Appears in Proceedings of STOC 94.

Motion Planning on a Graph

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.

*Department of Computer Science and Engineering, UCSD, La Jolla, CA 92093.
†IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.
‡IBM Tokyo Research Laboratory, Kanagawa 242, Japan. Work done at the IBM T.J. Watson Research Center.

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 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 accommodate 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 in polynomial time the decision version, which asks whether at all the robot can be moved from s to t. A number of related geometric motion planning problems are given in [8]. 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, in which there are no obstacles (thus all the robots have specified destinations), has been studied previously. Wilson [9] 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 showing that this bound is the best possible. Goldreich [2] has studied the complexity of determining the shortest move sequence for the GMPkR problem and shown that this is NP-hard in the case $k = n - 1$. Ratner and Warmuth [7] show that this is the case even if the graph is restricted to be a $\sqrt{n} \times \sqrt{n}$ grid (the case of interest in the generalized 15-puzzle). In our problem, we have

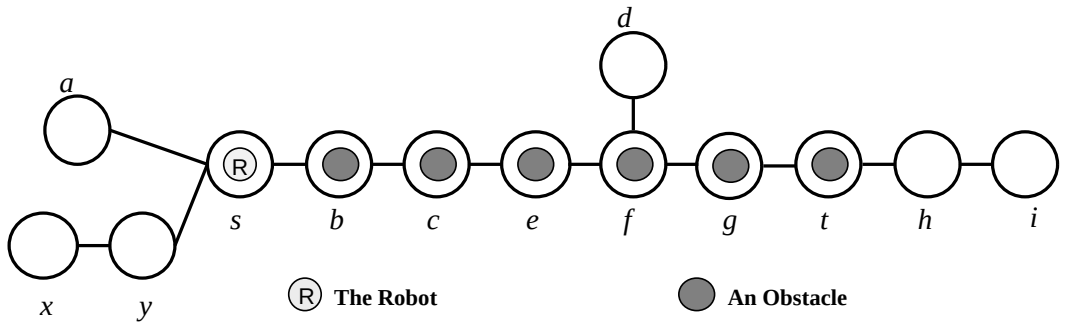

Figure 1: An instructive instance on a tree.

obstacles for which no destinations are specified; these obstacles have to be moved "somewhere" out of the way. The introduction of such obstacles appears to make the problem harder.

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 Appendix A. 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 Section 1.2, 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 that achieves a plan of length at most 7 times the optimal length. In Section 3 we give an 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 $O(l_{max}/l_{min})$

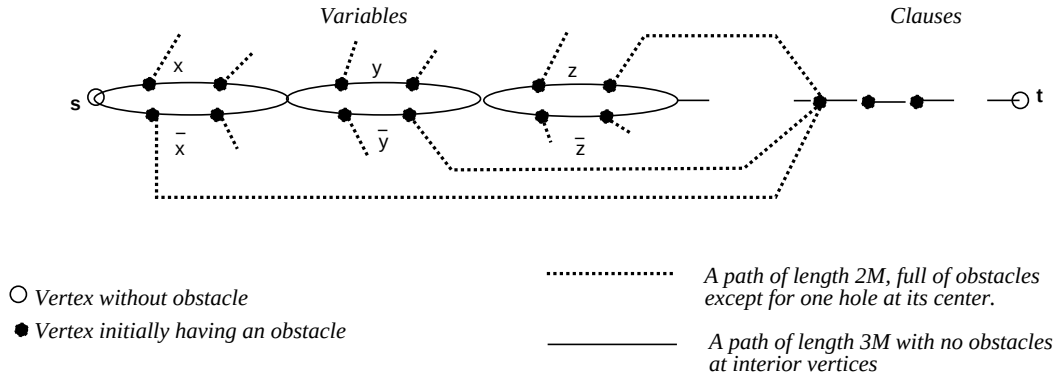

Figure 2: The NP-hardness reduction; the clause $x + y + \bar{z}$ is illustrated.

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 taken literally so that the vertex with the robot is also a hole. 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 "correct 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). This is so 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 directed path P. Often we specify an obstacle move by a source-destination pair, in which case the path taken is assumed to be a 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? For the moment let us 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. Below, several variations of the matching principle are used to simplify given plans and to obtain optimal subplans.

2. Trees

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 Appendix B.

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 directed 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 does not enter 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 subtree induced by 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 on this move 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. (*Notice that for some (e, n_1, n_2, n_3) the optimal (e, n_1, n_2, n_3)-respecting plan need not be canonical. However the optimal plan is canonical and (e, n_1, n_2, n_3) respecting for some (e, n_1, n_2, n_3) and this is all we require.*) 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 Appendix D. 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 Appendix C. The cost of an arc is the length of the path (in the tree G) between the obstacle and the corresponding (virtual) hole. 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 pairs (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 cost. 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 mincost flow problems 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 mincost flow problems on at most n networks with $O(n)$ nodes each.*

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 an 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 far 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 Appendix E. 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 $?_h(P)$ 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 $?_h(P)$, 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 $?_h(P)$. The following Lemma compares the cost of a plan based on P with our estimate $?_h(P)$.

Lemma 10: *For any $h\ge 2$ and path P from s to t such that $?_h(P)<\infty$, we can construct, in polynomial time, a complete plan with cost at most $O(k_{max}/l_{min}+1)?_h(P)$, 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 longest and the shortest chains of P. The cost of this plan is also bounded above by $O(k_{max}l(P)+?_h(P))$.*

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 $?_h(P)\ge(k_{max})^2$. 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 Claim 1. 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. Claim 2 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 GMP1R on weighted trees.

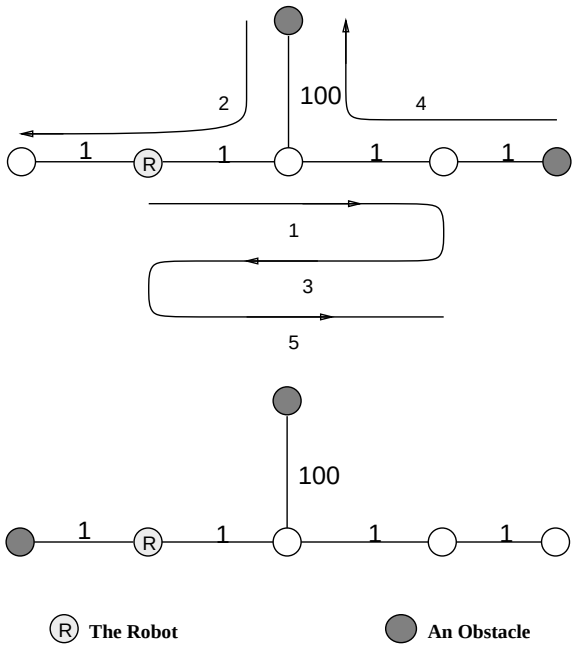

Figure 3: A wiggle in the robot's path

Claim 1: *There exists an optimal plan for* 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.

Claim 2: *In an optimal plan for* 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 Claim 1) towards the destination t.*

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

Theorem 13: *There exists a polynomial time algorithm to solve* GMP1R *on weighted trees.*

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; this is the GMPkR problem discussed in Section 1.
(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. Technical Report no. 792, Computer Science Department, Technion, Haifa. December 1993.

[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 "Warehousemen'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 diameter 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] D. Ratner and M. Warmuth. Finding a shortest solution for the $(N \times N)$-extension of the 15-puzzle is NP-hard. *Journal of Symbolic Computation*, 10:111–137, 1990.

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

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

A. *NP*-hardness reduction

Given an instance of 3-SAT in which each literal occurs at most twice, we construct G as follows. Consider the scheme suggested in Figure 2. The robot must travel from the leftmost vertex s to the rightmost vertex t. En route, it must first pass through a number of variable gadgets, setting one variable to either its true or complemented form as it passes through. Each of these gadgets consists of a cycle of length $6B$. There is an entry point and an exit point, and these are diametrically apart on the cycle. The two resulting paths between the entry point and the exit point correspond to the true and complemented settings for the variable. On each of these paths, at distance B from the entry and exit points, are nodes on each of which an obstacle is placed initially. To set a variable, the two obstacles on the true or complemented path (upper or lower in Figure 2) must be cleared out. This is done in M steps per obstacle, pushing the array of obstacles on the path leading to the clause; any other way requires at least $3M$ steps. After thus setting all the variables, it passes through a number of clause nodes, each of which initially has an obstacle on it. This obstacle can be cleared with M steps if the variables are set so that the clause is satisfied, but requires at least $2M$ steps otherwise. Set $M = 10(m+n)$ where m is the number of clauses and n is the number of variables. Then, knowing a satisfying assignment, we can bring the robot to t in $k = 9Mn+3Mm+M(2n+m)$ steps. Conversely, a solution of length k must correspond to a satisfying assignment. Note that, to let the robot make a short cut through the path packed with obstacles, we need to spend $M^2 > k$ steps to clear the path.

For the hardness of GMP1R on a planar graph, we first reduce a given 3-SAT instance to a planar 3-SAT by Lichtenstein's construction[6]. By a minor modification, we may arrange the edges out of each variable so that edges leading to positive literals and those leading to negative literals are separated into two consecutive blocks when we scan those edges clockwise. Then, we convert this graph into a planar version of the construction in the previous paragraph, by adding a robot path threading the variables and the clauses. To keep planarity, we may have to allow variables and clauses to appear in a mixed order; note, however, that this does not change the validity of the reduction.

B. Proofs of the canonical form lemmas for trees

In this appendix, we prove the lemmas in Section 2.2 that established crucial properties of the optimal plan on a tree. For the purposes of this appendix, we imagine that both obstacles and holes carry identities. When an obstacle goes from u to v in one move, a hole moves from v to u. We will rely heavily on this hole movement view. We consider two configurations equivalent if they are identical disregarding identities of obstacles and holes. Two plans are equivalent if their initial and final plans are respectively equivalent.

The following variation of the matching principle will be used frequently. Let γ_1 and γ_2 be two configurations with the same numbers of obstacles. Let V_1 be the set of vertices with an obstacle in configuration γ_1 (respectively, γ_2,). Consider the 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. In particular, if $v_1 = v_2 \in V_1 \cap V_2$ there is an edge with zero cost between them. Let μ denote the minimum cost matching on this bipartite graph as represented as a one-to-one mapping from V_1 to V_2. We call μ the *minimum cost obstacle-matching* from configuration γ_1 to γ_2. Such μ exists because G is connected. Then the plan consisting of obstacle moves $v \to \mu(v)$, $v \in V_1$, is an optimal plan for reaching a configuration equivalent to γ_2 from configuration γ_1, when we ignore the presence of the robot. Here, $\mu(v) = v$ necessarily for each $v \in V_1 \cap V_2$ so that moves $v \to \mu(v)$

for such v are ignored. Note that the roles of the obstacles and the holes are completely symmetric without the presence of the robot. Therefore, we may replace the word "obstacle" by "hole" in the above to get a *hole-matching* version of the matching principle.

We start with a lemma that applies not only to a tree but also to a general graph. We call a plan *monotonic* if it moves the robot along a simple path without changing direction. We call a plan *chain-monotonic* if every part of the plan in which the robot starts from an internal vertex u of some chain C and stays on C throughout is monotonic.

Lemma 14: *For any given plan for* GMP1R *on a general graph G, there is an equivalent plan with no greater cost that is chain-monotonic.*

Proof: Let C be a chain and let S be a plan in which the robot starts from an internal vertex u of C and ends up on a vertex v of C without ever leaving C. We show that S can be transformed into an equivalent monotonic plan without increasing the cost. Then we will be done, because any general plan can be transformed into a chain-monotonic one by repeating this process, without increasing the cost.

For $k \geq 1$, let $(v_0, v_1, \ldots, v_k)$ be the subpath of chain C such that $v_0 = u$ and $v_k = v$. For $1 \leq i \leq k$, let γ_i, $0 \leq i \leq k$, denote the configuration at the step when the robot enters v_k for the last time. Let μ_i denote the minimum cost obstacle-matching from γ_{i-1} to γ_i in the subgraph of G induced by the removal of vertex v_{i-1}. Our modified plan consists of k parts S_1, ..., S_k, where each S_i consists of obstacle moves $v \rightarrow \mu_i(v)$ for each v in the domain of μ_i, followed by the robot move from v_{i-1} to v_i. When we compare S_i to the original part of S to get from γ_{i-1} to γ_i, it is is not difficult to see that the matching principle still applies even though the robot may wander in the original plan, because it stays on the chain C. Thus, the cost of our plan is no more than that of the original. It is monotonic and produces a configuration equivalent to the final configuration of S. □

From here on, we assume that G is an unweighted tree. Given a configuration γ, we use notation $u \sim_\gamma v$ to mean that the path from u to v does not contain the robot. We often drop the subscript γ, leaving the configuration to the context.

We need several lemmas leading to the proof of Lemma 2.

Lemma 15: *Let S be a monotonic plan. Then there exists an equivalent monotonic plan S' with no greater cost which consists of a sequence of obstacle moves, followed by an uninterrupted sequence of robot moves, followed in turn by another sequence of obstacle moves.*

Proof: Let v_1 and v_2 be the initial and the final positions of the robot S and let P be the path from u to v. Let γ_1 and γ_2 be the initial and the final configuration respectively. We say an obstacle move from u to v occurs *ahead of the robot* if $u \sim v_2$ (and hence also $v \sim v_2$) at the moment of its execution in S. For a particular obstacle, the sequence of its moves consists of a possibly empty sequence of moves ahead of the robot, followed by a possibly empty sequence of moves not ahead of the robot. For each vertex v that has an obstacle in γ_1, let $f(v)$ denote the vertex on which the obstacle initially on v lands after completing the moves ahead of the robot. Observe first that $f(v)$ cannot be on P. We further claim that $f(u) \neq f(v)$ if $u \neq v$. Suppose to the contrary that $f(u) = f(v) = w$ for some $u \neq v$. Let w' be the vertex on P closest to w. When the robot is on w', the two obstacles initially on u and on v must then be on w, a contradiction. Now define a configuration γ by placing an obstacle that is on v in γ_1 on the vertex $f(v)$, and placing the robot on v_1. The first part of our plan S' is the

optimal plan to get from γ_1 to γ_2 obtained by the matching principle. This plan is valid for γ_1 because all the moves are ahead of the robot which is fixed at v_1. Then, we move the robot to v_2 through P. The last part of our plan is the optimal plan to get from γ' to γ_2, where γ' is the same as γ except that the robot is on v_2. □

Lemma 16: *Let v be a fork vertex and let v_1, v_2, and u be distinct vertices adjacent to v. Let S be a plan in which*
(1) the first step is a robot move from v_1 to v,
(2) the final step is a robot move from v to v_2,
(3) the penultimate step of the robot is not *from v_2, and*
(4) the robot visits u at some point.
Suppose moreover that, when the robot leaves v for the first time to an adjacent vertex other than v_1 or v_2, it is to u. Then, there exists a plan equivalent to S with no greater cost, in which the robot makes exactly four moves $v_1 \rightarrow v \rightarrow u \rightarrow v \rightarrow v_2$.

Proof: By Lemma 14 we assume without loss of generality that S is chain-monotonic. Let γ_1 and γ_2 denote the initial and the final configurations of S. For each vertex w adjacent to v, let T_w denote the connected component that contains w when v is removed from G. We have to consider several cases.

(Case 1) Suppose that the robot's second move in S is $v \rightarrow u$. Let the robot's penultimate move in S be $w \rightarrow v$. It is possible that $w = v_1$ or $w = u$ but not $w = v_2$ by our assumption (3). This last possibility occurs only when v_2 is also a fork vertex, because of the chain-monotonicity of S. By Lemma 15 we may assume without loss of generality that v_1, v, and u all have holes in γ_1. Similarly, we may assume that, in γ_2, all of v_2, v, and w have holes in γ_2. Let μ be the minimum hole-matching from γ_1 to γ_2. Note that the cost of hole moves in S is at least as large as the cost of μ. The idea is to construct a plan in which the robot makes the four-step move described above, and every hole movement specified by μ is executed at some appropriate timing. We have several cases to consider.
(Case 1.1) $w = u$. Since both v and u have holes in γ_1 and in γ_2, $\mu(v) = v$ and $\mu(u) = u$, i.e., μ does not move these holes. While the robot is still on v_1, make all moves of holes specified by μ that are now executable. More precisely, for each vertex z such that $\mu(z)$ is defined and $\mu(z) \sim z$, move the hole on z to $\mu(z)$. Then, move the robot to u and make all moves specified by μ that are now made possible. This should bring a hole to v_2. Move the robot to v_2 and make all the remaining moves specified by μ.
(Case 1.2) $w \neq u$. Since v has a hole in γ_1 and in γ_2, $\mu(v) = v$. When the robot is still on v_1, make all moves of μ that are now executable, *except the move of the hole from u* even if $\mu(u) \neq u$. Move the robot to u and make all moves of μ now executable. If $\mu(u)$ is not in T_{v_2}, v_2 must now have a hole so we proceed as in Case 1.1. Suppose now that $\mu(u) \in T_{v_2}$. We would not have had a chance to move the hole on u to $\mu(u)$. However, w must have received a hole in this case because $w \notin T_{v_2}$. In this case, move the hole on w to $\mu(u)$ (which ensures that v_2 now has a hole), move the robot to v_2, move the hole on u to w, and finally make all the remaining moves specified by μ. In our plan, two extra traversals of edge (v, w) are made by holes but, on the other hand, at least two traversals of the same edge by the robot are removed. Thus, the overall cost is not increased. Note that the above argument is valid even if $w = v_1$.

(Case 2) The case where the robot's penultimate move is from u is treated similarly by a symmetric argument.

(Case 3) The remaining case is that the robot's second move is to v_2 and its second last move is from v_1. We may assume without loss of generality that, in both γ_1 and γ_2, there are three holes on v_1, v,

and v_2.
(Case 3.1) Suppose first that u has a hole in the final configuration γ_2. Let μ be the matching from γ_1 to γ_2. Before any robot move, make all executable moves of μ. If u has a hole at this point, move the robot to u, make moves of μ that are now executable, move the robot to v_2, and then execute the remaining moves of μ. If u does not have a hole, move the hole on v_2 to u, move the robot to u, move the hole on $\mu^{-1}(u)$ to v_2 (which should be possible because $\mu^{-1}(u)$ must be in T_{v_1}), and move the robot to v_2, executing each of the remaining moves of μ at an appropriate timing. We make two extra traversals of edge (v, v_2) but save at least two robot traversals of the same edge.
(Case 3.2) Now suppose that u does not have a hole in γ_2. Let γ_u be the configuration immediately before the robot's first move from v to u. Let μ_1 be the minimum cost hole-matching from γ_1 to γ_u and μ_2 the minimum cost hole-matching from γ_u to γ_2. Noting that u has a hole in configuration γ_u but not in γ_2, let $u' = \mu_2(u)$. By the minimality of μ, u' does not have a hole in configuration γ_u. Let γ_2' be the configuration obtained from γ_2 by moving the hole from u' to u. Let μ_2' be the matching from γ_u to γ_2' obtained from μ_2 by modifying its value at u by $\mu_2'(u) = u$. The cost of μ_2' is that of μ_2 minus the cost of the path from u to u'. Since the composition of μ_1 and μ_2' is a matching from γ_1 to γ_2', the minimum-cost matching μ from γ_1 to γ_2' has a cost no greater than the total cost of μ_1 and μ_2'. Our plan is to first get to the configuration γ_2' bringing the robot to v_2 and then finish by bringing the hole on u to u'. This first part is similar to Case 3.1. Before any robot move, make all moves of μ that are immediately executable. Suppose first that u has a hole at this point. Move the robot to u, then make moves of μ which are now executable. If $u' \notin T_{v_2}$, move the robot to v_2, complete the remaining moves of μ, and finally move the hole from u to u'. If, on the other hand, $u' \in T_{v_2}$, move the hole on v_1 (which is available because $u' \neq v_1$) to u', move the robot to v_2, move the hole on u to v_1 and complete the remaining moves of μ. The two extra traversals of edge (v_1, v) are compensated by the saving on the robot moves on the same edge. Now suppose that u does not have a hole after the fist stage of hole moves. If $u' \neq v_2$, it must be that $\mu(v_2) = v_2$, so move the hole still on v_2 to u, move the robot to u, move the hole from $\mu^{-1}(u)$ to v_2. From this point, proceed exactly in the same way as above. In this case we make two extra traversals of edge (v_1, v) and two extra traversals of edge (v, v_2). However, these are compensated by the savings on the robot moves on the same edges. So far, we have considered all the cases except when $u' = v_2$ and $\mu^{-1}(u) \in T_{v_1}$. In this case, our first step is to make all moves of μ immediately executable *except the move from* v_2. Then we move the hole on v_2 to u and move the robot to u. Make all moves of μ executable at this point, move the hole on v_1 to v_2, and move the robot to v_2. Before or after this robot move, as appropriate, move the hole from $\mu^{-1}(u)$ to $\mu(u') = \mu(v_2)$. Finally, move the hole on u to v_1 and execute all the remaining moves of μ. Again, the number of extra traversals are at most two on edge (v_1, v) and at most two on (v, v_2).
□

We remark that Lemmata 15 and 16 hold even if the tree is weighted. The following lemma, in exactly the form given here, applies only to an unweighted tree. Let u and v be adjacent vertices in the tree and let T_u and T_v denote the two trees that result from removing the edge (u, v). We call two configurations γ_1 and γ_2 (u, v)–*equivalent* if
(1) the two configurations are equivalent when restricted to T_v, and
(2) there is a one-to-one correspondence f from the holes of γ_1 in T_u to the holes of γ_2 in T_u such that the distance of a hole a to u equals the distance of the corresponding hole $f(a)$ to u.
Note that if γ_1 and γ_2 are (u, v)–equivalent then any plan with initial configuration γ_1 in which the robot stays in T_v can be executed as a plan with initial configuration γ_2 with minor modification and without any cost increase.

Lemma 17: *Let P be a path from s to t in the tree G and suppose S is a plan such that*

(1) the first step of S is a robot move from s into P,
(2) the last step of S is a robot move to t from inside P,
(3) the robot stays on P throughout, and
(4) those holes that are initially behind s (i.e., their path to t contain s) never move.
Than there is a quasi-monotonic plan with cost no greater than S that is (t', t)-equivalent to S, where t' is the vertex on P adjacent to t.

Proof: For each vertex v on P, let a_v denote the hole that is on v the first time the robot visits v in the plan S. As we will see, we may assume without loss of generality that a_v is on v each time the robot subsequently returns to v. To show that this assumption is valid, we give a transformation from an arbitrary plan S to an equivalent one with this property. Consider a pair of events (E, F) that occurs at a v on P: E is an event in which a hole leaves v and F is an event in which a distinct hole moves to v for the first time after E. We call such an event pair (E, F) at vertex v *bad* if the robot leaves v before E, returns to v after F, and never visits v between E and F. Let (E_0, F_0) be the bad event pair such that F_0 is the first to occur among the second events of all bad event pairs of S. Then event E_0 must be a move of the hole a_v from v to some vertex u. Let b_v be the hole that is brought to v in the second event of this pair. Let us extend the notation $x \sim y$ to allow a hole to appear as x or y instead of a vertex. For example, we write $a \sim v$ to mean that the current vertex of hole a is connected to v without going through the robot.

Our new plan exactly follows S before E_0. Then, instead of moving a_v from v to u, we leave a_v on v and place a *shadow* a'_v of a_v on u. This shadow is merely for the book-keeping purposes and should be ignored once the construction of our plan is completed. From this point on, our plan continues following S, except that when a_v moves in S, we move the shadow a'_v instead. If $a'_v \sim b_v$ holds at any point, we move the hole b_v to the vertex of a'_v, erasing the shadow a'_v and returning to the exact simulation of S. Note that the resulting plan would be equivalent to S, and the cost is not increased because we are simply shortcutting the moves of two holes into a move of one hole. Note also that this simulation is valid provided the shadow is erased before event F_0 and before it is stepped on by the robot. We show that this is indeed the case.

Suppose first that $a'_v \sim b_v$ never happens before event F_0. This means that the robot is always on the path a'_v– b_v. But, we have $a'_v \sim v$ immediately after E_0 and $b_v \sim v$ immediately before F_0. To achieve this while staying on the path a'_v–b_v, the robot would have to step out of the path P towards a'_v at some point, contradicting our assumption. Thus, it only remains to show that the robot does not step on the shadow before it is erased. Suppose the robot tries to step on the shadow a'_v for the first time after event E_0. At this moment we must have $a'_v \not\sim v$, because otherwise this step would be the second event of a bad event pair occurring earlier than F_0, contradicting the choice of the pair (E_0, F_0). But for the shadow to go from v's side of the robot to the other side on P, we must have $a'_v \sim b_v$ at some point, by a similar reasoning as before. Thus, we would have been able to erase the shadow before this stepping-on were attempted. This transformation gives a plan equivalent to and no costlier than S, with one fewer bad event pair. Repeat this until we have no bad event pairs. Rename the resulting plan S.

Now we have established that, for each v on P, the same hole a_v is on v every time the robot visits v in S. We say the robot *uses the hole a_v on v*. The same hole may be used on several vertices; we refer to the *first use*, *second use*, and so on, meaning the order on the path P from s to t. Suppose a hole a is used on distinct vertices u and v in this order, without being used on any other u' between u and v. For this to happen, there must be a fork vertex w between u and v and a neighbor w' of w not on P such that the following two events occur at S:

(1) a moves from w to w' while the robot is strictly between w and v.
(2) later, a moves from w' to w while the robot is strictly between u and w.
Note that for this to happen, w must be at least two edges apart from both u and v. We call the edge (w, w') the *sidestep edge* of the hole a to get from u to v. For each branch vertex w on P, let A_w denote the set of holes that have a sidestep edge out of w.

We are now ready to describe our plan. For each $v \neq t$ on the path P, let next(v) denote the next vertex after v on P towards t. Similarly, let prev(v) denote the vertex v' such that next$(v') = v$. Prior to any robot move, bring each hole used somewhere on P to the vertex of its first use. This must be possible because of the assumption (4) in the lemma. Let $w_1 \neq s$ be the branch vertex on P that is the closest to s. Then the path from s to next(w_1) must now be filled with holes, because it is impossible for the original plan S to use a hole on more than two vertices on this path. Now we move the robot monotonically to w_1. In general, the robot moves from a fork vertex w_i to the next fork vertex w_{i+1} monotonically. Before leaving w_i, we make sure that the path from w_i to next(w_{i+1}) is clear, making the robot sidestep or wiggle if necessary. (When w_i is the last fork vertex then we similarly consider the path from w_i to t.) This is done according to the following cases.
(Case 1) The next path from w_i to next(w_{i+1}) is already clear. There is nothing to do. Note that if $A_{w_i} =$ then this must always be the case.
(Case 2) The next path is not clear and A_{w_i} is non-empty. Among the side-step edges out of w_i for the holes in A_{w_i}, choose an arbitrary edge (w_i, w_i'). Our plan is to let the robot side-step to w_i': i.e., move the robot to w_i', move each hole in A_{w_i} from the vertex of its previous use to the vertex of its next use, and bring the robot back to w_i'. To do this, we need to bring a hole to w_i' first. The precise way of doing this is as follows. Immediately before the arrival of the robot to w_i, bring the hole on next(w_i) to w_i'. The robot goes to w_i' through w_i; now execute the forward moves of the holes as described above, move the hole on prev(w_i) to next(w_i), move the robot to next(w_i) through w_i. This leaves a hole on w_i' which is supposed be on prev(w_i) in the final configuration of the original plan. However, this difference does not destroy the (t', t)-equivalence we are claiming. The moves added in our plan that are not in the original plan are: one traversal of a hole through each of the edges (prev$(w_i), w_i)$ and (w_i, w_i'), two traversals of a hole through the edge $(w_i,$ next$(w_i))$, and two robot traversals through the edge (w_i, w_i'). On the other hand, we are saving at least 2 traversals of a hole through edge (w_i, w_i') and at least two robot traversals through each of the edges (prev$(w_i), w_i)$ and $(w_i,$ next$(w_i))$. Therefore, we have not increased the cost in our plan. (Here we are relying on the uniform cost of the edges). There is a possibility of undercounting in case next(w_i) is the next fork vertex. We consider this below.

What remains to be done is a close examination of the case when w_i and w_{i+1} are adjacent. If the robot in the original plan traverses the edge (w_i, w_{i+1}) five times or more, then there is no undercounting in the above analysis. The undercounting could occur only when the edge (w_i, w_{i+1}) is traversed three times and Case 2 applies to each of w_i and w_{i+1}. Suppose that fork vertices $w_i, w_{i+1}, \ldots w_j$, $j > i$, occur consecutively on P, that Case 2 applies to each of them, and that each edge (w_k, w_{k+1}), $i \leq k \leq j-1$ is traversed exactly three times by the robot in the original plan. Then the move of the robot in this part must in fact consist of a monotonic move forward from prev(w_i) to next(w_j), a monotonic move backwards from next(w_j) to prev(w_i), and a monotonic move forward again; otherwise there would be some w_k to which Case 2 does not apply. This implies that, for each hole a that appears in A_{w_k} for some w_k, no use of a occurs on the path from prev(w_i) to next(w_j). Therefore, in our plan, it suffices for the robot to side-step at only one fork vertex among $w_i, \ldots, w_j$. Now our saving exceeds the loss. □

Now we are ready to prove the quasi-monotonicity lemma.

Proof of Lemma 2: Let S be an optimal complete plan and let T_S denote the subtree of G consisting of the vertices visited by the robot after the robot leaves s in the direction of t for the first time. By Lemma 16, we may assume without loss of generality that each vertex in T_S not on the s–t path is a leaf attached to an internal vertex of the s–t path, which we call a branch vertex. Moreover, by the same lemma, we may assume that the robot passes each branch vertex w in the following way: reach w, sidestep to exactly one of the leaves of T_S adjacent to w, return to w, and leave w for t. Later, the robot may come back to w but not move further back through w; the next robot move must be towards t after such a return. Thus, the robot's move is almost quasi-monotonic except that it may oscillate between two consecutive branches. Apply Lemma 17 to make this traversal of each such path monotonic, working from s towards t and introducing some more branch vertices if required. □

Proof of Lemma 3: We need to consider the trajectory of the robot in the back up part. Now let S be an optimal complete plan as in the above lemma and let B_S denote the tree visited by the robot in the back up part of S, including s. We have several cases to consider.
(Case 1) B_S consists of a single vertex: s. We are done in this case.
(Case 2) B_S consists of more than one vertex and s is not a fork vertex in G. B_S must contain at least one fork vertex of G as a non-leaf vertex of B_S, because otherwise the back up part would not produce any preflow and an optimal plan would be one without a back up part. Let s' be the fork vertex of G contained in B_S that is the closest to s. Let u be the vertex adjacent to s' that the robot enters when it leaves the s–s' path for the first time. By the chain-monotonicity lemma, we may assume that the robot's move from s to u is monotonic. We again have a few cases according to the locations of the sources of the holes that flow out from behind s' in the original back up plan.
(Case 2.1) No hole flows out from behind u. Then our back up plan simply backs the robot up to u, lets the holes flow out from behind s', and returns the robot to s.
(Case 2.2) Some holes flow out from behind u. Note that the robot must visit at least one vertex not on the s–u path in order to generate a useful preflow. Suppose first that there is another vertex v not on the s–s' path adjacent to s', such that at least one hole flows out from behind v. Then, we follow the original plan until the robot reaches u, bring a hole to v from behind v, let other holes flow out from behind v, move the robot to v, let out the remaining holes behind s' that are to flow out, and finally bring the robot back to s. Note that we are not increasing the number of robot moves. Suppose now that the only flow of holes from behind s' is from behind u. In this case, we move the hole on u to v before the robot. Then, we back up the robot to v instead of u, let the holes behind u flow out, and return the robot to s. The extra cost of 2 for the hole move is compensated by the saving in the number of robot moves.
(Case 3) B_S consists of at least 2 vertices and s is a fork vertex of G. If the degree of s in B_S is 1 and all the flow from behind s comes through this edge of s in B_S, then there must be another fork vertex s' of G in B_S and the proof is the same as in Case 2. Otherwise, our modified back up plan simply visits one or two vertices behind s. The details are very similar to Case 2.
□

Finally we prove the main lemma of Section 2.1.

Proof of Lemma 4: Let S be a quasi-bitonic optimal plan as given by Lemma 3. By the proof above, we may assume that the back up part of S already satisfies the properties P1 and P2 in the definition of canonical plans. Also, the backward flow of obstacles in this back up part satisfies P5. Consider now the forward part of S. Because the sequence of robot moves from one sidestep vertex (or s) to the next sidestep vertex (or t) is monotone, by Lemma 15 we may assume that obstacles move only when the robot is on s or on a sidestep vertex. For each vertex v in T_S (recall that this is the trajectory of the robot's quasi-monotonic move from s to t), let a_v denote the hole that is on v when the robot

visits v. Along each monotone path, this assignment must be distinct. We say that hole a is used on $v \in T_S$ if $a = a_v$. Each hole may be used on more than one vertex; we speak of the first use, the second use, etc. referring to the order along the walk from s to t. Now we modify S as follows. Prior to any robot move, move each hole in the t-side of s that is used on any vertex in T_S to the vertex of its first use, unless there is already a hole on that vertex. If there is already a hole b on the vertex where a is first used, rename b to a and a to b. This renaming is done at no cost, thus saving the cost of moving a; this saving compensates for the potentially increased cost of moving b later. Then start the back up part. During this back up part, bring each remaining hole to the vertex of its first use in T_S. Such moves occur only when the robot is on one of the back up vertices. When the back up part is complete, every hole ever used in T_S must now be at the vertex of its first use. This means that the path to the first sidestep vertex is clear. Move the robot to the first sidestep vertex. In general, when the robot reaches a sidestep vertex, bring each hole used in the next monotone path from the vertex of its previous use, and then move the robot to the next sidestep vertex. We have not increased the cost of the plan in this conversion, because the sequence of the moves of each hole in the modified plan S' is a shortcut version of (if at all different from) that in the original plan S. If we examine the moves of *obstacles* in S', we can verify that all properties of a canonical plan are satisfied. □

C. Mincost flow network in the atomic cost computation

In this appendix, we describe the mincost flow network used in the computation of the atomic cost atomic-opt$(e, n_1, n_2, n_3, e', n_1', n_2', n_3')$. Recall that v and v' are the tails of e and e' respectively, w is the sidestep vertex adjacent to v, and P is the path from w to the head of e'. Let F be the subgraph of G consisting of the vertices simultaneously in the v–side of e and in the v–side of e' but not on P. Thus F consists of some subtrees hanging off the path P.

To construct the network let us examine the potential obstacles which use the path P. These are
(1) the n_1 preflow obstacles which move ahead of v before the robot,
(2) the n_2' backflow obstacles which move behind v' before the robot reaches v'
(3) the n_3' postflow obstacles which move behind v' after the robot has reached v', and finally
(4) the obstacles which are found on the path P in the original configuration.
For each of these obstacles we have a source in our layered network, which we will refer to as a type 1, 2, 3, or 4 source according to the 4 cases above.

The n_1 preflow obstacles at v must be pushed somewhere ahead of v. These obstacles can potentially be absorbed by holes on P, holes in F, or they can be pushed beyond v' becoming preflow obstacles for the vertex v'. The obstacles going into F correspond to the outward move in the definition of a canonical plan and therefore do not move once they reach their destinations. Thus, we have a node in the sink layer corresponding to each hole in F, which we will call an *F-type* sink. From each type 1 source, we draw an arc to each F-type sink and assign to it a cost equal to the length of the path from v to the hole corresponding to this sink. Each of the n_1' preflow obstacles also gives rise to a sink, which we call a *preflow-type* sink, with an edge from each of the type 1 sources. The cost of such an arc is the length of the v–v' path. The obstacles that get stored on v or w must get pushed behind v as backflow. Other obstacles stored on P must get pushed back as postflow when the robot sidesteps to w. Thus, each of these holes on P gives a node in the intermediate layer. From each type 1 source to each intermediate node, draw an arc with cost determined by a similar rule. Create one sink node for each n_2 backflow obstacle (we call these *backflow-type* sinks) and to each of them draw an arc from each of the intermediate nodes corresponding to the holes on v and w. Create a sink for

each of the n_3 postflow obstacles (*postflow-type* sinks) and to each of them draw an arc from each of the intermediate nodes corresponding to the holes on P except for those on v and w.

The obstacles on the path P (from the initial configuration) are either pushed behind v as backflow (legitimate only when preflow $n_1 = 0$) or (except for those on w and v) as postflow when the robot sidesteps to w, or are pushed ahead of v', or can be moved to holes in F. Thus each type 4 source node corresponding to an obstacle on v or w is directly connected to sink nodes of back-flow type, F-type, and preflow-type. Each type 4 source node corresponding to an obstacle not on v or w is directly connected to sink nodes of all types.

The n_2' backflow obstacles that move behind v' need to move to holes in F or can be pushed behind v. Thus each type 2 source is directly connected to F-type, backflow-type and postflow-type sinks.

The n_3' postflow obstacles that get pushed behind v' can be moved to any position on the path P (since this path has been cleared to let the robot move), or can be pushed behind v. We create a sink corresponding to each vertex of P, which we call a *P-type* sink, and to each of them draw an arc from each type 3 source. We also draw an arc from each type 3 source to each postflow-type sink.

D. Computing the optimal solution based on the recurrence

Given the recurrence for $\mathrm{opt}(e, n_1, n_2, n_3)$ in Section 2, we still need to give the base case of this recurrence and the final step to get from the values of opt to an optimal complete plan.

Let e_0 be the first edge on the s to t path. The base case of the recurrence is $\mathrm{opt}(e_0, 0, n_2, n_3)$. This corresponds to the optimal way of backing up to generate n_2 backflow and accept n_3 postflow later. Recall that B_S denotes the trajectory of the back up move of the robot and note first that the number of possible shapes of B_S for S canonical is at most n^2. Therefore, it suffices to give an optimal cost when the shape of B_S is fixed to a particular B. This cost is essentially a matching computation where the obstacles to be matched are the n_2 backflow obstacles at s, the obstacles in B, and the n_3 postflow obstacles at s. The obstacles of the first two types are to be matched to the holes further behind B (away from t) and the obstacles of the last type are to be matched to the vertices of B (which we know are clear when the robot crosses e_0) and the holes further behind B.

We are now able to compute $\mathrm{opt}(e_1, n_1, 0, 0)$ where e_1 is the last edge in the s to t path using the recurrence in Section 2.2 and the base case given above. Let $\mathrm{topt}(n_1)$, $n_1 \geq 0$, denote the cost of the optimal plan to store n_1 obstacles that flow across e_1 into t-side of e_1 appropriately. Note that $\mathrm{topt}(n_1)$ can be easily obtained by another application of the matching principle. Then the cost of the optimal complete plan is given by

$$\min_{n_1}(\mathrm{opt}(e_1, n_1, 0, 0) + \mathrm{topt}(n_1)).$$

E. Detailed analysis of the approximation for general graphs

In this appendix, we give more technical details of the approximation algorithm for general graphs described in Section 3.

We start with more precise definitions of the evacuation cost, the cycle cost, and the hole-fetch cost of a chain C. Given a configuration γ, the *evacuation cost* $\omega_\gamma(v)$ of a vertex v is 1 plus the distance of v from the closed fork vertex, if v has an obstacle in γ, and 0 otherwise. The *evacuation cost* of a chain

C, denoted by $\omega_\gamma(C)$ is the sum of the evacuation costs of all the vertices of C. Similarly, denote by $k_\gamma(C)$ the number of obstacles on chain C in configuration γ. Observe that for each chain C, we have $\omega_\gamma(C) > k_\gamma(C)^2/4$.

The *basic cycle cost* of C in G, denoted by $\chi(C)$, is the length of the shortest cycle of G that contains C, if one exists; otherwise, we set $\chi(C) = \infty$. The *h–cycle cost* of a chain C, denoted by $\chi_h(C)$, is defined as follows.

$$\chi_h(C) = \left\{ \begin{array}{ll} 0 & \text{if } h \geq l(C)+3 \\ \max(0, (l(C)+3-h)(\chi(C)-h)) & \text{if } h < l(C)+3 \end{array} \right\}.$$

In particular, if $\chi(C) = \infty$ and $h < l(C)+3$ then we set $\chi_h(C) = \infty$.

Suppose the robot traverses a chain C. We say a hole is *used* in this traversal if it co-resides with the robot on a vertex at some time in the traversal, including the time immediately before the robot enters C and the time immediately after it leaves C. Thus $l(C)+3$ holes are used for a traversal of C, if we count according to multiplicity.

Let C be a chain and u and v its endpoints. We say that a chain C *has h holes immediately available* in some configuration if there are vertices $u', v' \notin C$ adjacent to u and v respectively so that $C \cup \{u', v'\}$ holds h holes in this configuration. The *clear-h cost* of C in a configuration γ, denoted by $\alpha_{\gamma,h}(C)$, where $h \leq l(C)+3$, is the minimum cost to reach a configuration from $\tilde{\gamma}$ such that at least h holes immediately available for C, where $\tilde{\gamma}$ is defined to be the same configuration as γ except that the robot is replaced by a new obstacle.

The following lemma is needed for the proof of Lemma 10. We call a chain C *dense* in configuration γ if $k_\gamma(C) \geq l(C)/2$ and *sparse* otherwise. Thus, a chain is sparse if and only if it has strictly more holes than obstacles on it.

Lemma 18: *Let P be a simple path in G that consists of m chains $C_1, \ldots, C_m$ concatenated in this order, with endpoints $s \in C_1$ and $t \in C_m$. Let s' be the vertex adjacent to s in C_1. Let γ be a configuration and $h \geq 2$ be an integer such that*
(1) the robot is on s,
(2) s' has a hole,
(3)$\chi_h(C_i) < \infty$ for every $1 \leq i \leq m$,
(4)$min_{1 \leq i \leq m} \alpha_{\gamma,h}(C_i) < \infty$.
Let $k_i = k_\gamma(C_i)$ be the number of obstacles on C_i and let $k_{max} = \max_{1 \leq i \leq m} k_i$. Let I_{dense} be the set of indices i such that C_i is dense in γ. Then, there is a plan starting from γ that brings the robot to t and has length

$$O(\ l(P) + \sum_{1 \leq i \leq m} k_i^2 + \sum_{i \in I_{dense}} k_i k_{max} + \sum_{1 \leq i \leq m} \chi_h(C_i) + \min_{1 \leq i \leq m} \alpha_{\gamma,h}(C_i) + \max_{1 \leq i \leq m} \min_{h' \leq h} (\alpha_{\gamma,h}(C_i) + \chi_{h'}(C_i))\ ).$$

Proof: For each common endpoint of C_i and C_{i+1}, $1 \leq i \leq m-1$, choose an arbitrary neighbor w_i which is not the neighbor on C_i or C_{i+1} of this endpoint. In the following, we assume that w_i is distinct from each other and from vertices of P; otherwise the proof is similar except that we have a shorter plan due to the obvious short-cutting robot moves. Let $W = \{w_1, \ldots, w_{m-1}\}$. Let $w_0 = s$ and $w_m = f$ for convenience and let D_i denote the path between w_{i-1} and w_i through C_i, for $1 \leq i \leq m$. We call each D_i a *span*. Our plan consists of $m+1$ parts S_0, S_1, ..., S_m. The first part S_0 is responsible for bringing in h holes to $P \cup W$. Then in each part S_i, we move the robot from w_{i-1} to w_i, after bringing in appropriate number of holes to span D_i.

We first describe S_0. Let i_0, $1 \leq i_0 \leq m$, be such that the h–clear cost $\alpha_{\gamma,h}(C_{i_0})$ of C_{i_0} is the minimum among those of C_i, $1 \leq i \leq m$. Let S_0' be the minimum length plan that brings h holes to $C_{i_0} \cup \{w_{i_0-1}, w_{i_0}\}$ starting from $\tilde{\gamma}$. Recall that $\tilde{\gamma}$ is the same configuration as γ with the robot replaced by a new obstacle. The length of S_0' is at most $\alpha_{\gamma,h}(C_{i_0})+4$, because, by the definition of the clear–h cost, there is a plan of length $\alpha_{\gamma,h}(C_{i_0})$ that brings h holes to $C_{i_0} \cup \{w'_{i_0-1}, w'_{i_0}\}$ where w'_{i_0-1} and w'_{i_0} are some vertices at distance at most 2 from w_{i_0-1} and w_{i_0} respectively. Remove from S_0' all the moves of the holes which are already on $P \cup W$ in configuration γ, and let the resulting plan be S_0''.

Now if no hole moves through s in S_0'', set $S_0 = S_0''$; this plan can be executed even if the robot is on s. Suppose S_0'' makes at least one hole pass through s into P. This means that s is not of degree one and hence of degree at least 3 because it is an endpoint of chain C_1. Let a be the first hole that passes through s into P. Our plan S_0 exactly follows S_0'' up to the point immediately before hole a makes its move into P. We stop a at the vertex adjacent to s. At this point we have 3 holes around the robot, that is, on s itself and on two of its neighbors, because s' has a hole from the beginning by assumption. We now follow the rest of S_0''. Each time some hole passes through s into P, we can execute this move in S_0 by first making the robot step aside to an appropriate neighbor of s using the 3 holes. After following S_0'' in this way to the end, we finish by placing the robot and hole a together on a neighbor of s that is not s' and returning the hole originally on s' to s'. The length of S_0 thus constructed is $O(\alpha_{\gamma,h}(C_{i_0}))$. Finally, redefine w_0 to be the vertex on which the robot now resides and redefine W and D_1 to include this new w_0.

Let γ' denote the configuration after we have executed S_0. Note that at least h holes are on $P \cup W$ in configuration γ' and also that $w_{\gamma'}(C_i) \leq w_\gamma(C_i)$ and $k_{\gamma'}(C_i) \leq w_\gamma(C_i)$ for each C_i, i.e., neither the evacuation cost nor the number of obstacles of each chain has increased by executing S_0.

For each $1 \leq i \leq m$, let V_i be the set of the first h vertices of span D_i, counting from w_{i-1}; if $h \geq l(D_i)$ then V_i is the set of all vertices of D_i. Fix i, and assign to each $v \in V_i$ a distinct hole $h_i(v) \in H \setminus H_i$ according to the following criteria. If v has a hole a in configuration γ' then $h_i(v) = a$. Let $\mathrm{Left}(V_i)$ denote the set of vertices v such that v has an obstacle in γ' and v is closer to w_{i-1} than to w_i. Similarly let $\mathrm{Right}(V_i)$ denote the set of vertices with an obstacle which are closer to w_i. We say that the holes are *to the left of* V_i if they are on vertices in $(\bigcup_{1 \leq j \leq i-1} D_j) \setminus V_i$. Similarly, we say that the holes are to the right of V_i if they are on vertices in $(\bigcup_{i \leq j \leq m} D_j) \setminus V_i$. If the number of holes to the left of V_i is at least $|\mathrm{Left}(V_i)|$, we choose $h_i(v)$ for each $v \in \mathrm{Left}(V_i)$ from the holes to the left of V_i so as to minimize the sum of the distances $\sum_{v \in \mathrm{Left}(V_i)} \mathrm{dist}_P(v, \gamma'(h_i(v)))$, where $\gamma'(h_i(v))$ denotes the vertex on which the hole $h_i(v)$ is in configuration γ' and $\mathrm{dist}_P(u,v)$ denotes the distance between u and v along the path P. Similarly, if the number of holes to the right of V_i is at least $|\mathrm{Right}(V_i)|$ we choose $h_i(v)$ for each $v \in \mathrm{Right}(V_i)$ from the holes to the right of V_i subject to a similar distance minimization. Suppose that the number of holes to the left is not sufficient. Then we assign them all to the vertices in $\mathrm{Left}(V_i)$ and, to the rest of the vertices in $\mathrm{Left}(V_i)$ assign the holes to the left (which still remain after the assignment to the vertices in $\mathrm{Right}(V_i)$), minimizing the sum of the distances as above. When this happens, we say that the chain C_i is *right-forced* in the hole assignment. The symmetric case where C_i *left-forced* is treated similarly. Let i_r be the largest value of i such that C_i is sparse and right-forced (if any) and let i_r be the smallest value of i such that C_i is sparse and left-forced (if any). Chains C_{i_r} and C_{i_l} are treated specially in the following description of our plan.

Now, let γ_i denote the configuration at the beginning of each part of the plan S_i, $1 \leq i \leq m$. First suppose that $i \neq i_r$ and $i \neq i_l$. Each S_i starts by bringing to each $v \in V_i$ the corresponding hole $h'_i(v)$ from its location in γ_i, where h'_i is the assignment obtained from h_i as follows. The difference of h'_i from h_i is rather small and is required by the fact that we cannot bring a hole to or from w_{i-1} where

the robot is currently on. Let a_i denote the hole which is on w_{i-1} at the end of S_{i-1}. If $a_i = h_i(w_{i-1})$ then define $h'_i = h_i$. If $a_i = h_i(v)$ for some $v \neq w_{i-1}$ then set $h'_i(u) = h_i(u)$ for each $u \in V_i \setminus \{v, w_{i-1}\}$, $h'_i(w_{i-1}) = a_i$, and $h'_i(v) = h_i(w_{i-1})$. Finally, if there is no $v \in V_i$ such that $a_i = h_i(v)$, then set $h'_i(u) = h_i(u)$ for each $u \in V_i \setminus w_{i-1}$ and $h_i(w_{i-1}) = a_i$. Note that moving the holes according to h'_i costs no more than it would cost for moving the holes according to h_i ignoring the presence of the robot. Call the part of S_i described above the *clearing phase*. After the clearing phase, we start moving the robot towards w_i and, if stuck with an obstacle, proceed by cycling some of the h available holes through the shortest cycle containing C_i. This latter phase costs us at most $l(D_i)+\chi_h(C_i)+O(1)$ steps. We will later give an analysis of the cost of the clearing phase amortized over all i, $1 \leq i \leq m$.

Now suppose that $i = i_r$. Let h' be the value of h that minimizes $\alpha_{\gamma,h'}(C_i) + \chi_{h'}(C_i)$ subject to $h' \leq h$. Bring the closest h' holes on $P \cup W$ to C_i, let the robot traverse D_i using these holes, and then return these holes to where they were in the beginning of this part of the plan. The cost of this part is accounted separately. The initial and the final phases of hole moves can be done with cost $2\alpha_{\gamma,h'}(C_i) + 2?_{other}$, where $?_{other}$ is the total cost of the our plan minus the cost of the parts dealing with the chains C_{i_r} and C_{i_l}. The case $i = i_l$ is similar. Thus to get the required bound on the total cost, it suffices to bound $?_{other}$.

To analyze the cost of the clearing phases, consider the movement of a particular hole a. Let v_a denote the vertex on which a is in configuration γ' and let Z_a be the set of vertices z such that $h_i(z) = a$ for some $i \in \{1, \ldots, m\} \setminus \{i_r, i_l\}$. As we execute S_i, $i \in \{1, \ldots, m\} \setminus \{i_r, i_l\}$ in sequence, hole a visits the vertices of Z_a one by one, starting from the one closest to s (call it l_a), walking along P towards t and ending at the one closest to t (call it r_a). The cost of this walk is at most $4dist_P(v_a, l_a)+4dist_P(v_a, r_a)$. Thus the total cost of the clearing phases is bounded by $4\sum_{a\in A}(dist_P(v_a, l_a) + dist_P(v_a, r_a))$, where A is the set of holes assigned to some vertex by some h_i.

From now on, we focus on estimating the above summation. In the following analysis, we fix the configuration to γ' and leave its reference implicit. We need the following claims.

Claim 3: *1 Let u be an arbitrary vertex on P and let C_i be the chain u belongs to. The are at most $k_{max} + 3$ holes a in A such that (1) l_a is in $D_{i'}$ for some $i' < i$ and (2) u is an internal vertex of the path from l_a to v_a.*

Proof: Suppose that there are more than $k_{max} + 3$ holes that satisfy the above conditions. These holes are all located on the t-side of u. Let b be the hole which is located farthest away from u, breaking ties arbitrarily. Then there are at least $k_{max} + 2$ holes between u and the vertex of b. This is a contradiction for the following reason. Let D_j be the span containing l_b such that $h_j(l_b) = b$. Then, since the number of the obstacles in D_j is at most $k_{max} + 2$, h_j would have assigned these holes between u and the vertex of b to those obstacles rather than b. □

Note that, by symmetry, this claim holds with l_a replaced by r_a.

Claim 4: *2 Let v be a vertex in span D_i with an obstacle. Then, the number of obstacles between v and the vertex of $h_i(v)$ along P, that are in some sparse chain C_j such that $j \neq i$ and $k_j \leq k_i$, is at most $2k_i + 1$.*

Proof: Suppose there are more than $2k_i + 1$ such obstacles. Of these, at most k_i may belong to the chain the hole $h_i(v)$ is in. Since a sparse chain contains strictly more holes than obstacles, we have at least $k_i + 2$ holes outside of D_i and on the path from v to the vertex of $h_i(v)$. (Note that we need

to pay attention to the intersection of adjacent spans.) This is a contradiction to the minimization condition on the assignment h_i. □

Claim 5: *3 Let u be a vertex with an obstacle in a sparse chain C_j. Then, there are at most $k_j + 3$ pairs of the form (i, v) such that (1) v is a vertex with an obstacle in span D_i, (2) $i < j$ and $k_i < k_j$, and (3) the path from v to $h_i(v)$ along P contains u.*

Proof: For each $i < j$, let n_i denote the number of pairs of the form (i, v) that satisfy the above conditions. From the minimization condition of the hole assignment, n_i is at most $k_i + 2 - \sum_{i'} k_{i'}$, where the summation is over all i' such that $i < i' < j$ and $n_{i'} > 0$, because each chain $C_{i'}$ with $n_{i'} > 0$ is sparse and hence has strictly more than $k_{i'}$ holes. It is easy to very that $\sum_{1 \le i < j} n_i$ is at most $k_{i_1} + 4 < k_j + 4$ where i_1 is the smallest value of i such that $n_i > 0$. □

We are now ready to bound the summation $\sum_{a \in A} dist_P(v_a, l_a)$. We adopt the following accounting scheme. For each $a \in A$, we consider each vertex on the path from l_a to v_a to carry a unit cost, and we will charge each such unit cost to one of the m accounts $?_1, \ldots, ?_m$ according to the rules described below. Suppose $u \neq l_a$ is on the path from l_a to v_a along P and let i_a be such that $h_{i_a}(l_a) = a$.
(Case 1) u is in D_{i_a}.
(Case 1.1) C_{i_a} is dense or not right-forced. We charge the unit cost carried by u to $?_{i_a}$.
(Case 1.2) Suppose otherwise: C_{i_a} is sparse and right-forced. Let i'_a be the smallest value of i such that $i > i_a$ and C_i is sparse and right-forced. Such i'_a must exist because, by the definition of l_a, it cannot be that $i_a = i_r$. Charge the unit cost carried by u to $?_{i'_a}$. Note that this can only happen if $k_{i'_a} > l(C_{i_a})/2$.
(Case 2) u is not in D_{i_a}.
(Case 2.1) A hole is on u. Then h_{i_a} must assign this hole to some vertex in D_{i_a} due to the minimization condition. We charge the unit cost carried by u to $?_{i_a}$.
(Case 2.2) An obstacle is on u. Since this implies that $u \neq v_a$, u must be on P. Let C_i be the chain containing u; choose arbitrary one if u is the intersection of two chains.
(Case 2.2.1) C_i is dense. We charge the unit cost carried by u to $?_i$.
(Case 2.2.2) C_i is sparse.
(Case 2.2.2.1) $k_{i_a} > k_i$. We charge the unit cost carried by u to $?_{i_a}$.
(Case 2.2.2.2) $k_{i_a} \le k_i$. We charge the unit cost carried by u to $?_i$.

For each hole $a \in A$, apply the above rule to each vertex $u \neq l_a$ on the path from l_a to v_a along P. (We account similarly for the path from v_a to r_a.)

The contribution of each case above to $?_j$ is:
(Case 1.1) at most $O(\omega_\gamma(C_j))$;
(Case 1.2) at most $(k_j)^2$;
(Case 2.1) at most $\binom{k_j + 2}{2}$, because each unit cost thus charged to $?_j$ is associated with a unique pair of obstacles in D_j;
(Case 2.2.1) at most $(k_{max} + 3)k_j$ by Claim 1 if C_i is dense and 0 if it is sparse;
(Case 2.2.2.1) at most $(k_j + 2)(2k_j + 1)$ by Claim 2;
(Case 2.2.2.2) 0 if C_j is dense and at most $(k_j + 3)k_j$ by Claim 3 if C_j is sparse.

When C_j is dense, using the fact that $l(C_j) \le 2k_j$, we have

$$?_j \le O(k_j^2 + k_{max}k_j).$$

On the other hand, for sparse C_j we get

$$?_j \leq O(\omega_\gamma(C_j))).$$

Summing this up together with the bound on the length of S_0 and the lengths of the robot move phases,

$$?_{other} \leq O(l(P) + \sum_{1 \leq i \leq m} k_i^2 + \sum_{i \in I_{dense}} k_i k_{max} \sum_{1 \leq i \leq m} \chi_h(C_i) + \min_{1 \leq i \leq m} \alpha_{\gamma,h}(C_i)).$$

Adding the cost of the special parts S_{i_r} and S_{i_l}, we get the stated bound. □

Now the first bound of Lemma 10 immediately follows from the fact that $k_i k_{max} > O(\omega_\gamma(C_i) k_{max}/l(C_i))$ for dense C_i. The second bound also easily follows because $\sum k_{max} k_i \leq k_{max} l(P)$.