Position papers of the 2014 Federated Conference on Computer Science and Information Systems pp. 93–99
DOI: 10.15439/2014F325
ACSIS, Vol. 3

A self-stabilizing algorithm for locating the center of Cartesian product of K_2 and maximal outerplanar graphs

Halina Bielak
Institute of Mathematics
Maria Curie-Skłodowska University in Lublin
pl. Marii Curie-Skodowskiej 1, 20-031 Lublin, Poland
Email: hbiel@hektor.umcs.lublin.pl

Michał Pańczyk
Institute of Computer Science
Maria Curie-Skłodowska University in Lublin
ul. Akademicka 9, 20-033 Lublin, Poland
Email: mjpanczyk@gmail.com

***Abstract*—Self-stabilizing algorithms model distributed systems and allow automatic recovery of the system from transient failures. The center of a graph is the set of vertices with the minimum eccentricity. In this paper we investigate the self-stabilizing algorithm for finding the center of Cartesian product of K_2 and maximal outerplanar graphs.**

I. Introduction

LET $G = (V(G), E(G))$ be a simple, connected graph with the vertex set V and edge set E. The distance $d(i, j)$ between nodes i and j is the length of the shortest path connecting these two nodes. The maximum distance from a given vertex i to any other vertex in the graph G is called the eccentricity $ecc(i)$ of the vertex i. The set $C(G)$ of the vertices with the minimum eccentricity is called the center of the graph G (see Fig. 1).

The Cartesian product $G_1 \Box G_2$ of simple graphs G_1 and G_2 is a graph with the vertex set

$$V(G_1 \Box G_2) = V(G_1) \times V(G_2)$$

and the edge set

$$\begin{gathered} E(G_1 \Box G_2) = \{\{(u_1, u_2), (v_1, v_2)\} \mid \\ u_1, v_1 \in V(G_1) \wedge u_2, v_2 \in V(G_2) \wedge \\ ((u_1 = v_1 \wedge \{u_2, v_2\} \in E(G_2)) \vee \\ (u_2 = v_2 \wedge \{u_1, v_1\} \in E(G_1)))\}. \end{gathered}$$

In many cases it is important to locate the center of a graph, especially in distributed systems, where it allows placing a control center for the minimum cost of communication with peripherial nodes of the system. There are several known algorithms for locating centers in graphs. Bielak and Pańczyk [1] proposed algorithm finding weighted centroid in a tree. Farley [6] gave a linear time algorithm for vertex centers in trees. Also Hedetniemi et al. [11] gave linear time algorithm for center problems in trees. Goldman [9] and Kariv and Hakimi [12] gave an algorithm solving the center problem in networks.

A lot of research has been realized related to centers of graphs [14], [15], [3]. Distributed algorithms were also developed [2], [13]. In this paper we propose a self-stabilizing algorithm for locating the center of Cartesian product of complete graph K_2 and maximal outerplanar graph. The problem for maximal outerplanar graphs, in classical, sequential computing paradigm was solved by Farley and Proskurowski [7]. Let us define a maximal outerplanar graph as it was done in the mentioned paper [7], as a triangularization of a planar polygon (see Fig. 1). We define K_2 as a complete graph with two vertices.

In a maximal outerplanar graph $M = (V(M), E(M))$ every edge $p = \{i, j\} \in E(M)$ partitions the set of all vertices apart i and j into two distinct sets inducing connected subgraphs called sides. One of the sides may be empty. It is the case when the partitioning edge is a part of the exterior face of the graph. In fact, all the edges with one side empty form the unique Hamiltonian cycle.

Let us note that in a maximal outerplanar graph every two neighbors i and j have at most two common neighbors, each of them belonging to distinct

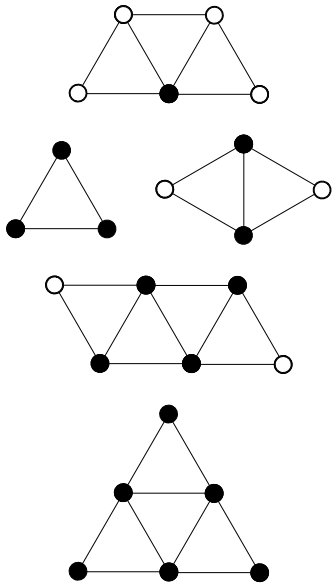

Fig. 1. Examples of centers (marked as black) in maximal outerplanar graphs.

sides of the edge $\{i, j\}$. Thus, it is sufficient to represent the side of the edge by one of the two common vertices adjacent to the edge. If a side is empty, we set $\emptyset$ as its representation.

In the Cartesian product of K_2 and any graph G, we can identify two layers of which every one is isomorphic to the graph G.

Farley and Proskurowski [7] introduced a notion of the edge eccentricity. Let us have the node i, its neighbor j and one of their common neighbors k ($\emptyset$ for the one nonexistent if applicable) in a graph G. For these three values we define $e(i, j, k)$ (edge eccentricity) in the following manner:

- the absolute value of $e(i, j, k)$ is equal to the eccentricity of the vertex i in the subgraph of G induced by $S_k \cup \{i, j\}$, where S_k is the side of the edge $\{i, j\}$ containing the vertex k,
- $e(i, j, k)$ is negative integer iff all vertices of $S_k \cup \{i, j\}$ at distance $d = |e(i, j, k)|$ from i lie at distance $d - 1$ from the vertex j.

The classical algorithm of Farley and Proskurowski [7] computes the edge eccentricity for every edge recursively using already computed values of the eccentricities for adjacent edges. It starts with outerface edges, for which the edge

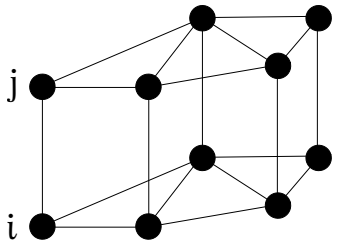

Fig. 2. An example of $K_2 \,\square$ maximal outerplanar graph.

eccentricity (on an empty side) is equal to -1. All the details can be found in [7].

In this paper we propose a self-stabilizing algorithm for finding the center in the Cartesian product of K_2 and a maximal outerplanar graph. In the following section we introduce the computational model used futher in the paper. In section III we show the notation useful in the algorithm. The algorithm is presented in section IV and its correctness and complexity is discussed in section V.

II. Computational Model

A notion of self-stabilizing algorithms on distributed systems was introduced by Dijkstra [4]. A survey in the topic can be found in the paper by Schneider [16], and further details in the book by Dolev [5]. The notions from graph theory not defined in this paper one can find in the book by Harary [10].

A distributed self-stabilizing system consists of a set of processes called computing nodes and communication links between them, which we can be modelled topologically by a graph. We assume that every node in the system runs the same algorithm and can change the state of the local variables. These variables determine the *local state* of a node. Moreover, nodes can observe the state of variables on themselves and their neighbor nodes. The state of all the nodes in the system determines the *global state*. In this paper we assume that every node has unique identificator (id).

Every self-stabilizing algorithm should have a class of global states defined, that are called the *legitimate states*, for which the system is stable, which means that no action should and can be done by the algorithm itself. Every other global state is called the *illegitimate* and for the algorithm to be correct there has to be some possibility

to make a move in this kind of state. Every possible sequence of moves made by the algorithm must end up with the legitimate state. Indeed, it is the aim of every self-stabilizing algorithm to bring the system to the legitimate (desirable) state, either after some alteration (from the outside of the system) of variables in the nodes had been done or after the system had been started.

Generally, an algorithm consists of a set of rules. A rule has the form:

label: **If** *guard*
then *assignment instructions*
where *definitions of objects*.

A *guard* is a logic predicate which can refer to variables in the node itself and its neighbors. A *label* and a *where* clause are optional. We say that a rule is *active* if its guard is evaluated to true. A node is *active* if it contains any active rule. If there is no active node in the graph, we say that the system is *stabilized*. Let us note that in a stabilized system no move can be made. One of the required property of a self-stabilizing algorithm is to make a system stabilized if and only if its state is legitimate.

We assume that active rules are triggered in an arbitrary order.

III. Notation

The main difficulty in enhancing the algorithm for locating the center in a maximal outerplanar graph to the Cartesian product of a maximal outerplanar graph and K_2 is to distinguish nodes that are placed in the same layer or not. Once it is done, the basic algorithm for a maximal outerplanar graphs can be easily adapted.

Theorem 1. *For every Cartesian product $G = K_2 \,\square\, M$, where M is a maximal outerplanar graph, the graph induced by $C(G)$ is the same as the Cartesian product of K_2 and the graph induced by $C(M)$.*

Proof: Let $x \in V(G)$, then $ecc_G(x) = ecc_M(x) + 1$. Thus, the center of G is the same as the sum of the centers of both layers of graph G. ∎

There are two types of neighbors (of any node $i \in V(G)$) in the graph (see Fig. 2). The first one consist of nodes in the same layer, which together form triangle faces of a maximal outerplanar graph. The second, say j, is a neighbor which belongs to the other layer, we call such two nodes i, j the *pairing* nodes.

We define the following predicate to determine whether any two neighbors are pairing nodes:

$$\text{pairing}(i,j) \;\Leftrightarrow\; j \in N(i) \wedge (N(i) \cap N(j)) = \emptyset.$$

We will use this predicate in the definition of $N_l(i)$, which we define as a set of neighbors from the same layer (maximal outerplanar graph):

$$N_l(i) = \{j | j \in N(i) \;\wedge \neg\text{pairing}(i,j)\}.$$

In the algorithm we will use the following notation:

- $n(i)$ — a variable storing the set of neighbor nodes (from the same layer) for the node i. Note that $n(i)$ is a variable, whose value may be incorrect at the beginning of the algorithm run, whereas $N_l(i)$ is a set which is determinable by the node i only, based on the topology of the network by looking at connections of node i; it can be computed only by the node i. Thus the $n(i)$ variable is set to allow a neighbor to determine other neighbors of the node i.
- $c(i,j)$ — a variable (stored in the node i) which stores the set of common neighbors for nodes i and j,
- $e(i,j,k)$ — a variable storing (in the node i) the edge eccentricity for the edge $\{i,j\}$ and the side containing the common neighbor k (of the nodes i and j; $k = \emptyset$ for an empty side),
- $opp(i,j,k)$ — a variable storing (in the node i) the representation of the side opposite to k (against the edge $\{i,j\}$),
- $v(i)$ — a variable storing (in the node i) the eccentricity of the vertex i, note that it is not the *edge* eccentricity, i.e. in a legitimate state $v(i) = \max_k |e(i,j,k)|$ for any $j \in N(i)$, and $v(i) \geq 1$ for any node i.
- $m(i,j,k)$ — a pair stored in the node i for inside the dual vertex $\{i,j,k\}$. After stabilization, its first element is the eccentricity of the center nodes. The second element of the pair is the direction, that the information about the eccentricity of the center comes from. If for the dual vertex $\{i,j,k\}$ the information comes from the region incident to the

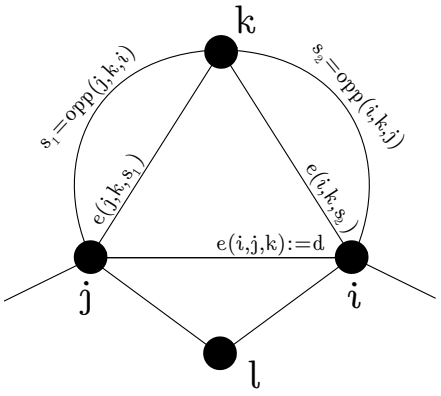

Fig. 3. Visualization of the rule 4.

edge $\{i, j\}$, then the direction is equal to $opp(i, j, k)$. In the case the information about the center eccentricity comes originally from the dual vertex $\{i, j, k\}$, then we set the direction to $\emptyset$.

IV. The Algorithm

In this section we present the rules of our algorithm (see Fig. 4). The first rule assigns the set of all neighbors of the node i in the same layer to its variable $n(i)$. Thanks to this, a node can know neighbors (in the same layer) of its neigbor, which is exploited in further rules. The second rule assigns in a node i the set of common neighbors in the same layer with node k: $c(i, k)$.

The rule 3a assigns outside edge eccentricity and sides of an edge. Rules 3b and 4 compute edge eccentricities according to [7] and both of them set $v(i)$ to proper value (see Fig 3).

The rule 5 propagates the minimum eccentricity through all the graph. The idea of the inside dual tree is used [8]. The information about the minimum eccentricity is propagated through the dual tree.

Note that in the rule 5 we used the function $MinEcc(i, j, k)$, returning the value of $m(i, j, k)$. The $MinEcc$ function is defined as follows (see Fig. 5):

Two projection functions are used in the above function: $fst((a, b)) \stackrel{\text{def}}{=} a$ and $snd((a, b)) \stackrel{\text{def}}{=} b$, which take the first and second element of the pair, respectively.

The $MinEcc$ function computes the minimum value over eccentricities and the direction that it comes from for the triangle region specified by three parameters i, j, k (see Fig. 5). The first step is to consider the node i itself as a candidate with the minimum value of the eccentricity available in the neighborhood. In this case the direction would

Function $MinEcc(i,j,k)$

```
1  v := v(i)
2  dir := ∅
3  if fst(m(k,i,j)) < v ∧ snd(m(k,i,j)) ∈
     {opp(k,j,i), ∅} then
4  |  (v, dir) := m(k,i,j)
5  end if
6  if fst(m(j,i,k)) < v ∧ snd(m(j,i,k)) ∈
     {opp(j,k,i), ∅ } then
7  |  (v, dir) := m(j,i,k)
8  end if
9  if fst(m(i,j,opp(i,j,k))) <
     v ∧ snd(m(i,j,opp(i,j,k))) ≠ k then
10 |  v := fst(m(i,j,opp(i,j,k)))
11 |  dir := opp(i,j,k)
12 end if
13 if fst(m(i,k,opp(i,k,j)) <
     v ∧ snd(m(i,k,opp(i,k,j))) ≠ j then
14 |  v := fst(m(i,k,opp(i,k,j))
15 |  dir := opp(i,k,j)
16 end if
17 return (v, dir)
```

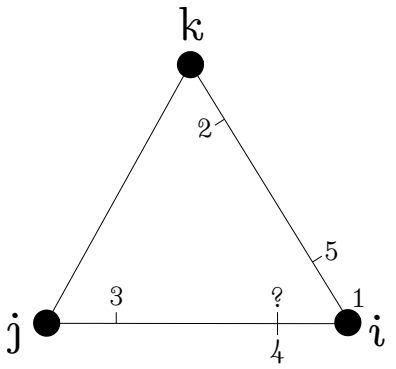

Fig. 5. The visualization of computation of the function $MinEcc(i, j, k)$. The numbers stand for the order of checking (and assigning if necessary) values of $m(\cdot, \cdot, \cdot)$. The order above is: 1. $(v(i), \emptyset)$ (lines 1–2 of the $MinEcc$ function), 2. $m(k, i, j)$ (lines 3–5), 3. $m(j, i, k)$ (lines 6–8), 4. $m(i, j, opp(i, j, k))$ (lines 9–12), 5. $m(i, k, opp(i, k, j))$ (lines 13–16). The question mark stands for $m(i, j, k)$, which is the computed value.

1: **If** $n(i) \neq N_l(i)$
then $n(i) := N_l(i)$
2: **If** $\exists_{k \in n(i)} c(i,k) \neq n(i) \cap n(k)$
then $c(i,k) := n(i) \cap n(k)$
3a: **If** $\exists_{j \in n(i)} (|c(i,j)| = 1 \wedge (e(i,j,\emptyset) \neq -1 \vee opp(i,j,\emptyset) \neq k \vee opp(i,j,k) \neq \emptyset))$
then $e(i,j,\emptyset) := -1$
$opp(i,j,\emptyset) := k$
$opp(i,j,k) := \emptyset$
where $\{k\} = c(i,j)$
3b: **If** $\exists_{j \in n(i)} (|c(i,j)| = 1 \wedge (e(i,j,k) \neq d \vee v(i) \neq \max(|e(i,j,k)|, 1))$
then $e(i,j,k) := d$
$v(i) := \max(|e(i,j,k)|, 1)$ **where**

$$q = \begin{cases} -(1 + e(j,k,opp(j,k,i))) & \text{if } e(j,k,opp(j,k,i)) > 0, \\ e(j,k,opp(j,k,i)) & \text{otherwise} \end{cases}$$

$$d = \begin{cases} |e(i,k,opp(i,k,j))| & \text{if } |e(i,k,opp(i,k,j))| \geq |q|, \\ q & \text{otherwise} \end{cases}$$

$\{k\} = c(i,j)$
4: **If** $\exists_{j \in n(i)} (|c(i,j)| = 2 \wedge \exists_{k \in c(i,j)} (e(i,j,k) \neq d \vee opp(i,j,k) \neq l \vee opp(i,j,l) \neq k \vee v(i) \neq \max(|e(i,j,k)|, |e(i,j,l)|)))$
then $e(i,j,k) := d$
$opp(i,j,k) := l$
$opp(i,j,l) := k$
$v(i) := \max(|e(i,j,k)|, |e(i,j,l)|)$
where

$$q = \begin{cases} -(1 + e(j,k,opp(j,k,i))) & \text{if } e(j,k,opp(j,k,i)) > 0, \\ e(j,k,opp(j,k,i)) & \text{otherwise} \end{cases}$$

$$d = \begin{cases} |e(i,k,opp(i,k,j))| & \text{if } |e(i,k,opp(i,k,j))| \geq |q|, \\ q & \text{otherwise} \end{cases}$$

$\{k,l\} = c(i,j)$
5: **If** $\exists_{j,k \in n(i)} (k \in c(i,j) \wedge m(i,j,k) \neq MinEcc(i,j,k))$
then $m(i,j,k) := MinEcc(i,j,k)$

Fig. 4. The self-stabilizing algorithm for finding the center in $K_2 \,\square$ maximal outerplanar graph.

be $\emptyset$ as it does not come from other region. In the second and third step, the node i checks the neighbor nodes k and j as a candidates for the minimum value. If the values in the nodes k or j come from regions incident to the edges $\{i,k\}$ or $\{i,j\}$, they are not trusted. It is to ensure that no wrong value can last in the region infinitely long time (number of moves). And last two steps, the values from two neighbor regions (incident to i) are checked.

V. Correctness and complexity

Now we prove some properties of the algorithm. We assume that n is the number of nodes in a layer (a maximal outerplanar graph) of the graph.

Lemma 1. *Algorithm consisting of rules 1–4 stabilizes in $\mathcal{O}(n^2)$ number of moves.*

Proof: The stabilization of the rule 1 is obvious as the guard does not depend on variables in neighbor nodes. So the rule 1 gets inactive in finite time. The rule 2, depends only on static (after stabilizing of the rule 1) information computed by the rule 1. Hence it stabilizes in limited by a constant number of moves per node, as rule 1 does.

The same applies to rule 3a, as it depends on variable values computed by two former rules,

because it is for an outerface edge (i.e. the edge belonging to Hamilton cycle), which is an initial case of the recursive classical algorithm. Once the $c(i,j)$ is properly computed in the node i, it never changes. Thus if any of the variables $e(i,j,\emptyset)$, $opp(i,j,\emptyset)$ or $opp(i,j,k)$ is in a wrong state, then all are correctly computed and also never change.

Now all the nodes have got rules 3a and 3b inactive. Then we consider the rule 4. Note that this rule is applicable only for graphs bigger than a triangle. Suppose there are two adjacent edges lying on an outerface of the graph. There has to be the third edge, which is also adjacent to them, and the edge eccentricities of this edge stabilize with rule 4. This is the first layer of proper rule 4 computation. Each next layer of proper computation of rule 4 depends on a previous layer. We have a finite graph, so the rule 4 stabilizes. As each layer of computation of rule 4 takes $\mathcal{O}(n)$ moves and there are $\mathcal{O}(n)$ layers, it all takes $\mathcal{O}(n^2)$ moves. The last layer of computation stabilizes by rule 3b, as it reaches the another outerface of the graph. ■

The following lemma describes the situation after stabilization of rules 1–4.

Lemma 2. *If the phase 1–4 has stabilized a system, then phase 5 will stabilize in $\mathcal{O}(n^2)$ number of moves.*

Proof: The pessimistic case would be if every dual vertex had wrong value for variable $v(i)$ — for example as a result of start up of the system — and having also wrong values of $m(\cdot,\cdot,\cdot)$ for every dual vertex (representing a region of the layer, i.e. maximal outerplanar graph). Let us assume that every $v(i)$ is less than the proper minimum eccentricity and there are no nodes i, j such that $v(i) = v(j)$. Then the pessimistic order of propagation of the $m(\cdot,\cdot,\cdot)$ values would be when the value $v(i)$ spreads the first (for some i) which is the biggest among all the other $v(k)$ (for all nodes k except i) but still it is less then the proper minimum eccentricity.

The above propagation takes $\mathcal{O}(n)$ moves. Note that now the dual tree is filled with improper value of some $v(i)$. But there are $n-1$ wrong candidates of the minimum eccentricity to spread left. Once again, the pessimistic case would be when the next value to propagate was the maximum among all the candidates, which is less then the spread in the tree.

Each of these phases takes $\mathcal{O}(n)$ moves and there are $\mathcal{O}(n)$ phases, so all pases of rule 5 run in $\mathcal{O}(n^2)$ moves. ■

Now we can formulate the following theorem.

Theorem 2. *The algorithm takes $\mathcal{O}(n^4)$ number of moves to stabilize.*

Proof: The computation in each of layers is independent, so by Lemmas 1 and 2 we get the result. ■

VI. Conclusions

In this paper we proposed a self-stabilizing algorithm for finding the center of the Cartesian product of graph K_2 and a maximal outerplanar graph. We hope the method similar to the presented here can be applied to other classes of graphs. The open question is if there exists an algorithm with better complexity — number of moves bringing a system to the legitimate state.

References

[1] Bielak, H., M. Pańczyk, M.: "A self-stabilizing algorithm for finding weighted centroid in trees"; Annales UMCS Informatica, AI XII, 2 (2012), 27–37; http://dx.doi.org/10.2478/v10065-012-0035-x.

[2] Bruell, S. C., Ghosh, S., Karaata, M. H., Pemmaraju, V.: "Self-stabilizing algorithms for finding centers and medians of trees"; SIAM Journal on Computing, 29 (1999), 600–614; http://dx.doi.org/10.1145/197917.198130.

[3] Chepoi, V., Fevat, T., Godard, E.,Vaxès, Y.: "A self-stabilizing algorithm for the median problem in partial rectangular grids and their relatives"; Algorithmica, 62 (2012), 146–168; http://dx.doi.org/10.1007/s00453-010-9447-4.

[4] Dijkstra, E. W.: "Self-stabilizing in spite of distributed control"; Communications of the ACM, 17 (1974), 643–644; http://dx.doi.org/10.1145/361179.361202.

[5] Dolev, S.: "Self-stabilization"; The MIT Press (2000).

[6] Farley, A. M.: "Vertex centers of trees"; Transportation Science, 16 (1982), 265–280; http://dx.doi.org/10.1287/trsc.16.3.265.

[7] Farley, A. M., Proskurowski, A.: "Computation of the center and diameter of outerplanar graphs"; Discrete Applied Mathematics, 2 (1980), 185–191; http://dx.doi.org/10.1016/0166-218x(80)90039-6.

[8] Fleischner, H. J., Geller, D. P., Harary, F.: "Outerplanar graphs and weak duals"; Journal of the Indian Mathematical Society, 38 (1974), 215–219.

[9] Goldman, A. J.: "Minimax location of a facility in a network"; Transportation Science, 6 (1972), 407–418; http://dx.doi.org/10.1287/trsc.6.4.407.

[10] Harary, F.: "Graph Theory"; Addison-Wesley, 1972.

[11] Hedetniemi, S. M., Cockayne, E. J., Hedetniemi, S. T.: "Linear algorithms for finding the jordan center and path center of a tree"; Transportation Science, 15 (1981), 98–114; http://dx.doi.org/10.1287/trsc.15.2.98.

[12] Kariv, O., Hakimi, S. L.: "An algorithmic approach to network location problems. I: The p-centers"; SIAM J. Applied Mathematics, 37 (1979), 513–538; http://dx.doi.org/10.1137/0137040.

[13] Korach, E., Rotem, D., Santoro, N.: "Distributed algorithms for finding centers and medians in networks"; ACM Transactions on Programming Languages and Systems, 6 (1984), 380–401; http://dx.doi.org/10.1145/579.585.

[14] Laskar, R., Shier, D.: "On powers and centers of chordal graphs"; Discrete Applied Mathematics, 6 (1983), 139–147; http://dx.doi.org/10.1016/0166-218x(83)90068-9.

[15] Rosenthal, A., Pino, J.: "A generalized algorithm for centrality problems on trees"; JACM, 36 (1989), 349–361; http://dx.doi.org/10.1145/62044.62051.

[16] Schneider, M.: "Self-stabilization"; ACM Computing Surveys, 25, 1, (1993); http://dx.doi.org/10.1145/151254.151256.