.38	295.19	**257.80**
		100	206.63*	273.00	300.60	258.62	283.38	**243.50**
		110	206.49*	262.61	268.92	244.86	264.00	**233.34**
		120	197.67*	246.91	271.02	233.63	258.48	**221.60**
		130	191.94*	233.63	255.99	221.82	249.23	**212.04**
		140	181.39*	224.08	248.47	209.74	244.73	**205.00**
		150	164.77*	218.19	234.82	202.82	232.00	**196.65**
nu3496	3496	10	756.63	1124.10	1025.36	970.82	926.16	**901.38**
		20	519.08	691.41	703.36	638.79	648.47	**566.66**
		30	396.16	531.76	544.15	493.57	495.53	**480.73**
		40	327.44	432.68	457.16	416.66	440.14	**403.45**
		50	287.71	397.56	374.54	356.68	365.90	**343.59**
		60	258.73	350.39	343.59	327.44	339.93	**306.41**
		70	240.37	320.15	320.15	295.33	300.46	**275.88**
		80	222.36	300.46	285.30	268.73	274.87	**254.40**
		90	203.44	283.33	263.52	252.21	254.95	**238.62**
		100	194.36	254.95	247.77	231.54	247.76	**222.36**
		110	179.50	238.62	235.70	222.35	233.92	**213.43**
		120	169.96	224.22	222.35	208.83	219.22	**200.68**
		130	161.09	206.62	208.82	200.68	206.15	**184.08**
		140	153.65	201.37	200.69	190.02	200.00	**177.17**
		150	149.07	186.34	190.02	180.27	186.34	**169.96**
ca4663	4663	10	10498.80	15121.02	15016.03	12874.86	14250.63	**12184.71**
		20	7023.86	9508.62	10398.78	8475.86	8455.11	**8414.57**
		30	5360.78	7456.44	8021.23	6668.44	7347.40	**6548.38**
		40	4605.70	6134.82	6485.64	5572.07	6209.95	**5367.60**
		50	3955.06	5274.49	5716.69	4875.24	5500.73	**4663.15**
		60	3650.00	4686.53	4890.49	4286.57	4665.76	**4084.69**
		70	3223.29	4159.89	4544.26	3883.90	4544.26	**3850.90**
		80	2908.70	3883.90	4112.92	3572.77	4139.91	**3499.91**
		90	2652.56	3594.59	3774.84	3217.35	3774.84	**3154.40**
		100	2543.89	3247.64	3668.26	2995.64	3478.50	**2874.45**
		110	2336.78	2981.42	3355.26	2827.49	3158.27	**2672.18**
		120	2188.91	2820.99	3063.44	2673.99	3032.64	**2569.58**
		130	2056.76	2711.49	2981.42	2557.46	2981.42	**2414.76**
		140	1976.81	2557.39	2796.67	2466.05	2707.39	**2290.68**
		150	1894.14	2466.05	2670.46	2289.65	2643.64	**2107.26**

TABLE 12. Results obtained by each tested algorithm over the sw24978, bm33708, and ch71009 instances from TSPLib. The best found solutions are highlighted.

				solution size			
instance	n	k	BKS	Gon	HS	Gon+	CDSh
sw24708	24708	25	1329.37	1618.03	1834.54	**1550.35**	1634.69
		50	925.71	1240.18	1226.55	**1088.45**	1099.11
		75	759.02	926.16	957.13	**862.32**	885.84
		100	685.77	798.08	869.54	**741.89**	752.95
bm33708	33708	25	1183.80	1540.65	1687.86	**1414.40**	1490.70
		50	823.27	1071.34	1190.82	**951.31**	981.21
		75	683.94	819.38	945.45	**766.84**	808.46
		75	593.48	693.21	745.35	**649.14**	680.58
ch71009	71009	25	4428.72	6295.54	6519.04	**5608.06**	5805.86
		50	3107.56	4039.83	4019.30	**3641.98**	3649.21
		75	2554.32	3162.71	3583.91	**2985.82**	3015.24
		100	2168.97	2777.36	2928.88	**2524.63**	2601.59

V. CONCLUSIONS

Under the assumption that $P \neq NP$, and since solving the problem of computing $(2-\epsilon)$-approximation solutions for the vertex k-center selection problem belongs to the NP-Hard class, the 2-approximated Sh, Gon and HS algorithms are examples of best possible polynomial algorithms for the vertex k-center problem. The analysis presented in Section III revealed that these three algorithms achieve their approximation bound by taking advantage of the same structural property of the k-center problem, namely, that in order to get the approximation guarantees, it suffices with selecting centers that are sufficiently far away from each other.

Unfortunately, even though these algorithms have the best possible worst-case performance, they tend to perform poorly on practice. This is mainly because they take conservative decisions in order to keep their approximation guarantees. For this reason, in this paper we also considered the 3-approximated CDS algorithm which consistently outperforms the Gon and HS algorithms over the *de facto* benchmark data sets from the literature.

In Section III, we showed that it is easy to prove the correctness of the Gon and HS algorithms, by showing that they are actually clever implementations of the Sh algorithm. While the Sh algorithm requires the optimal solution size ahead of time, the Gon algorithm cleverly eliminates such requirement by always selecting the farthest vertex. For its part, the HS algorithm performs a binary search over the set of possible values of r until it finds an r value less than or equal to r^* such that it allows its inner cycle to create a solution of size $2r$. Similarly, the proof of correctness for the CDS algorithm (Lemma 9) has a very similar structure to that of the proof of correctness for the Sh algorithm (Lemma 4) because it also takes advantage of the same structural property. This way, it is easy to observe how all these algorithms rely on the same structural properties of the vertex k-center problem. To some extent, the Gon and HS algorithms are equivalent, and the CDS algorithm can be considered as a deterministic perturbation of the Gon algorithm. Besides, we present a novel tight example for the CDS algorithm, which completes the demonstration that this algorithm is actually a 3-approximated one.

The experimental analysis confirmed the theoretical results by showing that the algorithms that become more unpractical as the input grows are the CDS, HS+, and CDSh+ algorithms. Besides, the same experiments show that the Gon, HS, CDSh, and Gon+ algorithms are able to generate good solutions very efficiently. Among the 2-approximation algorithms, the Gon+ and HS+ algorithms present the best performance. Despite its sub-optimal approximation factor, the CDS algorithm and its heuristic versions CDSh and CDSh+ had the best empirical performance. Among all the tested polynomial time algorithms, the one that keeps a better balance between execution time (efficiency) and quality of the generated solution (efficacy) is the CDSh algorithm, generating solutions with an average approximation factor (average execution time) of 1.047 (in 0.014 seconds), 1.117 (in 0.013 seconds), 1.159 (in 0.314 seconds), 1.170 (in 2.013 seconds), and 1.193 (in 689.723 seconds) over the *pmed* instances from OR-Lib, and the *small*, *medium*, and *large* instances from TSP-Lib, respectively. It is important to remark that, even though CDSh outperforms Gon+ on most instances, Gon+ had the best performance when tested over the large instances from TSPLib, with an approximation factor (average execution time) of 1.158 (in 914.406 seconds).

The BEA exact algorithm, which naturally follows from the basic reduction from the vertex k-center problem to the minimum dominating set problem, had a good performance on most instances of up to 4663 vertices. This is remarkable since it shows that one of the simplest IP formulations for the vertex k-center problem can be competitive with the other formulations from the literature by using a commercial optimization software with its default tuning parameters (Gurobi) [48]. However, there are some pathological instances, such as pcb3038, where BEA presents an extremely poor performance, with an average execution time of the order of weeks just to get an upper bound on the optimal solution size. Of course, due to its nature, and under the assumption that $P \neq NP$, any exact algorithm for the NP-Hard vertex k-center problem will run in exponential time.

Finally, as a byproduct of our experimental analysis, we corroborate most exact solutions reported in the literature and we also present novel best known solutions for a few extra instances, such as pcb3038 with values of k from 50 to 150.

REFERENCES

[1] R. Z. Farahani and M. Hekmatfar, Eds., *Facility Location: Concepts, Models, Algorithms and Case Studies*. Heidelberg, Germany: Springer, 2009.

[2] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Oper. Res.*, vol. 12, no. 3, pp. 450–459, Jun. 1964.

[3] M. E. Dyer and A. M. Frieze, "A simple heuristic for the p-centre problem," *Oper. Res. Lett.*, vol. 3, no. 6, pp. 285–288, 1985.

[4] J. Chuzhoy, S. Guha, E. Halperin, S. Khanna, G. Kortsarz, R. Krauthgamer, and J. Naor, "Asymmetric k-center is log* n-hard to approximate," in *Proc. 36th Annu. ACM Symp. Theory Comput. (STOC)*, Chicago, IL, USA, 2004, pp. 538–551.

[5] S. Vishwanathan, "An O(log*n) approximation algorithm for the asymmetric p-center problem," in *Proc. 7th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, Atlanta, GA, USA, 1996, pp. 1–5.

[6] S. Khuller and Y. J. Sussmann, "The capacitated k-center problem," *SIAM J. Discrete Math.*, vol. 13, no. 3, pp. 403–418, 2000.

[7] D. Chakrabarty, R. Krishnaswamy, and A. Kumar, "The heterogeneous capacitated k-center problem," in *Proc. Int. Conf. Integer Program. Combinat. Optim.* Cham, Switzerland: Springer, 2017, pp. 123–135.

[8] P. Brass, C. Knauer, H.-S. Na, C.-S. Shin, and A. Vigneron, "The aligned k-center problem," *Int. J. Comput. Geometry Appl.*, vol. 21, no. 2, pp. 157–178, 2011.

[9] J. Könemann, Y. Li, O. Parekh, and A. Sinha, "An approximation algorithm for the edge-dilation k-center problem," *Oper. Res. Lett.*, vol. 32, no. 5, pp. 491–495, 2004.

[10] S. Khuller, R. Pless, and Y. J. Sussmann, "Fault tolerant k-center problems," *Theor. Comput. Sci.*, vol. 242, nos. 1–2, pp. 237–245, 2000.

[11] S. Chechik and D. Peleg, "The fault-tolerant capacitated k-center problem," *Theor. Comput. Sci.*, vol. 566, pp. 12–25, Feb. 2015.

[12] S. Chaudhuri, N. Garg, and R. Ravi, "The p-neighbor k-center problem," *Inf. Process. Lett.*, vol. 65, no. 3, pp. 131–134, 1998.

[13] A. Lim, B. Rodrigues, F. Wang, and Z. Xu, "k-center problems with minimum coverage," *Theor. Comput. Sci.*, vol. 332, nos. 1–3, pp. 1–17, 2005.

[14] Y. Xu, J. Peng, and Y. Xu, "The mixed center location problem," in *Combinatorial Optimization and Applications* (Lecture Notes in Computer Science), vol. 10043. Cham, Switzerland: Springer, 2016.

[15] A. D. López-Sánchez, J. Sánchez-Oro, and A. G. Hernández-Díaz, "GRASP and VNS for solving the p-next center problem," *Comput. Oper. Res.*, vol. 104, pp. 295–303, Apr. 2019.

[16] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems. I: The p-centers," *SIAM J. Appl. Math.*, vol. 37, no. 3, pp. 513–538, 1979.

[17] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theor. Comput. Sci.*, vol. 38, pp. 293–306, 1985.

[18] D. S. Hochbaum and D. B. Shmoys, "A best possible heuristic for the k-center problem," *Math. Oper. Res.*, vol. 10, no. 2, pp. 180–184, 1985.

[19] J. Plesník, "A heuristic for the p-center problems in graphs," *Discrete Appl. Math.*, vol. 17, no. 3, pp. 263–268, 1987.

[20] M. S. Daskin, *Network and Discrete Location: Models, Algorithms, and Applications*. Hoboken, NJ, USA: Wiley, 2011.

[21] J. A. Pacheco and S. Casado, "Solving two location models with few facilities by using a hybrid heuristic: A real health resources case," *Comput. Oper. Res.*, vol. 32, no. 12, pp. 3075–3091, 2005.

[22] A. Kaveh and H. Nasr, "Solving the conditional and unconditional p-center problem with modified harmony search: A real case study," *Scientia Iranica*, vol. 18, no. 4, pp. 867–877, 2011.

[23] C. Contardo, M. Iori, and R. Kramer, "A scalable exact algorithm for the vertex p-center problem," *Comput. Oper. Res.*, vol. 103, pp. 211–220, Mar. 2019.

[24] D. S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*. Boston, MA, USA: PWS, 1997.

[25] W.-L. Hsu and G. L. Nemhauser, "Easy and hard bottleneck location problems," *Discrete Appl. Math.*, vol. 1, no. 3, pp. 209–215, 1979.

[26] D. B. Shmoys, "Computing near-optimal solutions to combinatorial optimization problems," in *Combinatorial Optimization* (Discrete Mathematics and Theoretical Computer Science). AMS Publications, 1995, pp. 355–397.

[27] R. Rana and D. Garg, "The analytical study of k-center problem solving techniques," *Int. J. Inf. Technol. Knowl. Manage.*, vol. 1, no. 2, pp. 527–535, 2008.

[28] B. Robič and J. Mihelič, "Solving the k-center problem efficiently with a dominating set algorithm," *J. Comput. Inf. Technol.*, vol. 13, no. 3, pp. 225–234, 2005.

[29] J. G. Diaz, R. M. Mendez, R. M. Mendez, and R. Quintero, "A structure-driven randomized algorithm for the K-center problem," *IEEE Latin Amer. Trans.*, vol. 13, no. 3, pp. 746–752, Mar. 2015.

[30] N. Mladenović, M. Labbé, and P. Hansen, "Solving the p-center problem with tabu search and variable neighborhood search," *Netw., Int. J.*, vol. 42, no. 1, pp. 48–64, 2003.

[31] C. A. Irawan, S. Salhi, and Z. Drezner, "Hybrid meta-heuristics with VNS and exact methods: Application to large unconditional and conditional vertex p-centre problems," *J. Heuristics*, vol. 22, no. 4, pp. 507–537, 2016.

[32] W. Pullan, "A memetic genetic algorithm for the vertex p-center problem," *Evol. Comput.*, vol. 16, no. 3, pp. 417–436, 2008.

[33] T. Davidović, D. Ramljak, M. Šelmić, and D. Teodorović, "Bee colony optimization for the p-center problem," *Comput. Oper. Res.*, vol. 38, no. 10, pp. 1367–1376, 2011.

[34] J. Garcia-Diaz, J. Sanchez-Hernandez, R. Menchaca-Mendez, and R. Menchaca-Mendez, "When a worse approximation factor gives better performance: A 3-approximation algorithm for the vertex k-center problem," *J. Heuristics*, vol. 23, no. 5, pp. 349–366, 2017.

[35] M. S. Daskin, "A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results," *Commun. Oper. Res. Soc. Jpn.*, vol. 45, no. 9, pp. 428–436, 2000.

[36] T. Ilhan, F. A. Ozsoy, and M. C. Pinar, "An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems," Dept. Ind. Eng., Bilkent Univ., Ankara, Turkey, Tech. Rep., 2002.

[37] S. Elloumi, M. Labbé, and Y. Pochet, "A new formulation and resolution method for the p-center problem," *INFORMS J. Comput.*, vol. 16, no. 1, pp. 84–94, 2004.

[38] A. Al-Khedhairi and S. Salhi, "Enhancements to two exact algorithms for solving the vertex P-center problem," *J. Math. Model. Algorithms*, vol. 4, no. 2, pp. 129–147, 2005.

[39] D. Chen and R. Chen, "New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems," *Comput. Oper. Res.*, vol. 36, no. 5, pp. 1646–1655, 2009.

[40] H. Calik and B. C. Tansel, "Double bound method for solving the p-center location problem," *Comput. Oper. Res.*, vol. 40, no. 12, pp. 2991–2999, 2013.

[41] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer-Verlag, 2003.

[42] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1990.

[43] F. Grandoni, "A note on the complexity of minimum dominating set," *J. Discrete Algorithms*, vol. 4, no. 2, pp. 209–214, 2006.

[44] N. Fan and J.-P. Watson, "Solving the connected dominating set problem and power dominating set problem by integer programming," in *Proc. Int. Conf. Combinat. Optim. Appl. (COCOA)*, in Lecture Notes in Computer Science, vol. 7402. Berlin, Germany: Springer, 2012, pp. 371–383.

[45] E. Minieka, "The m-center problem," *SIAM Rev.*, vol. 12, no. 1, pp. 138–139, 1970.

[46] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.

[47] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, 1962.

[48] *Gurobi Optimizer Reference Manual*, Gurobi Optim., Houston, TX, USA, 2018.

[49] J. E. Beasley, "OR-library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, Nov. 1990.

[50] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[51] J. G. Diaz, R. M. Mendez, J. S. Hernandez, and R. M. Mendez, "Local search algorithms for the vertex k-center problem," *IEEE Latin America Trans.*, vol. 16, no. 6, pp. 1765–1771, Jun. 2018.

JESUS GARCIA-DIAZ received the B.S. degree in communications and electronic engineering from the Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME), CDMX, Mexico, in 2009, and the M.S. and Ph.D. degrees in computer science from the Instituto Politécnico Nacional, CDMX, in 2013 and 2017, respectively. He is currently a CONACYT Research Fellow with the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico.

ROLANDO MENCHACA-MENDEZ received the B.S. degree in electronic engineering from the Universidad Autónoma Metropolitana, CDMX, Mexico, in 1997, the M.S. degree from the Instituto Politécnico Nacional, CDMX, Mexico, in 1999, and the Ph.D. degree in computer engineering from the University of California at Santa Cruz, in 2009. He is currently a Professor with the Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN), CDMX.

RICARDO MENCHACA-MENDEZ received the B.S. degree in computer science from the Escuela Superior de Cómputo (ESCOM), CDMX, Mexico, in 2001, the M.S. degree from the Insituto Politécnico Nacional, CDMX, in 2005, and the Ph.D. degree in computer science from the University of California at Santa Cruz, in 2013. He is currently a Professor with the Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN), CDMX.

SAÚL POMARES HERNÁNDEZ received the Ph.D. degree in computer science and telecommunications from the Institute National Polytechnique de Toulouse, France. Since 1998, he has been researching in the field of distributed systems and partial order algorithms. He is currently the Dean of academic affairs with the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico. He is also an Honorary Researcher (Chercheur Affilié) with the Laboratory for Analysis and Architecture of Systems, CNRS, Toulouse, France.

JULIO CÉSAR PÉREZ-SANSALVADOR received the B.S. degree in Computer Science from the Benemérita Universidad Autónoma de Puebla, Mexico, in 2005, the M.S. degree in computer science from the Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla, in 2007, and the Ph.D. degree in applied mathematics from Manchester University, U.K., in 2016. He is currently a CONACYT Research Fellow with the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico.

NOUREDDINE LAKOUARI received the B.S. degree in physics and the M.S. and Ph.D. degrees in computational physics from the Mohammed V University of Rabat, Morocco, in 2009, 2011, and 2015, respectively. He is currently a CONACYT Research Fellow with the Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Puebla, Mexico.