Graphbots: Cooperative Motion Planning in Discrete Spaces

Samir Khuller, Ehud Rivlin, and Azriel Rosenfeld, *Life Fellow, IEEE*

***Abstract*—Most previous theoretical work on motion planning for a group of robots has addressed the problem of path planning for the individual robots sequentially, in geometrically simple regions of Euclidean space (e.g., a planar region containing polygonal obstacles). In this paper, we define a version of the motion-planning problem in which the robots move simultaneously. We establish conditions under which a team of robots having a particular configuration can move from any start location to any goal destination in a graph-structured space. We show that for a group of robots that maintain a fixed formation we can find the "shortest" path in polynomial time, and we give faster algorithms for special kinds of environments.**

***Index Terms*—Graphs, motion planning.**

I. Introduction

MANY robotic tasks can be handled more efficiently and robustly by a group of robots. Various constraints may be imposed on the group by the common task at hand. These constraints may be internal, imposed by the nature of the task, or external, imposed by the environment. Consider, for example, a team of robots trying to cooperatively clean a large system of air-conditioning pipes (see Fig. 1). The group leader is assumed to have a map of the environment and should plan a path for the group that will take into consideration the constraints imposed by the topology of the air-conditioning system. The most common kinds of maps used in robot navigation are quantitative. Such maps directly represent geometric information about the environment. Alternatively, qualitative maps, such as topological graphs, can be used to model robot environments [15]. This kind of representation reflects qualitative information about the environment. In our case, the fact that pipes and their intersections (presumably) do not permit more than one robot at a time to pass motivates a topological representation of locations (intersections) and their connecting paths (the pipes joining them) by using a graph model (as in [13]). The group leader may impose other task-relevant constraints on the group's motion. These constraints might reflect a need to maintain communication, or mutual coverage, and might also reflect different abilities of the group's members.

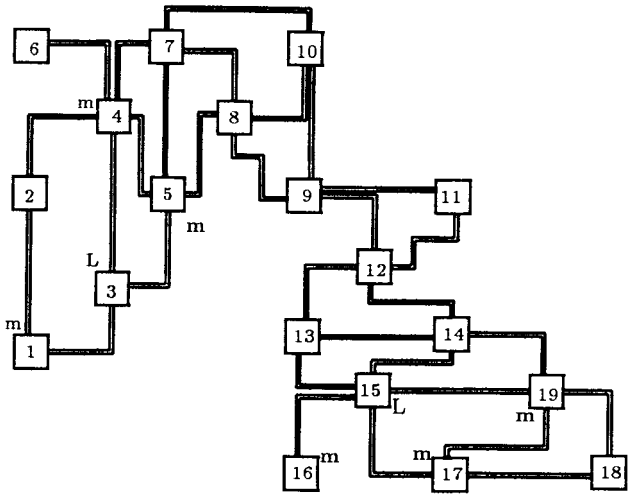

Fig. 1. Team of robots (one leader L and three team members m) in an air-conditioning system. Starting position in the lower right, goal position in the upper left.

In this paper, we study motion planning for a group of robots that operate in an environment that can be modeled discretely, i.e., as a graph, and where the group is trying to maintain a specific formation, represented by a subgraph.

A. Mobility in a Graph

We begin by formally defining our model for mobility of a graph-structured group of robots in a graph-structured space. Let G be a connected graph, which we usually refer to from now on as the "graph space," and let H be a connected subgraph of G, which we refer to from now on as the "group" [representing a cooperating team of (point) robots]. We allow H to incrementally "move" in G; as it moves, we require it to remain isomorphic to itself. Formally, this is defined as follows: two isomorphic subgraphs K, L of G are said to differ by a *local displacement* if corresponding nodes (under the isomorphism) are either identical or are neighbors. A *movement* or *motion* of H from S to T (which we call the start and target locations or "states") is defined by a sequence of subgraphs $S = H_0, H_1, \cdots,$ and $H_k = T$, all isomorphic to H, such that H_i and H_{i-1} differ by a local displacement $1 \leq i \leq k$. We say that H moves from S to T by a

Manuscript received August 12, 1995; revised August 27, 1996 and February 2, 1997. This work was supported by NSF Research Initiation Award CCR-9307462, NSF Career Award CCR-9501355S, and by ARPA and AFOSR under Grant F49260-95-1-0462.

S. Khuller is with the Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA (e-mail: samir@cs.umd.edu).

E. Rivlin is with the Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel, and Center for Automation Research, University of Maryland, College Park, MD 20742 USA (e-mail: ehudr@cs.technion.ac.il).

A. Rosenfeld is with the Center for Automation Research, University of Maryland, College Park, MD 20742 USA (e-mail: ar@cfar.umd.edu).

Publisher Item Identifier S 1094-6977(98)01525-9.

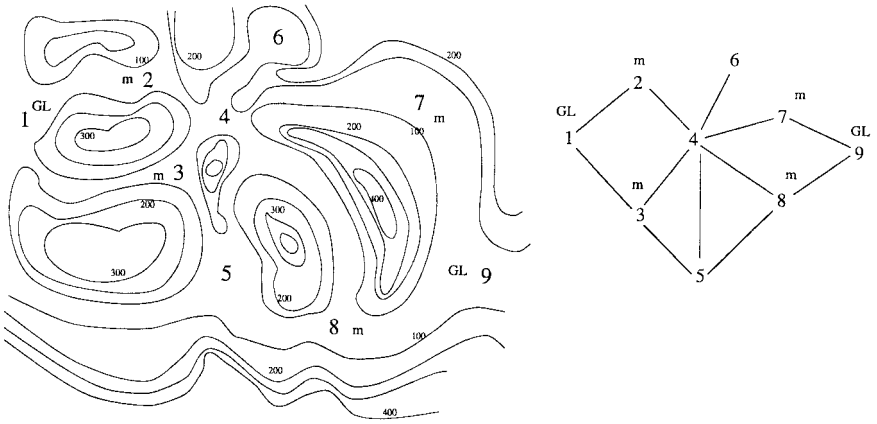

Fig. 2. Three UGV's operating in rugged terrain need to move from the start positions (2,1,3) to the goal positions (7,9,8) while maintaining their formation and minimizing their exposure. On the right is the graph representation.

sequence of k-local displacement steps. Note that a location (or "placement") of H in G is a subgraph of G that is isomorphic to H; two isomorphisms of H into G that have the same image are regarded as defining the same placement.

Example: Consider a group of unmanned ground vehicles (UGV's; see [16]) that need to move from a set of start positions to a set of goal positions in the area shown in Fig. 2. The group is heterogeneous, and the group leader (GL) can protect the members of the group when they are stationary and in a certain formation (which reflects constraints like distance, visibility, etc.). The nodes of the graph represent locations where a UGV can stop to communicate, reassess the situation, etc. Moving from one node to another is done along a relatively unexposed but unsecure path (see [22]). We represent the members of the team by single nodes, and each time they move, we require them to reestablish their formation, which defines the graph structure of the team. Note that a situation in which two robots occupy the same node represents a threat to the group and is therefore forbidden. The external constraints (the environment) motivate a graph-like representation for the locations and the unexposed paths. These tactical demands and the need for minimum exposure dictate the team's formation.

Our definition of movement is a discretized description of the way the group of UGV's is moving. The need to keep the time in which each UGV is moving to a minimum results in a series of steps, each of which represents a move of the group from one secure position to the next. Checking if such a series of moves is possible and trying to plan for the shortest such series is the topic of this paper.

B. Outline of Results

Let a group of robots in a formation be modeled by a graph H. The formation exists in an environment modeled by a "graph space" G. It is desired that the group maintain the same formation while moving around in G. This is not always possible; H can only occupy positions in which G is "big" enough to contain an isomorphic copy of H. Thus, H cannot always "move freely" in G. If H has only one or two nodes, it can obviously move freely in any connected graph G, but bigger H's cannot move freely in an arbitrary G.

In this paper we explore two kinds of questions.

- What conditions should G satisfy so that a given graph H can move freely in G? In other words, under what conditions can we guarantee the existence of a motion between *any two* valid placements of H in G? (We shall assume that there exist at least two possible locations, since if there is only one location, H cannot move at all.) We consider many different types of H's and establish conditions on G that permit a motion between any two valid placements of H. Our proofs easily lead to efficient algorithms for planning a motion between any two locations of H.
- Given a start state S and a target state T for the group, what is the complexity of finding a motion with the fewest local displacements from S to T (if one exists)?

We consider the first question for various types of H's. Even for very simple H's, this question turns out to be quite interesting and nontrivial. The results characterize various classes of graphs that permit free movement of various forms of H's, which we name after various types of arthropods[1] and which we illustrate in Fig. 3. In Section II, we give necessary and sufficient conditions on G for the movement of a two-vertex path, which we call a *tick*. In Section II-B, we give necessary and sufficient conditions on G for a three-vertex path, a *scorpion*, to be able to move freely in G, and in Section II-C, we give sufficient conditions for a three-vertex cycle, a *trilobite*, to move freely. These conditions do not generalize straightforwardly to paths or cycles (or cliques) that have more than three nodes; but in Section II-D, we give a simple sufficient condition on G for a $(k+1)$-node star, a k-legged *spider* consisting of a "central" node and k of its neighbors, to be able to move freely in G. A four-vertex cycle is referred to as a *jellyfish*, and it can move freely on a grid graph.

In Section III, we show that for any *fixed* graph H, the second question can always be answered in polynomial time. Unfortunately, the time complexity of the algorithm is quite high, albeit polynomial. For special kinds of graphs G, we give

[1] A major group of segmented invertebrates having jointed legs. (Strictly speaking, not all of our examples are arthropods.)

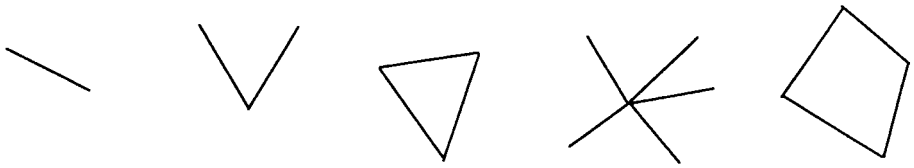

Fig. 3. Different formations that will be considered in the paper. From left to right: a tick, a scorpion, a trilobite, a five-legged spider, and a jellyfish.

faster algorithms. We also prove that the problem is NP-hard when H is part of the input. This reduction is also outlined in Section III.

In Section IV, we give an example involving the coordinated motion of three UGV's on rugged terrain. Section V contains a discussion and suggestions about future work.

C. Related Work

Several papers have addressed the problem of motion planning for multiple robots moving in Euclidean space (e.g., [8] and [20]). In [4] and [14], motion planning for a group of moving robots is addressed. The assumption is that the robots move sequentially. A similar assumption is made in [11], where motion planning for multiple mobile-tethered robots is described. The robot's movements and configurations are constrained by the tethers. In [19], control laws for governing the behavior of a team of agents are proposed. The paper describes a motion of a group that is parallel to the one-step motion in our model. These principles can be used by our planner after the next move is decided upon. Work similar in spirit to ours is that by Papadimitriou *et al.* [18]. Their work is similar in the sense of trying to capture a well-studied geometric motion-planning problem in a graph setting. In their work, it is assumed that the robot is a single vertex and the obstacles (also single vertices) are considered to be movable. The robot needs to plan a motion and has the power to request obstacles to move. (Another way to view this situation is to think of a group of robots located in the graph; one of them wants to plan a motion and may request the others to cooperate.) In our work, we have tried to model the movement of a group of robots that has a certain formation and that moves in the presence of fixed obstacles. Since the obstacles are fixed, we can simply delete the nodes occupied by them from the graph; thus, we never need to refer to them. (Similarly, fixed obstacles in the plane can simply be treated as holes in the plane.) A preliminary version of this paper appeared in [12].

II. Moving Robots Freely

A. Moving a Tick

A "tick" is modeled by a two-vertex graph linked by a single edge. The proof of the following proposition is obvious.

Proposition 2.1: A tick can move freely in any connected graph.

Note that if G has only two vertices (and is connected), then there is a unique placement of the tick. (Both isomorphisms define the same placement.)

B. Moving a Scorpion

A "scorpion" is modeled by a three-vertex graph linked by two edges. We will refer to the degree-two vertex as b (the coordinator or the "body") and the degree-one vertices as f (the "feet").

Theorem 2.2: A scorpion can move freely in G if and only if G does not contain a vertex v with two neighbors of degree 1.

Note that we regard the two placements of the scorpion on a three-vertex, two-edge graph as identical.

Proof: If G has such a vertex v, we can place the scorpion with b on v and the f's on the neighbors v_i of v. Any movement of the feet requires that both must move to v. If we move one foot to v, then b must leave v, and thus, the other foot will not be adjacent to b's new location. Thus, the scorpion cannot move from this location.

To prove the converse, assume there is no such vertex v. Let the initial placement of the scorpion be at u_1, u_2, u_3 (one f on u_1, b on u_2, and the other f on u_3), and let the target location be v_1, v_2, v_3. If there is a path joining a foot of the scorpion in the initial location to a foot of the scorpion at the final location, without passing through u_2 and v_2, then the scorpion can "creep" along this path to reach its destination. Let us assume that the only path from (u_1, u_2, u_3) to (v_1, v_2, v_3) goes through either u_2 or v_2 (or both). Since the degree of u_1 is not one (the case when the degree of u_3 is not one is identical), u_1 has a neighbor u_0 (other than u_2). Move the scorpion as follows. One foot f goes from u_1 to u_0, b goes to u_1, and the other foot f goes from u_3 to u_2. Now the scorpion can creep along the path to the desired destination. (At the other end, a similar orientation step may be required.) The details are left to the reader. □

Basically, to corner a scorpion, you have to completely immobilize it at its starting location. This proof easily extends to an algorithm that can move a scorpion from any initial location to a target location, if a motion exists.

C. Moving a Trilobite

A "trilobite" is modeled by a three-vertex graph linked by three edges (i.e., a clique of size three). We will give some conditions under which free motion of a trilobite is possible. Before discussing the details of the conditions that the graph must satisfy, we review some definitions [3].

Biconnected Graphs: A single vertex in a connected graph whose deletion disconnects the graph is called a *cut vertex* (also known as an *articulation vertex*). A graph with no cut vertices is called *biconnected*.

λ-Connected Graphs: A graph is said to be λ-connected if the deletion of *any* subset of $\lambda - 1$ vertices leaves the

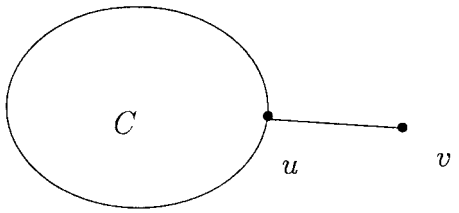

Fig. 4. Case in Lemma 2.5 when v has one neighbor.

graph connected. We also require that G contain at least $\lambda+1$ vertices.

Chordal Graph: A chordal graph is a graph in which each cycle of length of at least four has a chord. (A chord is an edge that connects two vertices that are not adjacent on the cycle.)

Perfect Elimination Orderings: A perfect elimination ordering (peo) is a numbering of the vertices from $(1, \cdots, n)$ such that, for each i, the higher numbered neighbors of vertex i form a clique. Thus, a peo is represented by a sequence σ of vertices.

Theorem 2.3) Fulkerson and Gross [9]: A graph G has a perfect elimination ordering if and only if G is chordal.

We will use this characterization of chordal graphs to prove the following theorem.

Theorem 2.4: A trilobite can move freely in a biconnected chordal graph that has at least three vertices.

Note that if G itself is a clique of size three, then a trilobite has only one possible location (recall that we do not count locations obtained by permuting the vertices as distinct locations).

Before we prove the theorem, we first prove the following simple lemma.

Lemma 2.5: Let G be a biconnected chordal graph that has at least three vertices. Consider a peo for G; then any vertex i ($i = 1, \cdots, n-2$) must have at least two neighbors with higher numbers.

Proof: We will prove this by contradiction. Start deleting vertices one at a time, starting with vertex 1. Stop the deletion process when we are about to delete vertex v that has fewer than two neighbors in the current graph. There are two possible cases.

Case 1) Vertex v Has One Neighbor u (see Fig. 4): The other vertices adjacent to v have been deleted. We now claim that u is a cut vertex (in G) that separates v from the other vertices in C (by our assumption, v's number in the peo is between 1 and $n-2$; thus, C has at least one vertex other than u). To prove this by contradiction, assume that P is the shortest path (in G) from v to any vertex in C that avoids u. Observe that P has at least two edges in it since v is not adjacent to any vertex in C (other than u). Consider the first vertex of P that we delete. There must be an edge between its two neighbors; this gives a shorter path from v to C, contradicting the assumption that P was the shortest path from v to some vertex in C (avoiding u).

Case 2) Vertex v Has No Neighbors: Let w be the most recently deleted vertex among the neighbors of v in G. By our assumption, about v, w had at least two neighbors when it was deleted; since these formed a clique, v must have a neighbor that has not been deleted as yet (by our assumption on w). Thus, this case cannot occur. □

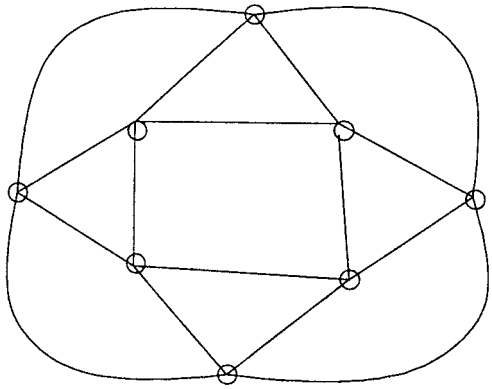

Fig. 5. Chordiality is not necessary for moving a trilobite freely.

Proof of Theorem 2.4: We now use Lemma 2.5 to prove the theorem by induction on the number of vertices of G. The base case is a chordal graph with three vertices. A biconnected graph on three vertices is a clique of size three and the theorem is trivially true for this graph. Consider a peo of G. Assume that we have deleted vertices $1, \cdots, i$ and have a graph G_i left. The subgraph G_i is chordal since it is an induced subgraph of G. G_i is also biconnected since it does not have any cut vertices. (If it has a cut vertex a separating it into two connected components C_1 and C_2, then when the last vertex in the peo is removed from the component that is exhausted first, this vertex will have only a as its neighbor; this is a contradiction to Lemma 2.5.) Since G_i is biconnected and chordal, by the induction hypothesis we know that the trilobite can move freely on G_i. What happens if we add vertex i back? We have to show that for each location of a trilobite that uses i, it can move to any location in G_i. In a placement of the trilobite using vertex i, let the other corners of the trilobite be at vertices x and y. Without loss of generality, assume that $x < y$. If $x \leq n-2$, then x has at least one other neighbor z with a higher number, such that y is adjacent to z (this follows from Lemma 2.5 and the definition of a peo). Thus, we can move the trilobite from i, x, y to x, z, y. Since x, y, and z are all in G_i, by the induction hypothesis the trilobite can also move to any other location in G_i. The other case to consider is when $x = n-1$ (in this case $y = n$). Since G_i has more than three vertices, vertex $n-2$ is distinct from vertex i. This vertex is also adjacent to both vertices n and $n-1$. We can move the trilobite from i, $n-1$, n to $n-1$, $n-2$, n. This concludes the proof. □

It is easy to see that valid locations of a trilobite cannot be separated by a cut vertex. However, we can construct graphs that are biconnected but not chordal, which allow free movement of the trilobite (see Fig. 5). Hence, chordiality is sufficient, but *not* necessary.

D. Moving a Spider

A k-legged "spider" is modeled by a $(k+1)$-vertex graph having a central vertex, denoting its "body" (vertex b), linked by edges to vertices $f_1, \cdots, f_k$, representing its "feet." We

give conditions under which free motion of a spider is possible. When $k = 1$, we have a one-legged spider, which is a tick; this case has already been dealt with. Section II-B gave an exact characterization of the graphs a two-legged spider (also called a scorpion) can move freely in. In the remainder of this subsection, we will assume that $k \geq 3$.

Theorem 2.6: A k-legged spider can move freely in a $(k-1)$-connected chordal graph. (We assume that the graph has at least two distinct placements of the spider.)

For the rest of the discussion in this subsection, we will assume that the underlying graph is $(k-1)$-connected and chordal. Observe that Menger's theorem [3] guarantees the existence of $k-1$ internally vertex-disjoint paths between any pair of vertices in G.

We first prove some interesting lemmas that are used in the proof of Theorem 2.6. Before we prove the lemmas, we introduce the following notation. Let $N(v)$ denote the set of vertices that are adjacent to vertex v. Let $N(u, v)$ denote the set of vertices that are adjacent to vertices u and v. In other words, $N(u, v) = N(u) \cap N(v)$.

Lemma 2.7) Common Neighbors Lemma: Let G be a graph that is $(k-1)$-connected and chordal. If x and y are two adjacent vertices, then $|N(x, y)| \geq k-2$.

Proof: Consider a set of $k-1$ vertex-disjoint paths from x to y. Clearly, one path is the edge from x to y. Let $P_i (i = 1, \cdots, k-2)$ be the other $k-2$ paths from x to y. Of all such paths, we can pick the set of $k-2$ paths that have the shortest total length. If path P_i has three or more edges, then, together with the edge (x, y), it creates a chordless cycle of length four or more, which is a contradiction to the assumption that the graph is chordal. Let P_i be (x, u_i, y). Clearly, all the vertices u_i are in $N(x, y)$. This completes the proof.

Lemma 2.8) Connectivity Lemma: Let G be a graph that is biconnected and chordal. For any vertex x, if G_x is the graph induced by the neighbors of x, then G_x is connected.

Proof: Suppose G_x were not connected. Let $C_1, \cdots, C_\ell$ be the connected components of G_x. Since G is biconnected, for any pair of vertices we must have a path in G avoiding x that connects the pair. Of all such paths, let P be the shortest path connecting two vertices y and z, such that $y \in C_i$ and $z \in C_j$ with $(i \neq j)$. Since there is no edge (y, z), this path must have at least two edges. In this case, the edge (x, y), together with $P(y, z)$ and the edge (z, x), form a chordless cycle of length four or more; this is because x has no edges to internal vertices of P. This is a contradiction to the assumption that G_x has at least two components. Hence, G_x is connected.

Lemma 2.9) Rotation Lemma: Consider a spider located with its body b at vertex v and its feet $f_1, \cdots, f_k$ at vertices $u_1, \cdots, u_k$. There is a legal motion that moves the k feet to any desired subset $S \subseteq N(v)$ with $|S| = k$.

Proof: By the Connectivity Lemma, G_v is connected; let T_v be a spanning tree of G_v. Using T_v, we will move the feet of the spider from their current locations to the vertices in S. Pick any foot f_i at vertex u_j. The vertices in S that currently do not have a foot on them are called "free" vertices, and those that have a foot on them are called "occupied" vertices.

Let P be a path in T_x that connects u_j with some free vertex in S. We start moving the foot f_i along the path P from u_j to the free vertex. If any vertex on the path is occupied by a foot f_p, we stop moving f_i and start moving f_p. In other words, we exchange the role of f_i and f_p. In this way, we can move all the feet to vertices in S.

Observe that the above lemma lets us reduce the problem of moving a spider from one location to another to the problem of moving the body without worrying about the placement of the feet.

Remark: The above lemma moves the feet one by one. If we want to move the feet as quickly as possible, it is possible to compute a set of $k-2$ shortest disjoint paths between the initial locations of the feet and the final locations. (Observe that the feet in "front" of and "behind" the body are not moved by the Rotation Lemma.) This works since the graph G is $(k-2)$-connected even after we delete the vertex occupied by the body.

Proof of Theorem 2.6: Assume that the start location of the spider's body is at vertex s and its target location is at vertex t. We will concentrate on moving the spider's body from vertex s to vertex t. Using the Rotation Lemma, we can always move the feet from any set of neighbors of a vertex to any desired set of neighbors of the vertex.

Let $P = (s = v_0, v_1, \cdots, v_\ell = t)$ be the shortest path in G from s to t. We will show how the spider can move from v_i to v_{i+1} by a local displacement. We first apply the Rotation Lemma and move the k feet to the set $S_i = v_{i-1} \cup v_{i+1} \cup N(v_i, v_{i+1})$. (The only cases in which this is an invalid set are $i = 0$ and $i = \ell$; these will be dealt with later.)

The body of the spider moves from v_i to v_{i+1}. The spider then moves the foot at v_{i+1} to v_{i+2} and the foot at v_{i-1} to v_i. The feet in the set $N(v_i, v_{i+1})$ do not need to move at all in this step. Observe that for the next local displacement, we apply the Rotation Lemma to move the feet to vertices in $N(v_{i+1}, v_{i+2})$ before we can move the body again. In particular, the feet are moved from the current location to $S_{i+1} = v_i \cup v_{i+2} \cup N(v_{i+1}, v_{i+2})$.

When $i = 0$, we define v_{-1} as any vertex in $N(v_0)$ that does not have a foot placed on it. When $i = \ell$, we define $v_{\ell+1}$ as any vertex in $N(v_\ell)$ that does not have a foot placed on it. □

Let n and m, respectively, denote the number of vertices and edges in G. Our proof also yields an efficient algorithm for finding the motion between the start and target locations. The shortest path from s to t can be found in $O(m)$ time by using breadth—first search. At each step, we can move the legs in $O(m)$ time. This yields an $O(nm)$ time algorithm for moving the spider from any start location to a target location (since the body moves for at most n steps).

Remark: Using a modification of the basic approach described above, we can design an algorithm with running time $O(m)$ (assuming that k is a fixed constant). The main bottleneck in our algorithm is due to the fact that we need to apply the Rotation Lemma each time the body moves one step. The following idea yields a linear time algorithm.

First compute $k-1$ vertex-disjoint paths $P_1, \cdots, P_{k-1}$ between s and t. (These can be obtained by finding a flow of value $k-1$ in an appropriately defined flow network [5]). Next,

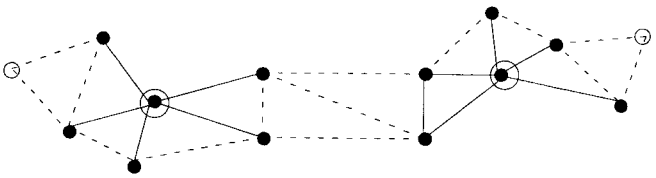

Fig. 6. A five-legged spider cannot move freely in a biconnected chordal graph.

preprocess these paths to make them minimal, so they have no chords that provide shortcuts. This is easily accomplished by labeling each vertex on P_i by its distance from s along the path and traversing P_i a second time, picking the neighbor with the highest label at each step. The new paths are called P_i', P_2', $\cdots$, P_{k-1}'.

Let $x_i (1 \leq i \leq k-1)$ be the vertex adjacent to s on the path P_i'. The spider does one rotation step to put $k-1$ feet on the x_i vertices. Observe that $s \cup_{i=1}^{k-1} x_i$ form a clique, since G is chordal. The spider can move its body to any neighbor x_j on one of the paths and drag the last foot behind it. We need to recompute chordless $k-1$ vertex-disjoint paths in $O(d(x_j))$ steps, where $d(x_j)$ is the degree of vertex x_i. This can be done by examining the neighbors of x_j on the paths P_i'.

The feet are once more moved onto the new set of paths and then the body moves again. Once the body reaches t, we need to do one more rotation step to finish the motion. This method saves the cost of doing a rotation step each time the body moves and thus, runs in $O(m)$ time.

E. Moving a Four-Legged Spider

We showed that we can move a three-legged spider in a biconnected chordal graph; this follows from the theorem in the previous section with $k = 3$. In this section, we prove the slightly stronger result that a four-legged spider can move freely in a biconnected chordal graph. This is the best possible, since we can show by an example that a five-legged spider cannot move freely in a biconnected chordal graph (see Fig. 6). Clearly, there are two feasible locations for the five-legged spider at the two vertices of degree five; but since these are the only two degree-five vertices, no movement is possible between the two locations of the spider.

The proof for the four-legged spider is more complicated, since the proof technique that worked for the three-legged spider in a biconnected graph does not work for four-legged spiders. (In particular, it is *not* the case that we can select a shortest path and move the body along this path, since there may be degree-three vertices on this path.)

Theorem 2.10: A four-legged spider can move freely in a biconnected chordal graph.

Proof: Let the spider's body be at vertex s, and let the target location of the body be t. By the Rotation Lemma, we can see that it is easy to "rotate" the legs without moving the body. Hence, we concentrate on moving b from s to t.

We prove the lemma by induction on the length of the shortest cycle that passes through both s and t. It is clear that such a cycle always exists since the graph is biconnected.

The base case is when this cycle has length three. In this case, vertices s and t share a common neighbor u. By Lemma 2.9, we can assume that one foot is on vertex u and the other is on vertex t. Assume that the third and fourth feet f_3 and f_4 are on vertices v_3 and v_4. We would like to move b from s to t; the foot at t must move to a destination location for the foot. One foot will continue to occupy u, and one foot will move to s. The fourth foot will have to be moved while the body moves from s to t.

Note that if s and t have two common neighbors, then, using Lemma 2.9, we can move two feet to those neighbors, and then move b from s to t (the foot at t moves to its final position, and one foot moves to s). Assume now that s and t have only one common neighbor. Note that G_s is connected (Connectivity Lemma), hence, there exists a path P_s from v_3 to t in G_s that does not use v_4 (if it uses v_4, replace v_3 by v_4 in the rest of the argument). Moreover, observe that in G_s vertex t has degree one (since s and t have only one common neighbor), and this neighbor is u. Hence, P_s passes through u.

We will move v_3 to the node of P_s just before u. There exists a path P_t in G_t from u to a final destination for a foot, which does not use s since s has degree one in G_t and its sole neighbor is u. We make the following moves in one step: the foot at u to the first node on P_t, the foot that came from v_3 on P_s to u, the foot at t to its final destination, b from s to t, and the foot at v_4 to s (see Fig. 7).

Let us assume by the induction hypothesis that there is a way to move the four-legged spider between two locations if a cycle of length $i-1$ passes through the two locations. We will show how to find a solution when the shortest cycle has length i. Let the shortest cycle passing through s and t be of length $i > 3$. We can think of this cycle as two internally vertex-disjoint paths P_1 and P_2 from s to t. Since this is the shortest cycle, and is of length of at least four, it must have a chord. Any such chord must clearly go from an internal vertex on P_1 to an internal vertex on P_2. (Any other chord would create a shorter cycle passing through s and t.) Of all such chords, pick a chord (x_1, x_2) with $x_i \in P_i$ that minimizes the length of the cycle $P_1(s, x_1)$; (x_1, x_2); $P_2(x_2, s)$. Clearly, this chord connects the two vertices that are the neighbors of s on P_1 and P_2. We first move the feet f_1 and f_2, such that they are on the paths P_1 and P_2, respectively. Assume that f_3, f_4 are on vertices v_3, v_4, respectively. Since G_s is connected (Connectivity Lemma), there is a path joining (v_3, v_4) to (x_1, x_2). Consider the shortest such path in the graph G_s. Assume it goes from v_3 to x_1 (the other case is similar). Let x_3 be the last vertex (other than x_1) on this shortest path. Clearly, it is adjacent to x_1.

First move f_3 from v_3 to x_3 along the shortest path in G_s. Now move b to x_1, f_1 to the next node on P_1, and f_4 to s. This results in a new location with a strictly shorter cycle passing through t and the new location of the body b (see Fig. 8). Applying the induction hypothesis completes the proof.

III. Shortest Motion

Assume that G has n vertices and H has ℓ vertices. There are at most a polynomial number of possible locations of H in G for any fixed-size graph H. We construct a new graph $G' = (V', E')$ in which each vertex corresponds to a possible valid location of H in G. There is an edge in E' between

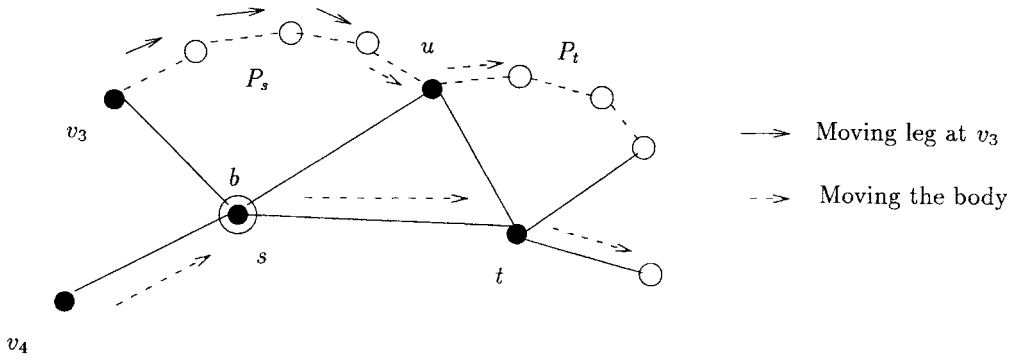

Fig. 7. Base case of induction for proof of Theorem 2.10.

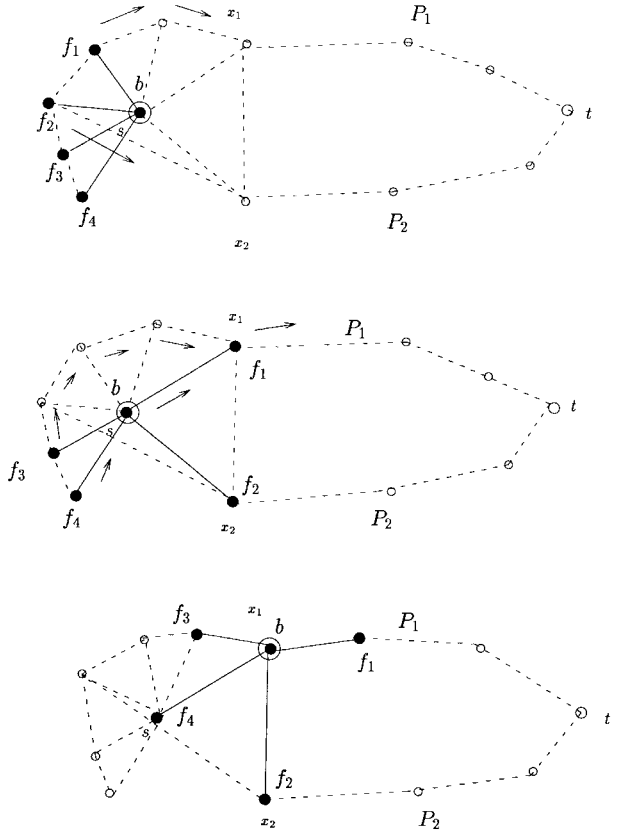

Fig. 8. Four-legged spider movement.

$u \in V'$ and $v \in V'$, if there is a local displacement between u and v of H. G' can be constructed in time polynomial in the size of H. In fact, by finding the shortest path in G' from the start state to the target state, we can determine the motion with the least number of local displacements in polynomial time for any fixed-size graph H. In fact, when the graph H is not fixed, not only is finding the shortest motion difficult, but even the problem of simply checking to see if there exists a motion between two valid placements is NP-hard, as we show at the end of this section.

A. Moving a Trilobite Quickly

The previous paragraph shows that we can move any arthropod from any start location to any target location in the least possible number of moves. Clearly, the complexity of the algorithm is rather high. In this subsection, we show how to move a trilobite from any start location to any target location (in a biconnected chordal graph) in a way that uses at most *one more* than the number of moves of an optimal solution. This algorithm has an $O(m)$ running time.

Theorem 3.1: In a biconnected chordal graph G, we can find a valid motion in $O(m)$ time, for which the number of steps is guaranteed to be at most one more than the optimal number of steps.

Proof: Let S denote the subgraph corresponding to the initial location of the trilobite and T denote the subgraph corresponding to the target location of the trilobite. If S and T share a common edge, then one step is enough to move the trilobite from S to T.

The key idea is to determine the shortest cycle passing through any two edges of S and T. This can be done in $O(m)$ time by using the algorithm by Suurballe and Tarjan[2] [21]. (The algorithm given in their paper is for the case of weighted graphs and has a running time of $O(m \log n)$. It is easy to modify it to work in $O(m)$ time when we are dealing with unweighted graphs.) We determine a motion for the trilobite by using only vertices on this cycle.

Lemma 3.2) Skip Lemma: If we have two placements of a trilobite, such that some pair of edges of these locations belong to a cycle of length three, then we can move the trilobite between these locations in a single move.

Proof: If the two locations do not share any vertices then the shortest cycle through some pair of edges has length at least four. Hence, the two locations must have a common vertex. In Fig. 9, we show two locations of a trilobite. The trilobite initially occupies vertices (1, 2, 3) and would like to move to the location (2, 4, 5). The arrows indicate how it can move in one step from the initial to the target location. □

If the cycle has length exactly three, then we can move the trilobite in exactly one step by using the Skip Lemma. If the cycle has length greater than three, we can move the trilobite so that the length of the cycle decreases by two at each step. We argue this as follows: Let the three vertices of the trilobite

[2]This paper deals with the problem of finding the shortest pair of disjoint paths between two specified vertices. We can reduce our problem to it by creating two artificial vertices, each connected to the vertices occupied by the trilobite.

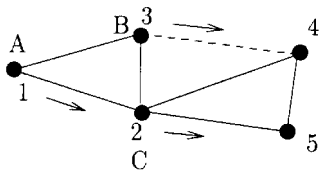

Fig. 9. Proof of Skip Lemma.

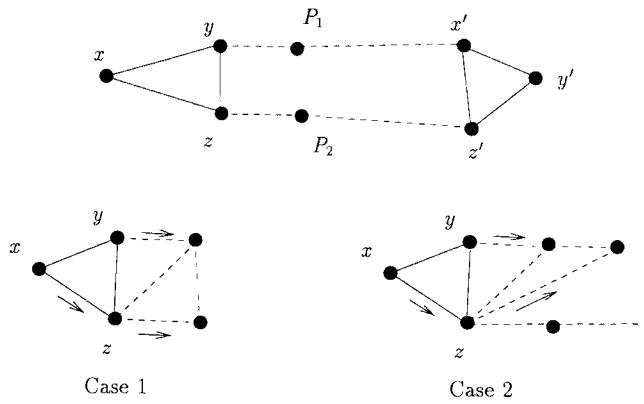

Fig. 10. Using the shortest cycle to move a trilobite.

be A, B, C. Let the vertices in G that the trilobite is placed on be x, y, z, respectively. Let the target vertices be x', y', z'. Let the shortest cycle pass through edges (y, z) and (x', z'). Let P_1 be the path (portion of the cycle) from y to x' and P_2 be the path (portion of the cycle) from z to z', such that they are disjoint (see Fig. 10). Since the graph is biconnected and chordal, and P_1 and P_2 are minimal paths, we must have edges from one of y, z to vertices on the two paths. Assume that there is an edge from z to the neighbor of y on P_1. There are two cases as shown in Fig. 10, and in each case, by applying the Skip Lemma, we can move the trilobite so that the cycle length decreases by two. The number of moves we make is, thus, $\lfloor c/2 \rfloor$, where c is the length of the cycle.

It is easy to show, by induction on the number of moves, that if there is a solution with p moves, then there must be a cycle of length at most $2p+2$ that passes through an edge of S and an edge of T. (Decompose the sequence of moves into the first move, followed by a sequence of $p-1$ moves. By the induction hypothesis, we can find a cycle of length $2p$; extending the cycle uses only two extra edges.) Since we have found the shortest cycle, $c \leq 2p+2$ and $\lfloor c/2 \rfloor \leq p+1$. Hence, the total number of moves is at most $p+1$, where p is the number of moves in an optimal solution.

Remark: This bound is tight for our algorithm, since it does not involve any moves in which all three vertices move simultaneously to three unoccupied vertices. Perhaps by using such moves we could obtain a solution involving the fewest moves in linear time.

B. NP-Hardness

Clique (G, k) is the problem of checking if the graph G contains a clique of size k. This problem is known to be NP-complete [10].

Theorem 3.3: Checking if there is a motion from a start state to a goal state is NP-hard.

Proof: We can reduce the clique problem to checking if there exists a motion that moves $H = K_{2k}$ from a start location to a target location. Construct a new graph $G' = (V', E')$, as follows:

$$
\begin{aligned}
V' &= V \cup (x_1, \cdots, x_{2k}) \cup (y_{2k+1}, \cdots, y_{4k}).\\
E' &= E \cup E_x \cup E_y \cup E''.\\
E_x &= [(x_i, x_j) | 1 \leq i \leq j \leq 2k].\\
E_y &= [(y_i, y_j) | 2k+1 \leq i \leq j \leq 4k].\\
E'' &= [(v, x_i), (v, y_j) | v \in V, 1 \leq i \leq 2k,\\
&\qquad 2k+1 \leq j \leq 4k].
\end{aligned}
$$

In other words, we add two cliques to G having $2k$ vertices each, and we add edges from each vertex in each clique to all the vertices in G.

In the start state, H occupies one clique completely, and in the target state, it occupies the other clique completely. If G contains a clique on k vertices, then we have two cliques in G' of $3k$ vertices each that share k vertices of G in common. We can move k vertices of H from one copy of K_{2k} to the k vertices in G forming the clique. In the next step, these k vertices move to the second copy of K_{2k}, and the remaining k vertices in K_{2k} occupy the k vertices in G forming the clique. One more step completes the motion. Thus, if G contains a clique of size k, the desired motion exists.

To prove the converse, notice that H cannot simultaneously occupy vertices from the two cliques of $2k$ vertices each (since there are no edges between them). Thus if G does not have a clique of size k, there is no motion from the start state to the target state.

IV. UGV Example

In this section, we expand the UGV example mentioned in Section I. Consider three UGV's, each having different special abilities (communication, supplies, coverage) and trying to move from a set of start positions to a set of goal positions. The UGV's try to maintain a trilobite formation, so that each pair of them always remain neighbors. The terrain map is shown in Fig. 11. The graph representation of the environment is shown in Fig. 12. We are trying to achieve a representation similar in spirit to the one described in [7], [15]. We connect nodes representing neighboring places that have a path between them that can be traversed easily (planar, or one that crosses a low saddle point that requires only one contour line of ascent).

Since the resulting graph is biconnected and chordal, we can use Theorem 3.1 to move our trilobite quickly. First, we determine the shortest cycle passing through any two edges of the trilobite in the start and goal locations. These are edges $(2, 3)$ in S and $(8, 10)$ in T. The cycle, which has length eight, is shown in Fig. 12. The planned motion will be on this cycle. Each time the trilobite moves by one step, the length of the cycle decreases by two. After making three moves, the current position and the target position share a common edge and only one more move is required. Thus, four moves were

Fig. 11. On the left is a portion of a Montezuma, NY, map. Part of the map, enlarged, is shown on the right with marked locations. The arrows indicate the first movements of the formation (see explanation below).

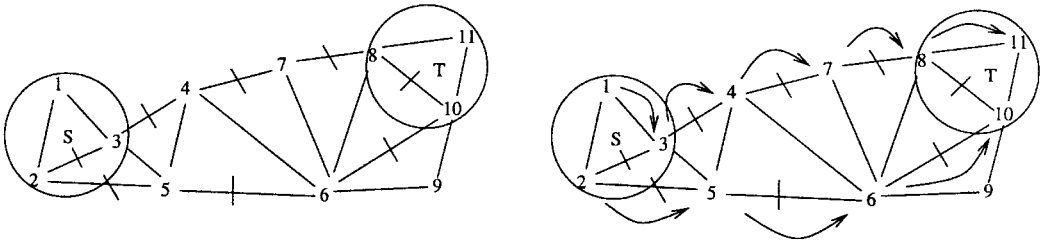

Fig. 12. On the right is the graph representation of the map. The start position is marked with an S, the target with a T. The shortest cycle is indicated by marks on the edges. The path is shown on the right.

sufficient to move our formation from S to T: $(1, 2, 3) \rightarrow (3, 5, 4) \rightarrow (4, 6, 7) \rightarrow (6, 10, 8) \rightarrow (8, 10, 11)$.

V. Discussion and Future Work

In this paper, we have defined a natural version of the motion-planning problem for a group of robots in a graph-theoretic setting. We have established conditions under which a team of robots having a particular graph structure can move from any start location to any goal destination in a graph-structured space. We have shown that, for a group of robots having a fixed formation, we can find the shortest motions in polynomial time, and we have given algorithms for cases in which the environment can be modeled (discretely) as special kinds of graphs. Efficient computation of shortest motions (or approximate shortest motions) remains a very interesting open problem. The main difficulty is in charging for the cost of rotations for movement of k-legged spiders.

Clearly, variations of our definition for motion could also be considered. For example, our definition allows the "exchange" of two vertices in a single step, but none of our results make use of such a motion, so that, in essence, we have disallowed it. (We did not do so explicitly to avoid cluttering our simple definitions.) In fact, in much of this paper, we have used a definition of movement in which we do not distinguish between automorphic images of H. If we strictly enforce our original definition, an automorphism of H is not necessarily a local displacement, since corresponding nodes may not be neighbors; thus, by this definition, a subpath of length ≥ 3 cannot move freely on a path, since it cannot reverse itself.

We could also consider stricter definitions of movement—for example, "rigid" movement, in which the distances in G between corresponding nodes of H remain the same. Specialized types of movements would also be of interest—for example, "translation," in which every node of H is required to move to a neighboring node.

It would also be of interest to study the free-movement problem in other special types of graphs. For example, it is clear that any H can move freely in G if G is a clique, a path, or a cycle. On the other hand, it is easy to see that only a path can move freely on a tree.

An interesting open problem is to characterize the class of graphs that a convoy of robots (or a *snake*) can move freely in. A snake is modeled by a four-vertex graph linked by three edges. We will refer to the four vertices as a, b, c, and d. Assume that the edges are (a, b), (b, c), and (c, d). To go from position (a, b, c, d) to another position, say (w, x, y, z), if there is a path from (a, b) to (w, z), the snake can just "creep" along this path. The problem becomes more difficult when the snake is not oriented in a position that allows it to start crawling from an endpoint. We have identified a list of

conditions that allow the snake to reorient itself, but this list does not appear to be exhaustive.

This paper has formulated a graph-theoretic framework for studying robot mobility. It would be of interest to develop graph-theoretic formulations of other types of robotic activities, such as sensing and manipulation. However, these activities do not have straightforward graph-theoretic counterparts, since a graph has no concept of direction (though it does have concepts of adjacency and distance). Thus, it is not clear how to define vision-like sensory processes in a general graph, since vision would seem to require some sort of concept of a "line of sight," which presumably would involve distinguishing between nodes that lie in given directions from a given node. Similarly, it is not clear how to define the concept of one subgraph (a "gripper") "grasping" an "object" subgraph; we can require that certain nodes of the gripper be adjacent to certain nodes of the object, but there is no obvious way of distinguishing stable from unstable grasps. In future papers, we plan to consider methods of augmenting a graph in order to allow direction-dependent concepts to be defined.

ACKNOWLEDGMENT

The authors would like to thank Y. Matias, S. Naor, B. Raghavachari, and H. Siegelmann for useful discussions.

REFERENCES

[1] J. Bares and W. L. Whitaker, "Configuration of an autonomous robot for Mars exploration," in *Proc. World Conf. Robot. Res.*, 1989, pp. 37–52.
[2] B. Bollobás, *Extremal Graph Theory.* London, U.K.: Academic, 1978.
[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications.* Amsterdam, The Netherlands: North Holland, 1977.
[4] Q. Chen and J. Luh, "Coordination and control of a group of small mobile robots," in *Proc. Int. Conf. Robot. Auto.*, 1994.
[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: MIT Press, 1989.
[6] B. Donald, J. Jennings, and D. Rus, "Analyzing teams of cooperating mobile robots," in *Proc. Int. Conf. Robot. Auto.* 1994.
[7] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Using multiple markers in graph exploration," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 859–865, 1991.
[8] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.
[9] D. Fulkerson and O. Gross, "Incidence matrices and interval graphs," *Pacific J. Math*, vol. 15, pp. 835–855, 1965.
[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.
[11] S. Hert and V. Lumelsky, "The ties that bind: Motion planning for multiple tethered robots," in *Proc. Int. Conf. Robot. Auto.*, 1994.
[12] S. Khuller, E. Rivlin, and A. Rosenfeld, "Graphbots: Mobility in discrete spaces," in *Proc. Int. Colloq. Automata, Languages, Programming*, 1995.
[13] B. Kuipers and T. Levitt, "Navigation and mapping in large-scale space," *AI Mag.*, vol. 9, pp. 61–74, 1988.
[14] R. Kurazume and S. Nagata, "Cooperative positioning with multiple robots," in *Proc. Int. Conf. Robot. Auto.*, 1994.
[15] T. Levitt and D. Lawton, "Qualitative navigation for mobile robots," *Artif. Intell.*, vol. 44, pp. 305–360, 1990.
[16] E. G. Mettala, "The OSD tactical unmanned ground vehicle program," in *Proc. DARPA Image Understanding Workshop*, 1992, pp. 159–171.
[17] P. J. M. McKerrow, *Introduction to Robotics.* Reading, MA: Addison-Wesley, 1993.
[18] C. H. Papadimitriou, P. Raghavan, M. Sudan, and H. Tamaki, "Motion planning on a graph," in *Proc. Foundations Comput. Sci. Conf.*, 1994.
[19] L. Parker, "Designing control laws for cooperative agent teams," in *Proc. Int. Conf. Robot. Auto.*, 1993.
[20] D. Parsons and J. Canny, "A motion planner for multiple mobile robots," in *Proc. Int. Conf. Robot. Auto.*, 1990.
[21] J. Suurballe and R. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, pp. 325–336, 1984.
[22] Y. A. Teng, D. DeMenthon, and L. S. Davis, "Stealth terrain navigation," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 96–109, 1993.

Samir Khuller received the the B.Tech degree from the Indian Institute of Technology, Kanpur, India, in 1986, and the M.S. and Ph.D. degrees from Cornell University, Ithaca, NY, in 1989 and 1990, respectively.

He is currently an Associate Professor in the Computer Science Department at the University of Maryland, College Park. His research interests are in graph algorithms, discrete optimization, and computational geometry. He has published approximately 30 journal papers on these topics.

Dr. Khuller received a National Science Foundation Career Development Award in 1995, and in 1996, he received a Dean's Teaching Excellence Award.

Ehud Rivlin received the B.Sc. and M.Sc. degrees in computer science and the M.B.A. degree from the Hebrew University, Jerusalem, Israel, and the Ph.D. degree in computer science from the University of Maryland, College Park.

Currently, he is a Senior Lecturer in the Computer Science Department at the Technion—Israel Institute of Technology, Haifa. His current research interests are in machine vision and robot navigation.

Azriel Rosenfeld (M'60–F'72–LF'96) received the Ph.D. degree in mathematics from Columbia University, New York, NY, in 1957, rabbinic ordination (1952) and Dr. Technol. degrees from Linköping University, Linköping, Sweden, in 1980 and Oulu University, Oulu, Finland, in 1994, respectively.

He is a tenured Research Professor, a Distinguished University Professor (since 1995), and Director of the Center for Automation Research at the University of Maryland, College Park. He also holds affiliate professorships in the Departments of Computer Science and Psychology and in the College of Engineering. He is widely regarded as the leading researcher in the world in the field of computer image analysis. Over a period of 35 years, he has made many fundamental and pioneering contributions to nearly every area of that field. He wrote the first textbook in the field (1969), was founding Editor of its first journal (1972), and was Co-Chairman of its first international conference (1987). He has published more than 25 books and 500 book chapters and journal articles and has directed more than 50 Ph.D. dissertations.

Dr. Rosenfeld won the IEEE's Emanuel Piore Award in 1985; he is a founding Fellow of the American Association for Artificial Intelligence (1990) and of the Association for Computing Machinery (1993); he is a Fellow of the Washington Academy of Sciences (1988) and won its Mathematics and Computer Science Award in 1988; he was a founding Director of the Machine Vision Association of the Society of Manufacturing Engineers (1995 to 1988), won its President's Award in 1987 and is a certified Manufacturing Engineer (1988); he was a founding member of the IEEE Computer Society's Technical Committee on Pattern Analysis and Machine Intelligence (1965), served as its Chairman (1985 to 1987), and received the Society's Meritorious Service Award in 1986, its Harry Goode Memorial Award in 1995, and became a Golden Core member of the Society in 1996; he received the IEEE Systems, Man, and Cybernetics Society's Norbert Wiener Award in 1995; he received an IEEE Standards Medallion in 1990, and the Electronic Imaging International Imager of the Year Award in 1991; he was a founding member of the Governing Board of the International Association for Pattern Recognition (1978 to 1985), served as its President (1980 to 1982), won its first K.S. Fu Award in 1988, and became one of its founding Fellows in 1994; he was a Foreign Member of the Academy of Science of the German Democratic Republic (1988 to 1992); and he is a Corresponding Member of the National Academy of Engineering of Mexico (1982).