2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

Shortest Distance Problems in Graphs Using History-Dependent Transition Costs with Application to Kinodynamic Path Planning

Raghvendra V. Cowlagi and Panagiotis Tsiotras

Abstract— **A new algorithm is presented to compute the shortest path on a graph when the node transition costs depend on the prior history of the path to the current node. The algorithm is applied to solve path planning problems with curvature constraints.**

I. INTRODUCTION

The problem of planning a path for an autonomous vehicle in a given environment, while avoiding obstacles, has been studied for several years [1], [2], [3], [4]. Detailed surveys of path planning and motion planning algorithms are provided, for example, by Latombe [5], Hwang and Ahuja [6], and more recently, by LaValle [7]. In its most generality, the problem of path planning is to find a path from a given initial point to a given destination in the environment such that it does not intersect any obstacles, and such that the resulting path can be followed by the vehicle. A time parameterization along the path leads to a *trajectory* that must be followed by the vehicle. During the last step, the trajectory needs to obey the constraints imposed by the associated vehicle dynamics, while also minimizing a certain cost function, e.g. time of travel. The computation of a suitable trajectory from the starting point to the goal destination is called the motion planning problem.

Apart from the obvious lack of optimality, the approach of addressing separately the geometric and dynamic parts of the problem may also lead to dynamically infeasible paths; the reason being that the geometric path planner has no prior knowledge of the dynamic limitations of the vehicle. If we want to ensure that the overall scheme will always generate feasible paths, we have to bridge the gap between the geometric and dynamic layers. This can only be achieved if certain information about the dynamic envelope of the vehicle is passed to the path planner.

In this paper we propose a new scheme to include information about the class of dynamically feasible paths early on (viz at the geometric layer). Furthermore, we do this in an numerically efficient manner that is based on a non-trivial modification of Dijkstra's algorithm for the solution of shortest-path problems on graphs. In that sense, our algorithm is of more general interest than just vehicle path/motion planning. It can be used to search for shortest paths on a graph whenever the node transition costs depend on the prior history of visited nodes. We show with simple examples the benefits of the proposed approach over the standard short-sighted strategy of initially planning a path without taking into account the dynamic envelope of the vehicle.

R. V. Cowlagi is a graduate student at the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, Email: rcowlagi@gatech.edu

P. Tsiotras is with the Faculty of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA, Email: tsiotras@gatech.edu

II. CELL DECOMPOSITIONS

Geometric path planning methods based on cell decomposition partition the obstacle-free configuration space into convex, non-overlapping regions, called cells, and then employ techniques, such as Dijkstra's algorithm, to search the connectivity graph for a sequence of adjacent cells from the initial point to the goal [5, Ch. 5 and 6]. One of the most extensively used approximate cell decomposition techniques is the quadtree method [8], [9], [10]. Multiresolution schemes that use local/global cell decompositions of varying fine/coarse resolution have been developed in [11], [12], [13], and [14]. Other hierarchical path planning techniques, which allow paths to travel through MIXED cells in early iterations and then refine those paths in subsequent iterations, so that they include only FREE or FULL cells, appear in [8], [15], [16].

A. Capturing Curvature Information

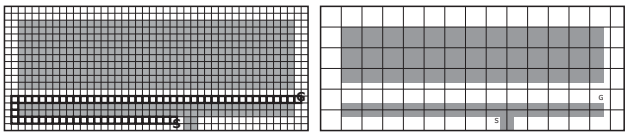
Current path planning algorithms based on cell decompositions of the obstacle-free space work exceedingly well for generating paths when no kinematic or dynamic constraints are present. However, the motion of the actual vehicle must obey such constraints. Without additional assumptions, there is no guarantee that a feasible trajectory satisfying these constraints will even exist within the channel of cells computed by the path planning algorithm. At first glance, one may argue that this is only an artifact of an inappropriate choice of the edge cost function in the associated graph. Below we provide a counter-example to this argument.

Consider the path planning problem depicted in Fig. 1, where S denotes the initial position, G denotes the goal, and the dark areas are obstacles. Consider two vehicles A and B, whose minimum radii of turn are kinematically constrained by $r^{\mathrm{A}}_{\min}$ and $r^{\mathrm{B}}_{\min}$ respectively, such that $r^{\mathrm{A}}_{\min} \leq \ell/2$ and $r^{\mathrm{B}}_{\min} > \ell$. Clearly, the dashed path in Fig. 1 is feasible for vehicle A, but not for vehicle B. A path planning algorithm for B ought to result in the bold path shown in Fig. 1.

Figure 2(a) depicts the same problem with a uniform cell decomposition of cell size $d = \ell/6$. The channel containing the dashed path of Fig. 1 is denoted by cells with bold outlines. Such a channel is obviously not traversable by vehicle B. However, notice that no *pair of successive cells* is *by itself* infeasible, i.e. a channel defined by two successive cells alone always contains a feasible path. Stated differently,

Fig. 1. Counterexample for path planning without kinematic constraint.

for any two adjacent cells, there is no cell-dependent property associated with the two adjacent cells that can be penalized by an edge cost function in order to prevent the graph search from generating a channel such as the one shown in Fig. 2(a).



(a) (b)

Fig. 2. (a) No pair of successive cells is itself infeasible; (b) Cells are too large, all cells are MIXED.

It may be further argued that a feasible path is guaranteed to exist in *any* channel if the dimensions of the cells are large enough. Indeed, Ref. [17] shows that a curvature-bounded path with local curvature less than or equal to $1/r_{\min}$ exists in a polygonal channel if the width w of the channel satisfies the inequality $w \geq \tau r_{\min}$, where τ satisfies a certain polynomial equation. The above counter-example also serves to illustrate that such a choice of cells may be too restrictive in practice. As illustrated in Figure 2(b), the dimensions of the cells may be too large to capture the details of the environment. In Figure 2(b), $d \approx \ell/2$.

B. Working with Multiple Cells

In this paper, we propose the following approach to plan a path using cell decompositions, while incorporating path curvature information: On the topological graph associated with a given cell decomposition, we define a cost function based on $k-$tuples of nodes, for some fixed $k > 2$, such that the elements of each $k-$tuple are pairwise adjacent. The question of feasibility of traversal through $k-$tuples of cells (rather than traversal through two successive cells only) allows for more general definitions of cost functions. In particular, we can introduce transition costs that capture the maximum approximate curvature for any path lying inside the channel. As a result, we can ensure that a feasible *trajectory* will always exist inside the computed channel of cells at the geometric, path-planning layer, even before invoking the motion-planning task. It should be noted that similar ideas have been explored in [18] and [19], for the specific case of $k = 1$.

III. A Graph Search Algorithm for Histories of Cell Transitions

We consider a uniform decomposition $\mathscr{C}_d$ of the environment $\mathscr{W}$, consisting of N cells, such that every cell in $\mathscr{C}_d$ is a square of size d. A cell $c_i \in \mathscr{C}_d$ will be identified by the location (x_i, y_i) of its center in some specified set of Cartesian axes. We may then construct a graph $\mathscr{G} \triangleq (\mathscr{V}, \mathscr{E})$, such that each element in the set of nodes $\mathscr{V}$ corresponds to a unique, obstacle-free cell. We label the nodes as $1, 2, \ldots, N$. Two nodes are *adjacent* if the corresponding cells are geometrically adjacent[1]. The edge set $\mathscr{E} \subset \mathscr{V} \times \mathscr{V}$ consists of all pairs (i, j), $i, j \in \mathscr{V}$ with nodes i and j adjacent. We now describe two path planning problems on the graph $\mathscr{G}$ associated with $\mathscr{C}_d$.

A. A Cell History-Based Cost

We introduce a non-negative edge cost function $g : \mathscr{E} \to \mathbb{R}_+$ that assigns to each pair of adjacent nodes in $\mathscr{G}$ a non-negative number (the cost of transitioning between the two nodes defining the edge). For given initial and terminal nodes i_S, $i_G \in \mathscr{V}$, an *admissible path* $\pi \triangleq (j_0^\pi, j_1^\pi, \ldots, j_P^\pi)$ in $\mathscr{G}$ is such that $j_k^\pi \in \mathscr{V}$, $(j_{k-1}^\pi, j_k^\pi) \in \mathscr{E}$, $k = 1, \ldots, P$, with $j_0^\pi = i_S$, $j_P^\pi = i_G$, and $j_p^\pi \neq j_r^\pi$, for $p, r \in \{0, \ldots, P\}$, with $p \neq r$.

Problem 1. Let the cost of an admissible path π be

$$J(\pi) = \sum_{k=1}^{P} g((j_{k-1}^\pi, j_k^\pi)). \tag{1}$$

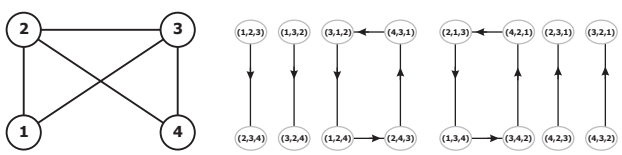
Find an admissible path π^* in $\mathscr{G}$ such that $J(\pi^*) \leq J(\pi)$ for every admissible path π in $\mathscr{G}$.

A class of algorithms used to solve Problem 1 are the *label correcting algorithms* (cf. [20], [21]). The proposed algorithm falls in the same category, as do the well-known Bellman-Ford, Dijkstra, [20], [21], and A* [7] algorithms. Label correcting algorithms progressively search for the least cost path starting from i_S and ending at node $i \in \mathscr{V}$, by iteratively reducing an estimate of the least cost to i, called the *label* of the node i. These algorithms also maintain a set $\mathscr{P}$ (referred to as the OPEN list in [20], [21]), which contains the nodes whose labels can potentially be reduced from their current value, as well as a *backpointer* function $b : \mathscr{V} \to \mathscr{V}$, which records the immediate predecessor of each node $i \in \mathscr{V}$ in the optimal path from i_S to i.

Consider now a different path planning problem with a cost function that depends on fixed-length sequences of nodes. To this end, define $\mathscr{V}_H \triangleq \{(i_0, i_1, \ldots, i_H) : (i_{k-1}, i_k) \in \mathscr{E},\ k = 1, \ldots, H,\ i_p \neq i_r,\ \text{for } p, r \in \{0, \ldots, H\},\ \text{with } p \neq r\}$, where H is a non-negative integer. We associate with each element of $\mathscr{V}_H$ a non-negative cost function $\tilde{g}_H : \mathscr{V}_H \to \mathbb{R}_+$. The problem statement is then as follows:

Problem 2a. For any admissible path $\pi \triangleq (j_0^\pi, \ldots, j_P^\pi)$ in $\mathscr{G}$, such that $j_0^\pi = i_S$, $j_P^\pi = i_G$, assume that i_S and i_G are

[1]We consider 4-connectivity for this work, that is, cells that have two vertices in common are said to be adjacent.



(a) $\mathscr{G} \triangleq (\mathscr{V}, \mathscr{E})$ (b) $\mathscr{G}_H \triangleq (\mathscr{V}_H, \mathscr{E}_H)$

Fig. 3. Example of construction of graph $\mathscr{G}_H$ for a given graph $\mathscr{G}$. In this example, $H = 2$.

sufficiently far away from each other, such that $P \geq H+1$, and let the cost associated with π be

$$\tilde{\mathscr{J}}(\pi) \quad = \quad \sum_{k=H+1}^{P} \tilde{g}_{H+1}\left((j_{k-H-1}, j_{k-H}, \ldots, j_k)\right). \quad (2)$$

Find an admissible path π^* in $\mathscr{G}$ such that $\tilde{\mathscr{J}}(\pi^*) \leq \tilde{\mathscr{J}}(\pi)$ for every admissible path π in $\mathscr{G}$.

Note that $\mathscr{V}_0 = \mathscr{V}$ and $\mathscr{V}_1 = \mathscr{E}$, and consequently, Problem 2a reduces to Problem 1 when $H = 0$. Henceforth, we will only consider the case $H \geq 1$. It is then possible to transform Problem 2a into an equivalent problem on a "lifted" graph, whose nodes are the elements of $\mathscr{V}_H$. We define the adjacency relations on $\mathscr{V}_H$ as follows: an element $I \in \mathscr{V}_H$ is adjacent to $J \in \mathscr{V}_H$ if $(J^{(H+1)}, I^{(H+1)}) \in \mathscr{E}$, $J^{(k)} = I^{(k-1)}$, for every $k = 2, \ldots, H+1$, and $J^{(1)} \neq I^{(H+1)}$, where $I^{(k)}$ denotes the k^{th} element of the $(H+1)-$tuple I. Let $\mathscr{E}_H \subset \mathscr{V}_H \times \mathscr{V}_H$ be the edge set of all pairs (J, I), such that I is adjacent to J. Figure 3 a pair of graphs $\mathscr{G}$ and $\mathscr{G}_H$, for $H = 2$.

For given initial and terminal nodes i_{S}, $i_{\mathrm{G}} \in \mathscr{V}$, an admissible path $\Pi \triangleq (J_0^\Pi, J_1^\Pi, \ldots, J_Q^\Pi)$ in $\mathscr{G}_H \triangleq (\mathscr{V}_H, \mathscr{E}_H)$ is such that $J_k^\Pi \in \mathscr{V}_H$, $(J_{k-1}^\Pi, J_k^\Pi) \in \mathscr{E}_H$, $k = 1, \ldots, Q$, and $J_0^{\Pi,(1)} = i_{\mathrm{S}}$, $J_Q^{\Pi,(H+1)} = i_{\mathrm{G}}$. Note that every admissible path $\Pi \triangleq (J_0^\Pi, J_1^\Pi, \ldots, J_Q^\Pi)$ in $\mathscr{G}_H$ uniquely corresponds to an admissible path $\pi \triangleq (j_0^\pi, j_1^\pi, \ldots, j_P^\pi)$ in $\mathscr{G}$, with $P = (Q+1)H$ and $J_k^{\Pi,(\ell)} = j_{kH+\ell-1}^\pi$, for $k = 0, 1, \ldots, Q-1$, and $J_Q = (j_{P-H}, \ldots, j_P)$. Since we assumed in Problem 2a that $P \geq H+1$, we may assume here that $Q \geq 1$.

For every pair of adjacent nodes in $\mathscr{G}_H$ we define a non-negative cost function $g_H : \mathscr{E}_H \to \mathbb{R}_+$ such that $g_H((J, I)) = \tilde{g}_{H+1}((J^{(1)}, \ldots, J^{(H+1)}, I^{(H+1)}))$ for every $(J, I) \in \mathscr{E}_H$. Consider now the following shortest path problem on $\mathscr{G}_H$.

Problem 2b. Let the cost of an admissible Π in $\mathscr{G}_H$ be

$$\mathscr{J}(\Pi) \quad = \quad \sum_{k=1}^{Q} g_H\left(J_{k-1}^\Pi, J_k^\Pi\right). \quad (3)$$

Find an admissible path Π^* such that $\mathscr{J}(\Pi^*) \leq \mathscr{J}(\Pi)$ for every admissible path Π in $\mathscr{G}_H$.

The expression for $\mathscr{J}(\Pi)$ in (3) is of the form (1), and thus Problem 2b may be solved using a label correcting algorithm, such as Dijkstra's algorithm. However, as we shall demonstrate in the sequel, this approach is computationally expensive. In the next section, we present a modification of the general label correcting algorithm that solves Problem 2a directly, without first transforming it to Problem 2b.

B. Description of Proposed Algorithm

For every node $j \in \mathscr{V}$, and integer L such that $1 \leq L \leq H$, we define the set $\mathscr{T}_{j,L} \triangleq \{I_{j,m} \in \mathscr{V}_L : I_{j,m}^{(L+1)} = j\}$. We index the elements of $\mathscr{T}_{j,L}$ with the natural numbers $1, \ldots, \left|\mathscr{T}_{j,L}\right|$. Let $T_L \triangleq \max\left\{\left|\mathscr{T}_{j,L}\right| : j \in \mathscr{V}\right\}$. The algorithm allows the user to specify L, and it maintains a family of *history* functions $h_m : \mathscr{V} \to \mathscr{V}_H \cup \{\text{NULL}\}$, $m = 1, \ldots, T_L$ instead of the back-pointer function in the standard label correcting algorithm. Along with multiple histories, the algorithm also maintains multiple labels for each node, i.e., functions $d_m : \mathscr{V} \to \mathbb{R}_+$, $m = 1, \ldots, T_L$. Consequently, $d_m(j)$, $m = 1, \ldots, \left|\mathscr{T}_{j,L}\right|$ is an estimate of the least cost of the path from i_{S} to j in which the predecessor history of j of length L is $I_{j,m}$. By definition, we set $d_m(j) = \infty$ for $m = \left|\mathscr{T}_{j,L}\right| + 1, \ldots, T_L$ if $\left|\mathscr{T}_{j,L}\right| < T_L$. Finally, as in the standard label correcting algorithm, the proposed algorithm maintains a set $\mathscr{P}$ which contains the nodes whose labels can potentially be reduced from their current value.

For the sake of clarity, we first describe the proposed algorithm for the case $L = H$. The algorithm can be easily generalized to the case $L \neq H$ (Section III-D).

INITIALIZATION

1) For every $j \in \mathscr{V}$ and $m = 1, \ldots, T_H$, set $d_m(j) = \infty$, and $h_m(j) = \text{NULL}$.
2) For every $j \in \mathscr{V}$, set $\mathscr{N}_j \triangleq \left\{J_m \in \mathscr{V}_{H+1} : J_m^{(1)} = i_{\mathrm{S}}, \left(J_m^{(2)}, \ldots, J_m^{(H+2)}\right) = I_{j,m} \in \mathscr{T}_{j,H}\right\}$
3) Set $\mathscr{P} = \left\{(j,m) : \mathscr{N}_j \neq \varnothing, \text{ and } J_m \in \mathscr{N}_j\right\}$.
4) For every $(j,m) \in \mathscr{P}$, set $d_m(j) = \tilde{g}_{H+1}(J_m)$ and $h_m(j) = \left(J_m^{(1)}, \ldots, J_m^{(H+1)}\right)$.

ITERATIVE STEPS

while $\mathscr{P} \neq \varnothing$, repeat Steps 1) through 4):

1) Compute $(j^*, m^*) = \arg\min\left\{d_m(j) : (j,m) \in \mathscr{P}\right\}$.
2) Set $\mathscr{P} = \mathscr{P} \backslash \{(j^*, m^*)\}$, and for each $j \in \mathscr{V}$ such that $(j^*, j) \in \mathscr{E}$, perform Step 3).
3) For every $m = 1, \ldots, \left|\mathscr{T}_{j,H}\right|$ set

$$\mathscr{H}_m \quad \triangleq \quad \left\{k \in \{1, \ldots, \left|\mathscr{T}_{j^*,H}\right|\} : h_k(j^*) \neq \text{NULL}, \left(h_k^{(3)}(j^*), \ldots, h_k^{(H+1)}(j^*), j^*, j\right) = I_{j,m} \in \mathscr{T}_{j,H}\right\}.$$

4) For every m such that $\mathscr{H}_m \neq \varnothing$,

$$D_{k,m} \triangleq d_k(j^*) \quad + \quad \tilde{g}_{H+1}\left(\left(h_k^{(2)}(j^*), I_{j,m}\right)\right), \quad \forall k \in \mathscr{H}_m$$

$$\textbf{if} \quad d_m(j) \quad > \quad \min_{k \in \mathscr{H}_m}\left\{D_{k,m}\right\}, \quad (4)$$

$$\textbf{then} \quad \mathscr{P} \quad = \quad \mathscr{P} \cup \{(j,m)\}, \quad (5)$$

$$d_m(j) \quad = \quad \min_{k \in \mathscr{H}_m}\left\{D_{k,m}\right\}, \quad (6)$$

$$\mu \quad \triangleq \quad \arg\min_{k \in \mathscr{H}_m}\left\{D_{k,m}\right\}, \quad (7)$$

$$h_m(j) \quad = \quad \left(h_\mu^{(2)}(j^*), \ldots, h_\mu^{(H+1)}(j^*), j^*\right). \quad (8)$$

The algorithm terminates when $\mathscr{P} = \varnothing$, at which point it returns the functions d_m and h_m, for $m = 1, \ldots, T_H$. For every node $i \neq i_{\mathrm{S}}$ in $\mathscr{V}$, we may then calculate the optimal path from i_{S} to i as follows: define $r_0 \triangleq \arg\min\left\{d_m(i) : m \in \{1, \ldots, |\mathscr{T}_{i,H}|\}\right\}$, $i_0 \triangleq i$. If $h_{r_0}^{(1)}(i_0) = i_{\mathrm{S}}$, then defining $P = H+1$, we have that a path from $j_0 = i_{\mathrm{S}}$ to $j_P = i$ is $(j_0, j_1 \ldots, j_P) = (h_{r_0}(i_0), i)$. Otherwise, let r_k and i_k to be such that $i_k \triangleq h_{r_{k-1}}^{(H+1)}(i_{k-1})$ and $I_{i_k, r_k} = h_{r_{k-1}}(i_{k-1})$, for $k = 1, \ldots, M$, where M is such that $h_{r_M}^{(1)}(i_M) = i_{\mathrm{S}}$. Letting $P = M + H + 1$, the path $(j_0, j_1, \ldots, j_P)$ from $j_0 = i_{\mathrm{S}}$ to $j_P = i$ is given by $(j_0, \ldots, j_H) = h_{r_M}(i_M)$, and $j_{H+\ell} = i_{M-\ell+1}$ for $\ell = 1, \ldots, M+1$.

In Section III-C, we show that the resultant path is indeed optimal, and that its cost, given by $d_{r_0}(i)$, is equal to the least

cost incurred from i_S to i. We first demonstrate the proposed algorithm via a simple example.

Example 1: Consider the graph shown in Fig. 4, where $i_S = 1$ and let $L = H = 1$. Let $\tilde{g}_2$ be a non-negative cost function given by the lookup Table I (for brevity, the values of only some of the elements of $\mathscr{V}_2$ are shown).

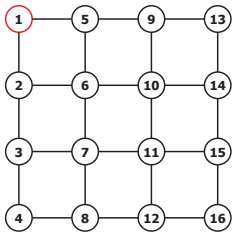


Fig. 4. Graph for Example 1.

Note that

$$\left|\mathscr{T}_{j,1}\right| = \begin{cases} 2, & j \in \{1,4,13,16\}, \\ 3, & j \in \{2,3,5,8,9,12,14,15\}, \\ 4, & \text{otherwise}, \end{cases}$$

and hence $T_1 = 4$, i.e., the algorithm maintains at most 4 histories and labels for each node. We index the elements of $\mathscr{T}_{j,1}$ as $1,2,3,4$ corresponding to UP, RIGHT, DOWN, LEFT edges of j, with reference to Fig. 4. If a particular edge is absent, the corresponding index applies to the next edge. For example, the indices of $(5,6),(10,6),(7,6),(2,6) \in \mathscr{T}_{6,1}$ are 1,2,3, and 4 respectively, while the indices of $(5,1),(2,1) \in \mathscr{T}_{1,1}$ are 1 and 2 respectively.

Applying the algorithm, we obtain that Initialization Step 2) results in $\mathscr{N}_3 = \{J_1^3\} = \{(1,2,3)\}$, $\mathscr{N}_6 = \{J_1^6, J_4^6\} = \{(1,5,6),(1,2,6)\}$, $\mathscr{N}_9 = \{J_3^9\} = \{(1,5,9)\}$, and $\mathscr{N}_j = \varnothing$ for $j \in \{1,\dots 16\}\backslash\{3,6,9\}$. Consequently, Step 3) results in $\mathscr{P} = \{(3,1),(6,1),(6,4),(9,3)\}$. Using Table I, Initialization Step 4) results in $d_1(3) = 5, d_1(6) = 2, d_4(6) = 6$ and $h_1(6) = (1,5), h_4(6) = (1,2), h_1(9) = 8, h_3(9) = (1,5)$.

Next, Iterative Step 1) results in $(j^*,m^*) = (6,1)$. Consequently, Step 2) results in $\mathscr{P} = \{(3,1),(6,4),(9,3)\}$, and Step 3) is performed for nodes 2, 5, 7 and 10. In particular, for node 2, Step 3) results in $\mathscr{H}_1 = \mathscr{H}_3 = \varnothing$, since for $m = 1,3$ we have $I_{2,m} = (1,2),(3,2)$ respectively, which do not satisfy the set-membership condition in Step 3). However, for $m = 2$, $I_{2,m} = (6,2) = (j^*,j)$, and we have $\mathscr{H}_2 = \{1,4\}$. Hence, for node 2, Step 4) is performed only for $m = 2$. Using Table I, we see that $\min_{k\in\mathscr{H}_2}\left\{d_k(6) + \tilde{g}_2\left(h_k^{(2)}(6),6,2\right)\right\} = 2+5 = 7$ and $\mu = 1$. Hence, (6) and (8) result in $d_2(2) = 7$, and $h_2(2) = (h_1^{(2)}(6),6) = (5,6)$, while (5) results in $\mathscr{P} = \{(3,1),(6,4),(9,3),(2,2)\}$.

Steps 3) and 4) are repeated for nodes 5,7, and 10, after which we have $\mathscr{P} = \{(3,1),(6,4),(9,3),(2,2),(7,1),(10,4),(5,2)\}$, and $d_2(5) = 18, d_1(7) = 9, d_4(10) = 7$. Consequently, Step 1) of the next iteration of the algorithm results in $(j^*,m^*) = (3,1)$.

C. Optimality and Performance

The definition of an admissible path Π for Problem 2b required only $J_Q^{\Pi,(H+1)} = i_G$, where $J_Q \in \mathscr{V}_H$ was the last element of Π. The first H elements of J_Q were unspecified,

TABLE I

COST FUNCTION USED IN EXAMPLE 1.

$I \in \mathscr{V}_2$	$\tilde{g}_2(I)$	$I \in \mathscr{V}_2$	$\tilde{g}_2(I)$
$(1,2,3)$	5	$(5,6,2)$	5
$(1,2,6)$	6	$(5,6,7)$	7
$(1,5,6)$	2	$(5,6,10)$	8
$(1,5,9)$	8	$(2,6,5)$	12
$(2,6,10)$	5	$(2,6,7)$	8

which implies that different admissible paths in $\mathscr{G}_H$ could possibly have different terminal nodes. In the proposed algorithm, we recognize this fact as multiple histories for every node $i \in \mathscr{V}$, denoted by the elements of $\mathscr{T}_{i,H}$. We denote the optimal cost from i_S to i, *given* a particular $I_{i,m} \in \mathscr{T}_{i,H}$, by $\mathscr{J}_{i,m}^*$. The following result shows that the proposed algorithm computes $\mathscr{J}_{i,m}^*$ for every $i \in \mathscr{V}$ and $m = 1,\dots,|\mathscr{T}_{i,H}|$.

Proposition 1: For every node $i \in \mathscr{V}$, if there exists at least one admissible path from i_S to i with a particular $I_{i,m} \in \mathscr{T}_{i,H}$, for some $m \in \{1,\dots,|\mathscr{T}_{i,H}|\}$, then the proposed algorithm terminates with $d_m(i) = \mathscr{J}_{i,m}^*$. Otherwise, the algorithm terminates with $d_m(i) = \infty$.

Proof: We first show that the algorithm terminates after a finite number of iterations. Each time a pair (j,m) is added to $\mathscr{P}$ in (5), the value of $d_m(j)$ is reduced. Since there is a finite number of nodes and T_H is finite, there is also a finite number of paths containing no cycles from i_S to i. Since the number of reductions of $d_m(j)$ is at most equal to the number of possible paths from i_S to i, it follows that the number of possible reductions of $d_m(j)$ is also finite. Hence, each pair (j,m) is added to $\mathscr{P}$ a finite number of times. Since a (j,m) pair is always deleted from $\mathscr{P}$ at each iteration, it follows that $\mathscr{P} = \varnothing$ after a finite number of iterations and the algorithm terminates.

Next, suppose there exists at least one admissible path in $\mathscr{G}$ from i_S to i associated with a particular $I_{i,m_P} \in \mathscr{T}_{i,H}$. Since there are finitely many paths from i_S to i with no cycles, and since cycles have non-negative cost, there exists an optimal path. Suppose $\pi^* \stackrel{\Delta}{=} (j_0, j_1, \dots, j_P)$ is an optimal path with $j_0 = i_S$, $j_P = i$, and $(j_{P-H},\dots,j_P) = I_{i,m_P}$. Also, let m_k be such that $(j_{k-H},\dots,j_k) \stackrel{\Delta}{=} I_{j_k,m_k}$, for $k \in \{H,\dots,P\}$. Since π^* is optimal, the cost from $j_0 = i_G$ to node $j_k \in \pi^*$ with the given I_{j_k,m_k}, must be $\mathscr{J}_{j_k,m_k}^*$, for every $k \in \{1,\dots,P\}$.

Next, for the sake of contradiction, suppose $d_{m_P}(i) > \mathscr{J}_{i,m_P}^*$ after the algorithm terminates. This implies that $d_{m_{P-1}}(j_{P-1}) > \mathscr{J}_{j_{P-1},m_{P-1}}^*$, for otherwise, when (j_{P-1},m_{P-1}) were removed from $\mathscr{P}$, the condition (4) of Iterative Step 4) of the algorithm would have been satisfied due to the optimality of π^*. Furthermore, the minimization in (4) would have been satisfied by $k = m_{P-1} \in \mathscr{H}_{m_P}$, and (6) would have resulted

$$\begin{aligned} d_{m_P}(i) &= \min_{k\in\mathscr{H}_{m_P}}\left\{d_k(j_{P-1}) + \tilde{g}_{H+1}\left(\left(h_k^{(2)}(j_{P-1}), I_{i,m_P}\right)\right)\right\} \\ &= \mathscr{J}_{j_{P-1},m_{P-1}}^* + \tilde{g}_{H+1}\left(\left(h_{m_{P-1}}^{(2)}(j_{P-1}), I_{i,m_P}\right)\right) = \mathscr{J}_{j_P,m_P}^*, \end{aligned}$$

since $h_{m_{P-1}}^{(2)}(j_{P-1}) = j_{P-H-1}$ by (8). By similar arguments, $d_{m_P-1}(j_{P-1}) > \mathscr{J}_{j_{P-1},m_{P-1}}^*$ implies that $d_{m_{P-2}}(j_{P-2}) > \mathscr{J}_{j_{P-2},m_{P-2}}^*$ and so on, leading to the conclusion that $d_{m_{H+1}}(j_{H+1}) > \mathscr{J}_{j_{H+1},m_{H+1}}^*$. However, Initialization Step 4)

TABLE II

COMPARISON OF EXECUTION TIMES: $L = H$.

$\lvert\mathscr{G}\rvert$	H	Avg. time ratio	$\lvert\mathscr{G}\rvert$	H	Avg. time ratio
100	1	1.1472	100	3	1.475
400	1	1.3680	225	3	1.531
1600	1	1.7551	400	3	1.536
100	2	1.8176	36	4	2.544
400	2	2.3000	64	4	2.519
900	2	3.8407	100	4	2.849

precludes this situation, thus leading to a contradiction. Hence, we must have $d_{m_P}(i) = \mathscr{J}^*_{i,m_P}$ after the termination of the algorithm. ■

Proposition 1 shows that the algorithm computes $\mathscr{J}^*_{i,m}$ for *every* node $i \in \mathscr{V}$ and *every* history $I_{i,m} \in \mathscr{T}_{i,H}$. However, since we only need to compute the optimal path and optimal cost from i_{S} to i_{G}, we may explore the possibility of terminating the algorithm earlier. Proposition 2, stated without proof for the lack of space, implies that this is indeed possible.

Proposition 2: Each pair (j,m), $j \in \mathscr{V}$, $m = 1,\ldots,|\mathscr{T}_{i,H}|$ enters the set $\mathscr{P}$ at most once during the execution of the algorithm.

In Iterative Step 4) the conditions (4) and (5) imply that a pair (j,m) is added to $\mathscr{P}$ only when the value of $d_m(j)$ can be reduced. It follows from Proposition 2 that once a pair (j,m) enters $\mathscr{P}$, the value of $d_m(j)$ cannot be reduced further. The implications of this fact for the implementation of the proposed algorithm are: first, since $d_m(j^*)$ cannot be reduced further if $(j^*,m) \in \mathscr{P}$, Iterative Step 3) need to be performed only for every $m \in \{1,\ldots,\left|\mathscr{T}_{j,H}\right|\}$ such that $(j,m) \notin \mathscr{P}$; and second, since we only seek to find an optimal path from i_{S} to i_{G}, the algorithm may be terminated as soon as all possible histories of i_{G} enter $\mathscr{P}$, i.e., the condition of the **while** loop of the Iterative Steps may be modified as:

while $(\mathscr{P} \neq \varnothing)$ and $((i_{\mathrm{G}},m_{\mathrm{G}}) \in \mathscr{P}$ for at least one $m_{\mathrm{G}} \in \{1,\ldots,\left|\mathscr{T}_{i_{\mathrm{G}},H}\right|\})$ repeat Steps 1) through 4).

Table II shows, for several sample test cases, the ratios of the total computational times required for the construction of the graph $\mathscr{G}_H$ and the execution of Dijkstra's algorithm on $\mathscr{G}_H$ over the execution times of the proposed algorithm. The graphs $\mathscr{G}$ used for Table II were topological graphs arising from uniform cell decompositions.We note that although the *complexity* of the two algorithms is the same (polynomial in N and exponential in H), the *absolute* execution time of the proposed algorithm is lower than that of Dijkstra's algorithm on $\mathscr{G}_H$, primarily due to the fact that the proposed algorithm does not explicitly construct the lifted graph. Furthermore, the adjacency list of $\mathscr{G}_H$ requires k times the memory required by the proposed algorithm for storing the multiple histories of each node, where k is the connectivity.

D. General Algorithm

The general algorithm allows the user to specify a parameter L, which changes the execution time of the algorithm. The parameter L is the length of histories of each node for which a record of the least cost-to-come is maintained. By Proposition 2, the upper bound on the number of iterations of the algorithm is NT_L, which could be much smaller than NT_H if $L < H$. On the other hand, the resultant path is no longer guaranteed to be optimal. Although analytical results on bounds of sub-optimality when $L < H$ are not yet available, nonetheless, our numerical examples show that the level of sub-optimality seems to be quite acceptable, in return for significantly reduced execution times. The algorithm for $L < H$ is a simple generalization of the algorithm presented in Section III-B and the its details are left to the reader.

Table III shows ratios of the total computational times required for the direct solution of Problem 2b using Dijkstra's algorithm over the execution times of the proposed algorithm for several sample graphs of varying dimensionality. The algorithm was found to achieve the optimal solution on a large number of occasions, but there are of course no guarantees that this will always be the case when $L < H$, and sporadic cases of sub-optimal results indeed do occur. A similar observation has been noted in [18], for the specific case of $H = 1$ and $L = 0$.

TABLE III

COMPARISON OF EXECUTION TIMES: $L < H$.

$\lvert\mathscr{G}\rvert$	H	L	Avg. time ratio	$\lvert\mathscr{G}\rvert$	H	L	Avg. time ratio
100	2	1	2.145	64	4	1	116.9
400	2	1	3.230	64	4	2	35.85
100	3	1	15.81	64	4	3	8.864
400	3	1	39.82	36	5	1	427.0
121	3	2	5.648	64	5	1	1185
361	3	2	7.375	64	5	2	395.6
36	4	1	46.63	64	5	3	97.39

IV. APPLICATION TO PATH PLANNING

Here we consider the relatively simple, yet important, example of finding channels containing paths whose curvature is bounded by an a priori given upper bound. We use the same notation as in Section III.

We define the *channel* $\mathscr{W}^\pi \subset \mathscr{W}$ associated with a path π in $\mathscr{G}$ given by $\mathscr{W}^\pi = \bigcup_{k=0}^{P} c_k$. We say that π is *feasible* if there exists a geometric path of maximum curvature $1/r_{\min}$, for a given $r_{\min}$, which lies entirely within $\mathscr{W}^\pi$. A necessary condition for π to be feasible is that, for every $k = 1,\ldots,P-H-1$, the fixed-length sub-path $(j_k, j_{k+1},\ldots,j_{k+H+1})$ be also feasible. In view of the previous observation we define the *tile* associated with the fixed-length sub-path $(j_k, j_{k+1},\ldots,j_{k+H+1})$ of π to be the succession of cells $(c_{j_k},\ldots,c_{j_{k+H+1}})$. For a given H, it is possible to identify a finite set of distinct tiles $\mathscr{A} = \{\mathscr{A}^1,\ldots,\mathscr{A}^{a_H}\}$, such that the tile associated with any element of $\mathscr{V}_{H+1}$ can be obtained via a sequence of geometric operations of translation, rotation, and reflection of one of the tiles in that set. For example, for $H = 3$ and $r_{\min} = 3d$, where d is the size of each cell in $\mathscr{C}_d$, the feasible tile set contains seven tiles, as shown in Fig. 5.

Now, let $\mathscr{A}_{\mathrm{aug}}$ be the set of tiles obtained via any finite number of geometric operations on the elements of the set $\mathscr{A}$, and let $\mathscr{V}_{\mathrm{feas}} \triangleq \{I \in \mathscr{V}_{H+1} : (c_{I^{(1)}},\ldots,c_{I^{(H+2)}}) \in \mathscr{A}_{\mathrm{aug}}\}$. We will use the set $\mathscr{A}_{\mathrm{aug}}$ to define the cost function $\tilde{g}_{H+1}$ in (2). For instance, Fig. 6 shows the application of the proposed algorithm to the problem presented in Sec. II-A, using the binary cost function ($H = 3$)

$$\tilde{g}_4(I) = \begin{cases} 0, & I \in \mathscr{V}_{\mathrm{feas}}, \\ \infty, & I \in \mathscr{V}_{H+1} \backslash \mathscr{V}_{\mathrm{feas}}. \end{cases}$$

The channel of cells marked in red is the result of the proposed algorithm, while the channel marked in blue is the result of performing a standard graph search on $\mathscr{G}$ using a cost defined on the edge set only ($H = 0$). The corresponding geometric paths of minimum possible curvature inside the channels are also shown in the same figure. The red curve is longer of course, but it has lower maximum curvature.

Figure 7 shows another, more interesting example, where the environment is cluttered with randomly distributed obstacles. The corresponding geometric paths of minimum possible curvature inside the channels are also shown in the same figure. Most interestingly, Fig. 7(b) shows the corresponding optimal velocity profile from the solution of the minimum-time problem along each path [22]. Although the red path is longer, a vehicle following this path will take less time than a vehicle following the shorter (but with more and sharper turns) blue path. This example demonstrates the benefits of the proposed algorithm for the solution of motion planning problems.

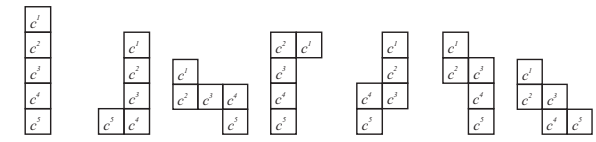
V. Conclusions

We have introduced an algorithm for finding shortest distance paths in general graphs when transitions between the graph nodes depend on prior path history. The proposed algorithm is based on an extension of Dijkstra's algorithm for two-node transitions. We have applied this algorithm for the solution of path planning and motion planning problem subject to kinodynamic constraints. By including curvature information of a feasible path at the path (geometric) planning step we can ensure better paths at the motion planning step than those obtained if the two planning steps were applied sequentially, as is typically the current practice.

Acknowledgment: This work has been supported in part by ARO award no. W911NF-05-1-0331 and NASA NRA NNX08AB94A.

References

[1] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. Association for Computing Machinery*, vol. 40, no. 5, pp. 1048–1066, November 1993.



(a) $\mathscr{A}^1$ (b) $\mathscr{A}^2$ (c) $\mathscr{A}^3$ (d) $\mathscr{A}^4$ (e) $\mathscr{A}^5$ (f) $\mathscr{A}^6$ (g) $\mathscr{A}^7$

Fig. 5. Illustration of tiles for $H = 3$

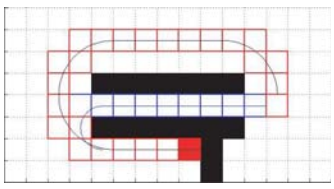
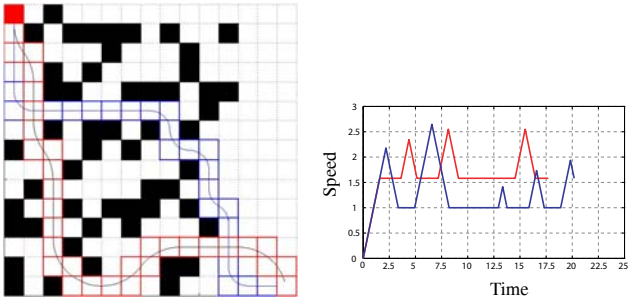


Fig. 6. The example of Fig. 1 solved using the proposed algorithm.



(a) Environment (b) Optimal velocity profile

Fig. 7. Example of a cluttered environment.

[2] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Intl. J. Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.
[3] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Intl. J. Robotics Research*, vol. 21, no. 3, pp. 233–255, March 2002.
[4] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.
[5] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
[6] Y. K. Hwang and N. Ahuja, "Gross motion planning - a survey," *ACM Computing Surveys*, vol. 24, no. 3, pp. 219–291, September 1992.
[7] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
[8] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robotics and Automation*, vol. RA-2, no. 3, pp. 135–45, September 1986.
[9] H. Samet, "The quadtree and related hierarchical data structures," *Computing Surveys*, vol. 16, no. 2, pp. 187–260, June 1984.
[10] H. Noborio, T. Naniwa, and S. Arimoto, "A quadtree-based path planning algorithm for a mobile robot," *J. Robotic Systems*, vol. 7, no. 4, pp. 555–74, 1990.
[11] P. Tsiotras and E. Bakolas, "A hierarchical on-line path planning scheme using wavelets," in *European Control Conf.*, Kos, Greece, July 2–5 2007, pp. 2806–2812.
[12] R. V. Cowlagi and P. Tsiotras, "Beyond quadtrees: Cell decompositions for path planning using the wavelet transform," in *Proc. 46th IEEE Conf. Decision and Control*, New Orleans, LA, Dec. 12–14 2007, pp. 1392–1397.
[13] ——, "Multiresolution path planning with wavelets: A local replanning approach," in *Proc. 2008 American Control Conf.*, Seattle, WA, June 11-13 2008, pp. 1220–1225.
[14] B. J. H. Verwer, "A multiresolution workspace, multiresolution configuration space approach to solve the path planning problem," in *Proce. 1990 IEEE Intl. Conf. Robotics & Automation*, 1990, pp. 2107–12.
[15] D. K. Pai and L.-M. Reissell, "Multiresolution rough terrain motion planning," *IEEE Trans. Robotics & Automation*, vol. 14, no. 1, pp. 19–33, February 1998.
[16] S. Behnke, "Local multiresolution path planning," *Lecture Notes in Artificial Intelligence*, vol. 3020, pp. 332–43, 2004.
[17] S. Bereg and D. Kirkpatrick, "Curvature-bounded traversals of narrow corridors," in *Proc. Twenty-first Annual Symposium on Computational Geometry*, Pisa, Italy, 2005, pp. 278–287.
[18] E. Rippel, A. Bar-Gill, and N. Shimkin, "Fast graph-search algorithms for general aviation flight trajectory generation," *J. Guidance, Control, and Dynamics*, vol. 28, no. 4, pp. 801–811, July-August 2005.
[19] S. Winter, "Modeling costs of turns in route planning," *GeoInformatica*, vol. 6, no. 4, pp. 345–361, 2002.
[20] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2000, vol. 1.
[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.
[22] E. Velenis and P. Tsiotras, "Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding horizon implementation," *J. Optimization Theory and Applications*, vol. 138, no. 2, pp. 275–296, 2008.