Optimal Robust Path Planning in General Environments*

T. C. Hu, Andrew B. Kahng and Gabriel Robins

Abstract

We address robust path planning for a mobile agent in a general environment by finding minimum cost source-destination paths having prescribed widths. The main result is a new approach which optimally solves the robust path planning problem using an efficient network flow formulation. Our algorithm represents a significant departure from conventional shortest-path or graph search based methods; it not only handles environments with solid polygonal obstacles but also generalizes to arbitrary cost maps which may arise in modeling incomplete or uncertain knowledge of the environment. Simple extensions allow us to address higher-dimensional problem instances and minimum-surface computations; the latter is a result of independent interest. We use an efficient implementation to exhibit optimal path-planning solutions for a variety of test problems. The paper concludes with open issues and directions for future work.

1 Introduction

To effect its goals, an autonomous mobile agent will execute a sequence of source-destination navigational tasks in the agent's configuration space. For each navigational task, the *motion planning problem* asks for a minimum-cost feasible path [8] [11] [25] [37]. The cost of a given solution may depend on many factors, including distance traveled, time or energy expended, and hazard probabilities encountered along the path. In practice, additional constraints might arise from a dynamic environment (e.g., moving obstacles), incomplete knowledge of the terrain and a concomitant need to reason under uncertainty, or such physical considerations as gravity, friction, and the dynamics of the robot agent. A large body of work has addressed the motion planning problem; for a survey, the reader is referred to Latombe [25] or Mitchell [31]. We begin our exposition by reviewing several major problem formulations and solution methods.

The basic motion planning problem involves kinematic path planning for a rigid-body mobile agent which moves in an environment populated by static polygonal obstacles. In this simple paradigm, the

*T. C. Hu is with the Computer Science and Engineering Department, University of California at San Diego, La Jolla, CA 92093-0114. Andrew B. Kahng is with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA 90024-1596. Gabriel Robins is with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442. Partial support for this work was provided by NSF MIP-9110696, NSF Young Investigator Award MIP-9257982, ARO DAAK-70-92-K-0001, ARO DAAL-03-92-G-0050, and an IBM Graduate Fellowship. Correspondence should be addressed to A. B. Kahng (kahng@cs.ucla.edu).

agent's path is defined by a sequence of translations and rotations of the rigid body. Motion planning then entails computing a path from a source to a destination which avoids collisions between the agent and the obstacles, i.e., the path must lie entirely in the so-called free space. This basic kinematic formulation has been augmented in several ways. For example, multiple agents may be present: this yields an environment with moving obstacles, and motivates the concept of *configuration space-time* [30]. The solution space may also be limited based on joint articulation, or the physical characteristics of control structures. Various classes of holonomic and non-holonomic kinematic constraints are discussed by Alexander [2], Alexander and Maddocks [3], Barraquand and Latombe [5] [6], Canny [8], Canny, Donald, Reif and Xavier [9], Laumond [26] [27], Li, Canny and Sastry [29], and Nelson [34]. An entirely distinct field of study incorporates robot dynamics to establish additional constraints on (kinematically planned) path solutions [25]. For example, gravity and friction coefficients will constrain both the path solution and the speed of traversal when the agent must avoid tip-over, sliding, and other undesirable phenomena. Note that our present work does not model the dynamics of the agent: rather, we address a kinematic path planning formulation which captures important practical constraints.

In general, kinematic path planning has been widely studied (see, e.g., [8] [25] [37] for surveys); a number of special cases have been well-solved, and several complexity bounds have been established. If the path solution is not constrained to lie on a grid, Canny [8] showed that path planning problems are NP-hard when the dimension of the configuration space exceeds three. In the same work, Canny also developed the exponential-time *roadmap* (or silhouette) method for general kinematic motion planning. Exact path planning algorithms for polygonal objects navigating among polygonal obstacles were given by Avnaim, Boissonnat and Faverjon [4]. Papadimitriou [36] proposed an algorithm that finds a path with cost within a factor of $1+\epsilon$ of optimal, and runs in time polynomial in both $\frac{1}{\epsilon}$ and the total number of vertices. Shortest path planning for a point moving on the boundary of a polyhedron has been studied in the computational geometry literature as the *discrete geodesic* problem (e.g., [33] and the related work in [32]). One readily observes that kinematic planning approaches fall into two main classes: efficient algorithms that have an underlying "shortest-path" approach, and exponential-time algorithms that rely on implicit enumeration techniques. Before presenting a more detailed taxonomy of kinematic planning solutions, we mention the motivating considerations which have led to our new combinatorial approach to kinematic planning.

Our work focuses on kinematic planning subject to two practical extensions: (i) the need to incorporate uncertainty into a motion planning formulation, and (ii) the requirement of an error-tolerant solution. These respectively yield the notion of a general cost function in a given terrain, along with the notion of a robust path solution (we use the term "robust path" to indicate a path that maintains a prescribed minimum width); Section 2 gives a formal development of these two concepts. While the complexity of kinematic planning generally increases as more constraints are added to the basic formulation, any strategy that addresses the basic formulation will usually generalize to handle the more complicated formulations; this is also true for the new approach given in this paper. Consequently, we find that approaches in the literature fall naturally into only a few main categories.

The most straightforward path planning algorithms are based on *graph search* approaches. Such methods use the vertices of a graph to represent feasible points in the agent's configuration space; edges in the graph represent optimum subpaths (also called *canonical* paths) between these configurations. Variations include retraction methods based on Voronoi decomposition [35], and approximate cell decomposition [30]. For simple problem formulations, e.g., a point agent and convex polygonal obstacles, computational geometry techniques along with shortest-path algorithms afford efficient solutions, as surveyed by Schwartz et al. in [37]. Obviously, the complexity of any variant is dependent on the number of paths computed and the ease of computing each optimum subpath. With a number of formulations, exhaustive graph search methods such as depth-first branch-and-bound or A* constitute the only known optimal algorithms. As noted by Latombe [25], such enumerative approaches typically have exponential time complexity, and therefore efficient heuristics must be employed.

A common heuristic for reducing search complexity is the hierarchical solution method based on 2^k-trees (esp. quadtrees and octtrees) or dissections into triangular or rectanguloid cells [10] [25]. Unfortunately, the hierarchical solution approach cannot guarantee optimal solutions [41]. Alternatively, physical analog methods might be employed; these commonly involve a *potential field* which forces the agent to move toward the goal configuration while avoiding contact with obstacles [7] [15] [24]. Path-finding may then be accomplished by variational methods (e.g., [38]), or by classical gradient or subgradient algorithms. Other work has essentially hybridized analogs of graph search algorithms (notably the best-first and depth-first approaches) with either hierarchical or physical analog formulations. The fundamental problem with such methods is that the output can be a locally minimum solution that is not globally optimum; no efficient variants of the physical analog approach that can guarantee a global optimum solution appear possible [25].

The main contribution of this paper is a new algorithmic approach to *optimal* robust path-planning in environments with arbitrary cost maps. Essentially, we discard the usual mix of shortest-path algorithms and graph search techniques, and instead employ a more general combinatorial approach involving network flows [12]. The crucial observation is that a minimum-cost *path* which *connects* two locations s and t corresponds to a minimum-cost *cut-set* which *separates* two other locations s' and t'. We obtain our new motion planning methodology by using efficient network flow algorithms to exploit this duality between connecting paths and separating sets.

Our approach is guaranteed to find *optimum* solutions to the minimum-cost path and minimum-cost robust path problem formulations which we define below. A primary attraction is that our algorithm runs in polynomial time; in fact, it can be implemented in $O(d^2 \cdot N^2 \cdot \log N)$ time where N is the number of nodes in a discrete mesh representation of the environment, and d is the prescribed path width. Experimental results confirm that we can efficiently find optimal robust paths where current combinatorial methods are prohibitively expensive, and where variational or gradient heuristics only return locally optimum solutions. Our new approach also has several other desirable features:

1. the intrinsic planarity and geometry of robot motion planning yields a layered, bounded-degree network representation, resulting in added implementation efficiency;
2. via incremental flow techniques, the method is suitable for dynamic or on-line problems, as well as situations where early knowledge of partial solutions is helpful (i.e., requiring an *anytime* algorithm [14]); and
3. the method generalizes to provide a solution to the classical Plateau problem on minimal surfaces [39], a result which is of independent interest [22] [23].

Given these advantages, we believe that our approach may well provide a new, practical basis for certain classes of autonomous agent path planning problems.

The remainder of this paper is organized as follows. In Section 2, we formally define the problem of robust motion planning in a terrain with arbitrary cost function. Section 2 also develops our solution of robust motion planning via network flows, and describes an efficient implementation. Section 3 gives experimental results showing optimal width-d paths in environments ranging from random cost maps to the more traditional class of maps with solid polygonal obstacles. We conclude in Section 4 with several directions for future research.

2 Robust Path Planning by Network Flows

We begin this section by establishing notation and terminology. Our development focuses on the connection-separation duality which motivates the network flow approach.

2.1 Problem Formulation

We say that a subset of the plane is *simply connected* if it is homeomorphic to a disk (i.e., contains no holes); a subset of the plane is *compact* if it is closed and bounded.

Definition: A *region* is a simply-connected, compact subset of $\Re^2$.

Given a region R, we know by the Jordan curve theorem [13] that the boundary B of R partitions the plane into three mutually disjoint sets: B itself; the interior of R; and the exterior of R. We consider the problem of computing a path in R from source S to destination T, where S and T are disjoint connected subsets of the boundary B. A *path* is defined as follows:

Definition: Given a region R with boundary B, a *path* between two disjoint connected subsets $S \subset B$ and $T \subset B$ is a non self-intersecting continuous curve $P \subseteq R$ which connects some point $s \in S$ to some point $t \in T$.

Clearly the path P partitions R into three mutually disjoint sets: (i) the set of points of R lying strictly on the left side of P, which we denote by R_l (we assume that P is oriented in the direction from s toward t); (ii) the set of points of R lying on the right side of P, denoted by R_r; and (iii) points of P itself. This is illustrated in Figure 1. It is possible for at most one of R_l and R_r to be empty, and this happens exactly when P contains a subset of B between S and T. More precisely, R_l (resp. R_r) is empty if $P \supseteq B_l$ (resp. $P \supseteq B_r$), where B_l and B_r respectively denote the subsets of the boundary B lying clockwise and counterclockwise between S and T, i.e., $B_l = (B \cap R_l) - (S \cup T)$ and $B_r = (B \cap R_r) - (S \cup T)$.

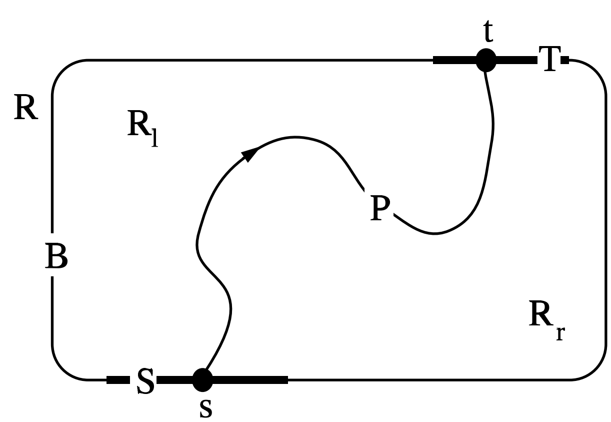

Figure 1: A path P between two points $s \in S$ and $t \in T$, where S and T are disjoint subsets of the boundary B of a region R.

As noted above, the goal of our work is to optimally solve motion planning when two practical constraints are incorporated: an *arbitrary* cost function defined over the region, and a *robust* path requirement. An arbitrary cost function corresponds to a *general environment*, which is in many ways more realistic than an environment consisting only of solid polygonal obstacles, and which moreover follows very naturally from the mobile agent having imperfect (probabilistic, fuzzy, or incomplete) knowledge of its surroundings. The imperfection of knowledge can be due to a variety of factors, including errors in positioning or orientation of the agent, staleness of information, and physical limitations of sensors. As an example, consider the following scenario: given probabilistic information about terrain difficulty (e.g., rivers, swamps, fields of landmines) and the locations of hostile entities, an agent wishes to travel with minimum probability of incurring a casualty or other damage. In this scenario, each point in the region will have an associated *weight*, or *cost* of traversal, corresponding to the level of danger. (In what follows, we use the terms "weight" and "cost" synonymously.) Multiple objectives may also be captured via this formulation: if the agent must reach his destination quickly, then the weight function will take into account both danger level and the time required to travel through each point. Note that this formulation subsumes the binary cost function of a basic environment with solid polygonal obstacles (cost $= 1$) and free space (cost $= 0$).

Formally, given a region R, we define a weight function $w : R \rightarrow \Re^{+}$ such that each point $s \in R$ has a corresponding positive weight $w(s)$. The *cost*, or *weight*, of a path $P \subseteq R$ is defined to be the integral of w over P. Optimal motion planning entails minimizing this path integral. To find a minimum-cost path $P \subseteq R$ between two points on the boundary of R, one might guess that Dijkstra's shortest path algorithm [12] provides a natural solution. However, application of Dijkstra's algorithm relies on an implicit assumption that the solution can be cast as an *ideal* path, i.e., a path of *zero* width. The relevance of this caveat becomes clear when we consider our second extension to the basic formulation - the requirement of a *robust* path solution.

Due to errors in location, orientation and motion, a robot agent is never able to exactly follow a given prescribed path. A path is error-tolerant if the agent will not come to harm when it makes a small deviation from the path: in the above scenario, a path which requires the agent to tiptoe precisely between landmines is not as error-tolerant as a path which avoids minefields altogether. With this in mind, we define a *robust* path to be one which maintains a *prescribed minimum width* (and which is thereby error-tolerant). This prescribed-width formulation also addresses the fact that robot agents are physical entities with physical dimensions: because physical agents do not occupy point locations in the environment, the minimum-cost path of zero width that is computed by a shortest-path algorithm may be infeasible (e.g., while an insect might easily exit a room through a barred window, a person cannot easily do so). In general, the optimum path for an agent of width d_1 cannot be obtained by simply widening or narrowing the optimum path for an agent of width d_2.

We now establish the relationship between a prescribed-width path requirement and the concept of *d-separation* [19]. In the following, we use $ball(x, d)$ to denote the closed ball of diameter d centered at x, i.e., the set of all points at distance $\frac{d}{2}$ or less from x. Our prescribed-width formulation, as well as the definition below of the robust path planning problem, assumes that an agent is of circular shape. If the agent has a different shape, e.g., an arbitrary polygon, then our new d-PATH algorithm (Section 2.2) assumes that the agent width is equal to the width of its smallest circumscribing disk.

Definition: Given two disjoint subsets S and T of the boundary of a region R, a set of points $\hat{P} \subseteq R$ is a *width-d path* between S and T if there exist $s \in S$, $t \in T$ and a path P connecting s to t such that $\hat{P} \supseteq \bigcup_{x \in P} \{ball(x, d) \cap R\}$, i.e., $\hat{P}$ contains the intersection of R with any disk of *diameter* d centered about a point of P.

Just as the path P between S and T will partition R into R_l, R_r, and P, the width-d path $\hat{P} \subseteq R$ between S and T also partitions R into three sets: (i) the set of points $\bar{R}_l = ((R - \hat{P}) \cap R_l) \cup B_l$, that is, the union of the left boundary B_l and all points in R that are to the left of $\hat{P}$; (ii) the set of points $\bar{R}_r = ((R - \hat{P}) \cap R_r) \cup B_r$; and (iii) the pointset $\hat{P}$ itself. We now obtain the definition of a d-separating path (see Figure 2):

Definition: Given two disjoint subsets S and T of the boundary of a region R, a set of points $\bar{P} \subseteq R$ is a *d-separating path* between S and T if $\bar{P}$ is a width-d path such that any point of $\bar{R}_l$ is distance d or more away from any point of $\bar{R}_r$.

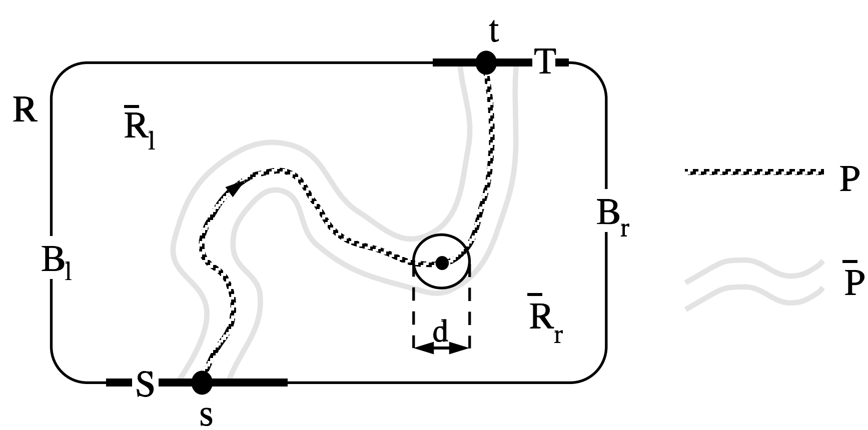

Figure 2: A d-separating path $\bar{P}$ of width d between two points $s \in S$ and $t \in T$ of the boundary of a region R. Here $\bar{R}_l$ is separated from $\bar{R}_r$ by a distance of d.

A d-separating path $\bar{P}$ between S and T is a *minimal d-separating path* between S and T if no subset of $\bar{P}$ satisfies the preceding definition. Because all points in R have positive cost and because we are interested in minimum-cost paths, the following discussion refers only to minimal d-separating paths. Given a specific width d, the d-robust motion planning problem can now be stated as follows:

d-Robust Motion Planning Problem (RMPP): Given an arbitrary region R with boundary B, weight function $w : R \rightarrow \Re^{+}$, a source $S \subseteq B$, a destination $T \subseteq B$, and a width d, find a d-separating path $\bar{P} \subseteq R$ between S and T which has minimum total weight.

While our formulation specifies an arbitrary weight function that is integrated to yield path cost, in most practical situations the region is discretized relative to a given fixed grid or sampling granularity (see, e.g., [7]). Thus, in the present work, we will assume a fixed-grid representation $\tilde{R}$ of the environment region R. With this discrete RMPP formulation, the cost of a path is defined to be the sum of the weights of the nodes covered by the path. Similarly, the notion of d-separation also naturally extends to the discrete grid:

Definition: Given a region R, a *discrete* d-separating path $\tilde{P}$ in the gridded region $\tilde{R}$ is the subset of the gridpoints of $\tilde{R}$ that is contained in some d-separating path $\bar{P}$ in R (Figure 3).

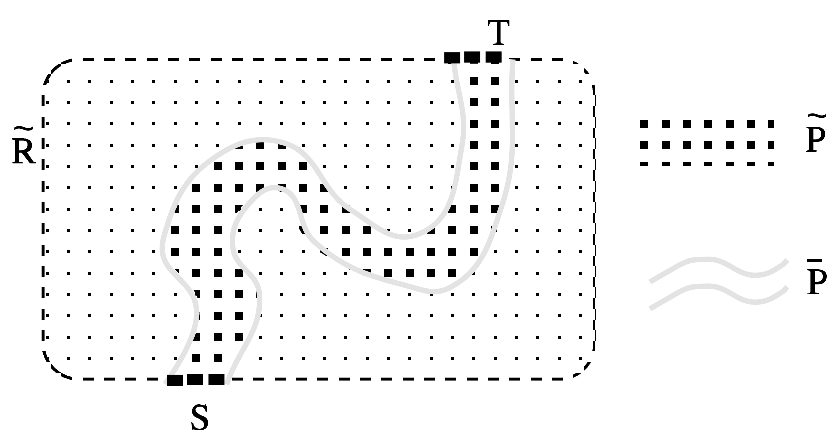

Figure 3: A discretized representation $\tilde{R}$ of a region R, and a discrete d-separating path $\tilde{P}$ in $\tilde{R}$. Note that $\tilde{P}$ is the set of lattice points covered by the continuous d-separating path $\bar{P}$ in R.

As before, a discrete d-separating path is *minimal* if no subset of it satisfies this definition. Analogously to the continuous case, a d-separating path partitions the gridded environment into two subsets, such that each gridpoint from one partition is a distance of at least d units away from any gridpoint in the other partition. We therefore have the following problem formulation:

d-Robust Motion Planning Problem in a Grid (RMPPG): Given a weighted gridded region $\tilde{R}$ with boundary $B \subset \tilde{R}$, a source $S \subseteq B$, a destination $T \subseteq B$, and a width d, find a discrete d-separating path $\tilde{P} \subseteq \tilde{R}$ between S and T which has minimum cost.

Intuitively, as the granularity quantum approaches zero the solution for the RMPPG instance will converge to the solution for the corresponding continuous RMPP instance. At this point, we observe that although RMPPG is a very natural problem formulation, it cannot be efficiently solved by traditional methods. In particular, efficient graph algorithms such as Dijkstra's shortest-path algorithm fail because the optimal d-separating path may self-intersect: when the shortest-path algorithm attempts to increment a path, it cannot determine how much of the increment should be added to the path cost (see Figure 4).[1]

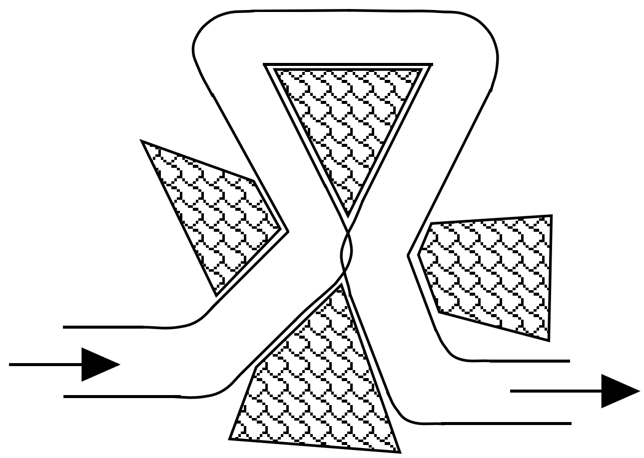

Figure 4: An optimal path with non-zero width can intersect itself (in this example the center gap is too narrow to allow the path to pass through it); efficient shortest-path algorithms fail to solve the robust motion planning problem for such instances.

In the special case where the cost function is binary, Dijkstra's algorithm is applicable via the following well-known technique [25]: augment the environment by growing each obstacle (as well as the region boundary) isotropically by $\frac{d}{2}$ units, then set the weight of each node in the free area to some constant, while the weight of any node in an area covered by an obstacle is set to infinity. A minimum-cost prescribed-width path in such an augmented environment would correspond to the center P of the d-separating path $\bar{P}$ that we seek (Figure 2). Unfortunately, this simple transformation fails for arbitrary weight functions: a general environment has no solid "obstacles" which can be "grown" in such a fashion.

[1] Recall that in an n-node edge-weighted graph $G = (V, E)$ with identified source $v_0 \in V$, the k^{th} phase of Dijkstra's algorithm, $k = 1, \ldots, n$, finds another node v_k for which the shortest pathlength d_{0k} in G is known; we know the optimum s-t pathlength when $v_k = t$. Although the RMPPG formulation above assumes a node-weighted G, we may easily obtain an edge-weighted graph (for all $v \in V$, add $\frac{w(v)}{2}$ to the weight of each edge incident to v) to which we may apply Dijkstra's algorithm. However, this transformation is correct only for computing the optimal zero-width path: Dijkstra's algorithm relies on the fact that d_{ij} can never be strictly less than $min_k(d_{ik} + d_{kj})$, but this may not hold when paths have non-zero width, as shown in Figure 4.

In fact, even in the case of a binary cost function, Dijkstra's algorithm may not be entirely suitable. Recall that our path cost model charges only once for each node covered by the path, reflecting the original motivation of minimizing the "path integral" of cost. Arguably, this is not unreasonable; for example, the cost of traversing a minefield may be very large, but once the mined area is cleared, the hazard associated with a second visit is zero. Similarly, when a path is plowed through snow or when a road is paved, the work is proportional to the actual area plowed or paved. Given this path cost formulation, Dijkstra's algorithm cannot be applied since it requires *fixed* edge costs. Figure 5 illustrates this, where for the path on the left it is appropriate to let the cost of the grid edge (y, z) be $\hat{w}(Z - Z \cap Y)$, where Y and Z are the sets of gridpoints in $ball(y, d)$ and $ball(z, d)$, respectively, and the weight function w is extended to sets of nodes, e.g., $\hat{w}(Z) = \sum_{z \in Z} w(z)$. However, this cost definition can charge more than once for a single visited region depending on where the path previously visited (Figure 5, right), and therefore edge costs cannot be fixed *a priori*.

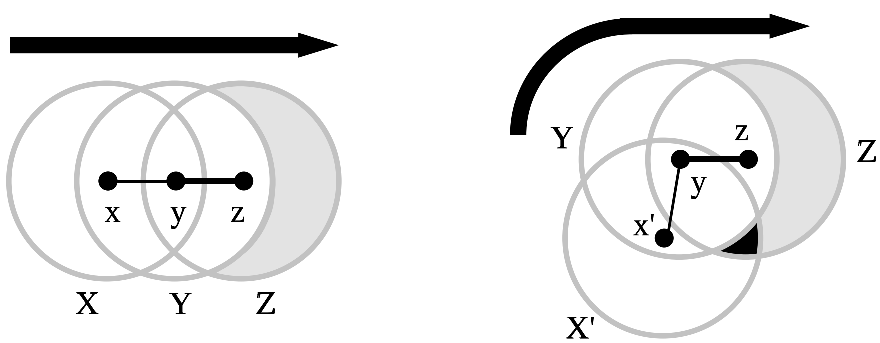

Figure 5: Dijkstra's algorithm fails for the general robust motion planning problem because the cost of an edge $e = (y, z)$ may depend on *many* previous nodes (e.g., x') on the path: in this example (shown on the right), the cost of incrementing the path by adding the edge e should be $\hat{w}(Z - Z \cap (Y \cup X'))$, rather than $\hat{w}(Z - Z \cap Y)$, otherwise the cost of the small dark region will be charged twice.

To summarize, we see that the traditional Dijkstra-based obstacle-growing method fails to solve the robust motion planning problem for two reasons: (i) the the obstacle-growing transformation does not apply to general cost functions, and (ii) fixed edge costs cannot be determined when the cost model charges exactly once for each node covered by the path. With this in mind, we now use a network flow approach to develop an efficient, optimal algorithm for the RMPPG problem.

2.2 A Network Flow Based Approach

To solve the robust motion planning problem in a grid, we use ideas from network flows in continua [21].[2] For completeness of our exposition, we first review several key concepts from the theory of network flows [16] [28]. A *flow network* $\eta = (N, A, s, t, c, c')$ is a directed graph with node set N; a set of directed arcs $A \subseteq N \times N$; a distinguished *source* node $s \in N$ and a distinguished *sink* node $t \in N$; an *arc capacity* function $c : A \to \Re^+$ which specifies the capacity $c_{ij} \geq 0$ of each arc $a_{ij} \in A$; and a *node capacity* function $c' : N \to \Re^+$ which specifies the capacity $c'_i \geq 0$ of each node $n_i \in N$. To handle undirected graphs, we may replace each undirected arc a_{ij} by two directed arcs a_{ij} and a_{ji}, each having capacity c_{ij}.

A *flow* in η assigns to each arc a_{ij} a value ϕ_{ij} with the constraint that $0 \leq \phi_{ij} \leq c_{ij}$. An arc a_{ij} is called *saturated* if $\phi_{ij} = c_{ij}$. We insist on *flow conservation* at every node except s and t, and we require that the flow through each node n_j does not exceed the capacity of that node:

$$\sum_i \phi_{ij} \ = \ \sum_k \phi_{jk} \ \leq \ c'_j \qquad n_j \neq s, t$$

A node n_j is called *saturated* if $\sum_i \phi_{ij} = c'_j$. Since flow is conserved at every node, the total amount of flow from the source must be equal to the total flow into the sink; we call this quantity the *value* Φ of the flow:

$$\Phi = \sum_i \phi_{si} \ = \ \sum_j \phi_{jt}$$

A flow with the maximum possible value is called a *maximum flow.* An *s-t cut* in a network is a set (N', A') of nodes $N' \subseteq N$ and arcs $A' \subseteq A$ such that every path from s to t uses at least one node of N' or at least one arc of A'. The *capacity* $c(N', A')$ of a cut is the sum of the capacities of all nodes and arcs in the cut. A classical result of linear programming duality states that the maximum flow value is equal to the minimum cut capacity; this is the *max-flow min-cut* theorem [16]:

Theorem 1: Given a network $\eta = (N, A, s, t, c, c')$, the value of a maximum *s-t* flow is equal to the minimum capacity of any *s-t* cut. Moreover, the nodes and arcs of any minimum *s-t* cut are a subset of the saturated nodes and saturated arcs in some maximum *s-t* flow. □

Recall our earlier observation that any *s-t* path will separate, i.e., cut, R_l from R_r. In particular, an inexpensive *s-t* path will correspond to an inexpensive cut between two appropriately chosen nodes s'

[2] We note that Mitchell [32] also extends the ideas of flows in continua, but in a very different way. The results in [32] develop a theory of flows in polyhedral domains, with a view to such practical applications as motion planning of *many* agents through a congested region (i.e., "rapid deployment"), the routing of multiple wires in a printed circuit board, etc.

and t'. Since a subset of the nodes and arcs saturated by the maximum s'-t' flow will yield this s'-t' cut, it is natural for us to derive the desired s-t path via a maximum-flow computation in a network where capacities correspond to travel costs in R. The remainder of this section describes how we accomplish this.

To transform robust motion planning in a region R into an instance of network flow, we first superimpose a mesh network topology over R, then assign node weights in this network according to the weighting function $w : R \rightarrow \Re^+$. This yields a representation that corresponds to the underlying RMPPG instance.

We guarantee a *robust* path solution by ensuring that any separating node set in the mesh topology satisfies the prescribed width-d requirement. Toward this end, we define the *d-neighborhood* of a node v in the mesh to be the set of all nodes at distance d or less units away from v, and we then modify the mesh topology by uniformly connecting each node to all other nodes in its d-neighborhood, where d is the prescribed path width. The resulting network is called a *d-connected* mesh, and has the property that no nodeset of width less than d is a d-separating set. An illustration of this construction for $d = 2$ is given in Figure 6. We note that the concept of a d-neighborhood was first investigated by Gomory and Hu [21].

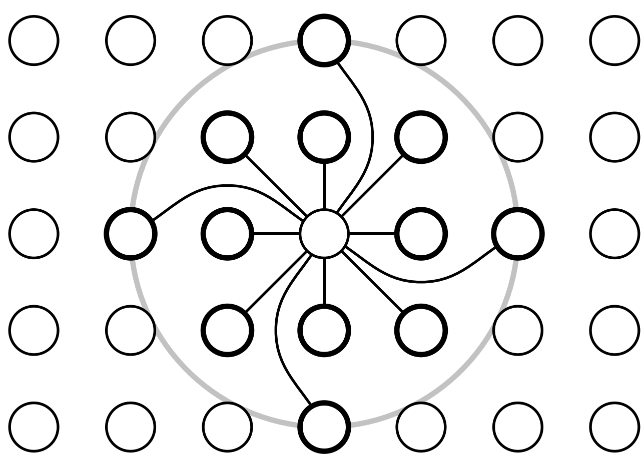

Figure 6: A node and its d-neighborhood ($d = 2$).

Finally, we choose nodes s' and t' such that the minimum s'-t' cut is forced to lie along some path between s and t. We accomplish this by making s' and t' respectively into a source and a sink, then connecting each to a contiguous set of nodes corresponding to part of the boundary of the original region R. This completes the transformation; Figure 7 gives a high-level illustration of the construction.

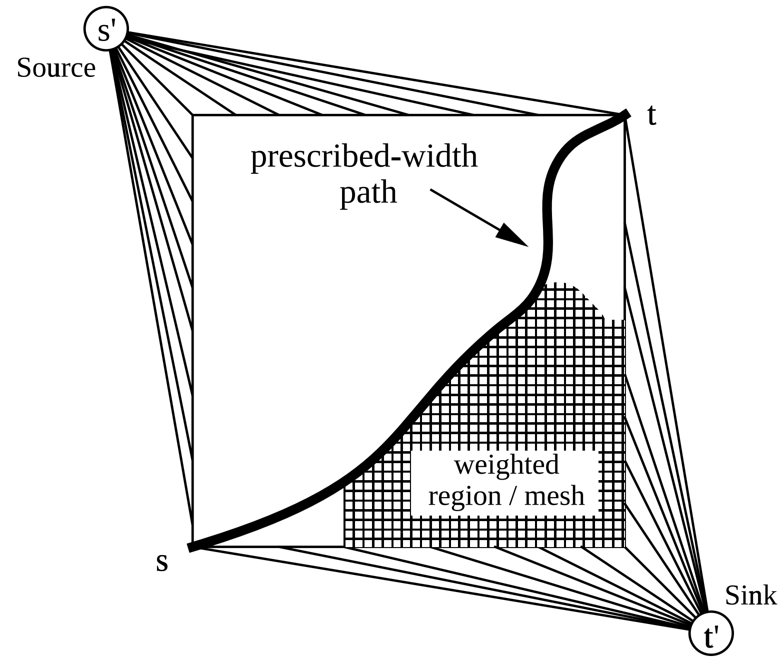

Figure 7: A robust motion planning instance transformed into a network flow instance.

Observe that up to this point, we have converted a motion planning instance into an undirected, *node-capacitated* (node-weighted) flow instance. However, network flow algorithms typically assume that the input is an *arc-capacitated* network (with infinite node capacities). Therefore, in order to use a standard maximum flow algorithm on our network, we must transform an instance having both node and arc capacities into an equivalent arc-capacitated maximum flow instance. To accomplish this, we use the standard device of splitting each node $v \in N$ with weight $w(v)$ into two unweighted nodes v' and v'', then introducing a directed arc from v' to v'' with capacity $w(v)$. Also, each arc $(u, v) \in A$ of the original network is transformed into two infinite-capacity directed arcs (u'', v') and (v'', u'). Thus, each arc (v', v'') of the resulting directed network will, when saturated, contribute the original node weight $w(v)$ to the minimum cut value. This transformation is illustrated in Figure 8 [20]. The overall size of the network increases by only a constant factor via this last transformation, i.e., the final directed arc-capacitated network will have only $2|N|$ nodes and $|N| + 2|A|$ arcs. Therefore, a maximum flow computation in the transformed network will be asymptotically as fast as in the original network.

Note that a maximum flow in the arc-capacitated transformed graph corresponds to a minimum arc-cut in the transformed graph (by the max-flow min-cut theorem), which in turn corresponds to a minimum node-cut in the original graph since the transformation preserves minimal cutset costs. Moreover, the

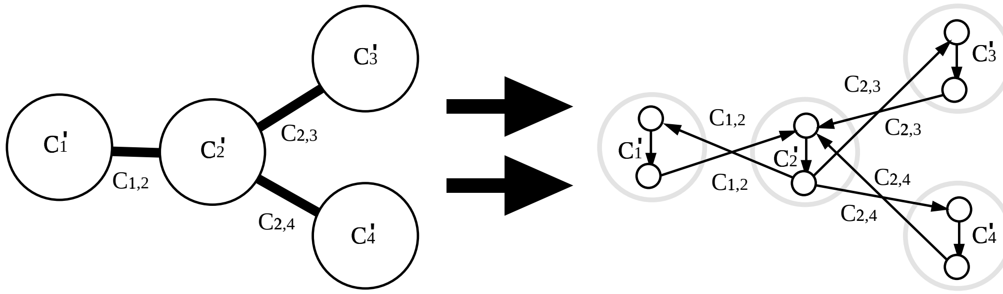

Figure 8: Transformation of a node- and arc-capacitated flow network to an arc-capacitated flow network: arc capacities c_{ij} remain infinite, while original node capacities (node weights) c'_i induce directed arc capacities in the transformed network.

"width" of the cut can be no less than d since, as discussed above, the connection of each node to all nodes in its d-neighborhood guarantees that any separating node set will have the prescribed width. A formal summary of our algorithm, which we call d-PATH, is given in Figure 9.

d-**PATH:** Finding a d-robust path in a weighted region
Input: Region R, weight function $w : R \to \Re^+$, width d, grid size g, source s and destination t on boundary of R **Output:** A discrete d-separating path $\bar{P} \subseteq R$ connecting s and t
Create a d-connected mesh topology η of size $g \times g$ over R with all arc capacities set to ∞ **Assign** node weights (capacities) in η according to weight function w **Set** boundary node weights (capacities) to ∞ **Transform** node/arc-capacitated network η into arc-capacitated network η' **Add** source node s' and sink node t' to η' **Connect** s' to B_l, the boundary nodes of R, clockwise from s to t **Connect** t' to B_r, the boundary nodes of R, clockwise from t to s **Set** capacities of all arcs adjacent to s' or t' to ∞ **Compute** maximum s'-t' flow in η' **Output** all nodes incident to arcs in the minimum s'-t' cut of η'

Figure 9: Finding a d-separating path of minimum cost in an arbitrary weighted region, i.e., an optimal solution to the RMPPG problem.

We conclude this section with the observation that the max-flow min-cut theorem [16] and the existence of efficient algorithms for maximum flow (e.g., [12] [16]) together imply the following:

Theorem 2: The d-PATH method of Figure 9 outputs an optimal solution to the RMPPG problem in time polynomial in size of the mesh representation of the region R. □

2.3 A Test Implementation

There are numerous algorithms for computing maximum flows in networks [1] [16] [21]. To demonstrate the viability of our approach, we have used an existing implementation of Dinic's network flow algorithm [18]. Starting with an empty flow, the Dinic algorithm iteratively augments the flow in stages; the optimal flow solution is achieved when no flow augmentation is possible. Each stage starts with the existing flow, and attempts to "push" as much flow as possible along shortest paths from the source to the sink in a residue network wherein each arc has capacity equal to the difference between its original capacity and its current flow value. After the current flow has been thus augmented, newly saturated arcs are removed and the process iterates. Since there can be at most $|N|-1$ such stages, each requiring time at most $O(|A| \cdot |N|)$, the total time complexity of the Dinic algorithm is $O(|A| \cdot |N|^2)$.

If we have a total of $|N|$ nodes in our mesh graph, the time complexity of the Dinic algorithm is $O(|N|^3)$. In practice, more efficient flow algorithms are available [1]. For example, by using the network flow algorithm of [17], we obtain the following:

Theorem 3: For a given prescribed path width d, the d-PATH method solves the RMPPG problem in $O(d^2 \cdot |N|^2 \cdot \log \frac{|N|}{d^2})$ time, where $|N|$ is the number of nodes in the mesh representation of the region R.

Proof: Each node in the mesh induced by the method has no more than d^2 adjacent arcs, so that $|A| = O(d^2 \cdot |N|)$. The network flow algorithm of [17] operates within time $O(|A| \cdot |N| \cdot \log(\frac{|N|^2}{|A|}))$. The overall time complexity of d-PATH is therefore $O(d^2 \cdot |N|^2 \cdot \log \frac{|N|}{d^2})$. $\square$

If d is a constant (i.e., the agent size is much smaller than the environment size), then the time complexity of d-PATH becomes $O(|N|^2 \cdot \log |N|)$ for any fixed d. The time complexity may be further reduced in cases where the terrain cost function may only take on values from a fixed, bounded range. In this case, we may apply the maximum flow algorithm of [1] to obtain an overall time complexity of $O(|N|^2)$ for the d-PATH algorithm.

3 Simulation Results

Our current d-PATH implementation uses ANSI C code to transform an arbitrary robust motion planning instance into a maximum-flow instance; we then use the Fortran-77 Dinic code of [18] to compute the flow, and invoke Mathematica [40] to draw the resulting path. We have tested our implementation on three classes of motion planning instances: uniformly weighted regions, environments with polygonal obstacles, and smooth randomly-costed terrains. For each of these input classes, the boundary of the region is a rectangle, and we look for a width-d path connecting s and t which are respectively in the top left and bottom right corners of the region. With each instance, we tested various values of d.

A uniformly weighted region has all node weights equal to the same constant. In such an instance we expect the solution path to resemble a straight line between s and t, with the straightness of the line improving as the mesh resolution and the width d both increase. Experimental results confirm this behavior.

Our test environments with polygonal obstacles are populated by polygons of random sizes, located throughout the region. Nodes in the clear areas are uniformly assigned a small constant weight, while nodes inside the obstacles have infinite weight. In such an environment, changing the prescribed width d may dramatically affect the optimum path topology with respect to the obstacles, since long detours may be required in order to avoid narrow passages between objects. This phenomenon was indeed confirmed by our experiments, as illustrated in Figures 10 and 11.

Finally, we tested our methodology on randomly-costed terrains, using a mesh resolution of 100 by 100 nodes and a range of d values. Each random terrain instance was generated as follows. All nodes in the mesh were initially assigned a weight of zero, except for a small random subset of the nodes which were each given a large random positive weight. Then, a weight redistribution step was iteratively used to increment each node's weight by a small random fraction of the total weight of its immediate neighbors until a smooth randomly-costed terrain was obtained. Figures 12 and 13 depict typical d-PATH output for the RMPPG problem in a random terrain. Regions of greater weight are denoted by darker shades, and regions of smaller weight are depicted by lighter densities. The optimum width-d path is highlighted in black.

We emphasize that even though the Dinic algorithm is not ideal for a mesh topology, typical running times used to generate and solve all of the above classes of instances are on the order of at most a few minutes on a SUN Sparc IPC (15.7 MIPS). We therefore conclude that our approach constitutes a viable new method for optimal robust motion planning in general terrains.

4 Conclusions and Open Problems

We have developed a polynomial-time algorithm which gives optimal solutions to the robust motion planning problem in a discretized environment. Our method is based on the duality between connecting paths and separating sets, and relies on a maximum-flow computation to find a minimum-cost path of prescribed width in an arbitrarily weighted region. The accuracy of the solution with respect to the

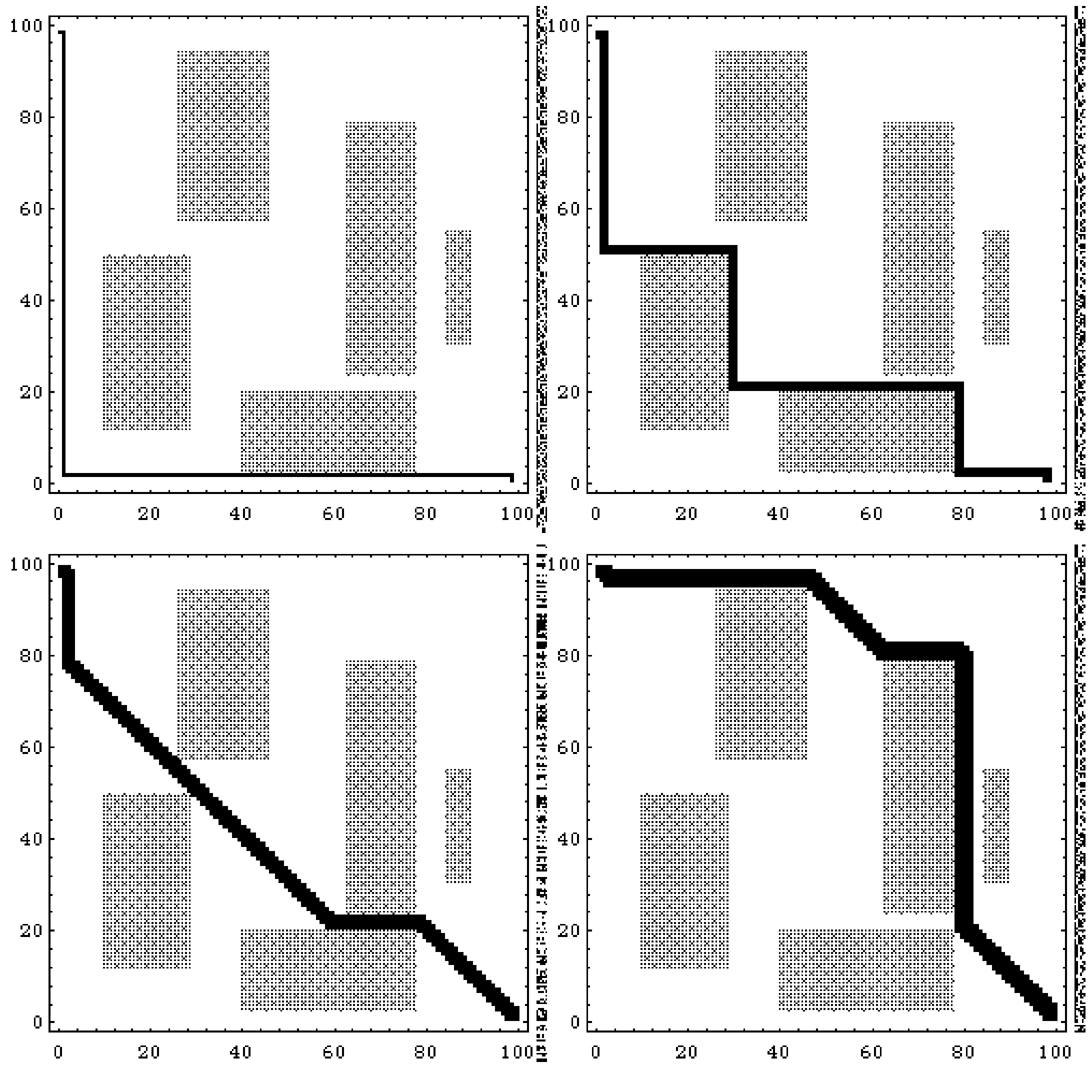

Figure 10: Robust motion planning in a region with polygonal obstacles. Note that the topology of the solutions changes as the prescribed width d is increased. The solutions shown correspond to widths $d = 1$ (top left), $d = 2$ (top right), $d = 3$ (bottom left), and $d = 4$ (bottom right).

continuous version of the problem depends on the gridsize, which is intrinsic to the input. It should be noted that the underlying network topology does not necessarily have to be a mesh. In particular, our d-PATH approach easily applies to instances where the underlying grid resolution varies across the region, with more crucial areas of the region being represented more finely, and less important areas having coarser resolution in order to increase efficiency. Our method enjoys a number of additional generalizations and applications, including a higher-dimensional extension which solves a class of discrete instances of the Plateau problem on minimal surfaces [22] [23].

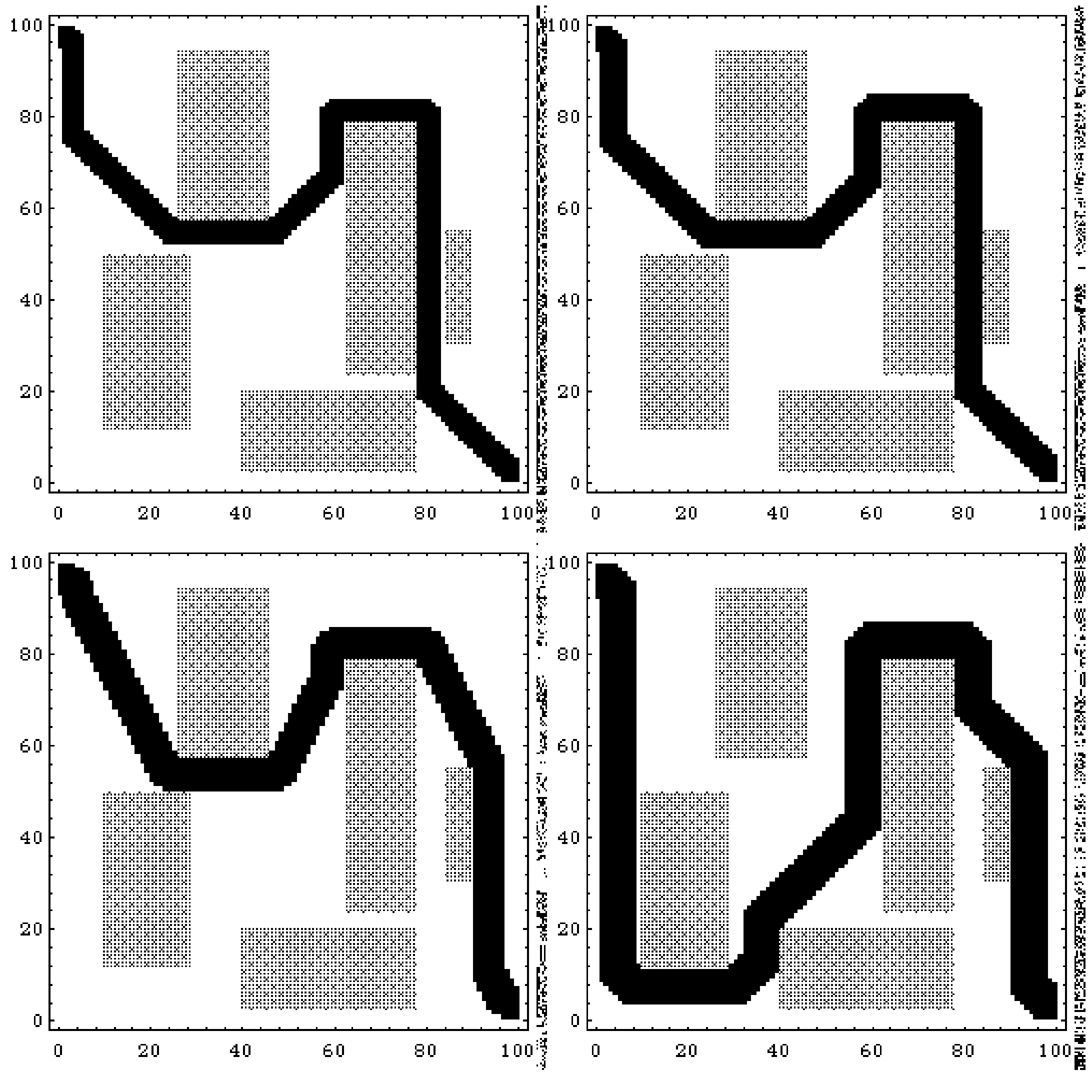

Figure 11: Robust motion planning in a region with polygonal obstacles (continued). The solutions shown correspond to widths $d = 5$ (top left), $d = 6$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

Chief among the future research goals is improvement of the time complexity of the network flow computation; substantial improvement is likely since the mesh is a highly regular and symmetric network that admits a concise representation. Additional research might also address more general path planning issues, such as (i) incorporation of kinematic and dynamic considerations, (ii) use of hierarchical approaches as a heuristic speedup, and (iii) addressing the case where the endpoints of the path are not necessarily on the boundary of the region.

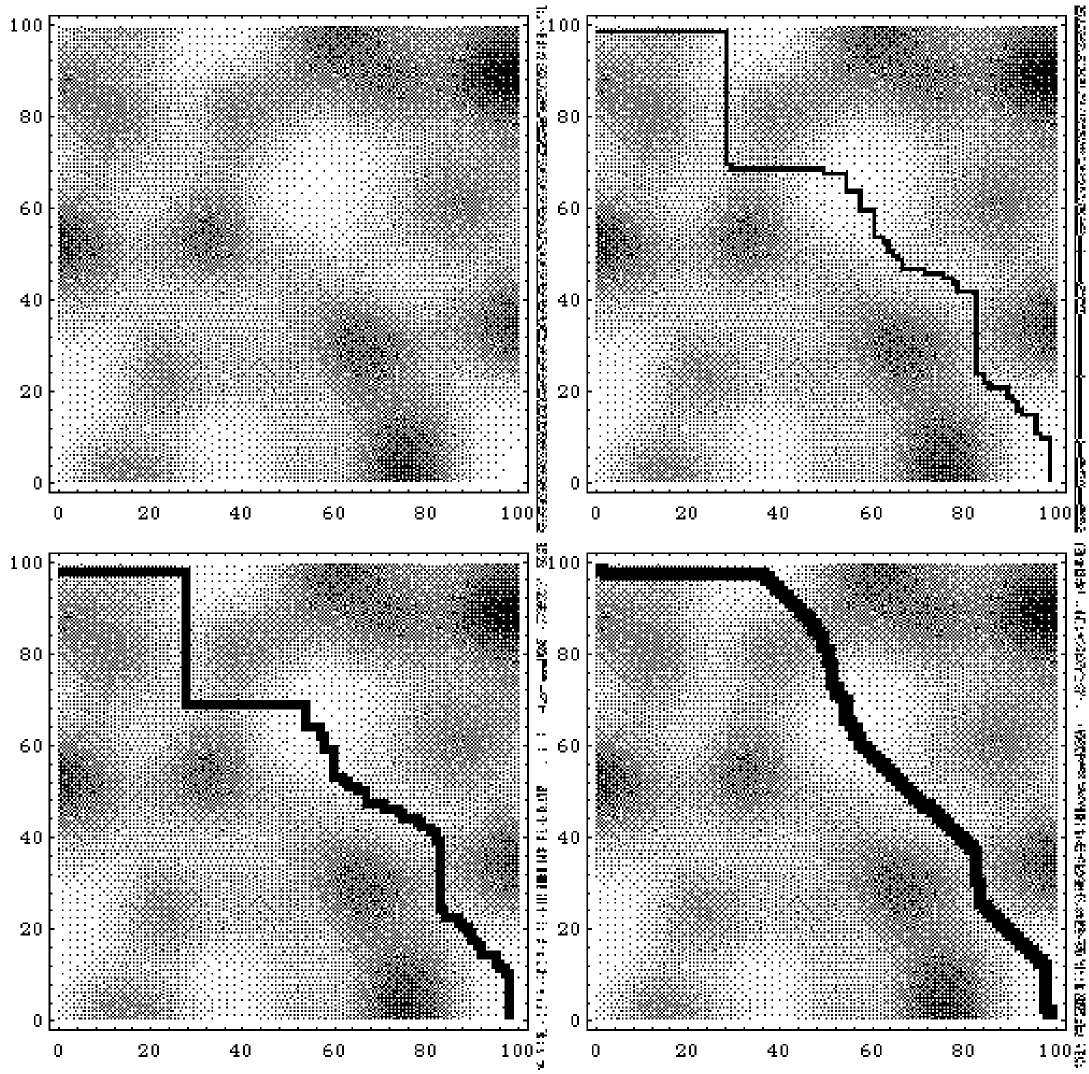

Figure 12: A randomly generated smooth terrain and its robust motion planning solutions. We see the terrain itself (top left), as well as solutions corresponding to widths $d = 1$ (top right), $d = 2$ (bottom left), and $d = 3$ (bottom right).

5 Acknowledgements

We are grateful to Professors Sheila Greibach and John Canny as well as to the anonymous referees for their helpful comments.

References

[1] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan, *Improved time bounds for the maximum flow problem*, Tech. Rep. CS-TR-118-87, Dept. of Computer Science, Princeton University, 1987.

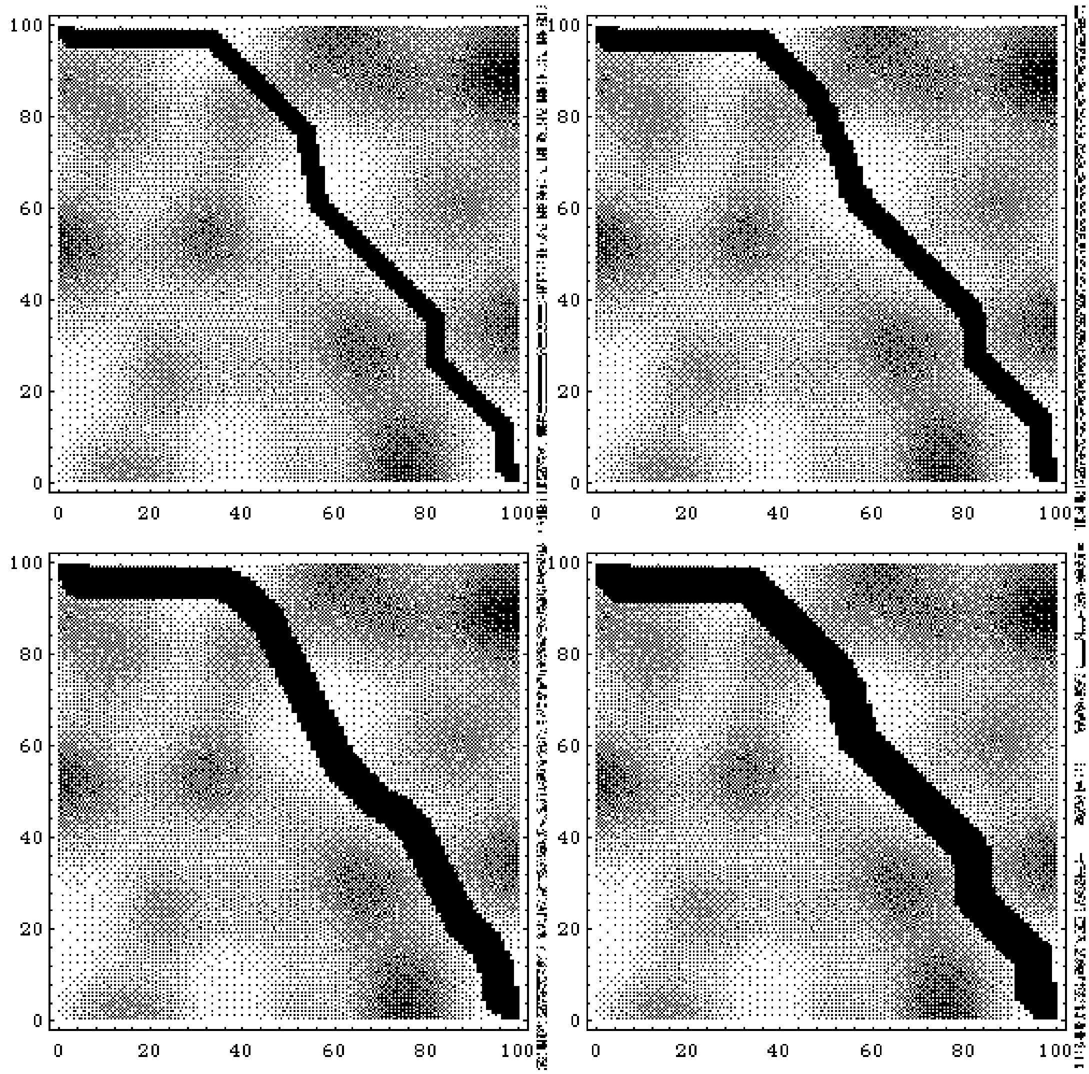

Figure 13: A randomly generated smooth terrain and its optimal robust motion planning solutions (continued). Here we see the solutions corresponding to widths $d = 4$ (top left), $d = 5$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

[2] J. C. Alexander, *On the motion of a trailer-truck, problem 84-12*, SIAM Review, (1985), pp. 578–579.

[3] J. C. Alexander and J. H. Maddocks, *On the maneuvering of vehicles*, SIAM J. Applied Math., 48 (1988), pp. 38–51.

[4] F. Avnaim, J. D. Boissonnat, and B. Faverjon, *A practical exact motion planning algorithm for polygonal objects moving amidst polygonal obstacles*, Tech. Rep. No. 890, INRIA, Sophia-Antipolis, France, 1988.

[5] J. Barraquand and J. C. Latombe, *On non-holonomic mobile robots and optimal maneuvering*, Revue d'Intelligence Artificielle, 3 (1989), pp. 77–103.

[6] ——, *Conrollability of mobile robots with kinematic constraints*, Tech. Rep. No. STAN-CS-90-1317, Department of Computer Science, Stanford University, Stanford, CA, 1990.

[7] ——, *Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles*, Proc. IEEE Intl. Conf. on Robotics and Automation, (1991), pp. 2328–2335.

[8] J. Canny, *The Complexity of Robot Motion Planning*, MIT Press, 1988.

[9] J. Canny, B. Donald, J. Reif, and P. Xavier, *On the complexity of kinodynamic planning*, in Proc. IEEE Symp. on Foundations of Computer Science, 1988, pp. 306–318.

[10] D. W. Cho, *Certainty grid representation for robot navigation by a bayesian method*, Robotica, 8 (1990), pp. 159–165.

[11] J. H. Connell, *Minimalist Mobile Robotics*, Academic Press, 1990.

[12] T. H. Cormen, C. E. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[13] Courant and Robbins, *What is Mathematics? An Elementary Approach to Ideas and Methods*, Oxford University Press, London, England, 1941.

[14] T. Dean and M. Boddy, *An analysis of time-dependent planning*, in Proc. AAAI, St. Paul, 1988, pp. 49–54.

[15] B. R. Donald, *Error Detection and Recovery for Robot Motion Planning with Uncertainty*, Ph.D. dissertation, Dept of EE&CS, MIT, 1987.

[16] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1961.

[17] A. V. Goldberg, E. Tardos, and R. E. Tarjan, *Network flow algorithms.* manuscript, March 1989.

[18] D. Goldfarb and M. D. Grigoriadis, *A computational comparison of the dinic and network simplex methods for maximum flow*, Annals of Operation Research, 13 (1988), pp. 83–123.

[19] R. E. Gomory, T. C. Hu, and J. M. Yohe, *r-separating sets*, Can. J. Math., XXVI (1974), pp. 1418–1429.

[20] D. Gusfield and D. Naor, *Efficient algorithms for generalized cut trees*, in Proc. ACM/SIAM Simposium on Discrete Algorithms, 1990, pp. 422–433.

[21] T. C. Hu, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.

[22] T. C. Hu, A. B. Kahng, and G. Robins, *Optimal solution of the discrete plateau problem*, Tech. Rep. CSD-920006, Computer Science Department, UCLA, January 1992.

[23] ——, *Solution of the discrete plateau problem*, Proc. of the National Academy of Sciences, 89 (1992), pp. 9235–9236.

[24] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Intl. J. of Robotics Research, 5 (1986), pp. 90–98.

[25] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

[26] J. P. Laumond, *Feasible trajectories for mobile robots with kinematic and environment constraints*, Preprints of the Intl. Conf. on Intelligent Systems, (1986), pp. 346–354.

[27] ——, *Finding collision-free smooth trajectories for a non-holonomic mobile robot*, in Proc. IEEE Conf. on Foundations of Computer Science, Milan, Italy, 1987, pp. 1120–1123.

[28] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt Rinehart and Winston, New York, 1976.

[29] Z. Li, J. F. Canny, and S. S. Sastry, *On motion planning for dexrerous manipulation, part i: The problem formulation*, in Proc. IEEE Intl. Conf. on Robotics and Automation, Scottsdale, AZ, 1989, pp. 775–780.

[30] T. Lozano-Perez, *Spatial planning: A configuration space approach*, IEEE Trans. on Computers, C-32 (1983), pp. 108–120.

[31] J. S. B. Mitchell, *An Algorithmic Approach to Some Problems in Terrain Navigation, in Geometric Reasoning, D. Kapur and J. L. Mundy, editors*, MIT Press, 1988.

[32] ——, *On maximum flows in polyhedral domains*, Journal of Computer and System Sciences, 40 (1990), pp. 88–123.

[33] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, *The discrete geodesic problem*, SIAM J. Computing, 16 (1987), pp. 647–668.

[34] W. Nelson, *Continuous-curvature paths for autonomous vehicles*, in Proc. IEEE Intl. Conf. on Robotics and Automation, April 1989, pp. 1260–1264.

[35] C. O'Dunlaing and C. K. Yap, *A retraction method for planning the motion of a disc*, J. Algorithms, 6 (1983), pp. 187–192.

[36] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, 1982.

[37] J. T. Schwartz, M. Sharir, and H. Hopcroft, *Planning, Geometry and Complexity of Robot Motion*, Ablex Publishing Corp., 1987.

[38] S. H. Suh and K. G. Shin, *A variational dynamic programming approach to robot-path planning with a distance-safety criterion*, IEEE Trans. on Robotics and Automation, 4 (1988), pp. 334–349.

[39] T. Tsuchiya, *On two methods for approximating minimal surfaces in parametric form*, Mathematics of Computation, 46 (1986), pp. 517–529.

[40] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer, Second Edition*, Addison-Wesley, Reading, MA, 1991.

[41] D. Zhu and J. C. Latombe, *New heuristic algorithms for efficient hierarchical path planning*, IEEE Trans. on Robotics and Automation, 7 (1991), pp. 9–20.