University of Ljubljana
Institute of Mathematics, Physics and Mechanics
Department of Mathematics
Jadranska 19, 1000 Ljubljana, Slovenia

Preprint series, Vol. 39 (2001), 750

THE OBNOXIOUS CENTER PROBLEM ON WEIGHTED CACTUS GRAPHS

Blaž Zmazek Janez Žerovnik

ISSN 1318-4865

March 23, 2001

Ljubljana, March 23, 2001

The obnoxious center problem on weighted cactus graphs

Blaž Zmazek
University of Maribor
PEF, Koroška cesta 160
2000 Maribor
SLOVENIA
blaz.zmazek@uni-mb.si

Janez Žerovnik *
University of Maribor
FS, Smetanova 17
2000 Maribor
SLOVENIA
janez.zerovnik@imfm.uni-lj.si

Abstract

The obnoxious center problem in a graph G asks for a location on an edge of the graph such that the minimum weighted distance from this point to a vertex of the graph is as large as possible. Weights on the vertices can be also viewed as a sensibility of the graph vertices. When each vertex of a graph is valued with one of the c different marks (weights) with respect to its sensitivity, we are solving obnoxious center problem in a graph with marked vertices. In this paper we present an algorithm which finds the obnoxious center in a weighted cactus with marked vertices in $O(cn)$ time, where n is the number of vertices and c is the number of different marks.

Categories and Subject Descriptors: G.2.2 [**Mathematics of Computing**]: *Graph algorithms, Network problems;* F.2.2 [**Theory of Computation**]: *Computations on discrete structures*

Math. Subj. Class. (2000): 90B80, 90C27, 68R10, 05C90

Key words: Location problems, center problem, obnoxious facilities, linear-time algorithm

*And Institute of Mathematics, Physics and Mechanics, Department of Theoretical Computer Science, Jadranska 19, 1111 Ljubljana, Slovenia. Supported in part by the Ministry of Science and Technology of Slovenia under the grant J2-1015.

1 Introduction

Distance is an important concept in applications of graph theory to computer science, operations research, chemistry and other fields. The literature on it so rich that there is an entire book dedicated to it [2]. Examples of applications include important invariants of computer networks and multiprocessor topologies [1] and popular topological indices such as the Wiener number in chemistry [9]. It may be interesting to note that the Wiener polynomial has a meaning both in chemistry [8, 14] and in computer science [7].

Distances in general graphs can easily be computed in $O(mn)$ time and consequently some of the distance based problems are polynomial. However, more efficient algorithms exist for special classes of graphs. For example, the Wiener number can be computed in linear time on trees [13], diameters of double loop networks can be computed in $O(\log n)$ time [10, 16], constant time algorithms exist for computing the distance related invariants on fasciagraphs and rotagraphs [11], etc.

A standard distance based topic in combinatorial optimization and operations research are location problems [5]. Location on networks is a topic of great importance in fields such as transportation, communication and computer science. Recently obnoxious facility location problems received an increasing interest. Complexity issues regarding placement of several obnoxious facilities were considered by Tamir [15]. On a tree, his algorithm has time complexity $O(kn \log^2 n)$, where k is the diameter of the tree, which was an improvement over $O(n^2)$ algorithm of [6]. In [3] an algorithm for obnoxious center problem on a tree was given with time complexity $O(kn \log n)$. On a path and a star and on a tree with all weights equal, the same authors gave a linear algorithm [3].

In this paper we consider the case where there is c different weights, which we call *marks*. We give an algorithm of time complexity $O(cn)$. If the number of marks c is constant, which is a practical assumption, our algorithm has linear time complexity. This improves on the results of [3]. Using the same general idea, it is possible to devise linear algorithms for invariants such as the weighted Wiener number and the Szeged number, and perhaps for some other distance related invariants. In some cases this would improve the best known results, and we will give details elsewhere. In some other cases, linear algorithms are already known. For example, our result is comparable to the linear-time algorithm for solving the center problem on weighted cactus graphs [12]. A linear algorithm for the median

problem on weighted cactus graphs is given in [4].

The paper is organized as follows. In the next section we begin with definitions. In Section 3 we give details of the representation of cactus graphs. Section 4 gives a detailed description of the algorithm.

2 Definitions

A *weighted graph* $G = (V, E, w, \lambda)$ is a combinatorial object consisting of an arbitrary set $V = V(G)$ of *vertices*, a set $E = E(G)$ of unordered pairs $\{x, y\} = xy$ of distinct vertices of G called *edges*, and two *weighting functions*, w and λ. $w : V(G) \mapsto \mathbb{R}$ assigns positive real numbers (weights) to vertices and $\lambda : E(G) \mapsto \mathbb{R}$ assigns positive real numbers (lengths) to edges. As usual, we denote $n = |V|$ and $m = |E|$. In the problem we will be interested in the solution may either be a vertex of the graph or a point on some edge. Here we will therefore call *points* the elements of edges, which are interpreted as images of intervals, including endpoints of intervals (i.e. vertices of the graph). Later when considered time complexity of the algorithms we will assume that the function w has $c \leq n$ different values.

A *simple walk* from x to y is a finite sequence of distinct vertices $P = x_0, x_1, \ldots, x_\ell$ such that each pair x_{i-1}, x_i is connected by an edge and $x_0 = x$ and $x_\ell = y$. The *length* of the walk is the sum of weights of its edges, $l(P) = \sum_{i=1}^{\ell} \lambda(x_{i-1}x_i)$. For any pair of vertices x, y we define the *distance* $d(x, y)$ to be the minimum of lengths over all walks between x and y. If there is no such walk, we define $d(x, y) = \infty$. The distances between pairs of points on the edges can naturally be defined as follows. Each edge $(x, y) \in E$ has a positive length $\lambda(x, y)$. Thus we can interpret each edge as (the image of) a closed real interval $[0, \lambda(x, y)]$ of length $\lambda(x, y)$. For any point z in this interval we define its distance $d(x, z)$ to x as z and its distance $d(z, y)$ to y as $d(x, y) - z$.

A graph G is *connected*, if $d(x, y) < \infty$ for any pair of vertices x, y. A vertex v is a *cut vertex* if after removing v and all edges incident to it the resulting graph is not connected. A graph without a cut vertex is called *nonseparable*. A *block* is a maximal nonseparable graph. A *cycle* is an induced subgraph which is connected and in which every vertex is of degree two. A *cactus* is a graph in which every block of three or more vertices is a cycle. Alternatively, a cactus is a connecetd graph in which two cycles have at most one vertex in common.

The *center problem* on a graph G is to minimize

$$f(z) = \max_{x \in V} w(x)d(z,x)$$

over all points z on the edges of G. This objective function reflects the goal to locate a facility (center) z as close to the clients x as possible, so that the clients can quickly get services from the center in case of emergency.

In the case of an *obnoxious* facility one wants to maximize

$$g(z) = \min_{x \in V} w(x)d(z,x).$$

This objective function places the new location as far as possible from the sites (vertices) x of G. In this case important and highly sensitive sites (or highly obnoxious sites) receive small weights. In measuring the sensitivity of the sites we usually dispose of the finite set of marks. In this case each vertex of a graph can be valued with one of the c different marks (weights) with respect to its sensitivity.

3 DFS based representation of a cactus

There are several properties which generalize naturally from trees to cactuses. For example, the number of edges is linear, $O(m) = O(n)$. Also, if in a cactus cycles are considered as (super)-vertices, there is a unique shortest path between any two vertices. It is therefore not hopeless to have algorithms of the same time complexity as for trees.

Any cactus G can always be represented as a *rooted cactus*. This means that a vertex $v_0 \in V(G)$ is distinguished and called the *root* of G. (Any vertex may be chosen for the root.) All other vertices are indexed as $v_1, v_2, \ldots, v_{n-1}$ in a DFS order. It can easily be seen that in each cycle there is exactly one vertex v_j, which is the first vertex in DFS order on the cycle C, containing the edge v_iv_j. For this reason we call the vertex v_j the *root* of the cycle C. Clearly, each cycle in a cactus has a unique root, determined as a first vertex on the cycle in the DFS order. Moreover, each vertex on a cactus G which is a successor of a vertex with higher index is the root vertex of one or more cycles. Hence, all cycles in a cactus are determined with the root and its successor.

The rooted cactus can be represented with two arrays T_i and R_i, $i = 0, 2, \ldots, n-1$, where T_i denotes the unique predecessor of vertex v_i in the rooted tree, constructed with the DFS. The element R_i denotes the root vertex of the cycle containing v_i. If v_i does not lie on a cycle then $R_i = v_i$

and if v_i is on a cycle rooted at v_j, then $R_i = v_j \neq v_i$. Note that the array R describes all cycles in the cactus.

It is clear that when traversing the graph with a DFS algorithm, one can also obtain arrays F_i (father of v_i, i.e. each vertex has a pointer directing to its father), $ind(v_i) := |\{j|v_i = T_j\}|$ (in-degree, leaves of the tree have no son directing to them) within the same time complexity $O(m)$.

After a DFS run, each vertex v_i can be regarded as a root of a subcactus, which we will denote by G_i.

From the discussion above we infer that after a DFS run, one can get a representation of a cactus and, furthermore, one can decide whether a given graph is a cactus or not. More formally,

Lemma 3.1 *Let G be a cactus graph. In a DFS run, each new vertex v_i can have at most one neighbor among the visited vertices and the neighbor can only be on the direct path from the root v_0 to v_i.*

Proof. If, for some vertex two such neigbor existed, then there will be edges on two (or more cycles). Contradiction. A back neighbor which is not on a direct path from the root v_0 to v_i was visited already and hence the vertex v_i should have been visited as its son. Contradiction. □

Hence, if G is not a cactus graph, this can be observed while traversing G in the DFS order.

Lemma 3.2 *Cactus graphs can be recognized in linear time.*

Proof. DFS algorithm takes $O(m)$ time. □

4 The algorithm

With each vertex v_i of the graph G a c-tuple $D_i = (D_i(1), D_i(2), \ldots, D_i(k))$ will be computed, for which

$$D_i(k) = \min\{d(x_l, x_j) \mid w(x_l) = k\}$$

will eventually hold. After computing the distances of each vertex to closest k-weighted vertex, we can use the the linear algorithm for determining the optimal point on each edge. Optimal solution is then the point in which the best value over all edges is attained.

The objective function for points on edges is defined as follows. Assuming the c-tuples D_i are computed for each vertex v_i, then for the point z on edge v_iv_j the objective function is defined as

$$g(z) = \min_{x \in V} w(x)d(z,x) = \min\{g^i(z), g^j(z)\},$$

where

$$g^i(z) = \min\{k(D_i(k) + d(z, v_i)) \mid k = 1, \ldots c\}.$$

Recall that we want to find a point z with maximal $g(z)$.

Our linear algorithm for maximizing $g(z)$ over a cactus consists of four steps. First, a representation of the given weighted cactus is found. Then we compute D_i for each vertex. We start with edges incident to leaves and continue in reversed DFS order up to root. We correct D_i starting from root to leaves. Finally we compute $g(z)$ on every edge and we maximize it over the tree.

1. Find a representation of the (rooted) cactus G.
2. Traverse the cactus G in the reversed DFS order and compute the temporal $D_i(k)$.
3. Traverse the cactus G in the DFS order and compute the final $D_i(k)$.
4. For all edges find their optimal points and return the maximum.

We now give more details of each step.

Step 1: Cactus recognition
Run a DFS on G. By discussion above, it is clear that we can compute T_i, R_i, F_i (father of v_i, i.e. each vertex has a pointer directing to its father) and $ind(v_i) := |\{j | v_i = T_j\}|$ (in-degree, leaves of the tree have no son directing to them) in time $O(m)$.

Step 2: Computation temporal values of $D_i(k)$
Traverse the DFS tree in the reversed DFS order. Compute the temporal value $D_i(k)$ for a leave v_i and reduce the number of unvisited sons for its father by one. If the father has no more unvisited sons, put it in a FIFO (first in first out) queue. The computation of temporal $D_i(k)$ values is defined by:

1. if v_i is a leave then set $D_i(k) = 0$, if $w(v_i) = k$, and $D_i(k) = \infty$, if $w(v_i) \neq k$.
2. In general, a vertex v_i may have some sons with known temporal distances. Then we set $D_i(k)$ to be the minimum over the local value, and the corrected values of the sons. Formally,

$$D_i(k) = \min_{\{j \mid T_j = v_i \vee v_i = v_j\}} \{D_i^{(j)}(k)\},$$

where

- The local value is $D_i^{(i)}(k) = 0$, if $w(v_i) = k$, and $D_i^{(i)}(k) = \infty$, if $w(v_i) \neq k$.
- For each son v_j, for which $v_i v_j$ is not an edge of a cycle of G, (i.e., if $R_j = v_j$ and $T_j = v_i$) set
$$D_i^{(j)}(k) = D_j^{(k)} + \lambda(v_i, v_j).$$
- For each son v_j, for which $v_i v_j$ is an edge of a cycle C of G, (i.e., if $R_j = v_i$ and $T_j = v_i$), compute the distances $d(v_j, v_\ell)$ for all v_k on the cycle C rooted at v_j, and set
$$D_i^{(j)}(k) = \min_{\{\ell \mid R_\ell = R_j\}} \{D_\ell(k) + d(v_\ell, v_j)\}.$$

Lemma 4.1 *After Step 2, for every vertex v_i, and for every mark k, $D_i(k)$ is the minimal distance to a vertex v_ℓ with $w(v_\ell) = k$ in a subcactus with root v_i. More formally,*

$$D_i(k) = \min\{d(v_\ell, v_j) \mid w(v_\ell) = k \wedge v_\ell \in G_i\}.$$

Proof. Easy. □

Lemma 4.2 *Step 2 can be computed in $O(cm) = O(cn)$ time.*

Proof. One has to observe that the distances from one vertex of a cycle to all other vertices can be computed fast, i.e. in time proportional to the number of edges of the cycle. For example, by one walk in each direction around the cycle. By the way, the $D_i^{(j)}(k) = \min_{\{\ell \mid R_\ell = R_j\}} \{D_\ell(k) + d(v_\ell, v_j)\}$ can be computed within the same time complexity. Note however that c values have to be computed. Since each cycle has to be visited only once, when computing $D_i(k)$ for the root, the overall time complexity is $O(cm)$.

The rest is trivial. □

Step 3: Computation the final values of $D_i(k)\}$
Traverse the DFS tree in the DFS order. Compute the final values $D_i(k)$ as follows:

1. For each vertex v_i do

- For each son v_j, for which v_iv_j is not an edge of a cycle of G, (i.e., if $R_j = v_j$ and $T_j = v_i$) set

  $$D_j(k) = \min\{D_j(k), D_i(k) + \lambda(v_i, v_j).$$

- For each son v_j, for which v_iv_j is an edge of a cycle of G, (i.e., if $R_j = v_i$ and $T_j = v_i$), for each v_ℓ on the cycle compute $D_\ell(k)$ by traversing the cycle twice in each direction and at each move setting

  $$D_j(v_\ell) = \min\{D_j(\ell), D_N(v_\ell)(k) + \lambda(v_\ell, N(v_\ell)).$$

  $N(\ell)$ is a forward or backward neighbor of v_ℓ on the cycle, depending on the direction of the walk.

Lemma 4.3 *After Step 3, for every vertex v_i, and for every mark k, $D_i(k)$ is the minimal distance to a vertex $v_\ell \in V(G)$ with $w(v_\ell) = k$. More formally, $D_i(k) = \min_{v_\ell}\{d(v_\ell, v_j) \mid w(v_\ell) = k\}$.*

Proof. Sketch of the idea. After setting $D_j(v_\ell) = \min\{D_j(\ell), D_N(v_\ell)(k) + \lambda(v_\ell, N(v_\ell))$, the c-tuple will give the correct minimum computed over itself plus over the vertices, which are already taken into account in $D_N(v_\ell)(k)$. After traversing the cycle twice, the influence from (say, left) of every vertex to every other vertex is computed. Influence from other direction is computed by double walk around the cycle in the other direction. Details ommited in the abstract. □

Lemma 4.4 *Step 3 can be computed in $O(cn)$ time.*

Proof. As before, each cycle is considered only once and the computation on the cycle is linear in the length of the cycle. □

Step 4: Maximizing the $g(z)$
The problem of computation the maximum value of $g(z)$ on edge v_i, v_j can be transformed to the obnoxious center problem on a path P of at most $2c$ vertices placed on the real line with the following coordinates:

$$V(P) = \{-D_i(k) \mid D_i(k) < \infty\} \cup \{D_j(k) + d(v_i, v_j) \mid D_j(k) < \infty\}$$

Using the algorithm from [3] we find the maximal values of $g(z)$ on each edge. Step 4 is finished by choosing the point z with maximum value $g(z)$ over all edges of the tree.

Proposition 4.5 *The above algorithm solves the obnoxious center problem on a weighted cactus with marked vertices in $O(cn)$ time.*

Proof. The correctness of the algorithm is based on the fact that on each edge we consider all possible weighted distances in maximizing values $g(z)$.

Regarding to the time complexity of the algorithm, we observe that m maximizations of $g(z)$ on all edges of the cactus takes $O(cm) = O(cn)$ time.

Recalling the complexity of Steps 1 to 3 the whole algorithm runs in linear time. □

References

[1] J.-C.Bermond, F.Comellas and D.F.Hsu, Distributed Loop Computer Networks: A Survey, Journal of Parallel and Distributed Compututing 24 (1995) 2–10.

[2] F.Buckley and F.Harary, Distance in Graphs, Addison–Wesley, Redwood, CA, 1990.

[3] R.E.Burkard, H.Dollani, Y.Lin and G.Rote, The Obnoxious Center Problem on a Tree, Karl-Franzens-Universitaet Graz and Technische Universitaet Graz, Optimierung und Kontrolle, Bericht Nr. 128, 1998.

[4] R.E.Burkard and J.Krarup, A Linear Algorithm for the pos/neg-weighted 1-median Problem on Cactus, Computing 60 (1998) 193-215.

[5] M.S.Daskin, Network and Discrete Location : Models, Algorithms And Applications, Wiley, New York, 1995.

[6] Z.Drezdner and G.O.Wesolowsky, Location of Multiple Obnoxious Facilities, Transportation Science 19 (1985) 193-202.

[7] J.F.Fink and B.Elenbogen, Distance Distributions for Graphs Modeling Computer Science Networks, manuscript 2000.

[8] H.Hosoya, On some Counting Polynomials in Chemistry, Applications of Graphs in Chemistry and Physics, Discrete Applied Mathematics 19 (1988) 239–257.

[9] Discrete Mathematics 80 (1997). (Special issue on the Wiener index.)

[10] Y.Cheng and F.K.Hwang, Diameters of Weighted Double Loop Networks, Journal of Algorithms 9 (1988) 401–410.

[11] M.Juvan, B.Mohar and J.Žerovnik, Distance-related Invariants on Polygraphs, Discrete Applied Mathematics 80 (1997) 57–71.

[12] Y.-F.Lan, Y.-L.Wang and H.Suzuki, A Linear-Time Algorithm for Solving the Center Problem on Weighted Cactus Graphs, Information Processing Letters 71 (1999) 205–212.

[13] B.Mohar and T.Pisanski, How to Compute the Wiener Index of a Graph, Journal of Mathematical Chemistry 2 (1988) 267–277.

[14] B.E.Sagan, Y.-N.Yeh and P.Zhang, The Wiener Polynomial of a Graph, International Journal of Quantum Chemistry 60 (1996) 959–969.

[15] A.Tamir, Obnoxious Facility Location on Graphs, SIAM Journal of Discrete Mathematics 4 (1991) 550–567.

[16] J.Žerovnik and T.Pisanski, Computing the Diameter in Multiple-loop Networks, Journal of Algorithms 14 (1993) 226–243.