Finding Minimum Rectilinear Distance Paths in the Presence of Barriers

Richard C. Larson
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
Victor O. K. Li
University of Southern California, University Park, Los Angeles, California 90007

Given a set of origin-destination points in the plane and a set of polygonal barriers to travel, this paper develops an efficient algorithm for finding minimal distance feasible paths between the points, assuming that all travel occurs according to the rectilinear distance metric. By geometrical arguments the problem is reduced to a finite network problem. The nodes are the origin-destination points and the barrier vertices. The links designate those node pairs that "communicate" in a simple way, where communication implies the existence of a node-to-node rectilinear path that is not made longer by the barriers. The weight of each link is the rectilinear distance between its two corresponding nodes. Solution of the minimal distance path problem on the network procedes in two steps. First, for a given origin or root node, a tree is generated containing a minimal distance path to each node that communicates with the root node. Second, a modified Dijkstra-type iteration is utilized, starting with the nodes of the tree, sequentially adding nodes according to minimum "penalty distance," where the penalty is the extra travel distance caused by the barriers. The paper concludes with a discussion of the computational complexity of the procedure, followed by a numerical example.

I. INTRODUCTION

The rectilinear distance between two points (x_1, y_1), (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. This distance is sometimes called right-angle, L_1, or Manhattan distance. This paper develops an algorithm for finding minimum rectilinear distance paths between given origin-destination points, in the presence of polygonal barriers to travel. The barriers need have no special properties, such as convexity.

The problem arises in a number of areas:

(1) *Urban transportation*. Urban vehicles travelling on a street grid may be considered to be governed by the right angle metric. Barriers to travel could include cemeteries, parks, railroad yards, rivers, etc. Effective operation of these systems would require knowledge of "best paths" around such barriers. (Ironically, the motivation for this paper originated with transportation work by these authors in Manhattan, which revealed the inadequacy of the "Manhattan distance metric" as a measure of travel distance between two points due, primarily, to New York's Central Park.)

(2) *Plant and facility layout*. Travel on a plant or factory floor can often be mod-

CCC 0028-3045/81/030285-20$02.00

eled as right-angle. Barriers to travel would be impassable areas on the floor, such as sub assembly areas, shops, sections of assembly lines, etc. The algorithm proposed here not only could reveal optimal paths between points, but also could be used to examine the plant floor travel-time consequences of alternative floor layouts.

(3) *Locating power lines.* In sections of the midwest of the U.S. new power lines are only allowed to be located along the east-west or north-south boundaries between surveyed mile-square or quarter-mile square sections. The location of such a power line occurs, then, along a right-angle route. Barriers in this problem could include towns, highways, parks, or subsections otherwise not available for power line routing.

Other potential applications include design of printed circuit boards, certain cutting problems, and finding the path through a maze (perhaps for a robot).

In the literature there has been some work on related problems. A discretized version of our problem, in which both the barriers and the feasible travel region are assumed to be elements of a finite rectangular grid, has been considered by Damm [1]; in the discrete case, the network nodes are automatically constrained to be integer lattice points (not in the barriers), and thus the network structure and its properties are relatively straightforward. We are aware of no other work on the exact problem examined herein, allowing arbitrarily complex polygonal barriers and L_1 paths anywhere except in the barriers. The closest nondiscretized work pertains to the analogous problem using the Euclidean metric. In 1974 Wangdahl, Pollack, and Woodward [2] developed a procedure for finding the minimal distance path between two points in the presence of polygonal barriers, where each path step utilizes the Euclidean metric. The node set N consists of the origin and destination points and the vertices of the barriers; each finite length link joins a node pair for which a connecting straight line can be drawn between the nodes without intersecting any barriers. The proposed "visibility algorithm" (so named because connecting nodes could "see each other") utilizes a variant of dynamic programming, which essentially is Dijkstra's algorithm [3]. Independently in 1974, Vaccaro [4] solved a similar problem, but only for barriers represented by line segments. The Wangdahl et al. procedure appears to have been rediscovered in 1978 by Lozano-Perez and Wesley [5], who offer their "VGRAPH" procedure only for convex polygons. Very different applications motivated each of these three efforts: minimum trajectory pipe routing through a ship [Wangdahl et al.], routing urban vehicles [Vaccaro], navigating an early robot vehicle named SHAKEY [Lozano-Perez and Wesley].

The paper proceeds in three parts. In Sec. II we summarize results necessary to transform the problem in the x-y plane, with a noncountably infinite number of paths, to a finite network problem. In Sec. III we develop the details of the algorithm, POLYPATH, and discuss its computational complexity. In Sec. IV we present a numerical example.

II. TRANSFORMING TO A NETWORK PROBLEM

A. Preliminaries

We consider Euclidean 2-space R^2. Within R^2 there are n_Q origin-destination points

$$Q = \{q(i), i = 1, 2, \ldots, n_Q\}$$

and n_B polygonal barriers $B_m (m = 1, 2, \ldots, n_B)$ with vertices

$$V = \{v(k, m), k = 1, 2, \ldots, v_m; m = 1, 2, \ldots, n_B\},$$

where $v(k, m)$ is the kth clockwise-ordered vertex of barrier m and $v_m < \infty$ is the number of its vertices. The coordinates of $q(i)$ are $\{x(i), y(i)\}$ and the coordinates of $v(k, m)$ are $\{x(k, m), y(k, m)\}$. $B_m (m = 1, \ldots, n_B)$ is an open set, e.g., for convex barriers the intersection of v_m open half-planes. Hence, the vertices and sides of B_m are not contained within B_m. We assume (without loss of generality) that barriers do not intersect (i.e., $B_m \cap B_n = \emptyset$ = null set, for $n \neq m$).

A rectilinear path P in R^2 having $2(s + 1)$ steps is specified by a sequence of path vertices $V(P) = \{(x_0, y_0), (x_1, y_0), (x_1, y_1), (x_2, y_1), (x_2, y_2), \ldots (x_s, y_s), (x_{s+1}, y_s), (x_{s+1}, y_{s+1})\} = [\{x\}, \{y\}], s = 0, 1, 2, \ldots,$ such that

$$P = \{(x, y_k) \in R^2; x_k \leqslant x \leqslant x_{k+1} \text{ or } x_{k+1} \leqslant x \leqslant x_k, k = 0, 1, \ldots, s\}$$
$$\cup \{(x_k, y) \in R^2; y_{k-1} \leqslant y \leqslant y_k \text{ or } y_k \leqslant y \leqslant y_{k-1}, k = 1, 2, \ldots, s + 1\}.$$

That is, a path is a connected set of points proceeding in straight line segments horizontally from (x_0, y_0) to (x_1, y_0), then vertically to (x_1, y_1), then horizontally to (x_2, y_1), etc. To allow the first step to be vertical, simply set $x_1 = x_0$. In our development we consider the merged set of points $W = Q \cup V$. A path P_{ij} from $w(i) \in W$ to $w(j) \in W$ is specified by a set of path vertices originating at $(x_0, y_0) = (x_w(i), y_w(i))$ and terminating at $(x_{s+1}, y_{s+1}) = (x_w(j), y_w(j))$. A *feasible* path P_{ij} from $w(i)$ to $w(j)$ penetrates no barriers, hence satisfies the condition $P_{ij} \cap [\cup_{m=1}^{n_B} B_m] = \phi$. We assume the existence of at least one feasible path between each pair $w(i), w(j)$. The rectilinear length (or simply length) of path P is

$$L(P) = L_x(P) + L_y(P),$$

where

$$L_x(P) \equiv \sum_{k=0}^{s} |x_{k+1} - x_k| \quad \text{and} \quad L_y(P) \equiv \sum_{k=0}^{s} |y_{k+1} - y_k|.$$

The minimum rectilinear travel distance between two points (x, y) and (x', y') is $t([x, y], [x', y'])$, obtained by following a shortest length feasible path between the points.

The problem is as follows: For any given origin $q(i)$ find a minimal length feasible path and the corresponding travel distance to each possible destination $q(j)$. Repeated application over i of any solution algorithm yields minimal length feasible paths between all origin-destination pairs.

Two points $w(i) \in W$, $w(j) \in W$, having coordinates $(x_w(i), y_w(i))$ and $(x_w(j), y_w(j))$, are said to *communicate* if there exists at least one feasible path P_{ij} between them having length $L(P_{ij}) = |x_w(j) - x_w(i)| + |y_w(j) - y_w(i)| \equiv c_{ij}$. That is, two points communicate if the barriers cause no net increase in travel distance between them.

The minimal feasible travel distance between $w(i)$ and $w(j)$ is abbreviated $t(i, j)[= t([x_w(i), y_w(i)], [x_w(j), y_w(j)])]$.

A path P having path vertices $V(P) = \{(x_0, y_0), (x_1, y_0), \ldots, (x_{s+1}, y_{s+1})\}$ is a *staircase path* having *staircase vertices* if $\{x_k, k = 0, 1, \ldots, s+1\}$ and $\{y_k, k = 0, 1, \ldots, s+1\}$ each form monotone sequences, i.e., if $\operatorname{sgn}(x_{k+1} - x_k) = \operatorname{sgn}(x_k - x_{k-1})$ and $\operatorname{sgn}(y_{k+1} - y_k) = \operatorname{sgn}(y_k - y_{k-1}), k = 1, 2, \ldots, s$. [sign (0) is arbitrary].

We now state three elementary observations:

(1) Two points having coordinates (x_0, y_0) and (x_{s+1}, y_{s+1}) communicate if and only if there exists at least one feasible staircase path between them.
(2) Any path having minimal length joining two communicating points must be a staircase path.
(3) Any two points, joined by a straight line segment that has no points in common with any barrier, communicate. Thus, all pairs of adjacent vertices of a barrier communicate.

The third observation can be proved by constructing a staircase path along the straight line connecting the two points, keeping the path sufficiently close to the line so that no path step intersects a barrier (see Fig. 1). (In certain situations, as illustrated later in Figure 3(e), this procedure may require a countably infinite number of steps.)

B. Vertex-Seeking Tree

We now wish to construct a simple tree about each node which provides a basis for path formation. First consider a point $q(i) \in Q$ having coordinates $\{x(i), y(i)\}$. We wish to construct the "positive x probe" $r_{x^+}(q(i))$ from $q(i)$. To do this, extend a ray from $(x(i), y(i))$ horizontally in the direction of increasing x. There are three possibilities (see Fig. 2):

(1) The ray intersects no barriers, in which case $r_{x^+}(q(i)) = \{(x, y) \in R^2 : y = y(i), x \geqslant x(i))\}$, i.e., $r_{x^+}(q(i))$ is the ray.
(2) The ray intersects another point $w(j) \in W$ (either a barrier vertex or an origin-destination point), in which case $r_{x^+}(q(i)) = \{(x, y) \in R^2 : y = y(i), x(i) \leqslant x \leqslant x_w(j)\}$; i.e., $r_{x^+}(q(i))$ is the line segment connecting $q(i)$ and $w(j)$.
(3) The ray intersects side k of a barrier m, the point of intersection being a. The barrier side is thereby partitioned into two subsides having endpoints

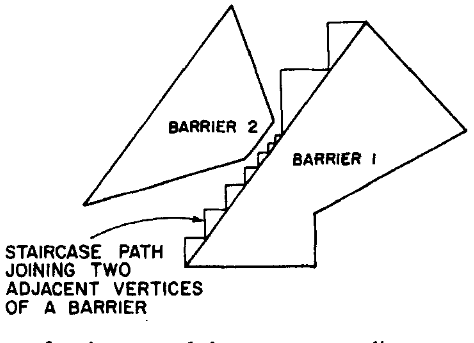

FIG. 1. Illustration of staircase path between two adjacent vertices of a barrier.

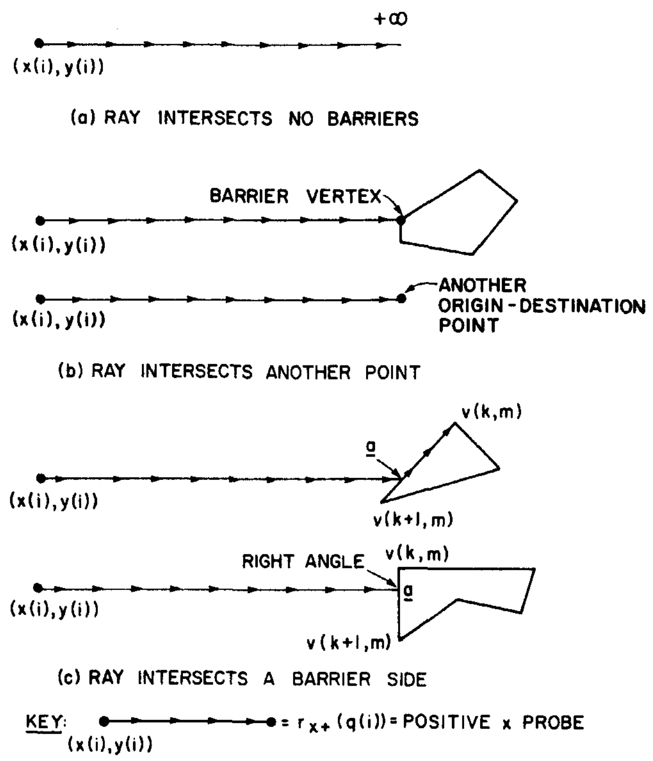
+∞
(x(i),y(i))
(a) RAY INTERSECTS NO BARRIERS
BARRIER VERTEX
(x(i),y(i))
ANOTHER ORIGIN-DESTINATION POINT
(x(i),y(i))
(b) RAY INTERSECTS ANOTHER POINT
v(k,m)
a
(x(i),y(i))
v(k+1,m)
RIGHT ANGLE
v(k,m)
a
(x(i),y(i))
v(k+1,m)
(c) RAY INTERSECTS A BARRIER SIDE
KEY: (x(i),y(i)) = r_{x+}(q(i)) = POSITIVE x PROBE

FIG. 2. The positive x probe: examples.

$(v(k, m), a)$ and $(a, v(k+1, m))$. Here $r_{x^+}(q(i))$ is the union of the line segment from $q(i)$ to a and that subside which lies at an obtuse angle to the segment from $q(i)$ to a. (In the case of right angle intersection $r_{x^+}(q(i))$ need not have this second component.)

In a like manner we can construct probes $r_{y^+}(q(i))$, $r_{y^-}(q(i))$, and $r_{x^-}(q(i))$ in the positive y direction, negative y direction, and negative x direction, respectively. The *vertex-seeking tree* about the *root node* $q(i)$ is

$$\text{VST}(q(i)) \equiv r_{x^+}(q(i)) \cup r_{x^-}(q(i)) \cup r_{y^+}(q(i)) \cup r_{y^-}(q(i)).$$

By construction of the tree, it is clear that all points on the tree communicate with $q(i)$.

Now consider a barrier vertex $v(k, m) \in V$. A similar vertex-seeking tree is constructed for $v(k, m)$, with the obvious condition that no point in the tree can penetrate barrier m. Thus, depending on the orientation of the barrier about $v(k, m)$, the number of feasible probes about $v(k, m)$ may be 4, 3, 2, 1, or even 0. (See Fig. 3.) We now state a fourth elementary observation:

(4) Suppose for two points $w(i) \in W$ and $w(j) \in W$, an x-probe from one and a y-probe from the other share at least one common point b. Then $w(i)$ and $w(j)$ communicate.

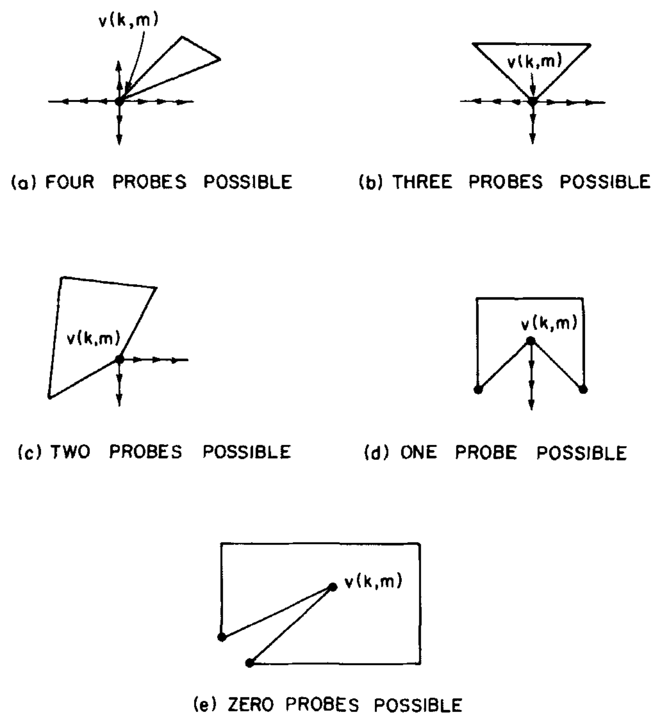

FIG. 3. Possible vertex-seeking trees from a barrier vertex.

We have now identified three conditions under which two vertices $w(i)$ and $w(j)$ communicate:

(1) If $w(i)$ and $w(j)$ are adjacent vertices of a barrier.
(2) If $w(i)$ is the root and $w(j)$ is a terminal point of a vertex-seeking tree.
(3) If $w(i)$ and $w(j)$ have x and y probes that share at least one common point b.

A pair of vertices $w(i)$ and $w(j)$ is called *simply communicating* if it satisfies at least one of these three conditions, assuming with condition (3) that there exists a point of commonality $b \notin W$ (otherwise each vertex communicates simply with b, but only communicates with the other vertex).

C. Vertex Following Paths

Theorem 1. Suppose two points $w(i) \in W$, $w(j) \in W$ communicate, but do not communicate simply. Then there exists at least one feasible staircase path P_{ij}^* between them containing a sequence of simply communicating vertices $\{w(i), w(k_1^*), \ldots, w(k_n^*), w(j)\}$.

Proof: Consider the case $x_w(j) \geq x_w(i), y_w(j) \geq y_w(i)$. We demonstrate the existence of $w(k_1^*)$, with the remaining $w(k_l^*)$ obtained by induction. Since $w(i)$ and $w(j)$ communicate, there exists at least one feasible staircase path P_{ij} between them, with path vertices $V(P_{ij}) = \{(x_w(i) = x_0, y_w(i) = y_0), (x_1, y_0), (x_1, y_1), \ldots, (x_s, y_s),$

$(x_w(j) = x_{s+1}, y_s), (x_{s+1}, y_w(j) = y_{s+1})\}$, such that $x_{k+1} \geqslant x_k$, $y_{k+1} \geqslant y_k (k = 0, 1, \ldots, s)$, and $L(P_{ij}) = (x_w(j) - x_w(i)) + (y_w(j) - y_w(i))$.

Now construct an alternate equal length feasible staircase path from $w(i)$ to $w(j)$. Beginning at $x = x_1$, create a new path step $[(x, y_0), (x, y_1)]$ and push it as far as to the right as possible up to and including x_2. If x_2 is reached without intersecting a barrier, continue to push rightward with the amalgamated path step $[(x, y_0), (x, y_2)]$. Eventually there must be an amalgamated path step $[(x, y_0), (x, y_k)]$ that intersects a barrier vertex or a barrier side for some $x = x_1'$, $x_k \leqslant x_1' < x_{k+1}$. If not, then x could reach $x_w(j)$, implying that $w(i)$ and $w(j)$ communicate simply, a contradiction.

Case 1. If the intersection is with a barrier vertex $v(k, m) = \{x(k, m), y(k, m)\}$, we have $x(k, m) = x_1'$, and $y(k, m) \geqslant y_w(i)$. Since $r_{x^+}(w(i))$ obviously intersects $r_{y^-}(v(k, m))$, $v(k, m) = w(k_1^*)$.

If the intersection is not with a barrier vertex, it must be with a barrier side. In this case, continue the step pushing and amalgamation process, with the intercepted barrier side being the step's lower end point.

Case 2. We may reach an endpoint or vertex $v'(k, m)$ of the barrier side, in which case the new path from $w(i)$ to $v'(k, m)$ has reproduced $r_{x^+}(w(i))$ and thus $v'(k, m) = w(k_1^*)$; or

Case 3. We reach a point x_1'' at which the amalgamated step intercepts a barrier vertex $v''(k, m)$; in this case $r_{x^+}(w(i)) \cap r_{y^-}(v''(k, m)) \neq \phi$, and thus $v''(k, m) = w(k_1^*)$.

One continues by induction, eventually reaching $w(j)$. ■ (See Fig. 4.)

Suppose two points $w(i) \in W$, $w(j) \in W$ do not communicate. Consider a minimal distance path P_{ij}' between $w(i)$ and $w(j)$, i.e., $L(P_{ij}') \leqslant L(P_{ij})$ for all feasible P_{ij}. The

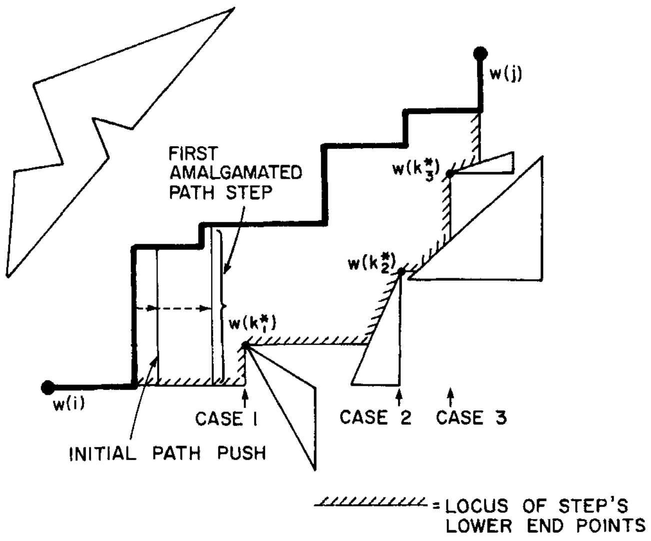

FIG. 4. Illustration of the path push and amalgamation process.

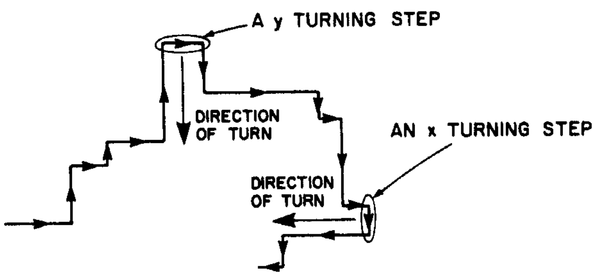

FIG. 5. *X* and *Y* turning steps.

path vertices are $V(P'_{ij}) = \{(x_w(i) = x_0, y_w(i) = y_0), (x_1, y_0), (x_1, y_1), \ldots, (x_s, y_s), (x_w(j) = x_{s+1}, y_s), (x_{s+1}, y_w(j) = y_{s+1})\}$. Since $w(i)$ and $w(j)$ do not communicate $L(P'_{ij}) > |x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|$. Hence there exists at least one point $x_k \in \{x_l\}$ such that $\operatorname{sgn}(x_k - x_{k-1}) \neq \operatorname{sgn}(x_{k+1} - x_k)$ or $y_k \in \{y_l\}$ such that $\operatorname{sgn}(y_k - y_{k-1}) \neq \operatorname{sgn}(y_{k+1} - y_k)$. For any such x_k the path step $[(x_k, y_{k-1}), (x_k, y_k)]$ is called an *x turning step*. And for any such y_k the path step $[(x_k, y_k), (x_{k+1}, y_k)]$ is called a *y turning step* (Fig. 5).

We now state without proof the following:

Lemma. All turning steps in a minimal distance path intersect a barrier vertex, with the barrier situated in the direction of the turn.

We now complete our geometrical arguments with

Theorem 2. Suppose two points $w(i) \in W$, $w(j) \in W$ do not communicate. Then there exists at least one minimal distance feasible path P^*_{ij} between them containing a sequence of simply communicating vertices $\{w(i), w(k^*_1), w(k^*_2), \ldots, w(k^*_n), w(j)\}$.

Proof: From the Lemma all turning steps in a minimal distance path P_{ij} contain a barrier vertex. But by the definition of turning steps, the $\{x_k\}$ and $\{y_k\}$ between two successive turning steps form monotone sequences, i.e., staircase subpaths. By Theorem 1 for each staircase subpath we can construct at least one feasible alternative staircase subpath containing a sequence of simply communicating vertices, beginning at one turning step barrier vertex and ending at the next one. In this manner we can construct the desired minimal distance path P^*_{ij}. ■

III. THE MINIMAL DISTANCE ALGORITHM: POLYPATH

Our results demonstrate that any minimal distance path in R^2 can be replaced by an equidistance path containing a sequence of simply communicating vertices. Thus the problem of finding a minimal distance rectilinear path between $q(i)$ and $q(j)$, in the presence of polygonal barriers, can be reduced to a finite network problem. The network nodes are the origin-destination points $q(i)$ and the polygon vertices $v(k, m)$. The arcs are bidirectional, having weights corresponding to the distances between simply communicating vertices.

The algorithm, called POLYPATH, contains three major steps:

1. Generation of the network.
2. Identification of all vertices communicating with a root vertex.
3. Finding the minimal distance path.

We consider each step next.

A. Generating the Network

The network for our problem is $G(W, L)$, where $W = Q \cup V$ is the set of n_W nodes or vertices and L is the set of links. Let S = set of simply communicating vertex pairs. For any $\{w(i), w(j)\} \in S$, the length of the link $l(i,j) \in L$ is $d_{ij} = |x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|$. For any $\{w(i), w(j)\} \notin S$, $d_{ij} = +\infty$ [or, equivalently, link $l(i,j)$ does not exist]. The matrix of link lengths is $D = (d_{ij})$. As above, we let $t(i,j)$ be the length of a minimal distance path between nodes $w(i)$ and $w(j)$. Given the problem statement, we know that the network is connected, i.e., that there exists at least one path between each pair of vertices.

Construction of the set S is straightforward. A vertex pair $\{w(i), w(j)\}$ is included in S if and only if (1) $w(i)$ and $w(j)$ are adjacent vertices of a barrier; or (2) $w(i)$ and $w(j)$ are contained in the same vertex seeking tree, with either $w(i)$ or $w(j)$ being the root vertex of the tree; or (3) $w(i)$ and $w(j)$ have x and y probes that intersect at a point other than a vertex $w(k)$. Generation of the vertex-seeking tree involves only tests for straight line intersection and, for obtuse angle checks, inequality tests. Similarly, checks for intersection of x and y probes involve only intersection tests.

A network minimal distance path between two points is called an *optimal nodal path*. It should be clear that traversal of any network link $l(i,j)$ corresponds to x-y travel along any one of a (usually) noncountably infinite number of staircase paths between the two simply communicating nodes $w(i)$ and $w(j)$. Thus, specifying a path on the network corresponds to specifying a unique sequence of nodes to visit but a nonunique path in R^2.

The number of optimal nodal paths between two origin-destination points can be large. Recalling the proof of Theorem 1, by pushing the path upward rather than downward we would have proven the existence of another optimal nodal path between two nonsimply communicating nodes $w(i)$ and $w(j)$. Thus, any two nonsimply communicating nodes have at least two (and possibly more) optimal nodal paths connecting them. Two noncommunicating nodes connected by an optimal nodal path with m turning steps typically have a total of 2^{m+1} or more different optimal nodal paths connecting them. (The number may be less than 2^{m+1} only when nodes at adjacent turning steps in an optimal nodal path communicate simply.)

B. Vertices Communicating with $w(i)$

In applications, typically many vertices (perhaps the great majority) communicate, though not simply. Thus it is beneficial to find a procedure for quickly identifying communicating vertices.

Consider $w(i)$ to be the root vertex. We wish to construct a tree $F(w(i)) \subset G(W, L)$ containing all nodes communicating with $w(i)$. $F(w(i))$ is usually not unique because of the possibility of multiple minimal distance paths; our construction will produce a tree $F(w(i))$ having a minimal number of nodes in the tree path from the root $w(i)$ to any $w(j) \in F(w(i))$.

To generate $F(w(i))$, we define eight "orientations" or "directions of travel" from $w(i)$, as indicated in Figure 6(a). We define the node orientation matrix $\Phi = (\phi_{ij})$, where

$$\phi_{ij} \equiv \text{orientation of node } w(j) \text{ with respect to node } w(i) (\phi_{ij} = 1, 2, \ldots, 8).$$

For instance, if $\phi_{ij} = 4$, then node $w(j)$ is northwest of node $w(i)$; $w(j)$ and $w(i)$ need not communicate. For computer storage purposes, it is useful to note that $\phi_{ji} = (\phi_{ij} + 3)_{\text{mod } 8} + 1$.

For each pair of communicating nodes $w(i)$ and $w(j)$ having orientation ϕ_{ij} we define a *permissable direction set* $\Gamma(\phi_{ij})$ such that

$$\Gamma(\phi_{ij}) \equiv \text{set of orientations about node } w(j) \text{ that may include staircase paths through } w(j) \text{ to } w(i).$$

For instance, $\Gamma(2) = \{1, 2, 3\}$, meaning that any vertex $w(j)$ northeast of a root vertex $w(i)$ may have staircase paths from directions 1, 2, or 3 passing through it to the root vertex. The sets $\Gamma(\phi_{ij})$ are $\Gamma(1) = \{1, 2, 3, 7, 8\}$, $\Gamma(2) = \{1, 2, 3\}$, $\Gamma(3) = \{1, 2, 3, 4, 5\}$, $\Gamma(4) = \{3, 4, 5\}$, $\Gamma(5) = \{3, 4, 5, 6, 7\}$, $\Gamma(6) = \{5, 6, 7\}$, $\Gamma(7) = \{5, 6, 7, 8, 1\}$, $\Gamma(8) = \{7, 8, 1\}$. (See Fig. 6(b) and 6(c)).

We say that $w(j)$ n-communicates with $w(i)$ if the *minimal number of* nodes in *all optimal nodal paths* from $w(i)$ to $w(j)$ is $n + 1$, $n = 0, 1, \ldots, n_W - 1$. $w(i)$ 0-communicates with itself. For some node $w(j)$ that n-communicates with $w(i)$, link $l(k, j)$ is a *predecessor link* for $w(j)$ if $l(k, j)$ is on the algorithm-selected $(n + 1)$-node optimal path from $w(i)$ to $w(j)$. For any such $l(k, j)$, $w(k)$ must $(n - 1)$-communicate with $w(i)$ and $\phi_{kj} \in \Gamma(\phi_{ik})$.

We construct the tree $F(w(i))$ by iteratively obtaining the sets of vertices that n-communicate with $w(i)$, together with the corresponding predecessor node for each vertex, for n first equaling 1, then 2, 3, etc. The necessary node-link sets are

$$\Delta_n(w(i)) \equiv \{(w(j), l(k, j)) \in G(W, L) : w(k)\ (n - 1)\text{-communicates with } w(i), d_{kj} < \infty, \phi_{kj} \in \Gamma(\phi_{ik})\}\ n = 1, 2, 3, \ldots, n_W.$$

As the boundary condition, we say that $\Delta_0(w(i))$ contains only $w(i)$ with no predecessor link, i.e., $\Delta_0(w(i)) = \{(w(i), \phi)]$. Clearly,

$$F(w(i)) = \bigcup_{n=0}^{n_W} \Delta_n(w(i)).$$

It is noted that some of the $\Delta_n(w(i))$ may be empty, and that $\Delta_n(w(i)) = \phi$ implies that $\Delta_{n+1}(w(i)) = \phi$. The iterative tree generation algorithm is as follows:

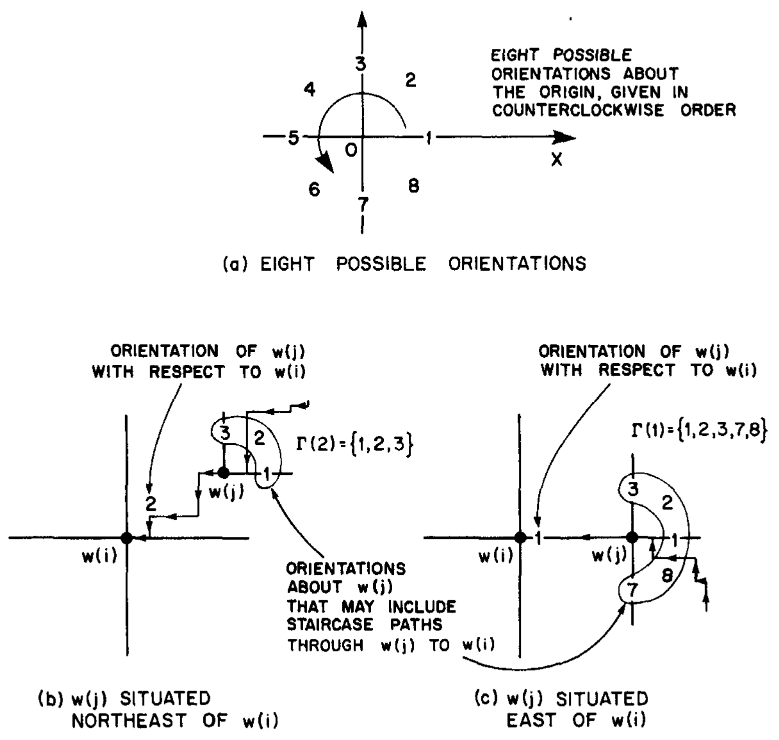
EIGHT POSSIBLE ORIENTATIONS ABOUT THE ORIGIN, GIVEN IN COUNTERCLOCKWISE ORDER
X
0
1
2
3
4
5
6
7
8
(a) EIGHT POSSIBLE ORIENTATIONS
ORIENTATION OF w(j) WITH RESPECT TO w(i)
Γ(2)={1,2,3}
w(i)
w(j)
ORIENTATIONS ABOUT w(j) THAT MAY INCLUDE STAIRCASE PATHS THROUGH w(j) TO w(i)
(b) w(j) SITUATED NORTHEAST OF w(i)
ORIENTATION OF w(j) WITH RESPECT TO w(i)
Γ(1)={1,2,3,7,8}
(c) w(j) SITUATED EAST OF w(i)

FIG. 6. Orientations and permissible direction sets.

Algorithm A1: Tree Generation

Step 1: $n = 0, \Delta_0(w(i)) = \{(w(i)), \phi)\}; \Delta_n(w(i)) = \{(\phi, \phi)\}, n = 1, 2, \ldots, n_W$.

Step 2: For each $w(k) \in \Delta_n(w(i))$, scan the kth rows of Φ and D to find all links $l(k, j)$ such that $d_{kj} < \infty$ and $\phi_{kj} \in \Gamma(\phi_{ik})$. For each such link $l(k, j)$, if its terminal node $w(j) \notin \cup_{m=0}^{n} \Delta_m(w(i))$, then

$$\Delta_{n+1}(w(i)) \leftarrow \Delta_{n+1}(w(i)) \cup \{w(j), l(k, j)\}.$$

Step 3: $n \leftarrow n + 1$

Step 4: If $n = n_W - 1$ or if $\Delta_n(w(i)) = \{(\phi, \phi)\}$, Stop. Else go to Step 2.

The fact that the minimal distance path from $w(i)$ to any node on the tree $F(w(i))$ contains a minimal number of links may be useful in applications.

Clearly for any $w(j) \in F(w(i))$, $t(i, j) = |x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|$. This computation could also be carried out in the above algorithm by setting $t(i, j) = 0$ in Step 1 and adding the computation $t(i, j) = t(i, k) + d_{kj}$ at the end of Step 2.

C. Finding All Minimal Distance Paths

We now develop an iterative procedure for finding the minimal distances and corresponding minimal distance paths from a root node $w(i)$ to all other nodes $w(j)$. The procedure is analogous to Dijkstra-type [3] iterations for finding minimal distance paths on a network, except that we begin with the tree $F(w(i))$ and focus on *penalty travel distances*, not absolute travel distances. Here, the penalty distance associated with an optimal path from $w(j)$ to $w(i)$ is

$$\epsilon(i,j) = t(i,j) - \{|x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|\}.$$

The penalty $\epsilon(i,j)$ represents the net increase in travel distance due to the presence of the barriers. Note that a minimal distance path corresponds to a minimal penalty path. Clearly for two nodes $w(i)$ and $w(j)$ that communicate, $\epsilon(i,j) = 0$.

Suppose we arrange the nodes in order of their penalty distances $\epsilon(i,j)$, using the index α_n = identification number of the node that has the nth smallest $\epsilon(i,j)$. If $F(w(i))$ contains e nodes, then the ordering of the first e nodes is arbitrary since their penalty terms are all 0. For any $m \geqslant e$, partition the nodes $\{w(i) \in W\}$ into two sets:

The "closed" nodes: $H_1(i|m) = \{w(j) \in W : j \in [\alpha_1, \alpha_2, \ldots, \alpha_m]\}$

The "open" nodes: $H_2(i|m) = W - H_1(i|m)$

$H_1(i|m)$ is the set of m least penalty nodes for $w(i)$.

Consider all the open nodes that communicate simply with at least one closed node, i.e., $H_2'(i|m) = \{w(j) \in H_2(i|m) : d_{\alpha_n j} < +\infty,\ n = 1, 2, \ldots, m\}$. For any open node $w(j) \in H_2'(i|m)$ define its m-stage *estimated penalty distance* for a trip to $w(i)$ to be

$$\hat{\epsilon}(i,j|m) \equiv \underset{n\in[1,\ldots,m]:d_{\alpha_n j}<+\infty}{\text{MIN}} [d_{\alpha_n j} + t(i,\alpha_n)] - (|x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|).$$

Here $\hat{\epsilon}(i,j|m)$ is our current "best guess" of the true penalty $\epsilon(i,j|m)$, given the identity of the m least penalty nodes. The best guess is generated by attaching each node in $H_2'(i|m)$ directly through one link to a node in $H_1(i|m)$ in a way which minimizes the total penalty.

Theroem 3. For any given $m = 1, 2, \ldots, n_W$, the m + 1st least penalty node is one with minimal estimated penalty distance, and its penalty distance is this estimate, i.e.,

$$\epsilon(i,\alpha_{m+1}) = \underset{j:w(j)\in H_2'(i|m)}{\text{MIN}} [\hat{\epsilon}(i,j|m)] \equiv \epsilon_0$$

and α_{m+1} is a node $w(j) \in H_2'(i|m)$ that achieves the minimum. (Ties can be broken arbitrarily).

Proof: 1. Suppose $\epsilon(i,\alpha_{m+1}) > \epsilon_0$. Then closed node (α_{m+1}) could be replaced by an open node $w(j)$ that achieves the minimum, resulting in a penalty smaller than $\epsilon(i,\alpha_{m+1})$, a contradiction.

2. Suppose $\epsilon(i, \alpha_{m+1}) < \epsilon_0$. Then there exists some open node $w(j'') \in H_2'(i|m)$ having penalty smaller than the indicated minimum. We need consider only $w(j'') \in H_2'(i|m)$ since a minimal distance path from any $w(j'') \in H_2(i|m) - H_2'(i|m)$ must pass through an open node simply communicating with a closed node in $H_1(i|m)$, incurring a penalty not less than that of the simply communicating node. But for any $w(j'') \in H_2'(i|m)$, a decrease in penalty from that node associated with ϵ_0 requires a new multilink path from $w(j'')$ eventually to some node in $H_1(i|m)$. The penalty of any such path cannot be less than the penalty of the penultimate node in the path, which itself is contained in $H_2'(i|m)$. And this penalty cannot be less than ϵ_0, the indicated minimum. We have reached a contradiction, and the theorem is proved. ■

We now have all the results necessary to find the minimal distance paths from any root vertex $w(i)$ to all other vertices $w(j)$. Basically, after identifying all nodes that communicate with $w(i)$, we add one node at a time to the tree of our least-penalty nodes. This is a Dijkstra-type iteration performed on first differences rather than magnitudes of travel distances. The tree containing the m least penalty nodes for $w(i)$ is fully specified by

$$\gamma(i|m) \equiv \text{set of pairs of nodes and predecessor links in the tree of } m \text{ least penalty nodes for } w(i)$$

We now detail

Algorithm A2: Finding Minimum Penalty Nodes

Step 1: Initialization

For all e $w(j) \in F(w(i))$, $\epsilon(i,j) = 0$, $t(i,j) = |x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|$. Assign each α_i, $i = 1, 2, \ldots, e$, to a unique vertex having $\epsilon(i,j) = 0$.

$$m = e$$

$$H_1(i|m) = \{w(i) \in W: i \in \{\alpha_1, \alpha_2, \ldots, \alpha_m\}\},$$

$$H_2(i|m) = W - H_1(i|m),$$

$$\text{From A1, } \gamma(i|m) = \bigcup_{n=0}^{m} \Delta_n(w(i)).$$

Step 2: First Iteration. For all $w(j) \in H_2(i|m)$, compute

$$\hat{\epsilon}(i,j|m) = \text{MIN}\,\{\infty, \underset{n \in [1,\ldots,m]: d_{\alpha_n j} < +\infty}{\text{MIN}} [d_{\alpha_n j} + t(i, \alpha_n)] - (|x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|)\}$$

Step 3: Subsequent Iterations

Set $\epsilon_0 = \underset{j: w(j) \in H_2(i|m)}{\text{MIN}} [\hat{\epsilon}(i,j|m)]$

α_{m+1} = any vertex $j (w(j) \in H_2(i|m))$ having $\hat{\epsilon}(i,j|m) = \epsilon_0$.

$t(i, \alpha_{m+1}) = \epsilon_0 + |x_w(i) - x_w(\alpha_{m+1})| + |y_w(i) - y_w(\alpha_{m+1})|$.

The predecessor link $l(\alpha_n, \alpha_{m+1})$ in $\gamma(i|m)$ is determined by that α_n, say α_n^0, which yields the minimum ϵ_0.

TABLE I. POLYPATH: Summary Description.

Step 1: Network Formulation
Network node set $W = Q \cup V$.
Construct set S of simply communicating vertex pairs:
$w(i)$, $w(j)$ are included as a pair in S if
(a) they are adjacent vertices of a barrier,
(b) they are contained in the same vertex seeking tree, with one as a root vertex,
(c) they have x and y probes that intersect at a point other than another vertex.
For each $\{w(i), w(j)\} \in S$, the length of link $l(i, j)$ is

$$d_{ij} = |x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|;$$

else $d_{ij} = +\infty$.
Compute the node orientation matrix $\Phi = (\phi_{ij})$.
Step 2: Find Vertices Communicating with $w(i)$
Execute Algorithm A1: Tree Generation
Step 3: Find Minimal Distance Paths to Remaining Vertices
Execute Algorithm A2: Finding Minimum Penalty Nodes
Repeat for all i, $i = 1, 2, \ldots, n_Q$, if entire matrix $T = (t(i, j))$ is desired.

$$\gamma(i|m+1) = \gamma(i|m) \cup \{w(\alpha_{m+1}), l(\alpha_n^0, \alpha_{m+1})\},$$
$$H_1(i|m) = H_1(i|m) \cup w(\alpha_{m+1}),$$
$$H_2(i|m) = H_2(i|m) - w(\alpha_{m+1}),$$
$$m \leftarrow m + 1.$$

Step 4: New Estimated Penalties
Compute

$$\hat{e}(i, j|m) = \text{MIN}\,\{\hat{e}(i, j|m-1), \underset{j:d_{\alpha_m j} < +\infty}{\text{MIN}} [d_{\alpha_m j} + t(i, \alpha_m) - (|x_w(i) - x_w(j)| + |y_w(i) - y_w(j)|)]\}$$

If $H_2(i|m) = \phi$ then exit, else go to Step 3.

Algorithm A2 completes the POLYPATH algorithm, a summary description of which is given in Table I.

D. Computational Complexity

The computational complexity of the POLYPATH algorithm, once the network is constructed, is usually less than that of the Dijkstra algorithm. Recall that in our problem there are n_Q origin-destination points, $\Sigma_{m=1}^{n_B} v_m$ barrier vertices, and $n_W = n_Q + \Sigma_{m=1}^{n_B} v_m$ nodes in the constructed network.

Suppose our problem is to generate the entire $(n_Q \times n_Q)$ minimal travel distance matrix between all origin-destination pairs $(q(i), q(j))$. We do this by executing POLYPATH n_Q times, once for each possible origin point. [This ignores possible exploitation of the symmetry of the matrix.] Each such execution is done on an n_W-node network, yielding n_W minimal distances, and straightforward application of the Dijkstra "node labeling" procedure would require approximately $(5/2)\,n_W^2$ additions and comparisons [6].

But in POLYPATH the tree generation algorithm A1 can be executed with no additions or comparisons, just simple storage manipulations (file creation), if the D and Φ matrices are appropriately rearranged to allow easy tracing of trees through permissable directions. This is accomplished by grouping together all nodes that are direction η from $q(i)$, for each $\eta = 1, 2, \ldots, 8$, and using indirect addressing to retrieve these nodes. Using this technique, the computer can sequentially create the communicating tree of nodes (the zero penalty nodes) with essentially no marginal computational effort, beyond the fixed one-time effort required to group nodes by direction.

Suppose a fraction f of nodes in the complete n_W-node network communicate. Then, Dijkstra-type iterations must be performed only on the $(1 - f)\,n_W$ positive penalty nodes. (These iterations are performed only on the penalty distances, not absolute distances, but the computational work is the same.) The number of additions and comparisons required for each possible origin point is thus reduced to approximately $(5/2)\,[(1 - f)\,n_W]^2$. The computational complexity associated with generating the entire $(n_Q \times n_Q)$ matrix (ignoring its symmetry) is thus approximately $n_Q\,\frac{5}{2}\,[(1 - f)n_W]^2$. If $f = 0.5$, say, POLYPATH reduces the amount of computational work compared to Dijkstra, by 75%.

T. Wong has programmed POLYPATH in PL/I [7]. He has solved problems of considerable complexity, including finding an optimal path from the center of a 27-node maze to the center of an 18-node maze.

IV. EXAMPLE

We now illustrate the algorithm with a simple two-barrier, six origin-destination point example, shown in Figure 7. This problem includes all of the different situations that can occur with a vertex-seeking tree: nodes with 0, 1, 2, 3, or 4 probes and probes terminating at another origin-destination point, at a barrier vertex, or intersecting a barrier side. We shall find the minimum rectilinear distance to all origin-destination points from point 1.

The first step is the generation of the link length (D) and orientation (Φ) matrixes. (Table II). Each entry (i, j) of the matrix includes the pair d_{ij} and ϕ_{ij}, where d_{ij} is the link length from node i to node j and ϕ_{ij} is the orientation of j with respect to i. Any (i, j) for which $d_{ij} = +\infty$ is left blank. Although this matrix generation involves tedious but simple inspections, it can be done efficiently by a computer.

Using the D and Φ matrixes, we invoke Algorithm A1 to generate the tree of vertices that communicate with node 1, i.e., $F(w(1))$ in our notation. It is noted that nodes 7, 8, 14, 15, 16, 1-communicate with 1, while nodes 3, 5, 9, 11, 2-communicate with 1. There are no n-communicating nodes for $n \geqslant 3$. It is noted that 10 of the original 16 nodes are included in the tree. (See Fig. 8.)

The final step is to apply Algorithm A2 to find the minimum rectilinear distances to other nodes not on the tree. We use the notation of Algorithm A2:

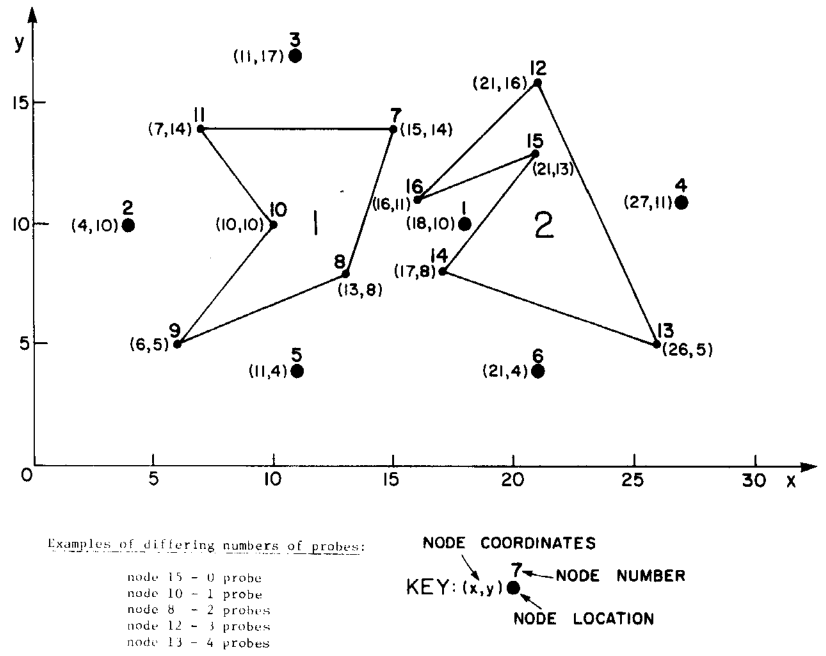

FIG. 7. Example: Two barriers, six origin-destination points.

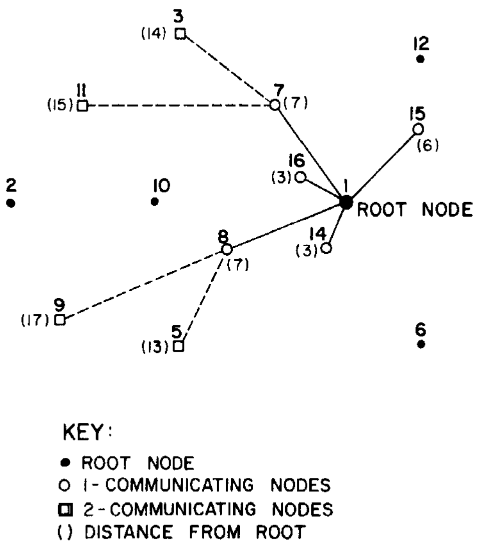

FIG. 8. Tree of vertices communicating with node 1 $[F(w(1))]$.

TABLE II. Link length and orientation matrices (D, Φ).[a]

Nodes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0 –						7 4	7 6						3 6	6 2	3 4
2		0 –	14 2		13 8				7 8	6 1	7 2	23 2				
3		14 6	0 –	22 8			7 8		17 6		7 6	11 8	27 8			11 8
4			22 4	0 –		13 6						11 4	7 6			
5		13 4			0 –	10 1	14 2	6 2	6 4				16 2	10 2		12 2
6				13 2	10 5	0 –	16 4	12 4	16 4				6 2	8 4		12 4
7	7 8		7 4		14 6	16 8	0 –	8 6	18 6		8 5	8 2	20 8	8 8		4 8
8	7 2				6 6	12 8	8 2	0 –	10 6				16 8	4 1		6 2
9		7 4	17 2		6 8	16 8	18 2	10 2	0 –	9 2	10 2	26 2	20 1	14 2		16 2
10		6 5							9 6	0 –	7 4					
11		7 6	7 2				8 1		10 6	7 8	0 –	16 2				
12		23 6	11 4	11 8			8 6		26 6		16 6	0 –	16 8			10 6
13			27 4	7 2	16 6	6 6	20 4	16 4	20 5			16 4	0 –	12 4		16 4
14	3 2				10 6	8 8	8 4	4 5	14 6				12 8	0 –	9 2	4 4
15	6 6													9 6	0 –	7 6
16	3 8		11 4		12 6	12 8	4 4	6 6	16 6			10 2	16 8	4 8	7 2	0 –

[a]Key: The matrix pair $(a(i,j)), b(i,j))$ in row i, column j is (d_{ij}, ϕ_{ij}). [Any (i,j) for which $d_{ij} = +\infty$ is left blank.] Note: $d_{ji} = d_{ij}$, $\phi_{ji} = (\phi_{ij} + 3)_{\text{mod } 8} + 1$.

Step 1: Initialization

10 closed nodes: $\{1, 3, 5, 7, 8, 9, 11, 14, 15, 16\}$,

$t(1,1) = 0, t(1,3) = 14, t(1,5) = 13, t(1,7) = 7, t(1,8) = 7,$
$t(1,9) = 17, t(1,11) = 15, t(1,14) = 3, t(1,15) = 6, t(1,16) = 3,$

$\alpha_1 = 1, \alpha_2 = 3, \alpha_3 = 5, \alpha_4 = 7, \alpha_5 = 8, \alpha_6 = 9, \alpha_7 = 11, \alpha_8 = 14,$
$\alpha_9 = 15, \alpha_{10} = 16,$

$m = 10,$

$H_1(1|10) = \{w(i) \in W: i \in \{\alpha_1, \ldots, \alpha_{10}\}\},$

$H_2(1|10) = \{2, 4, 6, 10, 12, 13\},$

$\gamma(1|10) = \{\{1, \phi\}, \{7, [7,1]\}, \{8, [8,1]\}, \{14, [14,1]\},$
$\{15, [15,1]\}, \{16, [16,1]\}, \{3, [3,7]\}, \{11, [11,7]\},$
$\{5, [5,8]\}, \{9, [9,8]\}\}.$

Step 2: First Iteration (See Table III)

TABLE III. Summary of computations for first iteration.

Using Closed Node \ Open Node	2	4	6	10	12	13
	$\hat{\epsilon}(1,2\|10)$	$\hat{\epsilon}(1,4\|10)$	$\hat{\epsilon}(1,6\|10)$	$\hat{\epsilon}(1,10\|10)$	$\hat{\epsilon}(1,12\|10)$	$\hat{\epsilon}(1,13\|10)$
3	14	㉖			16	28
5	12		14			
7			14		6	14
8			10			10
9	10			18		
11	⑧			⑭		
14			②			②
15						
16					④	

$\hat{\epsilon}(1, 2|10) = \text{MIN}\ \{14, 12, 10, 8\} = 8$ (hence circled in Table III).

Similarly,

$\hat{\epsilon}(1, 4|10) = 26, \hat{\epsilon}(1, 6|10) = 2, \hat{\epsilon}(1, 10|10) = 14,$

$\hat{\epsilon}(1, 12|10) = 4, \hat{\epsilon}(1, 13|10) = 2,$

$$\epsilon_0 = \min_{j: w(j) \in H_2(1|10)} \{\hat{\epsilon}(1, j|10)\} = \text{MIN}\ \{8, 26, 2, 14, 4, 2\} = 2,$$

$\alpha_{10+1} = \alpha_{11} = 6$ or 13 (Ties can be broken arbitrarily; choose $\alpha_{11} = 6$),

$t(1,6) = 2 + c_{16} = 2 + 9 = 11.$

Step 3: Second Iteration

Using closed node 6.

$$\hat{\epsilon}(1, j|11) = \hat{\epsilon}(1, j|10) \text{ for } j = 2, 10, 12, 13,$$

$$\hat{\epsilon}(1, 4|11) = 14,$$

$$\epsilon_0 = \min_{j: w(j) \in H_2(1|11)} \{\hat{\epsilon}(1, j|11) = \text{MIN}\ \{8, 14, 14, 4, 2\} = 2,$$

$$\alpha_{11+1} = \alpha_{12} = 13,$$

$$t(1, 13) = 2 + c_{1(13)} = 2 + 13 = 15.$$

This process continues through four more steps, at which point all nodes are closed and the algorithm is terminated.

V. EXTENSIONS

We conclude the paper with remarks on three possible extensions:

(1) The same algorithm can be used to develop optimal paths around smooth (differentiable) barriers. Each such barrier provides a node to the associated network at every point on the barrier perimeter at which a tangent to the perimeter is parallel to one of the two directions of travel.* (The definition of the vertex seeking tree must be modified in an obvious way when an eminating ray intersects a barrier.)

(2) For polygonal barriers, the observation above points to a procedure for reducing the number of barrier-generated nodes. Simply eliminate all barrier nodes from which only two probes (rather than 4, 3, 1, or 0) can be constructed and modify the definition of a vertex seeking tree accordingly.

(3) For the case in which the entire $n_Q \times n_Q$ travel distance matrix is desired, the ideas of this paper can be used in the Floyd-Warshall algorithm [8], [9]. Two different possibilities come to mind. First, one could find the set of communicating nodes directly using the "permissible direction" ideas of Section III and then perform Floyd-Warshall iterations only for the unknown distances (having positive penalties). Or, second, without any preprocessing, one could perform Floyd-Warshall iterations on the entire set of $n_Q \times n_Q$ *penalty* distances, immediately recognizing convergence of a travel time value whenever the penalty distance at an intermediate iteration is computed to be zero (implying communication between the corresponding two nodes).

References

[1] E. Damm, "Der Kurzeste Weg in einem Gitternetz mit Hindernissen," *Angew. Inf. Appl. Inf.*, **19**, 213-216 (1977).

[2] G. E. Wangdahl, S. M. Pollock, and J. B. Woodward, "Minimum-Trajectory Pipe Routing," *J. Ship Res.*, **18**, 46-49 (1974).

[3] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, **1**, 269-271 (1959).

*This is obviously not precisely true when the barrier includes straight line segments parallel to the directions of travel; in such cases, we revert to the original algorithm.

[4] H. Vaccaro, "Alternative Techniques for Modeling Travel Distance," Masters thesis in Civil Engineering, Massachusetts Institute of Technology, June 1974.
[5] T. Lozano-Perez and M. Wesley, "An Algorithm for Planning Collision-Free Paths Amongst Polyhedral Obstacles," IBM Thomas J. Watson Research Center, RC 7171, June 1978.
[6] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Oper. Res.*, 17, 395–412 (1969).
[7] T. Wong, "Minimum Rectangular Distance Paths Between Two Points on a Plane with Barriers," B.S. thesis in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1979.
[8] R. W. Floyd, "Algorithm 97, Shortest Path," *Comm. ACM*, 5, 345 (1962).
[9] S. Warshall, "A Theorem on Boolean Matrices," *J. ACM*, 9, 11–12 (1962).

Received June 5, 1979
Accepted September 29, 1980