

A FAST MARCHING ALGORITHM FOR HYBRID SYSTEMS

Michael S. Branicky, Ravi Hebbbar, and Gang Zhang

Electrical Engineering and Computer Science Dept.

Case Western Reserve University, Cleveland, OH 44106-7221 USA

{msb11,rxh20,gxz7}@po.cwru.edu

This paper describes an approach to solving optimal hybrid control problems using level set methods. Level set methods are powerful techniques for generating equipotential contours with applications in the realm of fluid mechanics, computer vision, material science, robotics, and geometry. This paper specifically deals with the problem of determining an optimal control path in a hybrid system by extending the "fast marching" method to a hybrid setting. We formalize the hybrid problem, provide an algorithm to solve it, and give a constructive proof of the algorithm's correctness. We also solve two examples in our hybrid setup and discuss upper- and lower-bounds of numerical solutions.

1. Introduction

Level set methods are numerical techniques designed to track the evolution of interfaces between two regions [1]. Consider a patch of ice on the surface of a body of water. As time progresses the boundary of the ice patch keeps morphing (increasing/decreasing) with time. The evolution of the boundary between ice and water can be solved using level set methods. Thus, in essence, level set methods provide a solution to the movement of fronts.

A special case of the general level set method is the *fast marching* level set method [2], in which the front always moves in one direction. The concept of the fast marching method can be visualized as the motion of a wave front emanating from a source, e.g., the evolution of wave fronts when a pebble is thrown in a pond of still water.

The fast marching method has been used to compute the evolution of wave fronts in state spaces of two and three dimensions. In this paper, we extend the method to hybrid systems, in which there are a number of discrete states, each with a constituent continuous state space belonging to either R^2 or R^3 . Related work in robotic obstacle avoidance includes the algorithms of Trovato [4]. There has also been some work in extending level set methods to deal with the evolution of Hamilton-Jacobi equations [5].

The paper is organized as follows. Sections 2, 3, and 4 review the fast marching method, its numerical algorithm, and an application, respectively. Section 5 presents our hybrid extension. In Section 6 we solve two examples. Section 7 discusses the computation of upper- and lower-bounds.

2. Fast Marching

Let us now focus our attention on the fast marching method, as described in [2]. It forms the basis of the

algorithm presented in this paper to solve for the optimal control path in a hybrid setup.

Now we formalize the problem discussed in [2]. It consists of the following elements:

- (1) State space: $X \subseteq R^2$ (or R^3). But in numerical approximations, an N by N grid is used. In this case, the grid points are separated by Δx , Δy (and Δz in 3D).
- (2) Speed function: $F: X \rightarrow R^+$. At each point in the state space, there is a corresponding wave propagating speed. For the grid case, the speed at the i, j th (i, j, k th) grid point is F_{ij} (F_{ijk}).
- (3) Start contour set $\Gamma \subseteq X$. The time of the start point is set to be zero.
- (4) Goal set $G \subseteq X$.
- (5) Time function: $T: X \rightarrow R^+$. The time at which the boundary crosses this point.

Within this setup, the problem is to compute the evolution of the wave front over time. Figure 1 shows a wave front in a 2D plane. Assume $\phi(x, t)$ to be a distance function such that $\phi(x, t = t_i) = d$ if the distance from the point x to the wave front at time $t = t_i$ is d . The points on the wave front at time $t = t_i$ are given by the set

$$\Gamma(t = t_i) = \{x \mid \phi(x, t = t_i) = 0\}$$

The equation of the evolving contour is obtained by solving the differential equation given as

$$\phi(x(t), t) = 0, \phi_0 = \phi(x(t), t = 0) \text{ given} \quad (1)$$

$$\Rightarrow \phi_t + \nabla(\phi(x(t), t)) \cdot \dot{x}(t) = 0 \quad (2)$$

$$\Rightarrow \phi_t + F \mid \nabla(\phi) \mid \quad (3)$$

since $n = \nabla(\phi) / \mid \nabla(\phi) \mid$

In the above equation, F is the speed with which the wave can propagate at any point; it can be a function of $x(t)$. If $F > 0$, it is shown in [1] that the equation reduces to that of a form of the Eikonal equation, given as

$$\mid \nabla(T) \mid F = 1 \quad (4)$$

where $T(x)$ is the time function, defined such that $T(x_i)$ is the time at which the front passes the point x_i . A numerical approximation scheme to solve Equation (4) is given in [1, 2] and described briefly in the next section.

3. Fast Marching - Numerical Approximation Scheme

In this section we briefly describe the fast marching level set method explained in [2]. As hinted above, the problem is simplified by considering the propagation of the wave

front on a 2D grid. The approximate solution to Equation (4) on a 2D grid is given as

$$[\max(D_{ij}^{-y} T, 0)^2 + \min(D_{ij}^{+x} T, 0)^2 + \max(D_{ij}^{-x} T, 0)^2 + \min(D_{ij}^{+y} T, 0)^2] = 1/F_{ij}^2 \quad (5)$$

However, as mentioned in [2], a different approximation to the gradient, introduced by [3], which is less diffusive is given as

$$[\max(\max(D_{ij}^{-x} T, 0), -\min(D_{ij}^{+x} T, 0))^2 + \max(\max(D_{ij}^{-y} T, 0), -\min(D_{ij}^{+y} T, 0))^2] = 1/F_{ij}^2 \quad (6)$$

In Equations (5) and (6), $D_{ij}^{-x} T$ and $D_{ij}^{+x} T$ are defined as

$$D_{ij}^{-x} T = (T_{ij} - T_{i-1,j})/\Delta x \quad \text{and} \quad D_{ij}^{+x} T = (T_{i+1,j} - T_{ij})/\Delta x$$

The algorithm to compute arrival times for all points of an N by N grid, using Equation (6), is reproduced from [2]:

1. Initialize
 - a. Alive points: All start points are tagged as alive points and the value of T for those points is assigned to be 0.
 - b. Narrow band points: All points that are members of the four-connectivity neighborhood of the alive points, that are not alive, are narrow band points. The time of arrival, T , for those points are evaluated according to Equation (6), after assigning values to faraway points as in c.
 - c. Faraway points: All other points on the grid are tagged as faraway points and their time of arrival, T , is set equal to ∞ .
2. Marching forward
 - a. Begin loop: Find i and j of the narrow band point with the smallest value of T .
 - b. Tag the point as an alive point and remove it from the narrow band list.
 - c. Tag its four-connectivity neighbors as narrow band points if they are either faraway or narrow band points. Remove the neighbors in the faraway list and put them in the narrow band list.
 - d. Compute the values of T for the neighbors by solving for the largest possible solution to Equation (6).
 - e. Return to step a.

We will now look into the problem of solving Equation (6) to compute the arrival times of the neighbors of an alive point. Figure 2 shows five points on the grid, where the arrival time of the point E needs to be evaluated. Let T_A , T_B , T_C , and T_D be the arrival times of the four points. It is required to compute the arrival time of the point E , i.e. T_E .

First, define $T_x = \min(T_A, T_C)$, $T_y = \min(T_B, T_D)$, and $f = 1/F_E$. If $T_x + f \leq T_y$, then $T_E = T_x + f$. If $T_y + f \leq T_x$, then $T_E = T_y + f$. Otherwise, T_E is computed by solving the following quadratic equation and taking the largest possible root:

$$(T_E - T_x)^2 + (T_E - T_y)^2 = f^2$$

It can be easily verified that T_E computed according to the above-mentioned rules is the exact solution of Equation (6).

4. Robotic navigation examples

The fast marching level set method provides an attractive solution to time optimal path generation for robotic navigation. Obstacles in the path of the robot can be represented as regions of very low speed and the rest of the configuration space (C-space) can be modeled as regions with very high speed. Knowing the start point of the robot, level sets can be generated based on the fast marching method. The trajectory from the goal point to the start point which is orthogonal to the level set contours at every point gives the time optimal path from the start point to the goal point. Figure 3 illustrates a few examples in which the start point is the grid point represented by an asterisk and the goal point is the grid point represented by a circle.

5. Extension to hybrid system problems

We are now in a position to extend the concept to solve for time optimal paths for hybrid systems. A *hybrid system* is one that has both continuous dynamics and discrete dynamics associated with it [6]. Hybrid systems can be modeled as *hybrid automata* which have finite set of control locations with edges between them [6, 7]. The finite set of *control locations* is the finite set of discrete states and the *edges* are the discrete transitions between the states. Each state has continuous dynamics encoded in it and the discrete transitions are characterized by a guard and a reset or jump relation. Figure 4 illustrates a hybrid automaton representing a hybrid system.

The hybrid automaton of Figure 4 has two states with the continuous dynamics encoded in the states. The *guards* are the condition checks associated with the edges. A *reset relation* is one in which the value of a variable is set to some constant value and a *jump relation* is one in which the value of a variable is set based on the values of other variables or itself.

Now we formalize the problem for the hybrid situation. It consists of the following:

- (1) State space: $S = \bigcup_{q=1}^N \{q\} \times X_q$, $X_q \subseteq R^2$ (or R^3).

The state is described as the pair of discrete mode and the continuous state. There are N modes, and in the numerical algorithm, we will assume that the continuous state spaces are partitioned into grids. In each constituent state space, X_q , we have a grid of points with spacing Δx_q , Δy_q (and Δz_q).

In a very special case, $X_q \equiv X$ for all $q \in \{1, 2, \dots, N\}$. An application to this case will be discussed in Section 6.

- (2) Speed function: $F: S \rightarrow R^+$. For each mode, for each grid point, there is a corresponding speed value $F_{q,ij}(F_{q,ijk})$.
- (3) Starting set: Start contour set $\Gamma \subseteq S$. The time of the start set point is set to be zero.
- (4) Goal set $G \subseteq S$.
- (5) Time function: $T: S \rightarrow R^+$. The time at which the boundary crosses this point.

- (6) Jump function: $J: S \rightarrow 2^S$. For each $s \in S$, $J(s)$ is a subset of S which are the *jump successors* or destination points of s .

Hence, the neighbors of a point $p \in S$ are defined as

$\mathbf{N}(p) = \{4\text{- (or 6-) neighborhood, in } X_q \text{ if } p \in \{q\} \times X_q\}$;
see Figure 2; $\bigcup J(p)$.

In addition, for $s \in S$, we define its *jump predecessors* to be the set

$\mathbf{P}(s) = \{p \mid s \in J(p)\}$.

- (7) Jump delay function: $C: \bigcup_{s \in S} S \times J(S) \rightarrow R^+$

If a jump occurs from a state in one mode to a state in another mode, a jump delay (or jump cost) is incurred.

The fast marching algorithm for solving a hybrid system problem is as follows.

- (i) Initialize:

- Alive points: All start points are tagged as alive points and the value of T for those points is set to 0.
- Narrow band points: $\text{NB} = \{p \in S \mid \exists s \in S, s \text{ is alive, such that } p \in \mathbf{N}(s) \text{ and } p \text{ is not alive}\}$. The time of arrival, T , for each of these points is computed according to the method described in (ii).d below, after setting the values of faraway points as in c.
- Faraway points: All other points on the grid are tagged as faraway points with T equal to ∞ .

- (ii) Marching forward:

- Begin loop: Find q , i , and j of the narrow band point with the smallest value of T .
- Tag the point as an alive point and remove it from the narrow band list.
- Tag its neighbors as narrow band points if they are either faraway or narrow band points. Remove the neighbors in the faraway list and put them in the narrow band list.
- Compute the values of T for the neighbors:

For each neighbor point $p = (q, x_q)$, first we can get an evaluation value T_1 from its four-connectivity (or six-connectivity in 3D) neighbors in the same mode by solving for the largest possible solution to the quadratic Equation (5). Then obtain another evaluation from its jump predecessors, i.e., the sum of the arrival time of its corresponding points in another mode and the jump delay to the current mode:

$$T_2 = \min \{T_s + C(s, p) \mid s \text{ is alive and } p \in J(s)\}$$

The recomputed value for point p is $T_p = \min(T_1, T_2)$

- Return to step a.

- (iii) Stopping criterion: One goal point is alive.

A proof of the algorithm's correctness is in the Appendix.

Once the propagation is complete, the time optimal path is again determined through a backward trajectory from the goal point to the start point that follows the negative gradient. In each discrete state, this trajectory is orthogonal to the level set contours. If a jump successor, s , is hit in the

process, we examine its predecessors. If the least of their times of arrival plus jump costs is equal to that of s , we follow the jump (backwards) to that predecessor, from which the process continues.

6 Application to Hybrid Systems Problems

We now solve two hybrid examples using the modified fast marching method. In the first example, we introduce restricted switching between states. In the second one, we relax the restriction and thus introduce arbitrary switching. In both examples, the continuous dynamics of each state is specified on a 2D grid.

6.1 Start point in one state and goal point in another

In this model, we consider a start point in one discrete state and a goal point in some other discrete state, with transitions from one discrete state to another being explicitly defined at a specific subset of locations. To explain the concept, let us consider four floors of a building as four discrete states. Let us assume that the start point is the building entrance. Each floor is connected to its neighboring floors by stairs. The problem is to find the time optimal path from the entrance (triangle at the lower left-hand corner of floor 1) to anywhere in the building.

Each floor also has obstacles in it. We assume a nominal speed of unity, except within the obstacles, where it is set to a very low number (0.0001). This defines the continuous dynamics for each discrete state. The stairs represent the discrete transitions between these states, where a jump relation defines the time of traversal. The stairs are assumed to be free of obstacles and hence have constant delays associated with them (1.0). The grid point corresponding to the building entrance is the one from which the wave front propagation starts; it continues on that level the same way as is done in the case of the single state case explained in Section 3. As soon as the wave front hits a stair, however, we have a hybrid problem. The front must enter the next level and the propagation must continue on both of these levels. Therefore, we use our extension, as described in Section 5. That is, as long as a grid point does not have a stair to any of the other floors, it is considered to have only four neighbors. Otherwise, suppose a grid point, say p , has a stair connecting to the next floor at an attachment point there, say s . In this case, s is the jump successor of p , p is the jump predecessor of s , and the cost of taking the stair from p to s is given by the (in this case, constant) jump delay function.

We then propagate the front through all four floors and find the optimal path using our algorithm. Figure 5 shows the four floors with obstacles (outlined regions), the stairs connecting the floors (dotted lines), and a time optimal path (bold lines) from the entrance to a point on the fourth floor. The top left sub-figure represents the first floor, the top right the second floor, the bottom left the third floor, and the bottom right the fourth floor.

6.2 Start point and goal point in all states

In this model, we consider the case where the start and goal points are replicated in all the discrete states and *switching* between discrete states can occur everywhere on the grid (to another discrete state, while maintaining the

same continuous state). Specifically, we consider the example of maneuvering a vehicle with four gears from a start point to a goal point. The four distinct continuous dynamics for each gear are encoded in each of the four discrete states in terms of a speed profile, specified at each grid point. Transition between discrete states is arbitrary, e.g., one may switch directly from first to fourth gears.

Thus, every grid point has four neighbors in the same discrete state and a jump successor in each of the other discrete states, corresponding to the ability to switch gears. A jump delay function defines the jump cost incurred in switching from one gear to another. Here, we used a cost of 0.25 times the number of gears jumped.

The process begins from the four start points (triangles, one in each gear) simultaneously, each being assigned a cost of zero. The time of arrival of all other points in all the gears is computed according to our modified fast marching algorithm. The goal point (open circle) is also considered in all gear states simultaneously (because we only care about the end continuous state, regardless of the gear). The time optimal path is then computed as described at the end of Section 5. Figure 6 illustrates the speed profile for the four states. Figure 7 illustrates the time optimal path. In both figures, gears one through four appear in a clockwise fashion starting from the upper left.

7. Upper- and Lower-bounds while Fast Marching

The fast marching level set method provides a numerical approximation to the continuous differential equation in each mode of the hybrid problem. Now we focus our attention on deriving approximate solutions that are upper- and lower-bounds of the true analog problem. The arrival time of each grid point is calculated based on the speed profile, and we assume that we know the propagation speed at each grid cell center. If we use for that value an upper bound of the analog speed function in that whole grid cell, instead of the speed function sampled at that grid point, to do the fast marching calculation, we can obtain a lower bound of the arrival time at each grid point (because the arrival time is inversely proportional to the speed at each point). Similarly, if we use a lower bound of the speed function over the cell instead of the sample at each grid point to do the fast marching calculation, we obtain an upper bound of the arrival time approximation to each grid point.

Based on the above idea, we can compute upper- and lower-bounds for the gear switching problem of Section 6.2. The optimal arrival time from the start point to the goal point and the upper- and lower-bounds for different grid sizes are compared in Figure 8. We can see that the upper- and lower-bounds converge to the solution of the real problem (defined piecewise continuously on a 100×100 grid) as the grid size becomes larger.

8. Conclusions

This paper was aimed at studying level set methods and investigating the possibility of employing level sets in solving time optimal path problems in hybrid systems. Specifically, we extended the "fast marching" method to a hybrid setting. We formalized the hybrid problem, provided an algorithm to solve it, and gave a constructive

proof of the algorithm's correctness. Two hybrid control problems were solved using our new formulation. We also demonstrated a way to numerically compute upper- and lower-bounds of solutions.

Acknowledgments: This work supported by the National Science Foundation, under grant DMI97-20309. We wish to thank Claire Tomlin for raising the issue of upper- and lower-bounds at the AAAI Spring Symposium on Hybrid Systems and AI, Stanford University, March 1999.

References

1. J.A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Sciences*, Cambridge University Press, 1996.
2. J.A. Sethian, A fast marching level set method for monotonically advancing fronts, *Proc. Nat. Acad. Sci.*, 93(4):1591-1595, 1996.
3. E. Rouy and A. Tourin, *SIAM J. Num. Anal.*, 29:867-884, 1992.
4. K.I. Trovato, *A* Planning in Discrete Configuration Spaces of Autonomous Systems*, Ph.D. Thesis, Univ. of Amsterdam, Sept. 1996.
5. C. Tomlin, J. Lygeros, and S. Sastry, Computing controllers for nonlinear systems, *Proc. Hybrid Systems: Computation and Control*, Nijmegen, The Netherlands, March 1999.
6. M.S. Branicky, *Studies in Hybrid Systems: Modeling, Analysis, and Control*, Sc.D. Thesis, Electrical Eng. and Computer Science Dept., M.I.T., June 1995.
7. R. Alur *et. al.*, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: R.L. Grossman *et al.*, eds., *Hybrid Systems*, LNCS 736, pp. 209-229. Springer, 1993.
8. M.S. Branicky and R. Hebbbar, Fast marching for hybrid control, *Proc. IEEE Computer-Aided Control System Design*, Kona, HI, Aug. 1999. To appear.

Appendix

The proof of correctness of the traditional fast marching method (that is, without jump predecessors) is given in [2]. For the hybrid case, consider the matrix of neighboring values given in Figure 2 and assume that the time value of point E needs to be recomputed. Let P_1, P_2, \dots, P_m be the jump predecessors of E . We need to show that whenever a "trial" value is converted into an alive value, none of the recomputed neighbors obtain new values less than the accepted value. Then we will always be marching ahead in time, and thus the correct "upwind" nature of the algorithm will be respected. For simplicity, we consider the situation of two dimensions; the case of three dimensions follows similarly.

Part 1: One of the jump predecessors, say P_i , is the grid point that forces E to be recomputed. From the algorithm, if P_i is the last point to be converted from trial to alive, we know that T_{P_i} is greater than or equal to the time value of all other alive points, and is less or equal to the time value

of all the other trial points (narrow band points). For simplicity, we denote the jump delay from P_k to E as $C_k = C(P_k, E)$ for all $k = 1, 2, \dots, m$.

From $A, B, C,$ and $D,$ we obtain an evaluation value of E : $T_{E-from-ABCD}$.

According to the algorithm, the recomputed value of E is set as:

$$T_E = \min(\min_{1 \leq k \leq m} (T_{P_k} + C_k), T_{E-from-ABCD}) \quad (7)$$

Case 1: We should show that $T_{E-from-ABCD} \geq T_{P_i}$.

Case 1.1: if $A, B, C,$ and D are all in the faraway list, we are done, because the time values of these four points are all greater than T_{P_i} , so the recomputed value from these four grid points satisfies $T_{E-from-ABCD} \geq T_{P_i}$.

Case 1.2: If at least one of them, say $A,$ is alive, we claim $T_{E-from-A} \geq T_{P_i}$.

Suppose $T_{E-from-A} < T_{P_i}$ when P_i is converted to alive. Then sometime earlier, when A was converted to alive, E became a trial point and received an evaluation $T_{E-from-A} < T_{P_i}$, then the time value of E will never increase according to the algorithm. Then when we convert P_i from trial to alive, E has a time value which is less than T_{P_i} . Therefore according to the algorithm, we should have converted E to alive before, not P_i , which is a contradiction.

Case 1.3: A, B, C and D are either trial points or faraway points. Suppose A is trial and has the smallest trial value of these four grid points. We have known from the proof in [2] that $T_A \leq T_{E-from-A} \leq T_A + f$. We now recall that since P_i is the latest grid point that is converted to alive, T_{P_i} is less than or equal to T_A . Hence $T_{E-from-A} \geq T_A \geq T_{P_i}$.

Case 2: We should also show $\min_{1 \leq k \leq m} (T_{P_k} + C_k) \geq T_{P_i}$. Assume that $T_{P_j} + C_j = \min_{1 \leq k \leq m} (T_{P_k} + C_k)$. It remains to show $T_{P_j} + C_j \geq T_{P_i}$.

Case 2.1: If P_j is trial or faraway, then we are done because P_i has the smallest trial value when it is converted from trial to alive. So then $T_{P_j} \geq T_{P_i}$, and since we have $C_j \geq 0$, we conclude $T_{P_j} + C_j \geq T_{P_i}$.

Case 2.2: If P_j is alive, we proceed by contradiction. Suppose $T_{P_j} + C_j < T_{P_i}$ when P_i is converted from trial to alive. Therefore, sometime earlier, when P_j was converted to alive, we received evaluation of E from P_j :

$$T_{E-from-P_j} \leq T_{P_j} + C_j \quad (8)$$

Thus E is a trial point (in the narrow band). According to the algorithm, the time value of a trial point can only

decrease. So when P_i is converted from trial to alive, E has a time value, say T_E' :

$$T_E' \leq T_{E-from-P_j} \quad (9)$$

Hence we have $T_E' \leq T_{E-from-P_j} \leq T_{P_j} + C_j < T_{P_i}$, when P_i is converted to alive and E is trial point. This contradicts the fact that P_i has the smallest trial value when it is converted to alive. Hence when P_j is alive, $T_{P_j} + C_j \geq T_{P_i}$.

So we have shown $\min_{1 \leq k \leq m} (T_{P_k} + C_k) \geq T_{P_i}$ and $T_{E-from-ABCD} \geq T_{P_i}$. Therefore, the recomputed value of E satisfies

$T_E = \min(\min_{1 \leq k \leq m} (T_{P_k} + C_k), T_{E-from-ABCD}) \geq T_{P_i}$ if P_i is the grid point that forces E to be recomputed.

Part 2: One of $A, B, C,$ or $D,$ say $A,$ forces E to be recomputed. We know that T_A is greater than the time value of all other alive points, and is greater or equal to the time value of all the other trial points (narrow band points). Since A is the latest grid point that is converted from trial to alive.

$$T_E = \min(\min_{1 \leq k \leq m} (T_{P_k} + C_k), T_{E-from-A}) \quad (10)$$

We already know from the proof in [2] that $T_A \leq T_{E-from-A} \leq T_A + f$. We must also show $\min_{1 \leq k \leq m} (T_{P_k} + C_k) \geq T_A$. Assuming that $T_{P_j} + C_j = \min_{1 \leq k \leq m} (T_{P_k} + C_k)$, we must show $T_{P_j} + C_j \geq T_A$.

Case 1: If P_j is a far away point or trial point (in narrow band), then the time value of P_j is greater than A when A is converted to alive, and also $C_j \geq 0$, so $T_{P_j} + C_j \geq T_A$.

Case 2: If P_j is alive, we use contradiction. Suppose $T_{P_j} + C_j < T_A$. Because P_j is alive, E was in the narrow band when P_j was converted into alive. At that time E received an evaluation from P_j , say $T_{E-from-P_j}$, which is less than or equal to $T_{P_j} + C_j$. According to the algorithm, the time value of a trial point can only decrease. So when A is converted from trial to alive, E has a time value, say T_E' , with $T_E' \leq T_{E-from-P_j}$. Therefore, $T_E' \leq T_{E-from-P_j} \leq T_{P_j} + C_j < T_A$. So E has a trial value less than A when A is converted from trial to alive, which is a contradiction to A 's conversion.

Hence, we have $\min_{1 \leq k \leq m} (T_{P_k} + C_k) \geq T_A$ and therefore the recomputed value of E satisfies

$T_E = \min(\min_{1 \leq k \leq m} (T_{P_k} + C_k), T_{E-from-A}) \geq T_A$ when A forces E to be recomputed.

Now we have shown that whenever a "trial" value is converted into an alive value, none of the recomputed neighbors obtain new values less than the accepted value. Thus the algorithm constructs an upwind solution. Q.E.D.

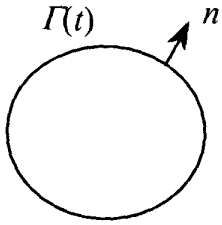


Figure 1

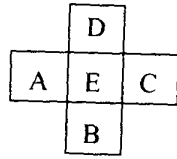


Figure 2

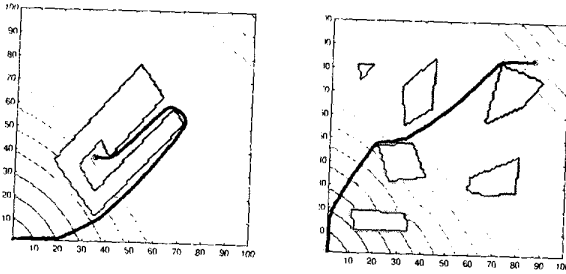


Figure 3

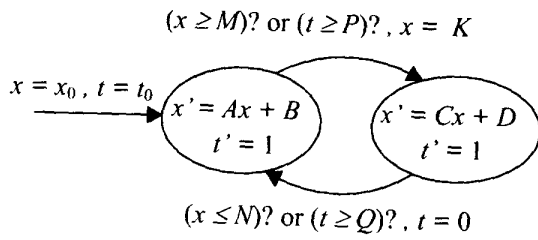


Figure 4

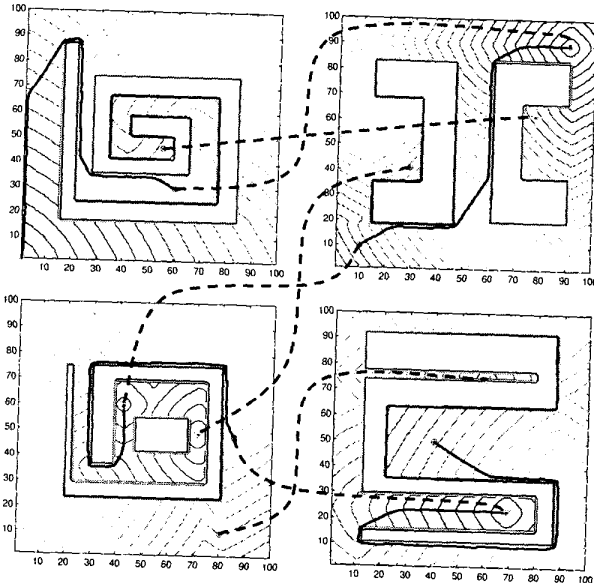


Figure 5

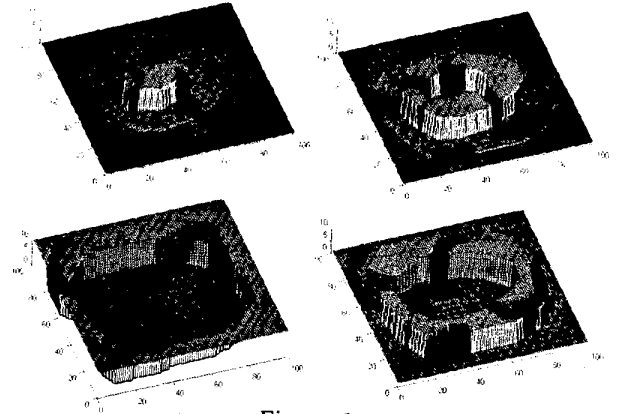


Figure 6

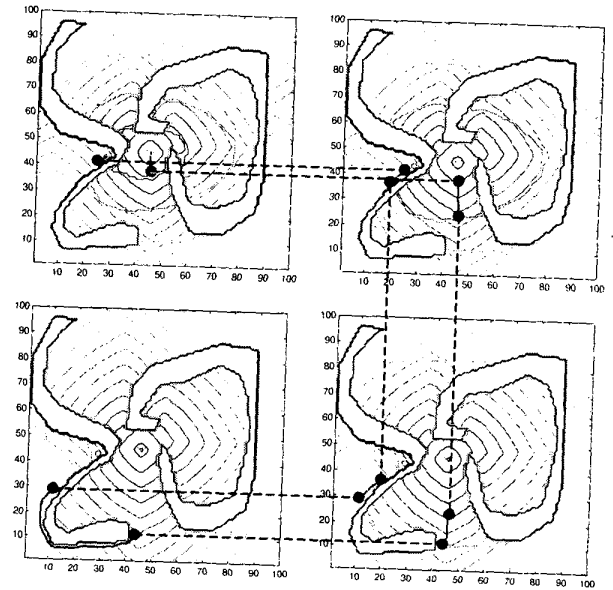


Figure 7

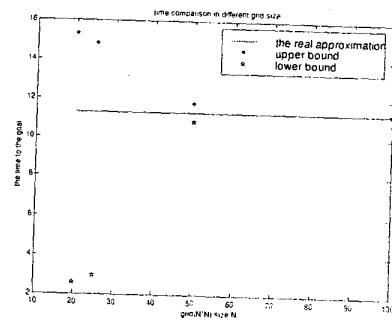


Figure 8