Motion Planning on a Graph (Extended Abstract)

Christos H. Papadimitriou * Prabhakar Raghavan † Madhu Sudan †
Hisao Tamaki †

Abstract

We are given a connected, undirected graph G on n vertices. There is a mobile *robot* on one of the vertices; this vertex is labeled s. Each of several other vertices contains a single movable *obstacle*. The robot and the obstacles may only reside at vertices, although they may be moved across edges. A vertex may never contain more than one object (robot/obstacle). In one *step*, we may move either the robot or one of the obstacles from its current position v to a vacant vertex adjacent to v. Our goal is to move the robot to a designated vertex t using the smallest number of steps possible.

The problem is a simple abstraction of a robot motion planning problem, with the geometry replaced by the adjacencies in the graph. We point out its connections to robot motion planning. We study its complexity, giving exact and approximate algorithms for several cases.

1. Overview

We are given a connected, undirected graph G on n vertices. There is a mobile *robot* on one of the vertices; this vertex is labeled s. Each of several other vertices contains a single movable *obstacle*. The robot and the obstacles may only reside at vertices, although they may be moved across edges. No vertex may ever contain more than one movable entity (robot or obstacles). In one *step*, we may move either the robot or one of the obstacles from its current position v to a vacant vertex adjacent to v. Our goal is to move the robot to a designated vertex t using the smallest number of steps possible. Let us call this *graph motion planning with one robot*, or GMP1R for short.

There are two motivations for studying GMP1R (and related problems we will mention in Section 4):

1. GMP1R strips away the geometric considerations from the following geometric motion planning problem: move an object in a geometric scene from one position to another, moving obstacles out of the way if necessary. Motion planning problems for robots in geometric environments have relatively high complexities (most practical motion planning problems are *PSPACE*-hard, or worse). In fact, a problem closely related to the geometric version of GMP1R is *PSPACE*-hard (see Section 4 for details). From a complexity-theoretic viewpoint, GMP1R enables us to study how much of this complexity stems from geometric considerations, and how much from purely combinatorial ones.

2. The algorithms we devise for GMP1R can be applied to practical motion planning problems in relatively uniform settings, in which geometry does not play a substantial role. Candidates arise in environments with regular geometries such as buildings or factory floors in which the movable entities are identical (say, carts of the same size moving in corridors that can accom-

*Department of Computer Science and Engineering, UCSD, La Jolla, CA 92093.

†IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

modate one cart at a time, or vehicles moving on a network of tracks). Packet routing using the *deflection* or *hot-potato* model [3] provides a related example (see Section 4). In some of these cases, a better cost metric may account for the physical lengths of edges. In others, intersections in the building/railroad may be able to hold more than one obstacle/robot at a time. In Section 4 we outline extensions of some of our results to these variants.

Our formulation of GMP1R seeks to minimize the number of steps to move the robot from s to t. In fact, the characterization and algorithms we give will implicitly solve the decision version — which asks whether at all the robot can be moved from s to t — in polynomial time. A number of related geometric motion planning problems are given in [7]. It is possible to formulate graph-theoretic analogs of all of these problems, and this is discussed in Section 4. We note that Frederickson (see [1] and references therein) has studied a different planning problem that he calls motion planning on a tree; our problem and methods are completely different from his.

An obvious generalization of GMP1R is GMPkR, where we have k robots with respective destinations. A special case of GMPkR, where there are no additinal obstacles (thus all the movable objects have their destinations), is previously studied. Wilson [8] studies the case $k = n - 1$, which is the "15-puzzle" played on a general graph, and gives an efficiently checkable characterization of the solvable instances of the problem. Kornhauser, Miller, and Spirakis [5] extend his result to any $k \leq n - 1$ and also give an upper bound of $O(n^3)$ on the number of steps to solve any solvable instance. They give an example for which this bound is optimal. Goldreich [2] examines the issue of computing the shortest move sequence for the problem studied by Kornhauser et al. and shows that determining the shortest move sequence is NP-hard.

Figure 1 depicts an instance of GMP1R on a tree that invalidates a number of plausible characterizations of the optimal plan (and thereby a number of simple algorithms). Here the *only* feasible plan for moving the robot from s to t is, essentially: (1) move the robot to a; (2) move the obstacles at b and c to x and y; (3) move the obstacles on vertices e through t to the right, so that the occupy vertices g through i; (4) move the robot to the *sidestep vertex* d; (5) move the obstacles currently on g and t back towards the source past f, clearing the way for the robot to move from d to t.

The example shows that (1) the robot may temporarily have to move away from the source, both initially from s and later on to a sidestep vertex; (2) the motion of some obstacles, too, may be non-monotone — here some obstacles first move to the right along the s–t path, and then again to the left.

1.1. Our results

Theorem 1: *Given an instance of GMP1R and a positive integer k, it is NP-complete to decide whether a solution of length k exists. The problem remains NP-complete when restricted to a planar graph.*

The proof of *NP*-hardness is by a reduction (Figure 2) from 3-SAT, and immediately yields a hardness of approximation result — the solution length cannot be approximated to an arbitrarily small constant. Details of this reduction (and the extension to planar graphs) are given in the full paper. It is interesting to note that the graph used in this reduction belongs to a class for which our algorithm in Section 3 gives a constant-factor approximation. Membership in *NP* will follow from the characterization of the feasibility in Section 1.2 that provides a polynomial length solution for every feasible instance.

After some preliminaries in the following subsection, we study the problem on a tree in Section 2. Based on a canonical form lemma established in Section 2.1, we give in Section 2.2 a polynomial time algorithm that computes an optimal plan whenever G is a tree. This algorithm has a rather large running time, so in Section 2.3 we give a faster algorithm running in time that achieves a plan of length at most 7 times the optimal length. In Section 3 we give an

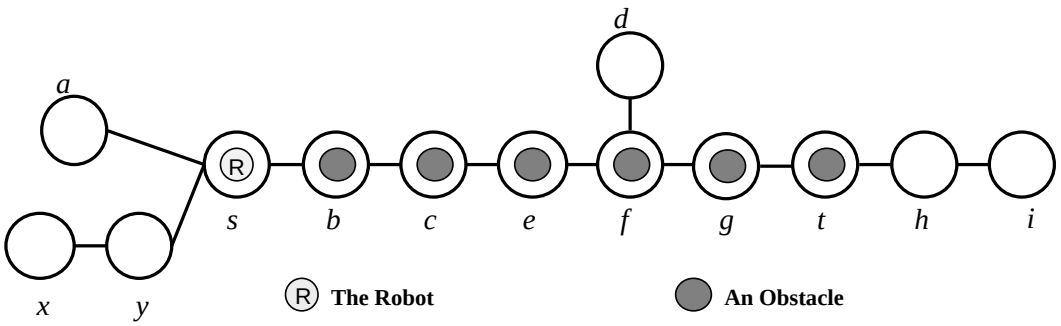

Figure 1: An instructive instance on a tree.

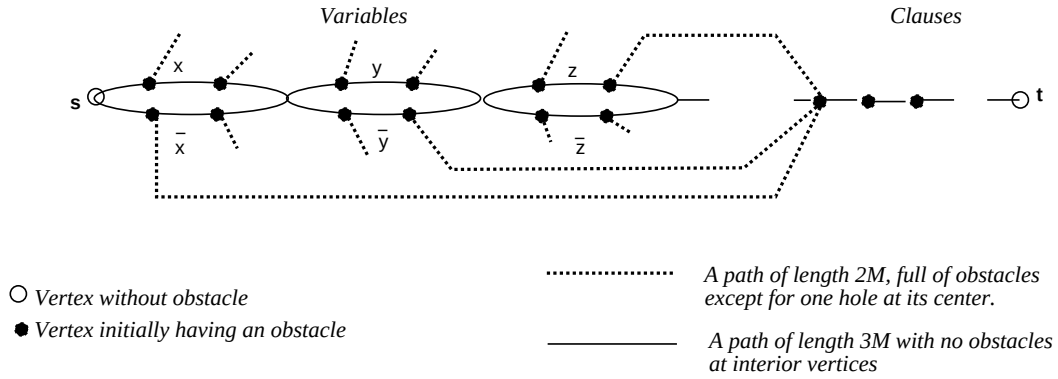

Figure 2: The NP-hardness reduction; the clause $x + y + \bar{z}$ is illustrated.

approximation algorithm for general graphs. The cost of the solution obtained by this algorithm is at most $O(\sqrt{n})$ times the optimal, and at the same time $O(l_{max}/l_{min})$ times the optimal, where l_{max} and l_{min} are the lengths of the longest and shortest paths of degree-2 vertices in G.

Some of our results may be extended to generalizations and variants of GMP1R; these are outlined in Section 4.

1.2. Preliminaries

A *hole* is a vertex that does not contain an obstacle. This should be literally taken so that the vertex with the robot is also a hole: a convention for technical convenience. When an obstacle is moved from v to the adjacent vertex w, we may think of it as a hole moving from w to v; we often use this notion in our descriptions. We call a path of G a *degree-2 chain* or simply a *chain* if all of its internal vertices are of degree 2 and neither of the endpoints is of degree 2. The *length* of a chain C, denoted by $l(C)$, is the number of the edges in the chain. A chain is *critical* if it does not belong to a cycle. We call a vertex of degree 3 or greater a *fork vertex*, and a vertex of degree 1 a *leaf*. Thus, an endpoint of a chain is either a fork vertex or a leaf.

We first address the feasibility question: given an instance, is it at all possible to bring the robot from s to t? When G is biconnected, it is sufficient (and clearly also necessary) to have 2 holes, because we can always move one of the holes wherever we want. Suppose that s and t belong to two distinct biconnected components and the length of the longest critical chain between these components is l. It is clear that $l + 3$ holes are necessary for the robot to cross this chain (unless s or t is an endpoint of this chain).

Having $l+3$ holes may not be sufficient if some of them are not available on the "right side" of the robot. However, feasibility can be determined by examining the robot's ability to reach a nearest fork vertex (in each of the k directions where k is the degree of s) because, once the robot is on a fork vertex with 2 more holes adjacent to it, all other holes can be moved freely by pushing the robot around this fork vertex. This gives us a simple necessary and sufficient condition, which shows that the feasibility problem is in P and the optimization problem is in NP.

In the subsequent analysis, we use the following alternative formulation of GMP1R. Suppose that a path P from u to v is filled with obstacles except for v and we move each of these obstacles one step towards v. We can view the net effect as moving the obstacle initially at u to v. We can also view it as a move of a hole from v to u. Note that this view extends to the more general case where the path P may contain vacant vertices other than v. Thus, in our new formulation, an obstacle is allowed to traverse a path in one move, provided the path does not contain the robot and its destination is initially vacant, paying the length of the path as the cost. The robot, however, is allowed to move only to an adjacent vertex in one step as before.

A *plan* for an initial configuration is a sequence of robot and obstacle moves starting from that configuration, where each obstacle move is specified by an oriented path P. Often we specify an obstacle move by a source-destination pair, in which case the path taken is assumed to be the shortest path. We call a plan *valid* if, when each move is executed, every robot move is to a vertex without an obstacle and the path of every obstacle move is clear of the robot. The *cost* of a plan is the total cost of the moves in the plan, where each robot move has a unit cost and the cost of an obstacle move is the length of the path of the move. We call a plan *complete* if it brings the robot from s to t. We call two plans *equivalent* if their initial and final configurations are respectively identical, disregarding the identity of the obstacles.

Consider the following subproblem. Let γ be a configuration and let U be an arbitrary set of vertices. What is the least cost plan to clear U entirely of obstacles? Let us first ignore the presence of the robot. Then, this subproblem has the following simple answer which we call the *matching principle*. Let $V_1 \subseteq U$ be the set of vertices in U with an obstacle in γ and let $V_2 \subseteq V(G) \setminus U$ be the set of vertices outside of U with a hole in γ. It is necessary that $|V_2| \geq |V_1|$ for our problem to have a solution. Consider a weighted bipartite graph on vertex sets (V_1, V_2) where there is an edge with cost l between $v_1 \in V_1$ and $v_2 \in V_2$ if and only if the shortest path between v_1 and v_2 in G has length l. Let μ denote the minimum cost matching from V_1 into V_2 in this bipartite graph. This matching μ can be viewed as a plan consisting of obstacle moves $v \rightarrow \mu(v)$, $v \in V_1$, which can be executed without interference with each other. In fact, it is not difficult to see that this is an optimal plan from γ to clear U. In the subsequent analysis of our problem with a robot, several variations of the matching principle are used to simplify given plans and to obtain optimal subplans.

2. Trees

In this section we study the case when G is a tree.

2.1. Canonical plans

In this subsection we define a canonical form for a complete plan, and show the existence of an optimal complete plan that is canonical. This will allow us to consider only canonical plans when we design algorithms in the later sections. The proofs of all the lemmas in this section can be found in the full paper.

In the example of Figure 1, we saw that the moves of the robot in an optimal plan may not be monotonic. It may back up from s, and may sidestep at several points on its way towards t. Our first goal is to establish that these are the only ways that the robot may deviate from a monotonic advance along the s–t path.

We call a sequence of robot moves *quasi-monotonic* along an oriented path P from u to v, if it starts from u towards v and, on arriving at each internal

vertex w of P, either (1) immediately proceeds to the next vertex on P, or (2) "sidesteps" to a vertex adjacent to w not on P, returns to w and proceeds to the next vertex on P. We call an internal vertex of P at which a sidestep occurs a *branch vertex* and a vertex to which the robot sidesteps a *sidestep vertex*. We call a valid complete plan S *quasi-monotonic* if the robot's walk in S is quasi-monotonic along the path from s to t. Let G_v denote the forest that results from removing a vertex v from the tree G. For a vertex $u \neq v$, the *u–side of v* is the tree of G_v that contains u. The u–side of an edge e is similarly defined. A vertex u is *behind* a vertex $v \neq u$ if u is not in the t–side of v. The following lemma asserts the existence of an optimal complete plan in which the robot, once having stepped into the t-side of s, behaves quasi-monotonically.

Lemma 2: *There exists an optimal complete plan which consists of two parts: (1) a* back up part *(which may be empty) in which the robot stays away from the t–side of s, followed by (2) a* forward part *which is a quasi-monotonic complete plan that brings the robot from s to t.*

As we saw in the example of Figure 1, the purpose of the back up part is to liberate some holes behind s, which would otherwise be unavailable until the robot reaches the first side-step vertex in its quasi-monotonic move towards t. For this purpose, it is always sufficient for the robot to monotonically back up to the closest fork vertex s' behind s, visit up to two vertices adjacent to s', which we call *back up vertices*, and return to s monotonically. We call a valid complete plan *quasi-bitonic* if it consists of a back up plan, which either is empty or takes the above form, followed by a quasi-monotonic forward plan.

Lemma 3: *There exists an optimal complete plan which is quasi-bitonic.*

We also want the obstacles in an optimal plan to behave nicely. Let S be a quasi-bitonic plan. We denote by B_S (T_S) the tree consisting of the set of vertices visited by the robot in the back up part (forward part, respectively) of S, including s. An obstacle move from a vertex v in T_S is *outward* if its destination is in the t–side of s and not in T_S; *forward* if its destination is in the t–side of v and in T_S; *backward* if its destination is either behind s or in the intersection of T_S and the s–side of v. We call a quasi-bitonic plan S *canonical* if it has the following properties.

(P1) No obstacle ever moves from outside of $B_S \cup T_S$ into $B_S \cup T_S$.

(P2) All obstacles that are in B_S move out of B_S without passing s, before the robot starts moving.

(P3) All outward and forward moves from T_S occur before the robot starts moving.

(P4) Each obstacle initially on T_S either moves once, outward or backward, or moves twice, first forward and then backward.

(P5) When an obstacle moves backward, it passes through at least one branch vertex. When the obstacle is from a vertex strictly between two branch vertices, the meaning of this statement is clear. We need clarifications for a few special cases. If the obstacle originates from a branch vertex v or a sidestep vertex adjacent to v, it passes another branch vertex. If the obstacle originates from a vertex on the path from s to the first sidestep vertex, then it passes through the fork vertex in B_S, i.e. the one the robot backed up to. Moreover, when an obstacle moves backward, the robot is on the sidestep vertex adjacent to the branch vertex it first passes through (or on a leaf of B_S in the above special case.)

Lemma 4: *There exists an optimal complete plan that is canonical.*

2.2. An exact algorithm

The algorithm for GMP1R below fully exploits the properties of canonical optimal plans. Hereafter, the only complete plans we will be concerned with are canonical plans.

Given a canonical plan S and an edge e from u to v lying on the path from s to t (all edges are thought of as directed towards the sink t), the subsequence of moves that only involve edges on the s-side of v (including e) is denoted $\mathsf{left}(S, e)$. (Here, think of

S as a plan in our original formulation of GMP1R in which each move of an obstacle is across a single edge.) This sequence $\mathsf{left}(S, e)$ is almost a valid plan except that it may pile up obstacles or holes on v. Similarly all moves on the t-side of e form an almost valid plan, which is denoted $\mathsf{right}(S, e)$.

Consider the flow of objects over e while executing a canonical plan. From the definition of a canonical plan, we can verify that this flow occurs in the following sequence. It starts with the *preflow* across e, i.e., obstacles which cross from u to v and is followed by some *backflow* across e, i.e., obstacles going from v to u. This is followed by the crossing of the robot over e. Lastly, there is a *postflow* of more obstacles going from v to u. In the following lemma we show that the amount of preflow, backflow and postflow completely characterize the decomposition of a plan into two pieces across an edge e.

We will say that a canonical plan S is (e, n_1, n_2, n_3)-*respecting* if the preflow across e is n_1, the backflow is n_2 and the postflow is n_3.

Lemma 5: *Let e be any edge on the path from s to t and left S_1 and S_2 be two (e, n_1, n_2, n_3)-respecting canonical plans. Then there exists an (e, n_1, n_2, n_3)-respecting canonical plan S such that $\mathsf{left}(S) = \mathsf{left}(S_1)$ and $\mathsf{right}(S) = \mathsf{right}(S_2)$.*

Proof [Sketch]: Decompose S_1 into partial plans $S_1^{(1)}, S_1^{(2)}, \ldots$ where $S_1^{(i)}$ is the sequence of moves which occur between the $(i-1)$st move across e and the ith move across e. Similarly decompose S_2. The plan S obtained by replacing $S_1^{(i)}$ with $\mathsf{left}(S_1^{(i)}), \mathsf{right}(S_2^{(i)})$, for all i is our target plan. An induction on i shows that the sequence of moves before the ith move across e is a valid sequence. It can also be verified that the resulting sequence is complete and canonical. □

The above lemma enables us to decompose the construction of the optimal plan into the constructions of the optimal plans to the left and the right of e, while making them (e, n_1, n_2, n_3)-respecting. Let $\mathrm{opt}(e, n_1, n_2, n_3)$ represent the cost of the left part $\mathsf{left}(S, e)$ of an optimal canonical (e, n_1, n_2, n_3)-respecting plan S. We focus on the task of computing $\mathrm{opt}(e, n_1, n_2, n_3)$.

Let e and e' be edges appearing on the s to t path in this order and suppose that the tail of e is a fork vertex. We define the cost of an "atomic" move as follows. Let us call a canonical plan S (e, e')–*atomic* if it causes the robot to sidestep at the tail of e, but not again until the robot crosses e'. The atomic cost $\text{atomic-opt}(e, n_1, n_2, n_3, e', n_1', n_2', n_3')$ is defined to the minimum over all (e, e')–atomic, (e, n_1, n_2, n_3)-respecting, and (e', n_1', n_2, n_3')-respecting plans S, of the quantity that is the cost of $\mathsf{left}(S, e')$ minus the cost of $\mathsf{left}(S, e)$.

The cost $\mathrm{opt}(e', n_1', n_2, n_3')$ can now be computed easily using the recurrence

$$\mathrm{opt}(e', n_1', n_2, n_3') = \min_{e, n_1, n_2, n_3} \{\mathrm{opt}(e, n_1, n_2, n_3) + \text{atomic-opt}(e, n_1, n_2, n_3, e', n_1', n_2, n_3')\},$$

where e ranges over all the e on the s–t path such that the tail of e is a fork vertex.

It is relatively straightforward to compute an optimal complete plan based on this recurrence for opt. A precise description of this is included in the full paper. We now concentrate on the harder task of computing $\text{atomic-opt}(e, n_1, n_2, n_3, e', n_1', n_2', n_3')$

Let v be the tail of e and v' the tail of e'. Let w be the vertex adjacent to v that is used as the sidestep point. Let P denote the path from w to the head of edge e'. We will show how to compute the $\text{atomic-opt}(e, n_1, n_2, n_3, e', n_1', n_2', n_3')$ given that w is the sidestep point. By minimizing over all possible ws we get $\text{atomic-opt}(e, n_1, n_2, n_3, e', n_1', n_2', n_3')$.

This computation turns out to be a minimum cost flow computation on an appropriately defined layered network. The network has three layers of nodes, the source set, the sink set and intermediate set. Arcs may go from the source layer to the sink layer directly, or may go from the source to the intermediate layer and from the intermediate layer to the sink layer. There is one source node in this network for each obstacle that ever uses P, and one sink node for each possible ultimate destination of these obstacles. Every arc of the network has unit capacity and the flow constraint is that (1) every source should have one unit of flow out of it and (2) at most one

unit of flow can enter each intermediate or sink node. There are $n_1 + n'_2 + n'_3$ sources corresponding to the obstacles that flow into P through v and v', in addition to the sources corresponding to the obstacles on P in the initial configuration. Each hole on the path from v to v' (excluding v and v') in the initial configuration, is a node on the intermediate layer. The sink layer has one node for each of the $n_2 + n_3$ obstacles that get pushed behind v, one for each of the n'_1 obstacles get pushed ahead of v', one for each hole on the subtrees hanging off the v–v' path and one for each vertex on P. The rules for arcs between these nodes can be inferred by analyzing the possible moves of obstacles in a canonical plan. The details are given in the full paper. The cost of an arc is the length of the path (in the tree G) between the obstacle and the corresponding (virtual) hole. Here, for example, if the source represents one of the preflow obstacles then the cost of the arc is determined as if the obstacle were at v. Similar rules are applied to compute the cost of the remaining arcs.

A minimum cost flow in the above network gives the cost of moving the obstacles so as to allow the robot to move from v to w to v'. Minimizing over all possible ws gives us atomic-opt$(e, n_1, n_2, n_3, e', n'_1, n'_2, n'_3)$.

We now analyze the complexity of the entire algorithm. The properties of a canonical plan imply that, if an (e, n_1, n_2, n_3)-respecting canonical plan uses the tail of e as a branch vertex, then $0 \leq n_2 \leq 2$ whenever $n_1 > 0$. Therefore, there are only $O(n^3)$ quadruples (e, n_1, n_2, n_3) among which we need to compute atomic-opt. Each of $O(n^6)$ atomic cost computations generates $O(1)$ flow problems when amortized over all possible paris (e, e'). Therefore, we need to solve $O(n^6)$ mincost flow problems. Once this is done, the optimal plan is obtained by solving a shortest path problem in a network in which these $O(n^3)$ quadruples are the nodes.

Theorem 6: *If there is a solution of finite cost for an instance of* GMP1R *on a tree, an optimal solution can be computed by solving $O(n^6)$ mincost flow problems on networks with $O(n)$ nodes each, and a shortest path problem on a graph with $O(n^3)$ nodes.*

2.3. A fast 7-approximation for trees

In this section, we describe an algorithm for GMP1R on a tree that gives a solution with cost at most 7 times the optimal. The idea is to simplify the problem by fixing the set of side-stepping vertices. Let T_0 be a subtree of G consisting of the path from s to t and, for each branch vertex v strictly between s and t, an arbitrary vertex adjacent to v but not on the s–t path. Recall that, for a canonical plan S, T_S (B_S) denotes the subtree consisting of the set of vertices visited by the robot in the forward part (back up part, respectively) of S.

Lemma 7: *There exists a complete canonical plan S with $T_S = T_0$ with cost at most 7 times the cost of the optimal complete plan.*

The proof is a simple simulation and is omitted. Our algorithm searches for an optimal canonical plan assuming that it side-steps according to T_0. This is done by solving a mincost flow problem very similar to the one we used in computing the atomic cost in the exact algorithm of Section 2.2. Since the side-step vertices are fixed, we can construct one flow graph (for each potential shape B of B_S) which corresponds to a complete canonical plan S with $T_S = T_0$ and $B_S = B$. The source layer consists of the obstacles initially in $T_0 \cup B$ and the sink layer consists of the holes outside of $T_0 \cup B$ and the vertices in $T_0 \cup B$. The intermediate layer consists of the vertices of T_0 which are initially vacant and thus potential temporary locations of obstacles between the forward and the backward moves. The rules for the presence and costs of the arcs are very similar to those in Section 2.2. We need to solve a flow problem for each of $O(n^2)$ potential shapes of B. However, combining the problems for B with minor differences, we can reduce the number of flow problems required down to n.

Theorem 8: *An approximate solution for* GMP1R *on a tree, with cost at most 7 times the optimal, can be computed by solving at most n mincost flow problems on a graph with $O(n)$ vertices.*

3. An approximation algorithm for general graphs

In this section, we consider general graphs and construct polynomial time algorithms to solve GMP1R approximately. In the following, we assume that s is a fork vertex. This assumption is justified, in the context of approximation , by the following observation.

Lemma 9: *Let s be an internal vertex of a chain C, and let s_1 and s_2 be two endpoints of C. Suppose further that t is not on this chain. Let S_i, $i = 1, 2$, be the optimal plan to bring the robot from s to s_i and let S_i' be the optimal plan to bring the robot to t starting with the final configuration of S_i. Then, for either $i = 1$ or 2, the plan S_i followed by S_i' is a solution to the original problem whose cost is at most three times the optimal.*

Proof: If the robot in the optimal plan exits C from s_i, the optimal cost must be greater than the cost of S_i. But S_i' can do no worse than undoing S_i and then executing the optimal s-to-t plan. □

A natural heuristic for an approximate solution is to let the robot follow the shortest path from s to t, with possible side-steppings. It is easy to see that the plan obtained by this heuristic can be as bad as $\Omega(l_{max})$ times the optimal, where l_{max} is the length of the longest chain in G. This is because a chain of length l packed with obstacles requires $\Omega(l^2)$ steps to clear, while the optimal plan may choose a slightly longer path with few obstacles. In fact, a much worse case exists. Suppose that there are two disjoint paths from s to t of different lengths. Each internal vertex of the shorter path has a single leaf attached to it. In the longer path, any two vertices having distance 2 on the path are connected by an additional chain of length 2. Thus, the chains on both paths are all of length 1. If there are only 2 holes in the entire graph, the longer path can be traversed in a number of steps that is linear in its length, while the shorter one requires a quadratic number of steps (at each robot move, a hole must be recycled through the cycle consisting of the two s–t paths). Yet another case in which the shortest path performs poorly is when there is a rich pool of holes "closer" to the longer path than the shorter path.

These examples motivate the following estimates of the cost of traversing a chain, in addition to the obvious estimate — its length.

1. *Evacuation cost* $\omega(C)$: the cost required to clear chain C assuming that an infinite source of holes is attached to each end of the chain.
2. *Cycle cost* $\chi_h(C)$: the cost to be spent for recycling holes when the robot traverses the chain using exactly h distinct holes. If h holes are enough to fill up the chain, this cost is set to 0. If there is no cycle containing C and h holes are not enough to fill up the chain, this cost is set to ∞.
3. *Hole-fetch cost* $\alpha_h(C)$: the cost required to bring h holes to chain C in the initial configuration, regarding the robot as one of the obstacles.

More formal definitions of these measures are given in the full paper. Note that all of these measures can be computed efficiently based on the initial configuration.

Suppose that the optimal plan uses at most h distinct holes in the robot's traversal of any single chain. Then its cost is at least $\sum_i (l(C_i) + \omega(C_i) + \chi_h(C_i))$, where the summation is over all chains C_i traversed by the robot in the optimal plan. Its cost is also at least $\max_i(\min_{2 \le h' \le h}(\alpha_{h'}(C_i) + \chi_{h'}(C_i))$, because the traversal of any chain C_i requires globally fetching some holes to C_i and recycling them. Moreover, if the optimal plan uses exactly h distinct holes on some chain, its cost is at least $\min_i \alpha_h(C_i)$. Therefore, the cost of the optimal plan is at least

$$\frac{1}{3}\{\sum_i (l(C_i) + \omega(C_i) + \chi_h(C_i)) + \max_i(\min_{2 \le h' \le h}(\alpha_{h'}(C_i) + \chi_{h'}(C_i)) + \min \alpha_h(C_i)\}.$$

For each path P from s to t, let $\Gamma_h(P)$ denote $\sum_i (l(C_i) + \omega(C_i) + \chi_h(C_i)) + \max_i(\min_{2 \le h' \le h}(\alpha_{h'}(C_i) + \chi_{h'}(C_i)) + \min_i \alpha_h(C_i)$,

where i indexes all chains in P. Our algorithm tries all values of h, finding for each a path P that minimizes $\Gamma_h(P)$, and constructing a plan in which the robot traverses this path, side-stepping at every fork vertex. Such a path can be found as follows. Let $\mathcal{C}_I$ denote the set of chains of G with the I smallest values of $\min_{2 \le h' \le h}(\alpha_{h'}(C) + \chi_{h'}(C)$ and let $\mathcal{D}_J$ denote the set of chains of G with the J largest values of $\alpha_h(C)$. Let G_{IJ} denote the network obtained from G by replacing each chain C in $\mathcal{C}_I \cap \mathcal{D}_J$ by an arc of length $l(C) + \omega(C) + \chi_h(C)$, and removing all other chains. Solve the s–t shortest path problem on each G_{IJ} and take the solution that minimized $\Gamma_h(P)$. The following Lemma compares the cost of a plan based on P with our estimate $\Gamma_h(P)$.

Lemma 10: *For any $h \ge 2$ and path P from s to t such that $\Gamma_h(P) < \infty$, we can construct, in polynomial time, a complete plan with cost at most $O(k_{max}/l_{min} + 1)\Gamma_h(P)$, where k_{max} is the maximum number of obstacles on a single chain of P in the initial configuration and l_{max} and l_{min} are the lengths of the logest and the shortest chains of P. The cost of this plan is also bounded above by $O(k_{max} l(P) + \Gamma_h(P))$.*

Combined with the above observations, the first part of this lemma immediately leads to:

Theorem 11: *There is a polynomial time $O(l_{max}/l_{min})$–approximation algorithm for GMP1R on a general graph, where l_{max} and l_{min} are the lengths of the longest and the shortest chains of G respectively.*

Although the bound in this theorem, as it is presented, appears to be sensitive to an addition of even one short chain to the graph, a closer look at the proof reveals that this is not the case: addition of short chains does not essentially change the bound on the approximation ratio as long as their total length is $O(l_{max})$. The second bound in Lemma 10 implies n/k_{max}–approximation because $\Gamma_h(P) \ge (k_{max})^2$. Combined with the first bound, we have:

Theorem 12: *There is a polynomial time $O(\sqrt{n})$–approximation algorithm for GMP1R on a general graph.*

4. Extensions and further work

In this section we outline extensions of our algorithms and directions for further work.

The case in which each edge of G has a positive weight associated with it is an interesting generalization of the unweighted case. The problem becomes significantly different, even in the case of a tree. For instance, the quasi-monotonicity of robot's motion does not hold any more, even in the case when the robot starts at a leaf (Figure 3). We refer to the movement depicted in Figure 3 as a *wiggle*. However we are able to establish that the path of the robot still adheres to a certain canonical form. This is described informally in Lemma 13. In the case where the robot does not start from a leaf vertex, the robot does not necessarily back up to the first branch vertex behind it, and the motion on the backward journey need not be simple. Lemma 14 addresses this issue informally. The canonical obstacle moves are similar to the unweighted case. These features suffice to establish the existence of polynomial time algorithm for the GMP1R on weighted trees.

Lemma 13: *There exists an optimal plan for the* GMP1R *on weighted trees for the case where s is a leaf, where the robot moves monotonically towards t, except to make* sidesteps *or* wiggles.

Lemma 14: *In an optimal plan for the* GMP1R *on weighted trees, the robot may start by moving backward initially. In such cases, there exists an optimal plan where the robot proceeds monotonically to some vertex s' behind it (without wiggles or sidesteps), and then proceeds almost monotonically (as in Lemma 13) towards the destination t.*

The proof is omitted. Using these lemmas, we set up a dynamic programming algorithm to solve the GMP1R on weighted trees.

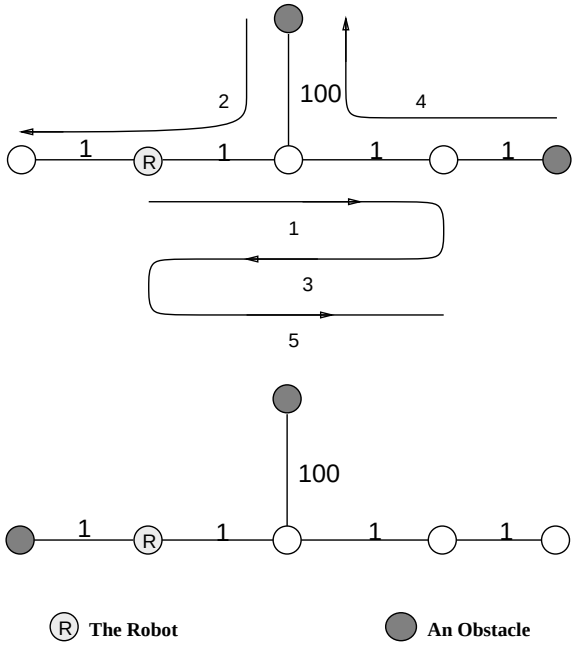

Figure 3: A wiggle in the robot's path

Theorem 15: *There exists a polynomial time algorithm to solve the* GMP1R *on the weighted tree.*

A solution to the weighted version of GMP1R extends immediately to the case when each vertex of G has a positive integral *capacity*, representing the number of objects that may sit on it at one time, because we can simulate a vertex with capacity k by a k-vertex star consisting of weight 0 edges. This models situations in which we have intersections at which objects can move past one another (say, a railway junction).

We conclude by mentioning the most interesting directions for further research:
(1) Show that a simple geometric version of GMP1R is *PSPACE*-hard. The related *warehouseman's problem* is *PSPACE*-hard[4]; however, that reduction breaks down if all the objects have the same size.
(2) Study the case of several robots, each with its own destination, i.e., GMPkR discussed in the introduction.
(3) In situations such as railway networks, several objects can move simultaneously under their own power. This version is closely related to *deflection routing* for packets [3]. Thus, we would study plans with parallel moves allowed, and study the number of steps to deliver every robot to its destination.

References

[1] G. Frederickson and D. J. Guan. Preemptive ensemble motion planning on a tree. *SIAM J. Comput.*, 21:1130-52, 1992.

[2] O. Goldreich. Shortest move-sequence in the generalized 15-puzzle is NP-hard. Manuscript, Laboratory for Computer Science, MIT, June 1984.

[3] B. Hajek. Bounds on evacuation time for deflection routing. *Distributed Computing*, 5:1–6, 1991.

[4] J. Hopcroft, J. T. Schwartz and M. Sharir. *On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehousmen's Problem". International Journal of Robotics Research*, 3:76–88, 1987.

[5] D. Kornhauser, G. Miller, and P. Spirakis. *Coordinating pebble motion on graphs, the diam-*

eter of permutation groups, and applications. Proceedings of the 25th Annual Symposium on Foundations of Computer Science, pages 241–250, 1984.

[6] D. Lichtenstein. *Planar Formulae and Their Uses. SIAM J. Comput.*, 11:329-343, 1982.

[7] J. T. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry and Complexity.* Ablex, 1987.

[8] R.M. Wilson. *Graph Puzzles, Homotopy, and the Alternating Group. Jounal of Combinatorial Theory (B)*, 16:86-96, 1974.