Graphbots: Mobility in Discrete Spaces

Samir Khuller *
Computer Science Department and UMIACS
University of Maryland, College Park, MD 20742.

Ehud Rivlin †
Computer Science Department, Technion, Haifa 32000, Israel
and Center for Automation Research
University of Maryland, College Park, MD 20742.

Azriel Rosenfeld ‡
Center for Automation Research
University of Maryland, College Park, MD 20742.

Abstract

Most previous theoretical work on motion planning has addressed the problem of path planning for geometrically simple robots in geometrically simple regions of Euclidean space (e.g., a planar region containing polygonal obstacles). In this paper we define a natural version of the motion planning problem in a graph theoretic setting. We establish conditions under which a "robot" or team of robots having a particular graph structure can move from any start location to any goal destination in a graph-structured space.

1 Introduction

Most previous theoretical studies of motion planning deal with "robots" that have simple geometries (e.g., polygonal) and that operate in simple spaces (e.g., in a planar region containing polygonal obstacles) [10]. This paper suggests that it may also be appropriate to study "robots" (or teams of robots) that operate in *discrete* spaces, i.e. in graphs, and that have discrete "geometries" (represented by subgraphs).

---

*Research supported by NSF Research Initiation Award CCR-9307462. E-mail : `samir@cs.umd.edu`.

†E-mail : `ehudr@cs.technion.ac.il`.

‡Research supported by ARPA under Contract DACA76-92-C-0009 (ARPA Order 8459). Email : `ar@cfar.umd.edu`.

1.1 Mobility in a Graph

We begin by defining our model for mobility of a graph-structured robot (or team 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 "robot" (though, as we shall see below, it might also represent, e.g., 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 $S = H_0, H_1, \ldots, H_k = T$, 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 sequence of k local displacement steps. Note that a location (or "placement") of H in G is a subgraph of G which is isomorphic to H; two isomorphisms of H into G which have the same image are regarded as defining the same placement.

In this model, the graph structure of the robot or team of robots might represent physical (but not necessarily rigid) connections between the robot's parts, or communication links between the team members. It is reasonable to require that this structure must be preserved when the robot (or team) moves. Moreover, the movement should be as "continuous" as possible; "jumps" from one position to another arbitrarily distant position should not be allowed. Our definition of movement is a very good discretized description of the way a real-world team of robots might move cooperatively (e.g., [6]); we represent the members of the team by single nodes, and each time they move, we require them to reestablish their lines of communication, which define the graph structure of the team. (Note that a team of robots might be operating in a complex environment such as a mine (a network of caves and tunnels), so that the space in which the team moves might have an arbitrary topology, represented by an arbitrary graph.) Our model can also be applied to the movement of a jointed robot (with telescoping limbs that meet at universal joints) that moves on a discrete space, where neighboring joints always occupy neighboring nodes in the space.

1.2 Outline of Results

A given graph H cannot always move *freely* in a given graph space G; H can only occupy positions in which G is "big" enough to contain an isomorphic copy of H. If H has only one or two nodes, it can obviously move freely in any connected 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, 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 non-trivial. The results characterize various classes of graphs that permit free movement of various forms of H's, which we will name after various types of arthropods.[1] In Section 2.1 we give necessary and sufficient conditions on G for the movement of a two-vertex path, which we call a *tick*. In Section 2.2 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 2.3 we give sufficient conditions for a three-vertex cycle (a *jellyfish*) to move freely. These conditions do not generalize straightforwardly to paths or cycles (or cliques) that have more than three nodes; but in Section 2.4 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. (The study of this class of graphs is motivated by CMU's Ambler [1]. In fact, our spider can be regarded as an abstract version of a physical legged robot that has freely pivoting leg sockets and telescoping legs and that walks on stepping stones.) These proofs yield efficient algorithms for moving these types of arthropods from one location to another.

In Section 3 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 faster algorithms. We also prove that the problem is NP-complete when H is part of the input. This reduction is also outlined in Section 3.

1.3 Related Work

Recently we learned of related work by Papadimitriou, Raghavan, Sudan and Tamaki [9]. The spirit of 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 "fleet" 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 robot that has a "shape" 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.) Clearly, the two models could be combined; one could study the problem of planning a cooperative motion for *many* robots, each of which has a certain shape.

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

2 Moving Robots Freely

2.1 Moving a Tick

A "tick" is modelled by a two vertex graph linked by a single edge. We leave the proof of the following proposition as an exercise for the reader.

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.)

2.2 Moving a Scorpion

A "scorpion" is modelled by a three-vertex graph linked by two edges. We will refer to the degree-2 vertex as b (the "body") and the degree-1 vertices as the f vertices (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 1 (the case when the degree of u_3 is not 1 is identical), u_1 has a neighbor u_0 (other than u_2). Move the scorpion as follows. Vertex f goes from u_1 to u_0, b goes to u_1 and 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! □

2.3 Moving a Jellyfish

A "jellyfish" is modelled by a three-vertex graph linked by three edges (i.e., a clique of size 3). We will give some conditions under which free motion of a jellyfish 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 graph connected. We also require that G contain at least $\lambda + 1$ vertices.

A Chordal Graph is a graph in which each cycle of length at least 4 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 $\{1, \dots, n\}$, 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 [5]) *A graph G has a peo if and only if G is chordal.*

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

Theorem 2.4 *A jellyfish 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 jellyfish 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; it must be the case that any vertex i ($i = 1, \dots, n-2$) has 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: 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, 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.

□

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 3 and the theorem is trivially true for this graph. Consider a `peo` of G. Assume that we have deleted vertices $1, \ldots, i$ 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 the last vertex to be removed in the `peo` from the component that is exhausted first 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 jellyfish can move freely on G_i. What happens if we add vertex i back? We have to show that for each location of a jellyfish that uses i, it can move to any location in G_i. In a placement of the jellyfish using vertex i, let the other corners of the jellyfish be at vertices x and y. W. l. o. g 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 jellyfish from i, x, y to x, z, y. Since x, y and z are all in G_i, by the induction hypothesis the jellyfish 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 jellyfish 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 jellyfish cannot be separated by a cut vertex. However, we can construct graphs that are biconnected but not chordal, which allow free movement of the jellyfish. Hence chordality is sufficient, but *not* necessary.

2.4 Moving a Spider

A spider is modelled by a $(k+1)$-vertex graph having a central vertex denoting its "body" (vertex b) linked by edges to vertices representing its "feet". We give conditions under which free motion of a spider is possible. We will refer to such a spider as a k-legged spider with feet $f_1, \ldots, f_k$. When $k = 1$, we have a 1-legged spider, which is an inchworm; this case has already been dealt with. Subsection 2.2 gave an exact characterization of the graphs a 2-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 ($i = 1, \dots, k-2$) 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 the path P_i has three or more edges, then together with the edge (x,y) it creates a chordless cycle of length 4 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 $C_1, \dots, C_\ell$ 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 4 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 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 $f_1, \dots, f_k$ at vertices $u_1, \dots, u_k$. 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 one wants 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 $P = [s = v_0, v_1, \ldots, v_\ell = t]$ 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 spider now moves the foot at v_{i+1} to v_{i+2}, and the foot at v_{i-1} to v_i. The body moves from v_i to v_{i+1}. 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 is not in $N(v_0, v_1) \cup v_1$. When $i = \ell$, we define $v_{\ell+1}$ as any vertex in $N(v_\ell)$ that is not in $N(v_\ell, v_{\ell-1}) \cup v_{\ell-1}$. □

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 BFS. 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 $P_1, \ldots, P_{k-1}$ between s and t. (These can be obtained by finding a flow of value k in an appropriately defined flow network [4]). Next, preprocess these paths to make them minimal, so they have no chords that provide shortcuts. This is easily accomplished by numbering 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.

Define $x_i (1 \le i \le k)$ to be the set of vertices 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 $s \cup_{i=1}^{k} x_i$ 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.

2.5 Moving a Four-Legged Spider

We know 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.Clearly, there are two feasible locations for the five-legged spider at the two vertices of degree-5; but since these are the only two degree 5 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-3 vertices on this path.)

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

We omit the details of this proof.

3 Shortest Motion

Assume that G has n vertices and H has ℓ vertices. There are at most $O(n^\ell)$ possible locations of H in G. 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 $u \in V'$ and $v \in V'$, if there is a local displacement between u and v of H. Clearly, 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 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-complete, as we show next.

3.1 Moving a Jellyfish 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 jellyfish from any start location to any target location (in a biconnected chordal graph) in a way that uses at most *twice* 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 a valid motion in $O(m)$ time for which the number of steps is guaranteed to be at most twice the optimal number of steps.*

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

The key idea is to determine the shortest cycle passing through any two edges of S and T. We determine a motion for the jellyfish using only the vertices on this cycle. Each time we move the jellyfish by one step, the length of the cycle decreases by one. The number of moves we make is $c - 2$ where c is the length of the cycle. After $c - 2$ moves, the current position and the target position share a common edge and only one more move is required. This means that $c - 1$ moves are sufficient.

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 only uses two extra edges.) Since we have found the shortest cycle, $c \leq 2p + 2$ and $c - 1 \leq 2p + 1$. Hence the total number of moves is at most $2p + 1$, 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. Perhaps by using such moves we could obtain a solution involving the fewest moves in linear time.

3.2 NP-completeness

`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 [7].

We can reduce the clique problem to checking to see 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:

$$V' = V \cup \{x_1, \ldots, x_{2k}\} \cup \{y_{2k+1}, \ldots, y_{4k}\}.$$

$$E' = E \cup E_x \cup E_y \cup E''.$$

$$E_x = \{(x_i, x_j) | 1 \leq i \leq j \leq 2k\}.$$

$$E_y = \{(y_i, y_j) | 2k + 1 \leq i \leq j \leq 4k\}.$$

$$E'' = \{(v, x_i), (v, y_j) | v \in V, 1 \leq i \leq 2k, 2k + 1 \leq j \leq 4k\}.$$

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.

4 Discussion and Future Work

1. In this paper we have proposed a model for motion planning in graphs. Clearly, variations of our definition 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.

2. 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.

3. It would also be of interest to study the free movement problem in 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.

4. It might be of interest to study arthropods that undergo metamorphosis!.

5. An interesting open problem is to characterize the class of graphs that a *snake* can move freely in. A snake is modelled 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 either $\{a, b\}$ 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 re-orient itself, but this list does not appear to be exhaustive. A simple characterization does not appear to be straightforward.

6. 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.

7. 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.

Acknowledgments:

We would like to thank Yossi Matias, Seffi Naor, Balaji Raghavachari and Hava Siegelmann for useful discussions.

References

[1] J. Bares and W. L. Whitaker, Configuration of an Autonomous Robot for Mars Exploration, In *Proc. of the World Conf. on Robotics Research*, vol. 1, pp. 37–52 (1989).

[2] B. Bollobás, *Extremal Graph Theory,* Academic Press, London (1978).

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications,* North Holland, Amsterdam (1977).

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, MA (1989).

[5] D. Fulkerson and O. Gross, Incidence Matrices and Interval Graphs, *Pacific Journal of Math.*, vol. 15, pp. 835–855 (1965).

[6] E. G. Mettala, The OSD Tactical Unmanned Ground Vehicle Program, In *Proc. of the DARPA Image Understanding Workshop*, pp. 159–171 (1992).

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, San Francisco, CA (1979).

[8] P. J. M. McKerrow, *Introduction to Robotics,* Addison-Wesley, Reading, MA (1993).

[9] C. H. Papadimitriou, P. Raghavan, M. Sudan and H. Tamaki, Motion Planning on a Graph, To appear in *Proc. of the* 35^{th} *Foundations of Computer Science Conference* (1994) .

[10] J. T. Schwartz, M. Sharir and J. H. Hopcroft, *Planning, Geometry and Complexity of Robot Motion,* Ablex Publishing Corp., Norwood, NJ (1987).