A Shortest Path Algorithm for Grid Graphs

F. O. Hadlock
Florida Atlantic University
Boca Raton, Florida

ABSTRACT

Grid graphs are a simple class of planar graphs for which the vertices can be assigned integer coordinates so that neighbors agree in one coordinate and differ by one in the other coordinate. Grid graphs arise in applications from the layout design of integrated circuits to idealized models of city street networks. In many applications, a shortest path between two given vertices is needed. The best known algorithms for the shortest path in a general graph of n vertices are of complexity $O(n^2)$. *However, if edge lengths are of uniform length, the shortest path can be determined in time* $O(n)$. *In this paper, taking advantage of the concept of direction present in grid graphs, an algorithm is developed which is* $O(n)$ *in the worst case and* $O(\sqrt{n})$ *in the best case.*

I. BACKGROUND

A grid graph may be defined as a planar graph whose vertices may be assigned integer coordinates so that neighbors agree in one coordinate and differ by one in the other coordinate. Alternatively one may think of a grid graph as being formed by starting with a rectangular region ruled off by horizontal and vertical lines spaced at unit intervals. Intersection points represent vertices while unit segments represent edges. This would be the complete grid graph. Any other grid graph may be formed from it by removing edges at will, along with vertices of zero degree. Several sample grid graphs are shown in Figure 1.

Networks, 7: 323-334

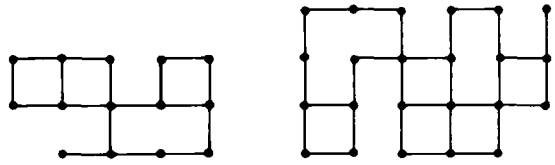


Fig. 1

Grid graphs arise in diverse applications. One application is in the automated design of integrated or hybrid circuit layout [5,7,8,9,10,12]. The circuit chip is imagined to be ruled off with lines parallel to its edges and spaced one unit apart. The result is a complete grid graph whose dual is also a grid graph used in representing circuit layout. As components and conducting paths are committed to the chip, a subgraph (representing the as yet free space) has edges and vertices deleted. Another application of grid graphs is as an idealized model of city street networks [4,6].

In both applications the problem arises of computing a shortest path from one vertex to another. In an arbitrary graph of n vertices, any vertex is potentially on the shortest path to the destination vertex. Consequently the shortest path is computed from the source vertex to every other vertex closer than the destination vertex in a general graph. Every known algorithm [1,3] for doing this is of time complexity $0(n^2)$. In Dijkstra's algorithm [2], this is due to the fact that every unincorporated vertex adjacent to the minimum distance tree must be considered for inclusion in the tree. Since there are 0(n) unincorporated vertices and this must be repeated 0(n) times until the tree includes every vertex closer than the destination, the required number of operations is $0(n^2)$.

In [7], Hart, Nilsson and Raphael have employed the idea of using a lower bound on the distance between two vertices to guide the search for the shortest path. Rubin [12] applies this idea to rectangular grids, using the Manhattan distance as lower bound. Since he treats non-uniform edge costs, the resulting algorithm must necessarily still search a list for the 'next cell to expand' (equivalent to 'adding a vertex to the minimum distance tree rooted at the source'). Hoel [8] introduces the idea of organizing the list of frontier vertices into a series of stacks, each stack containing cells of a given cost, in order to eliminate list searching. Although the shortest path algorithm to be presented is developed directly, it employs some ideas used in Rubin's algorithm and in Hoel's algorithms (both the ideas of using Manhattan distance as a lower bound and of using several stacks are employed).

In graphs with uniform edge lengths, there is no need to search for those vertices to include in the minimum distance tree--all unincorporated neighbors of those just included are included on the next pass. In this context, a shortest path can be computed by Moore's algorithm in $O(n)$ time. Since there is no sense of direction in a 'maze,' the shortest path is still computed to every vertex closer to the source than the destination. In this paper, an algorithm for shortest paths in grid graphs is developed. It takes advantage of both special properties of grid graphs: uniform edge lengths and direction. Called the minimum detour algorithm, it finds the shortest path in time between $O(\sqrt{n})$ and $O(n)$. It is compared with Moore's algorithm and is seen to operate at least as fast. For a special class of graphs, it operates approximately at least 8 times as fast.

II. DEFINITIONS

For two vertices P and Q, we will use the notation $d_M(P,Q)$ to refer to the Manhattan distance between P and Q. Thus $d_M(P,Q)$, the sum of the x and y distances between P and Q, is a lower bound to the actual distance $d(P,Q)$ in a particular grid graph. Thinking of P as initial or source vertex and Q as destination vertex, an intermediate vertex u has its neighbors partitioned into two sets. A neighbor v is a Q-positive neighbor if $d_M(v,Q) + 1 = d_M(u,Q)$ and is a Q-negative neighbor if $d_M(v,Q) - 1 = d_M(u,Q)$. A path α from P to x consisting of vertices $u_0 = P, u_1, \ldots, u_n = x$ has detour number d (with respect to Q) equal to the number of vertices u_i for which u_i is a Q-negative neighbor of u_{i-1}. Thus d is just the number of times the path moves away from Q. By d_x is meant the detour number of the shortest path from P to x.

Path Lengths and Detour Numbers

Given a grid graph along with two points P,Q, the set of paths from P to Q is partitioned into classes according to path length, $d(P,Q) + 2 \cdot k$ with $k \geq 0$. These classes are the same as those resulting from a partition by detour number. The simple relation between path length and detour number is given in the following theorem.

Theorem 1: A path α from P to Q has length $d_M(P,Q) + 2 \cdot d$ where d is the detour number of α with respect to Q.

Proof: The proof is by induction on d. For d = 0, the path length $|\alpha|$ must equal the Manhattan distance $d_M(P,Q)$. For if $|\alpha| > d_M(P,Q)$, there are points u_i on α (P is one such) for which the Manhattan distance $d_M(u_i,Q)$ is less than the remaining distance along α (from u_i to Q). Let u_m be the furthest such point along α (i.e., m is the maximal index for which $d_M(u_m,Q) < |\alpha'|$, $\alpha' = u_m,\ldots,u_n = Q$). Then $d_M(u_{m+1},Q) = |\alpha''|$ where $\alpha'' = u_{m+1},\ldots,u_n = Q$. Since $|\alpha''| = |\alpha'| - 1$, we have $d_M(u_{m+1},Q) \geq d_M(u_m,Q)$ which means u_{m+1} is a Q-negative neighbor of u_m and $d \neq 0$, a contradiction.

Suppose that $|\alpha| = d_M(P,Q) + 2 \cdot d$ for $d \leq k$. If α is a path with d = k + 1, let u_{i+1} be the first Q-negative neighbor of its predecessor on α. Partition α into $\alpha' = u_0,\ldots,u_i$ and $\alpha'' = u_{i+1},\ldots,u_n$. Let d', d" be the detour numbers of α' and α'' respectively. Then d' = 0 and d" = k so that $|\alpha'| = d_M(P,u_i)$ while $|\alpha''| = d_M(u_{i+1},Q) + 2 \cdot k$ by the induction hypothesis. Since u_{i+1} is a Q-negative neighbor of u_i, $d_M(u_{i+1},Q) = d_M(u_i,Q) + 1$. Then $|\alpha| = |\alpha'| + 1 + |\alpha''| = d_M(P,u_i) + 1 + d_M(u_i,Q) + 1 + 2 \cdot k = d_M(P,u_i) + d_M(u_i,Q) + 2 \cdot (k+1)$. Since u_i must be within the rectangle defined by P and Q, $d_M(P,u_i) + d_M(u_i,Q) = d_M(P,Q)$ and $|\alpha| = d_M(P,Q) + 2 \cdot d$.

Q.E.D.

As a consequence of Theorem 1, the shortest path from P to Q is a path α for which d is minimum; which just says that we minimize distance by detouring as little as possible. An algorithm which searches for a path α with detour number 0,1,2,..., in that order will find a shortest path. Such an algorithm is presented in the next section.

III. A SHORTEST PATH ALGORITHM FOR GRID GRAPHS

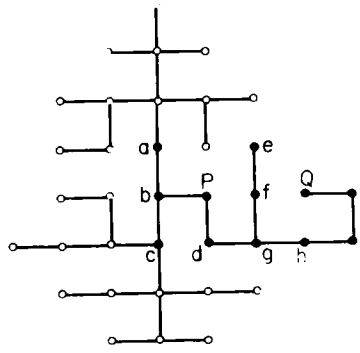
The following algorithm searches for a path α from P to Q, first with d = 0, then d = 1, etc. Vertices are initially unlabeled and labeled only when they are "officially" visited. Two stacks, the P and the N stacks, are used. Stacking Q-negative neighbors implies the N-stack while stacking Q-positive neighbors implies the P-stack. Explicitly the algorithm finds only the shortest path length; details concerning the information necessary to path construction are omitted.

Minimum Detour Algorithm

1) u ← P. d ← 0.
2) Official visit to u: label u as being visited. If u = Q then stop. Stack unlabeled Q-negative neighbors of u. If there are no unlabeled Q-positive neighbors, then go to 3). Let u ← v, v an unlabeled Q-positive neighbor and stack the additional Q-positive neighbor, if there is one. Repeat 2).
3) If P-stack is empty, then go to 4). Unstack v from P-stack. If v is labeled then repeat 3). Let u ← v and go to 2).
4) If N-stack is empty, then go to 5). Let P-stack <--> N-stack. Let d ← d + 1. Go to 3).
5) Print 'No path exists from P to Q.' Stop.

That the algorithm finds a shortest path is easily seen. As long as it can, it proceeds toward Q. At each step it remembers alternate vertices to visit towards Q (P-stack) and away from Q (N-stack). The current vertex is obtained as a positive step towards Q from the vertex just visited (Step 2) or as a positive step from a vertex visited previously during this pass, or as a negative step from a vertex visited during the last pass (Step 3). If no path exists with d negative vertices (or less), then one is sought with d + 1 negative vertices (Step 4). If every avenue has been explored without reaching Q, the algorithm halts (Step 5). The validity of the algorithm is demonstrated formally in a later section.

An Example: To illustrate the action of the algorithm, the shortest path from P to Q is computed in the grid graph shown in Figure 2(a). The values of v, d as well as the stack contents are shown in Figure 2(b). Stack contents and values shown are those which would exist when whatever action is due to the indicated step has been completed. The heavy vertices are those either placed on a stack or visited. The entire graph would comprise the minimum distance tree rooted at P which either Moore's or Dijkstra's algorithms construct.



(a)

Algorithm Step	v	d	P-stack	N-stack
1	P	0	Empty	Empty
2	P	0	Empty	d,b
3	P	0	Empty	d,b
4	P	1	d,b	Empty
3	b	1	d	Empty
2	b	1	d	a,c
3	d	1	Empty	a,c
2	g	1	Empty	a,c
2	h	1	Empty	a,c,f
2	h	1	Empty	a,c,f,i
3	h	1	Empty	a,c,f
4	h	2	a,c,f,i	Empty
3	i	2	a,c,f	Empty
2	j	2	a,c,f	Empty
2	Q	2	a,c,f	Empty

Stop

(b)

Fig. 2

IV. ANALYSIS

In order to demonstrate the validity of the minimum detour algorithm and to bound the number of vertices visited in searching for a shortest path, a theorem analogous to Theorem 1 is presented.

Theorem 2: Let α be a path from P to x with detour number d with respect to Q. Then $|\alpha| = d_M(P,Q) + 2 \cdot d - d_M(x,Q)$.

Proof: Let α" be a path from P to Q composed of α followed by a path α' from x to Q such that α' has detour number 0 with respect to Q (we can imagine the existence of such a path in those grid graphs in which none exists). Then the detour number of α" with respect to Q is d and so, by Theorem 1, $|\alpha"| = d_M(P,Q) + 2 \cdot d$. Since $|\alpha"| = |\alpha| + |\alpha'|$ and $|\alpha'| = d_M(x,Q)$ by Theorem 1, $|\alpha| = d_M(P,Q) + 2 \cdot d - d_M(x,Q)$.

Q.E.D.

From Theorem 2, the validity of the following theorem is easily established.

Theorem 3: Let x be a vertex visited by the minimum detour algorithm in its search for a shortest path from P to Q. The path constructed by the algorithm is a shortest path from P to x.

Proof: By Theorem 2, the path is of length $d_M(P,Q) + 2 \cdot d - d_M(x,Q)$ where d is its detour number. Since the Manhattan distances are constant, then if d is minimized, so is the path length. Clearly the algorithm minimizes d since, on a given pass, it visits all vertices which can be visited (from previously visited vertices) without detouring with respect to Q. At the same time it remembers those vertices with a single detour by placing them on the N-stack for the next pass. Thus a vertex is visited with the minimum detour possible.

Q.E.D.

Comparison

An algorithm, due to Moore [11], will find shortest paths in graphs with uniform edge lengths in O(n) time and is presented here for purposes of comparison. It will be seen that because intended for a more general class of graphs, Moore's algorithm, since it does not employ the idea of direction, visits as many vertices as the algorithm presented here. Both algorithms are restated here giving only those features relevant for comparison.

Moore's Algorithm

(1) Stack P on S_1.

(2) While S_1 is nonempty, do (5).

(3) If S_2 is empty, then halt.

(4) Interchange stacks S_1, S_2 and go to (2).

(5) Unstack vertex v from S_1. If v = Q, halt. Label v and stack all unlabeled neighbors on S_2.

Minimum Detour Algorithm

(1) Stack P on S_+.

(2) While S_+ is nonempty, do (5).

(3) If S_- is empty, then halt.

(4) Interchange stacks S_+, S_- and go to (2).

(5) Unstack vertex v from S_+. If v = Q, halt. Label v. Stack all unlabeled Q-positive neighbors on S_+ and all unlabeled Q-negative neighbors on S_-.

Obviously both algorithms are O(n) where n is the number of vertices since no vertex is ever unstacked more than once in Step (5) of either algorithm. Thus, to compare the algorithms, it is sufficient to characterize the relative number of vertices visited in Step (5). In Moore's algorithm, at the beginning of the kth execution of Step (2), stack S_1 contains all vertices of distance k - 1 from P. Before Q is reached, all vertices within distance d(P,Q) - 1 will be visited in Step (5) along with some of distance d(P,Q) depending on the order in which neighbors are stacked.

In the minimum detour algorithm, at the beginning of the kth execution of Step (2), stack S_+ contains some of the vertices reachable from P with detour number k - 1. The rest are added to S_+ and removed during the execution of (2). So the detour number of the path constructed to visit any vertex x is no greater than the detour number of the path to Q. From Theorem 2, the distance is $d_M(P,Q) + 2\cdot(k-1) - d_M(x,Q) \leq d_M(P,Q) + 2\cdot d$ where d is the detour number of the path constructed to Q. By

Theorem 3, $d_M(P,Q) + 2 \cdot d$ is $d(P,Q)$. Before Q is reached, all vertices x visited in Step (5) are those of distance $d(P,x) < d(P,Q)$ reachable with detour number $d_x < d_Q$. Also visited are some vertices of smaller distance and equal detour, depending on the order in which vertices are stacked. Now $d(P,x) = d(P,Q) \Rightarrow d_M(P,Q) + 2 \cdot d_x - d_M(x,Q) = d_M(P,Q) + 2 \cdot d_Q - d_M(Q,Q) \Rightarrow d_M(x,Q) = d_M(Q,Q)$ since on the last pass $d_x = d_Q$. Thus the only vertex x visited at equal distance to Q is Q itself. Thus the set of intermediate vertices visited by the minimum detour algorithm is $\{x \mid d(P,x) < d(P,Q) \text{ and } d_x \leq d_Q\}$ which is a subset of those visited by Moore's algorithm. Containment will almost always be proper since Moore's algorithm will generally visit vertices with detour number greater than d_Q or distance equal to $d(P,Q)$.

To get a more precise comparison of computation times, we introduce a class of grid graphs for which this is easy. The class of graphs includes all grid graphs with the following properties. Let the graph have center P and radius r. Let the vertex set include all vertices x such that $d_M(P,x) \leq r$ and the edge set include edges sufficient to satisfy $d(P,x) = d_M(P,x)$. This means every vertex, other than P, has an edge joining it to at least one P-positive neighbor. Since vertices in line with P have only one P-positive neighbor, all axis-edges belong. Non-axis vertices are connected to either or both P-positive neighbors. Since there are $2 \cdot r^2 - 2 \cdot r$ non-axis vertices, there are $3^{(2 \cdot r^2 - 2 \cdot r)}$ graphs of radious r, neglecting isomorphisms (dividing by 8 gives a lower bound to the number of non-isomorphic such grid graphs of radius r). For r = 2 there are $3^4 = 81$ graphs of which there are 15 non-isomorphic graphs displayed by Figure 3.

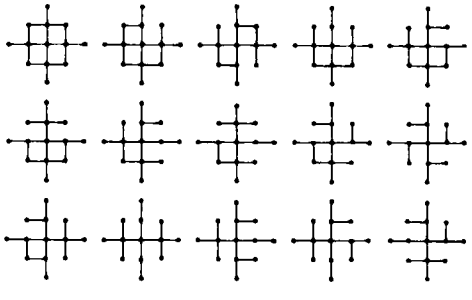


Fig. 3

Now let vertex Q be appended to one of these grid graphs, adjoining it to a perimeter vertex by a horizontal or vertical edge. Then Moore's algorithm visits every vertex in the graph in finding a shortest path from P to Q. Because $d_Q = 0$, the minimum detour algorithm visits some subset of the vertices lying within the rectangle defined by P,Q. If i,j are the absolute coordinates of Q, then there are $(i+1) \cdot (j+1)$ vertices within the rectangle. Since $i + j = r + 1$, the number of vertices $(i+1) \cdot (j+1)$ is bounded above by $[(r+3)/2]^2$. In some cases the shortest path will be found by visiting as few as r vertices (e.g., the complete grid graph). But in any event, no more than $[(r+3)/2]^2$ vertices will be visited. Since Moore's algorithm visits all $2r^2 + 2r + 1$, in the limit as $r \to \infty$, the minimum detour algorithm runs at least 8 times as fast for this class of grid graphs.

The graph in Figure 4 is an example of a grid graph in this class with r = 4. Q has been appended with coordinates i = 3, j = 2. There are 12 vertices within the rectangle defined by P,Q. Thus the most vertices visited by the minimum detour algorithm is 12 as opposed to $2 \cdot r^2 + 2 \cdot r + 1 = 41$ vertices visited by Moore's algorithm. The 1 to 8 ratio is approached as $r \to \infty$. It is clear that as Q approaches an axis, the size of the rectangle approaches r so that the upper bound on computation time approaches $O(\sqrt{n})$.

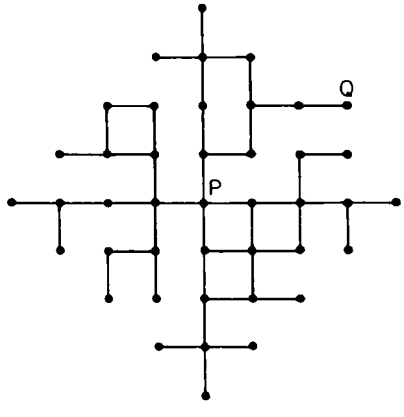


Fig. 4

V. CONCLUSIONS

The algorithm presented here is applicable to a special but useful class of graphs. The algorithm presented makes no attempt to remember the shortest path. However, this is easily incorporated by assigning a tentative predecessor label in Step 2 to each neighbor, Q-positive or Q-negative, and stacking this label with the neighbor. This predecessor label becomes part of the permanent label when the vertex is visited.

A dramatic example of the difference in computation times between the minimum detour algorithm and Dijkstra's algorithm is obtained by taking two distant points in a large, complete graph. Then the specialized algorithm presented here requires $0(d(P,Q))$ time or $0(\sqrt{n})$ time. This example actually arose in a practical context when the author was designing an interactive system for hybrid circuit layout [5]. Dijkstra's algorithm was used for shortest path computation in conductor routing. In the first stages of layout, a more or less complete grid graph represented the as yet free chip space. The user is required to wait while a minimum distance tree is constructed covering a diamond shaped area about the point to be connected. This algorithm, with nothing intervening, would trace immediately a path along the bottom and right side, or left side and top, of the rectangle defined by P,Q. The choice would depend on the priority given in Step 2 to Q-positive neighbors.

Of course with any of the algorithms due to Dijkstra, Lee, or Rubin, non-uniform edge lengths may be employed to penalize a route which results in crossing other conductor runs. But the algorithm presented here could be usefully employed in this situation by not allowing crossings. This algorithm would then more quickly find a crossing free route if such existed. Only if none existed or if the length of the crossing free route obtained was unsatisfactorily long would the more general (and slower) algorithm be employed.

An extension of this algorithm to a lattice graph [6] in n-dimensions is immediate. Another extension which is being considered is to an arbitrary graph G. Suppose the distance between every pair of vertices has been computed and is available. Now, because of busy lines or some other phenomenon, a subgraph G' remains and the shortest path in G' is desired. Can the lower bound of the distance in G be used to do better than $0(n^2)$?

REFERENCES

1. Christofides, N., *Graph Theory, An Algorithmic Approach*, Academic Press, New York, 1975.

2. Dijkstra, E. W., "A Note on Two Problems in Connection with Graph Theory," *Numerische Mathematik*, 1, 1959, pp. 269-271.

3. Dreyfus, S. E., "An Appraisal of Some Shortest-Path Algorithms," *Operations Research*, 17, 1968, pp. 395-410.

4. Hadlock, F. O., "Minimum Distance, Minimum Turn Vehicle Routes," in preparation.

5. Hadlock, F. O., "PRIMS, An Interactive System for the Design of Hybrid Circuit Layout," *Proc. of the Third Annual Conference on Computer Graphics, Interactive Techniques and Image Processing*, Philadelphia, 1976, pp. 293-301.

6. Hadlock, F. O. and F. Hoffman, "Block Patterns and Routing," *Proc. of the Sixth Southeastern Conference on Combinatorics, Graph Theory and Computing*, Boca Raton, 1975, pp. 385-400.

7. Hart, P. E., N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, Vol. SSC-4, 1968, pp. 100-107.

8. Hoel, J. H., "Some Variations of Lee's Algorithm," *IEEE Trans. on Computers*, Vol. C-25, No. 1, 1976, pp. 19-24.

9. Lee, C. Y., "An Algorithm for Path Connections and its Applications," *IRE Trans. of Electronic Computers*, Vol. EC-10, 1961, pp. 346-365.

10. Marshall, C. W., *Applied Graph Theory*, John Wiley & Sons, New York, 1971, p. 114.

11. Moore, E. F., "The Shortest Path Through a Maze," *Annals of the Harvard Computation Laboratory*, Vol. 30, Pt. II, 1959, pp. 285-292.

12. Rubin, F., "The Lee Path Connection Algorithm," *IEEE Trans. on Computers*, Vol. C-23, No. 9, 1974, pp. 907-914.

Paper received June 14, 1976.