JOURNAL OF COMPUTER AND SYSTEM SCIENCES 29, 48–53 (1984)

Finding Pseudoperipheral Nodes in Graphs*

JAN K. PACHL

Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

Received December 13, 1982; revised September 14, 1983

The SPARSPAK [5] algorithm for finding pseudoperipheral nodes in graphs with n nodes and e edges has the worst case time complexity $\Omega(e\sqrt{n})$. The function $e\sqrt{n}$ is also an asymptotic upper bound for the worst case time complexity of the problem of finding pseudoperipheral nodes.

1. OVERVIEW

SPARSPAK, Waterloo Sparse Linear Equations Package, contains a subroutine called Pseudoperipheral Node Finder, whose goal is to find a node with large eccentricity in a given sparse graph [4, 5]. In their book, George and Liu ask whether the execution time of the subroutine can be worse than linear in the number of edges [5, p. 75].

This paper answers that question: the *worst case* execution time of the subroutine on graphs with n nodes and e edges is at least $\Omega(e\sqrt{n})$. No upper bound of the same order seems to be known for the SPARSPAK algorithm, but there is another algorithm for finding pseudoperipheral nodes, whose worst case execution time is $O(e\sqrt{n})$.

2. THE SPARSPAK PSEUDOPERIPHERAL NODE FINDER

Let $G = (X, E)$ be a graph with the set X of nodes and the set E of edges. Assume that for every two nodes $x, y \in X$ there is a path from x to y; the length of the shortest such path is called the *distance* between x and y and denoted $d(x, y)$. The *eccentricity* of $x \in X$ is defined by

$$l(x) = \max\{d(x, y) \mid y \in X\},$$

and the *diameter* of G by

$$\delta(G) = \max\{l(x) \mid x \in X\} = \max\{d(x, y) \mid x, y \in X\}.$$

A node $x \in X$ is called *peripheral* if $l(x) = \delta(G)$.

* This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A7403.

0022-0000/84 $3.00

Experience shows that several node ordering algorithms used in sparse matrix computations perform well when their starting nodes have large eccentricity. Peripheral nodes are expensive to find; the best algorithms known have time complexity $O(M(n) \log n)$ for dense graphs [2] and $O(ne)$ for sparse ones [3]. (Here $M(n)$ is the time complexity of matrix multiplication.) SPARSPAK uses pseudoperipheral nodes instead. We say that $x \in X$ is a *pseudoperipheral node* if there exists $y \in X$ such that

$$l(x) = d(x, y) = l(y).$$

The term is used in a different meaning in [4], where $x \in X$ is said to be pseudoperipheral if $l(x)$ is "close" to $\delta(G)$. The present terminology is less vague, and it remains consistent: The pseudoperipheral node finder in SPARSPAK indeed finds a pseudoperipheral node.

The following description of the SPARSPAK pseudoperipheral node finder employs a function *Furthest—from* (x), which returns $y \in X$ such that $d(x, y) = l(x)$: if there are several such y then one is selected arbitrarily.

ALGORITHM S.

$x_0 :=$ any element of X
$j := 0$
$x_1 :=$ *Furthest—from* (x_0)
repeat
  $j := j + 1$
  $x_{j+1} :=$ *Furthest—from* (x_j)
until $d(x_{j+1}, x_j) = d(x_j, x_{j-1})$
claim x_j is pseudoperipheral

We first consider the question of how many times Algorithm S calls the function *Furthest—from.*

2.1. THEOREM. *If $w(n)$ denotes the worst case number of calls to Furthest—from by Algorithm* S *on graphs with n nodes, then*

$$w(n) = \Omega(\sqrt{n}).$$

Proof. There is a sequence of graphs $G_1, G_2, \ldots$, such that for each $k = 1, 2, \ldots$

(i) G_k has $n = k^2 + 9k + 3$ nodes and n edges;

(ii) there is a node x_0 of G_k such that Algorithm S starting at x_0 calls the function *Furthest–from* $2k + 1$ times.

Figures 1 and 2 show two graphs in the sequence, G_2 and G_3. For a general $k \geqslant 1$, the graph G_k consists of a cycle whose nodes are, consecutively, $y_0, y_1, \ldots, y_{6k+1}$, and

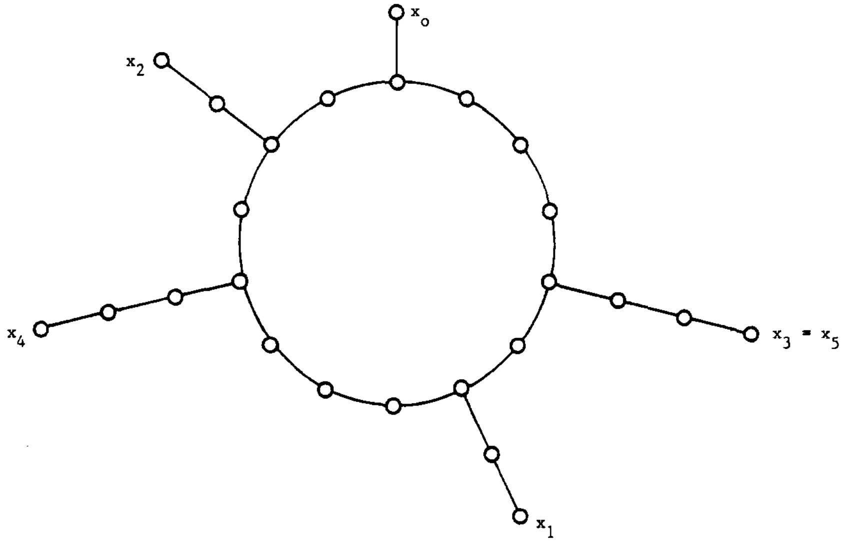

FIG. 1. The graph G_2.

linear segments attached to certain nodes in the cycle. A segment of length s is attached to the node y_j if and only if either $j = 2i$, $s = i + 1$ and $0 \leqslant i \leqslant k$, or $j = 3k + 2i$, $s = i + 1$ and $1 \leqslant i \leqslant k$.

From (i) and (ii) it follows that on a graph with $n = k^2 + 9k + 3$ nodes and n edges the algorithm makes $2\sqrt{n + 69/4} - 8 = 2\sqrt{n} + O(1)$ calls to the function. ∎

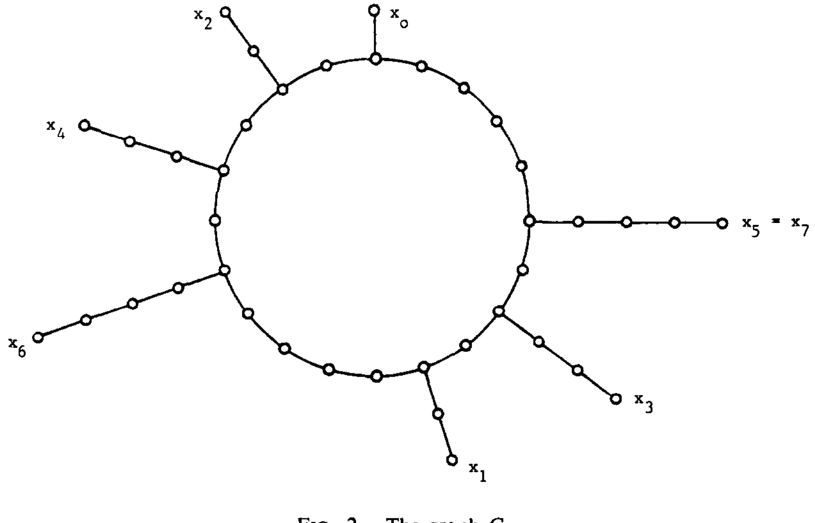

FIG. 2. The graph G_3.

Can one claim that Algorithm S is in some sense the best algorithm for finding pseudoperipheral nodes? The following conjecture is a rather weak form of the claim; it states that Algorithm S is not worse (in terms of the worst case execution time, modulo constant) than the algorithm in the next section. Conjecture:

> *There is a constant c such that, for every graph on n nodes and for every starting node x_0, Algorithm S calls the function Furthest—from at most $c\sqrt{n}$ times.*

3. The Worst Case Execution Time

In SPARSPAK, the node $y = $ *Furthest—from* (x) is computed by the breadth first search [3, p. 12]. The graph is represented by its incidence lists [3, p. 4]. If we assume the uniform cost criterion [1, 1.3] then one call to *Furthest—from* requires time proportional to e, the number of edges.

Hence the conjecture at the end of Section 2 states that the worst case execution time of Algorithm S for the graphs with n nodes and e edges is $O(e\sqrt{n})$; and from 2.1 it follows that $\Omega(e\sqrt{n})$ is a lower bound for the algorithm.

Although we do not know whether the complexity of Algorithm S is really $O(e\sqrt{n})$, we are now going to see that there is another algorithm, whose complexity is not worse than $O(e\sqrt{n})$.

Let $G = (X, E)$ be a graph with n nodes and e edges, and let k be a positive integer. We say that a set $Y \subseteq X$ is *k-discrete* if $d(x, y) > k$ whenever $x, y \in Y, x \neq y$.

3.1. Lemma. *There is an algorithm that constructs a maximal k-discrete set Y of nodes, and whose worst case execution time is O(me), where m is the cardinality of Y.*

Proof. Denote

$$B_k(x) = \{y \in X \mid d(x, y) \leqslant k\}.$$

If $B_k(x)$ is computed by the breadth first search, then the worst case execution time of the following algorithm is $0(me)$.

$S := \varnothing$
repeat
    $x :=$ any element of X
    $S := S \cup \{x\}$
    $X := X - B_k(x)$
until $X = \varnothing$
claim S is a maximal k-discrete set ∎

3.2. Lemma. *If $n \geqslant k/2$ then every k-discrete set $Y \subseteq X$ has at most $2n/k$ nodes.*

Proof. Let h be the largest integer not exceeding $k/2$.

The sets $B_h(x)$ and $B_h(y)$ are disjoint when $x, y \in Y$, $x \neq y$. Moreover, if $n \geqslant k/2$ then every $B_h(x)$ has at least $k/2$ elements (because G is connected). Hence the cardinality of Y is at most $n/(k/2) = 2n/k$. ∎

3.3. LEMMA. *There is an algorithm to find, for every $Y \subseteq X$, two nodes $x_0, y_0 \in Y$ such that*

$$d(x_0, y_0) = \max\{d(x, y) | x, y \in Y\};$$

the worst case execution time of the algorithm is $O(me)$, where m is the cardinality of Y.

Proof. All distances $d(x, y)$ for a given x can be computed by the breadth first search starting at x, which requires time $O(e)$. Therefore all the distances $d(x, y)$, $x, y \in Y$, can be computed in time $O(me)$. ∎

We are ready to construct the $O(e\sqrt{n})$ algorithm for finding pseudoperipheral nodes.

3.4. THEOREM. *There is an algorithm that finds a pseudoperipheral node in worst case time $O(e\sqrt{n})$.*

Proof. Let k be the smallest integer not smaller than $\sqrt{n}$.

The algorithm has three parts:

1. Find a maximal k-discrete set $Y \subseteq X$.
2. Find $x_0, y_0 \in Y$ such that

$$d(x_0, y_0) = \max\{d(x, y) \mid x, y \in Y\}.$$

3. Execute Algorithm S with starting node x_0.

By 3.1, 3.2 and 3.3, steps 1 and 2 can be executed in worst case time $O(e\sqrt{n})$. To estimate the execution time of step 3, observe that for any two nodes $x, y \in X$ there are $x', y' \in Y$ such that $d(x, x') \leqslant k$ and $d(y, y') \leqslant k$ (because Y is maximal k-discrete). Therefore

$$l(x_0) \geqslant d(x_0, y_0) \geqslant \delta(G) - 2k.$$

The sequence $x_0, x_1, \ldots$ generated in step 3 satisfies

$$\delta(G) - 2k \leqslant l(x_0) < l(x_1) < \cdots \leqslant \delta(G).$$

Hence the algorithm in step 3 repeats its loop at most $2k$ times. It follows that the worst case execution time for step 3 is $O(e\sqrt{n})$. ∎

4. Concluding Remarks

The *worst case* time cost of the algorithm in Section 3 is $O(e\sqrt{n})$, which is not worse than the *worst case* time cost of Algorithm S. Nevertheless, Algorithm S seems to execute in time $O(e)$ on "typical" graphs arising in sparse matrix computations, and is therefore better in practice.

The space cost of both algorithms is dominated by the memory needed to store the graph; it is proportional to $n + e$.

Acknowledgments

I wish to thank K. Booth, B. Danloy, A. George and D. Rotem for their comments and helpful suggestions.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1974.
2. K. S. Booth and R. J. Lipton, Computing extremal and approximate distances in graphs having unit cost edges, *Acta Inform.* **15** (1981), 319–328.
3. S. Even, "Graph Algorithms," Computer Science Press, Potomac, Md., 1979.
4. A. George and J. W. H. Liu, An implementation of a pseudoperipheral node finder, *ACM Trans. Math. Software* **5** (1979), 284–295.
5. A. George and J. W. H. Liu, "Computer Solution of Large Sparse Positive Definite Systems," Prentice–Hall, Englewood Cliffs, N.J., 1981.