Stealth Terrain Navigation

Y. Ansel Teng, Daniel DeMenthon, and Larry S. Davis, *Member, IEEE*

Abstract— **A method for solving visibility-based terrain path planning problems using massively parallel hypercube machines is proposed. A typical example is to find a path that is hidden from moving adversaries. This kind of problem can be generalized as a time-varying constrained path planning problem, and is proven to be computationally hard. The method we propose is an approximation based on both temporal and spatial sampling. Since a 2-D grid cell representation of terrain can be embedded into a hypercube with extra links for fast communication, our method can be very efficient when implemented on hypercube machines. The time complexity is in general $O(T \times E \times \log N)$ using $O(N)$ processors, where T is the number of temporal samples, E is the number of adversary agents, and N is the number of grid cells on the terrain. It is also shown that the method can be applied to several realistic problems with a variety of path optimizations. All algorithms have been implemented on the Connection Machine CM-2 and results of experiments are presented.**

I. Introduction

THIS paper describes a method for solving visibility-based path planning problems over natural terrain using massively parallel hypercube machines. These problems arise in the development of both autonomous and teleoperated systems for vehicle navigation in a battlefield scenario. Generally speaking, this kind of problem is an instance of path planning on a 2-D space with time-varying constraints. It is well-known that path planning with moving obstacles is computationally hard [26]. However, visibility constraints are potentially harder than obstacle constraints because their sizes and distributions can change in time with little coherence. Furthermore, the analysis of visibility requires intensive computation. So our problem is potentially harder than path planning under moving obstacle constraints. However, by transforming these problems to a discretized formalism, many basic computations can be arranged in a regular pattern, and thus can be done in parallel efficiently. This motivated our attempt to explore digital approximation techniques on a massively parallel machine to solve these problems.

II. Previous Work

Path planning is studied extensively in robotics. Previous related work can be classified into three categories: path planning with moving obstacles, path planning for terrain navigation, and visibility analysis on terrain. Our work combines these results with digital approximation techniques on a powerful parallel machine so that we can solve realistic problems in an acceptable amount of time.

Manuscript received March 17, 1991; revised May 26, 1992. This work was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 6350 and by the U.S. Army Engineer Topographic Laboratories under Contract DACA 76-88-C-0008.

The authors are with the Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, MD 20742.

IEEE Log Number 9205796.

A. Path Planning with Moving Obstacles

Reif and Sharir [26] showed that path planning in the presence of moving obstacles is generally hard. Among their results, they provided a polynomial time algorithm for the 2-D asteroid avoidance problem for a bounded number of obstacles. To be more precise, their algorithm runs in $O(n^{2(k+2)}k)$ time, where n is the total number of vertices and edges of the obstacles, and k is the number of obstacles, which is assumed constant. Clearly, if the number of obstacles is not bounded, this algorithm will have an exponential complexity. Later Canny [5] proved that this problem is NP-hard.

Although this problem has been proved to be hard, it is nevertheless a very important problem. Researchers have developed algorithms to plan the best path under different assumptions and approximations. One early practical algorithm was proposed by Kant and Zucker [17], using a path-velocity decomposition approach that first finds a minimum length path among the static obstacles, and then determines the speed along the path to avoid collision with the moving obstacles. Later they incorporated another low-level mechanism called a local avoidance strategy [18] so that the robot can modify the path to deal with unexpected changes of movement of the obstacles. By this decomposition, they reduced the complexity of the problem by reducing the dimension of the search space. However, they also restricted the possible solutions to be within that reduced space. This is the common drawback of all practical algorithms.

Erdmann and Lozano-Perez [11] worked on a more general problem: planning the motion of multiple moving objects among several stationary obstacles. They took a prioritized approach that assigns a priority to each moving object and plans the motion of one object at a time, with the order determined by the priority. When the planner plans the path for an object, it has to take into consideration both the stationary obstacles and the moving objects that have already been planned for. This problem is considered to be more general in the sense that planning a path among both stationary and moving obstacles can be treated as planning a path for the lowest priority object, which must take the responsibility of avoiding collision with all other objects. They represented the moving obstacles in space-time by a set of slices. Each slice is the configuration space at the time when some obstacle changes its velocity, so all obstacles move at constant velocity between any two consecutive slices. The planner will return a piecewise linear path in which each segment connects two

consecutive slices. The time complexity of the algorithm is $O(rn^3)$, where n is the total number of edges in the environment, and r is the number of slices constructed. There is a trade-off between the complexity and the quality of the path. If r is small, the complexity is low, but the returned path can be very poor in that the robot is able to change its velocity only occasionally, or may even fail to find a collision-free path. This can be improved by adding extra configuration space slices between the existing ones, but at the price of increased complexity.

An important tool for path planning among static obstacles is the visibility graph [19], [20]. Fujimura and Samet [12], [14] generalized this to the accessibility graph, on which they developed an algorithm for planning a time-minimal path among moving obstacles under the assumption that the robot can move faster than any obstacle in the environment.

Tychonievich and others proposed a maneuvering-board approach for path planning with moving obstacles [29], [31]. They used a geometric constraint-based reasoning methodology for obstacle avoidance. One advantage of this method is the ability to incorporate other navigation constraints into a uniform geometric representation. Their planner will return a series of linear local paths toward the final goal. This is suitable for nautical navigation because the direction of motion does not change frequently in such cases. However, in terrain navigation, change of direction is common, and is quite natural for minimizing energy consumption, so a good planner should be able to take advantage of this property and find a better path. In their method, if the constraints cannot be fully satisfied simultaneously, they take a hierarchical approach in which constraints have different priorities.

Discretized approximation is another dimension for practical algorithms. Fujimura and Samet [13] also designed an algorithm using A^* search on a nonuniformly discretized space-time search space. Their discretization, which they called "spatial indexing," is based on an octree representation. Their algorithm features the ability to plan a path under a predetermined range of velocities, accelerations, and centrifugal forces, with the restrictions that the robot can change its acceleration only at the discretized points and that acceleration takes on discrete values.

B. Path Planning on Terrain

Mitchell explored algorithmic approaches to several terrain navigation problems. His work is summarized in [22]. The major problems discussed are the (static) obstacle avoidance problem, the discrete geodesic problem, and the weighted region problem. All of these problems can be characterized as finding an optimal path with different weighting schemes over the terrain. He worked on both digital terrain models (DTM's) and polyhedral terrain models, but mostly on the latter. The most important idea in his approach is building a *shortest path map* by using a continuous version of Dijkstra's shortest path algorithm [8]. A shortest path map is a subdivision of the plane into regions each of which is the locus of all goal points whose shortest paths from the source have the same topology (e.g., passing through the same sequence of obstacle vertices, in the obstacle avoidance case). This continuous Dijkstra algorithm simulates the propagation of a wavefront from the source, in much the same way that expansion operates in the Dijkstra or A^* algorithm. He also discussed the extension of the weighted region problem to the maximum concealment problem and the least-risk watchman route problem, which are visibility-based problems in a static environment, with the visibility defined loosely by detecting the existence of a "mountain range" between the adversary and the region being analyzed. What we need for our problem is an integration of the weighted region problem and the geodesic problem in a time-varying environment with more accurate analysis of visibility. This is probably beyond what can be accomplished using polyhedral models presently, and justifies the digital approximation approach.

Bitz and Kung [1] proposed an algorithm for planning a least-cost path on DTM with known traversal cost associated with each grid cell. The algorithm applies the dynamic programming technique and accesses the terrain information in a highly regular way. They implemented their algorithm for the least-cost path from a single source to all other points on the Warp computer. The worst-case running time is $O(n^4/k)$, using $k(k < n)$ processors for a map of size $n \times n$.

Metea and Tsai [21] proposed a path planning algorithm for a combat engagement scenario, which considers the traversal cost and the threat from the adversary on each grid cell and assigns an undesirability factor to each cell. Their algorithm optimizes the global route by using a dynamic programming technique and a hierarchical representation of terrain. Their representation of terrain is similar to the quadtree, but with a 10×10 division on each cell. Sensors are used to update the knowledge about the environment, and a rule-based production system is incorporated to modify the plan during navigation. They implemented their algorithm on a Vax 11/780 and they also discussed the parallelization of the algorithm.

Dorst and Trovato [10] developed an algorithm for optimal path planning by cost wave propagation in a discretized metric configuration space. A similar scheme for finding collision-free paths among obstacles can be found in [33]. Although they were dealing with static obstacles in an artificial environment, their concept can be adapted to deal with moving obstacles and terrain. For example, the metric can be interpreted as the traversal cost between terrain grid cells, and the 2-D space can be extended to space-time to deal with moving obstacles.

C. Visibility Analysis on Terrain

Visibility analysis is a fundamental problem in computer graphics for removing hidden lines and hidden surfaces from the display. A taxonomy of various algorithms can be found in [30]. Although any 3-D object-space hidden surface algorithm can be used for visibility analysis on terrain, it will not be very efficient since the algorithm does not use the fact that a terrain is considered as a special $2\frac{1}{2}$-D space that can only be viewed from above the surface.

Algorithms specially designed for terrain visibility analysis can be categorized by the underlying terrain models. In polyhedral terrain models, Jung [16] made a comparison

between three different algorithms, namely the angular-sweep algorithm, the edge-sorting algorithm, and the triangle-sorting algorithm. A parallel algorithm is proposed by Reif and Sen [25], which runs in time $O(\log^4(n+k))$ using $O((n+k)/\log(n+k))$ processors in a CREW PRAM model, where n and k are, respectively, the input and output sizes.

Sharir [28] proposed an $O(n \log^2 n)$ algorithm for computing the shortest watchtower that can observe the entire terrain. Cole and Sharir further investigated several aspects of terrain visibility in [6], including the problem of locating a set of observation points that jointly their visibility can cover the entire terrain. They showed that in the case of a single point, its existence and location can be determined in $O(n \log n)$ time, but finding a minimal set of points for this task is NP-hard.

In digital terrain models, terrain is represented by elevation data in regular grids. Due to the regularity of the structure, practical parallel algorithms are encouraged by the advent of massively parallel machines with limited degree of regular connections. For example, Blelloch [2] implemented a visibility algorithm with his scan operation model on the Connection Machine CM-2. We adopted his algorithm in our project; it will be explained in detail later.

Dehne and Pham [7] proposed a visibility algorithm for binary images on hypercube or perfect-shuffle computers. They use a special data structure called a *sector tree* so that parallel prefix operations can be applied efficiently. The algorithm runs in $O(\log^2 n)$ for an $n \times n$ image once the structure is built. However, the mapping between the image pixels and the sector tree is not discussed; it may involve some extra communication time.

III. Basic Example: Planning a Safe Path on Digital Terrain

A. Problem Definition

Given a digital terrain map with elevation data at each grid cell, and information about a friendly agent and adversary agents moving on the ground, we want to find a path for the friendly agent from its current position to a final goal such that the movement of the friendly agent is hidden from the adversaries by taking advantage of the terrain. We refer to this property as *safety* in the following discussion.

The information about the friendly agent includes its current position, the final goal location, and its maximal speed. In this example, speed is determined by considering only the Euclidean distance on the X-Y plane. The information about each adversary agent includes its current position and its predicted motion trajectory. We assume the movements of the adversaries are not affected by the movement of the friendly agent since it is hidden from them.

Although we are able to plan a complete path at once, the feasibility of such a path is questionable due to the uncertainty of the prediction about the motion of the adversaries. Thus our approach is to determine a time interval over which we are fairly sure of the motion of the adversaries, and make a subplan for that interval to a subgoal. The subgoal is chosen to be closer to the final goal and to have good observability in its vicinity, from which the activities of the adversaries can be observed to update the information about their motion. Then this subplan process is iterated until the agent reaches the final goal. Thus two important time factors are defined for each subplan:

1) subplan interval—the interval of time we are planning for in each subplan;
2) thinking time—the time required to generate a subplan. Since our plan is time-dependent; we have to know when the agent can start moving. We regard the thinking time as a constant known at the beginning of the subplan, which will be justified later by the complexity of our algorithm.

The criteria for a subgoal are as follows.

1) It is safe at the end of the subplan interval (and preferably remains safe for a certain period).
2) It must be reachable at the end of the subplan interval through a safe path.
3) It is closer to the goal.
4) It has good observation points in its vicinity.

So the complete path planning contains the following loop:

while the final goal is not reached
begin
evaluate each cell that can be reached in one subplan interval;
choose a subgoal;
plan a path to the subgoal;
execute the subplan and update information;
end

Since the last step, the execution of the subplan, is not part of the planning, the rest of this paper will focus on the first three steps in the loop, namely the *subplanning process*.

B. Algorithms for Subplan

The first difficulty comes from the definition of safety. As we mentioned in the introduction, it is very difficult to represent the visibility map of a moving object analytically. So we adopt a temporal-sampling approach, i.e., we determine a sampling period in advance, compute the visibility maps only once for each sampling period, and define safety by these samples.

In order to determine the cells that can be safely reached in one subplan interval, we make an incremental computation of the safely-reachable region for each time step, denoted as RS_t, where t is the time index. It is natural to set the length of one time step as the length of a sampling period for computing the safety map. The computation starts at the end of the thinking time, where $RS_{\text{end\_of\_think\_time}}$ is initialized as the current location of the friendly agent. For each time step, we compute RS_t by first expanding RS_{t-1} according to its maximal speed, and then curtailing it by the regions visible from the predicted adversary positions at time t. See Fig. 1. After iterating this process, t is incremented to the end of the subplan interval, and it is guaranteed that there is a safe path to any cell in the region $RS_{\text{end\_of\_interval}}$. Cells in this region are evaluated by the criteria mentioned in the previous subsection and the best cell is chosen as the subgoal.

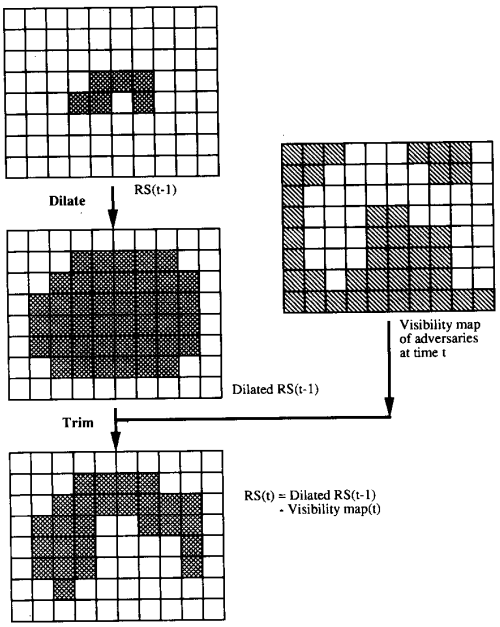


Fig. 1. Building the RS structure incrementally by dilating and trimming. In the visibility map, the shaded cells are visible to the adversaries.

The path from the starting point to the subgoal is extracted from the RS structure backward in time. We start at the subgoal in $RS_{\text{end\_of\_interval}}$ and dilate it according to the speed constraint. The result defines the set of cells the agent has to reach at one time step before the end of the subplan interval if it is to reach the subgoal at the end of the interval. Then we compute the intersection of this set and the RS at that time. The result is a set of cells that can be on our path. It is guaranteed that this set is nonempty by the method employed to construct the RS structure. We have to choose one cell from this set according to some goodness measure; we currently choose the cell with the least distance to the straight line between the last path point and the initial location of the friendly agent, which tends to minimize the path length. Then from the chosen path point we repeat the dilation-intersection-selection procedure, until the time index is decremented back to the end of the thinking time. Since the current position is the only cell in the first RS region, the path must terminate at the current location. The terms *forward dilation* and *backward dilation* denote, respectively, the processes of the construction of the RS structure and the extraction of the path. The subplanning process is summarized in this algorithm:

Algorithm Subplan
begin
{Step 1: Finding the safely-reachable region by forward dilation}
$RS_{\text{end\_of\_think\_time}}$ = the current position of the friendly agent;
for t from the end of the thinking time to the end of the subplan interval
begin
$RS_t = dilate(RS_{t-1})$;
Predict the adversary positions at this moment;
Analyze the visibility from the adversary agents to the potentially reachable region;
$RS_t = RS_t$ Nonvisible region;
end;
{Step 2: Choose a subgoal}
Evaluate cells in $RS_{\text{end of interval}}$ and choose the best one as the subgoal;
{Step 3: finding the path by backward dilation}
$PATH_{\text{end\_ of\_ interval}}$ = the position of the subgoal;
for t from the end of the subplan interval back to the end of the thinking time
begin
$PATH_t$ = dilate$(PATH_{t+1})$;
$PATH_t = PATH_t \cap RS_t$;
$PATH_t$ = best cell$(PATH_t)$;
end;
return$(PATH)$;
end algorithm

Fig. 2 illustrates this algorithm in space-time. Although the RS structure is constructed by dilation at the maximal speed of the agent, this does not mean that the agent is always moving at its maximal speed. A forward dilation of RS_t computes the region that can be reached at time $t+1$ from a point in RS_t at a speed bounded by maximal speed. Consequently, although the path covers a time interval from the end of the thinking time to the end of the subplan interval, the agent may not always be moving during this interval, since the same terrain cell may be chosen as the path points for several consecutive time steps. In such cases, the agent would simply remain at that cell for the corresponding interval: the cell can be the starting location, the subgoal, or any intermediate point along the path.

Assume the terrain map is a square with a total of N grid cells. In our implementation, a 2-D array of processors of the same size is allocated to store the terrain map as well as the RS structure. One processing element (PE) is assign to each terrain cell. It stores the height information and all the corresponding elements of the RS structure for that cell. Let T be the number of sampling periods in one subplan interval, i.e., the number of temporal samples considered in one subplan interval, and E be the number of adversary agents. The time complexity of this algorithm is $O(T \times E \times VA)$ where VA stands for the time complexity of one visibility analysis.

IV. Algorithm for Visibility Analysis

Visibility analysis requires intensive computation and is used frequently during path planning. In fact, it dominates the running time for the entire process. Our implementation is based on the algorithm developed by Blelloch [2] using hypercube machine models. Hypercube machines provide the flexibility to embed a mesh-connected array of any dimension and the efficiency to perform parallel prefix operations along any axis of the array in logarithmic time. These advantages will be used extensively in our algorithms; this kind of parallel prefix operation will be referred to as a *scan* operation.

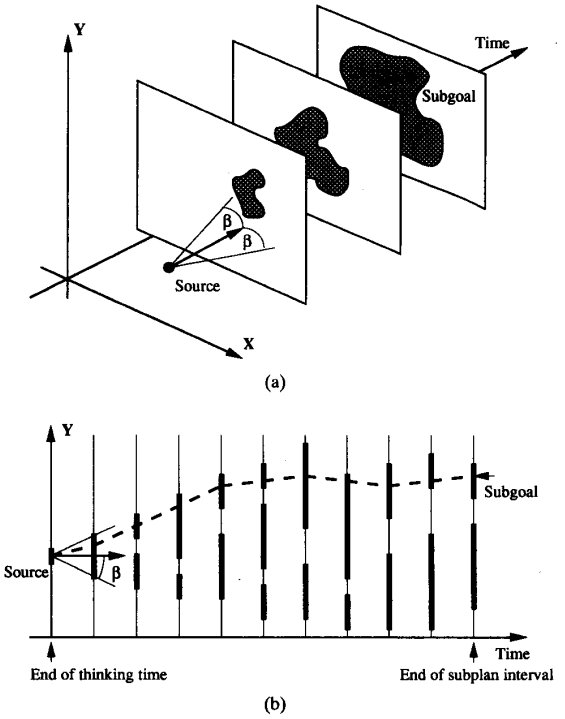


Fig. 2. Finding a path through the RS corridor. (a) A perspective view. The RS regions are shown by the shaded area on each time slice, with the speed constraint shown by the angle β. (b) A side view of this problem. Thick vertical line segments show the RS regions and the dashed line is an example of a valid path.

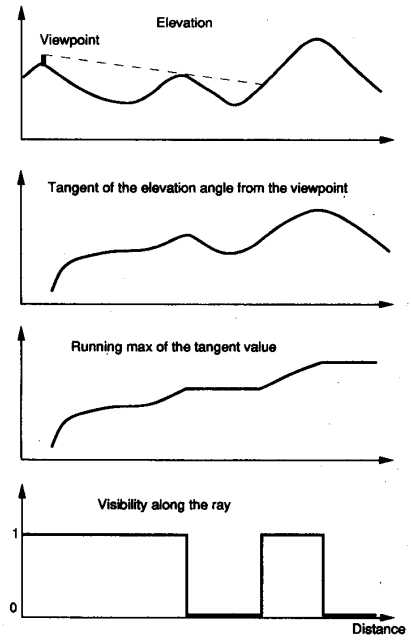


Fig. 3. Computing visibility along a ray. The X-axis represents the horizontal distance from the viewpoint. The effect of the middle hill is shown by the dashed line in the elevation plot. In practice, the tangent value of an elevation angle is used instead of the elevation angle itself. Also the height of the viewer is added to the elevation of the viewing point.

The visibility analysis algorithm computes the visible subregion of a given region from a given viewing point. Therefore, it is referred to as a *point-to-region* visibility analysis (PRVA) algorithm. Since the underlying terrain is represented by an array of processors, each corresponding to a grid cell of the terrain, the result is returned as a *visibility map* by indicating the visibility of each grid cell with a flag in the associated PE. For computational efficiency, the region is assumed to be an upright rectangle. For a region of arbitrary shape, we may obtain the maximal and minimal X and Y coordinates of the region and use the rectangular box determined by these coordinates instead. On a hypercube machine, these coordinates can be obtained in $O(\log N)$ time for a DTM with N grid cells.

A. Visibility Along a Ray

A point is visible to a given viewing point if and only if the line segment joining them in the three-dimensional space lies completely above the terrain. The basic parallel algorithm for visibility analysis is to compute the visibility w.r.t. a given viewing point for every grid cell along a *ray*, where a ray refers to a directed line segment on the X-Y plane. Suppose we have a list of processing elements (PE's) representing this ray. Each of them contains the elevation of the corresponding grid cell on the ray with the first cell as the viewing point. The visibility of each grid cell from the viewing point can be determined by first computing the elevation angle from the viewing point to each cell and then comparing its angle to the running maximum, that is, the maximal angle among all cells closer to the viewing point. A cell is visible if its elevation angle is greater than the running maximum. These steps are illustrated in Fig. 3.

The total complexity is dominated by the computation of the running maximum that can be done by a scan operation in $O(\log K)$ time on a hypercube machine using K processors, where K is the number of grid cells along the ray.

B. Point-to-Region Visibility Analysis

The PRVA algorithm is based on the ray-visibility algorithm. The idea is to find a minimal set of rays that covers all the grid cells in the region, allocate a list of processing elements to each ray, let each processor get the elevation of its corresponding cell, then use the ray-visibility algorithm to determine the visibility of each corresponding cell, and send back the result to that grid cell on the terrain.

Here we define and clarify some terminology in our discussion. Geographically, we use the term *ray* to refer to a directed line segment on the X–Y plane. It also refers to a list of PE's from the computational point of view. The processor structure representing a set of rays is called a *ray structure*. We do not distinguish between the two meanings of rays as long as no ambiguity occurs. The same principle applies to the terrain cells and points. In a DTM, terrain is actually a tessellation of square cells in a regular grid pattern. Each cell has a representing grid point in the center, indexed by the

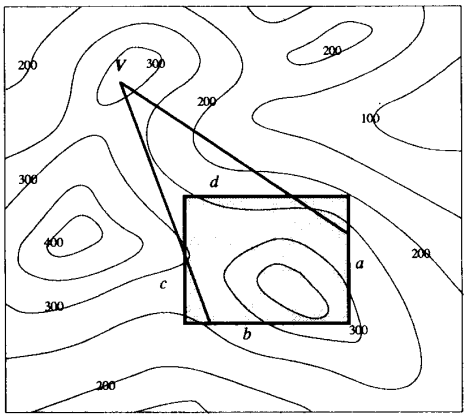


Fig. 4. The definition of far sides. Only side a and b are far sides because the line segments from the viewing point V to them pass through the interior of the region.

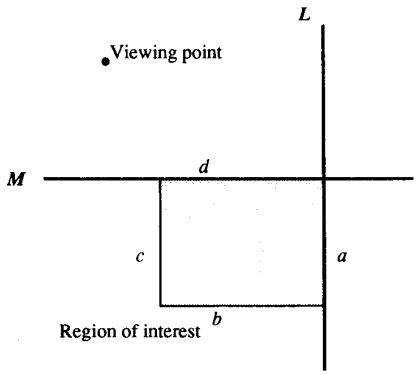


Fig. 5. The test of far sides. Side a is a far side since the viewing point and the region of interest lie on the same side of L, the line embedding side a, while side d is not a far side since they are on different sides of M.

X and Y coordinates. We usually use the term "point" to emphasize a specific location and the term "cell" to imply a piece of the tessellation with unit area. Computationally, both terms refer to the PE assigned to that grid cell. Two processor structures are used in the analysis: one for the terrain map and the other for the ray structure. We usually refer to the processors of the former as terrain cells and the latter as PE's of a ray or in a ray structure.

As mentioned above, we assume the region to be an upright rectangle. We define a *far side* as a side of the rectangle such that when drawing a line from the viewing point to any point on that side, except the end points, the line will pass through the interior of the rectangle. See Fig. 4. A useful property of far sides is that the viewing point and the rectangle lie on the same side of the line containing a far side. This property leads to a good test for far sides. See Fig. 5. The minimal set of rays covering all grid cells in a rectangular box will be the rays from the viewing point to all grid points on the far sides. Therefore the PRVA can be accomplished by constructing a ray structure for each far side of the rectangle and running the ray-visibility algorithm for each ray in the ray structure.

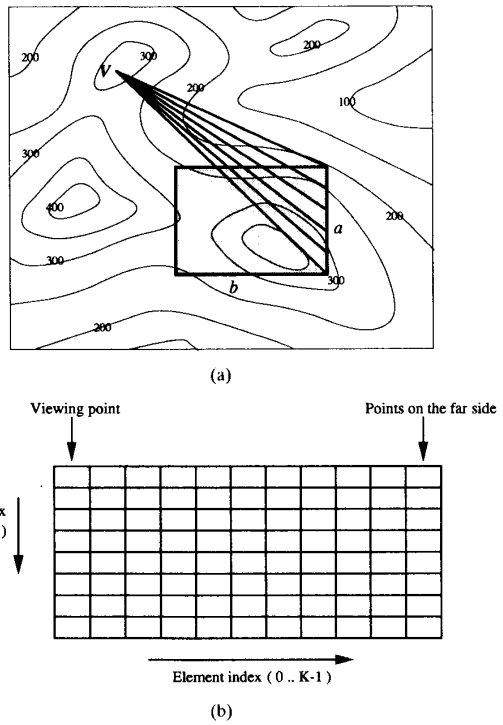


Fig. 6. Ray and ray structure. (a) A ray structure for side a is constructed: the rays are shown by the line segments between the viewing point V and side a of the rectangle. The ray structure consists of all rays for this side. (b) The ray structure is allocated as a 2-D array.

Let L be the length of a side of the rectangle, and K be the maximal number of grid cells on a single ray to this far side. An $L \times K$ 2-D array of processors is allocated for the ray structure, as shown in Fig. 6. Each row in the ray structure corresponds to a ray. A PE in the ray structure is identified by its ray index and element index. Ray index ranges from 0 to $L-1$, indicating which ray the processor is on, while element index ranges from 0 to $K-1$, indicating its rank in its ray. The key point in constructing the ray structure is to find the corresponding grid cell in the terrain map for each processor in the ray structure so that each processor can obtain the corresponding height to perform the ray visibility algorithm. The coordinates of the viewing point and the end points of the far side are first broadcast to the entire ray structure. Using the ray index, each processor can find the end point of its own ray. Then, using the element index, each processor can find its corresponding grid cell by the digital differential analyzer (DDA) technique [15]. The time complexity for constructing the ray structure is constant.

After construction of the ray structure, each processor obtains the elevation information from its corresponding terrain cell and the ray-visibility algorithm is conducted along all rays simultaneously. The result will be sent back to its corresponding grid cell. If several results are sent back to the same grid cell, they are combined by an OR operation.

The time complexity is thus $O(\log K + Comm)$, using $L \times K$ processors, where $Comm$ stands for the complexity of the

global communication to read the elevation data and write the visibility flag. In the construction of the ray structure, since we actually map a triangular area of the terrain into a rectangular 2-D array of processors, several processors in the ray structure may be mapped to the same terrain cell. The closer to the viewing point, the more processors will be mapped to the same cell. Thus numerous concurrent reads/writes occur in the global communication between the two processor structures. This does not only result in congestion in the communication network but also create a long list of destination addresses in the terrain cells near the viewing point such that it may run out of the limited memory in each processor.

The solution is based on the observation that all PE's corresponding to the same terrain cell have the same element index and consecutive ray indexes, as shown by an example in Fig. 7. Using this property, concurrent reads/writes can be eliminated by grouping the PE's according to the terrain cell they are mapped to and allowing only one PE from each group to participate in the global communication. In each group, the data can be distributed to or combined from every member by segmented scan operations. As this operation is conducted along the dimension of size L, its complexity is $O(\log L)$. The complexity of an exclusive read/write operation should be no more than that of a permutation. In the worst case, a permutation on an N-node hypercube can be performed in $O(\log^2 N)$ time [4], while a randomized routing algorithm can achieve an expected $O(\log N)$ complexity with high probability [32]. Since both L and K are bounded by $\sqrt{N}$, the overall time complexity of the PRVA algorithm is thus $O(\log N)$ expected with high probability.

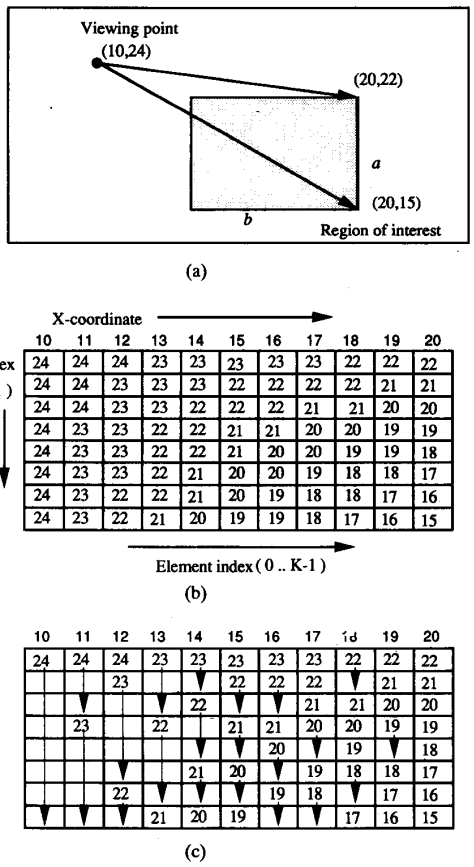


Fig. 7. Using scan operations for height information. (a) The coordinates involved in this example. (b) The corresponding Y-coordinate for each element of the ray structure. All elements in the same column have the same X-coordinate, which is shown awt the top of each column. (c) The arrowed lines show where the scan works.

V. The Planning Algorithm and Its Extensions

In an attempt to provide a more general framework for solving more complex path planning problems, we recast the solution procedure in a more abstract framework. There are two constraints in the previous example, namely reachability and safety. Reachability can be generalized as motion constraints on the agent itself, and safety can be generalized as constraints derived from the environment, which refer to factors that will not change in response to our actions, e.g., the terrain and the movements of the adversary agents. Therefore, each slice of the RS structure can be interpreted as a *qualified region* under these constraints.

Our planning algorithm is a variation of the dynamic programming paradigm. It fills up a 3-D table, i.e., the RS structure, using values previous obtained in the table and values from another precomputed 3-D table, i.e., the visibility maps. It keeps the whole table for efficiency in finding the path backward because the computation of the qualified region is too expensive to be repeated. The key point that makes this algorithm suitable for parallel processing is to represent all the constraints in digital maps efficiently. The safety constraint was discussed in the previous section. In this section, we first discuss the motion constraint; then we show that we can solve other path planning problems with appropriate modifications of dilation and trimming to optimize over a variety of constraints.

A. Dilation

In our framework, the motion constraint is satisfied by a process called dilation, which corresponds to the expansion of a region by the maximal distance the agent can move in one unit of time. Assume S is a set of points. The dilation of S by radius r is defined as

$$S^r = \{p \mid \exists q, q \in S \wedge \overline{pq} < r\}.$$

The definition is concise, but it is hard to compute S^r from S on a digital map exactly due to quantization errors. In our implementation, we use alternating four/eight-neighbor propagation to approximate the dilation. Fig. 8 illustrates this process. This method, also known as the octagonal distance transformation, was proposed by Rosenfeld and Pfaltz [27] for parallel computation of distance transformations. There are some inevitable errors involved in this process, which can accumulate through the iterations. The maximum error of octagonal distance transformation w.r.t. Euclidean distance transformation is about 15% [3]. Using more sophisticated approximation techniques can reduce this error, but at the price of additional running time. Discussion of distance transformation

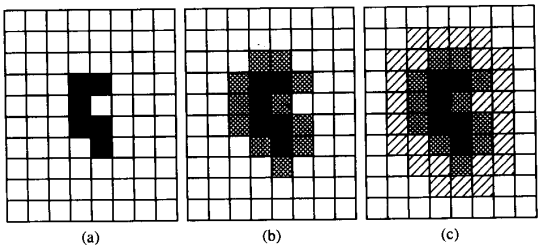


Fig. 8. An octagonal dilation with radius 2. (a) The region before dilation is in black. (b) The region expanded by four-neighbor propagation is shown by the cross-hatched cells. (c) The region expanded by the following eight-neighbor propagation is shown by the slash-hatched cells.

and length estimation on digitized spaces can be found in [3], [9], [23].

B. Finding a Relatively Safe Path Instead of an Absolutely Safe Path

The example mentioned in Section III returns a safe path if there is one. But it will return a null path if there is no absolutely safe path. Also, it will sometimes make an awkward choice of subgoal if all points in the direction of the final goal are exposed to the adversaries at some moment, even if for only one time step. If these situations occur, we would rather have a path that is safe for most of its length and can lead us to a good subgoal, than have nothing useful.

We formalize the previous example by using some standard concepts from optimization theory [24]. Let

$$P(t) \qquad t \in [t_0, t_n]$$

be the path, which is the set of problem variables. For the planning of an absolutely safe path, we maximize an objective function

$$Q(P(t_n)) \qquad t \in [t_0, t_n]$$

under the constraints

$$\left|\frac{dP(t)}{dt}\right| < V_{\max}$$

and

$$\text{Safety}(P(t)) = 1$$

for all $t \in [t_0, t_n]$. Note that the objective function $Q()$ depends only on the subgoal, i.e., $P(t_n)$ in our previous example.

To realize a relatively safe path we move the safety constraint into the objective function so that it can be optimized together with the quality measure of the subgoal. The new objective is to maximize

$$Q'(P(t)) \qquad t \in [t_0, t_n]$$

under the constraint

$$\left|\frac{dP(t)}{dt}\right| < V_{\max}$$

where $Q'()$ depends on the safety of the entire path and the quality of the subgoal.

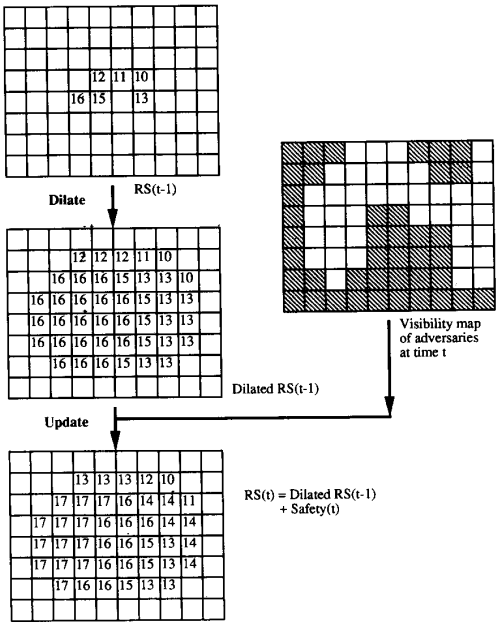


Fig. 9. Building the RS structure incrementally by dilating the path credit and adding the new safety information.

In our framework, there are two methods for satisfying the constraints and optimizing the objectives: the first approach is by control structure, and the second is by evaluation functions. In the basic example, the constraints are strictly enforced by the program control structure, i.e., dilation for the speed constraint and trimming for the safety constraint, while the objective function is optimized by an evaluation function that computes the weighted sum of the subgoal criteria for each grid cell. Using control structure reduces the size of the solution space, but it will also reduce the flexibility to perform various ways of optimization.

To optimize safety together with the criteria for the subgoal, we should avoid trimming the qualified region at each time step. Instead, we keep in each grid cell a value representing the quality of the best path to this cell and propagate this information during the dilation process. In general, we refer to this value as *path credit*, which in this case can be a count of safe steps along the path. See. Fig. 9 and compare it to Fig. 1. So the data structure for each qualified region is not simply a bit map, but a table in which a value is associated with each cell representing the safety of the safest path among all paths to the cell at that moment. At the end of the planning interval, we can use this information to optimize the choice of the subgoal in a variety of ways, which will be discussed later. Once we choose the subgoal, the path can be traced backward by the standard gradient-following technique [10]. As a result, this extension to our framework provides the flexibility to deal with more complex planning problems. The complexity will increase as the size of the data structure increases, but it should remain in the same order as the original one.

C. Path Planning for a Group of Agents

A more complicated problem is to plan a path for a group of agents so that they can maintain a given configuration and mutual visibility during movement. It is easy to see that all requirements except speed can be absorbed as part of the objective function to be optimized. However, too much flexibility makes the solution space intractable. Especially in this case, the flexibility of the group's possible configurations makes it difficult to measure safety and intervisibility. So we take their configuration as a constraint and restrict our attention to the following more specific problem: given a group of agents located along a line segment, find a path for the group that satisfies the speed constraints and optimizes the following factors:

- safety—hidden from the adversary agents;
- intervisibility—the agents can see their partners;
- configuration—the agents maintain their configuration as a line segment but the orientation of the line segment and the spacing between the agents can change. However, there is a limit on the length of the line segment, and we do not allow the spacings to change significantly along the path.

Using the same formalism as in the previous subsection, we can write the general form of this optimization as optimizing

$$Q(P_k(t))$$

where $P_k(t)$ is the path for the kth agent and Q is the quality measure over all paths. However, for each time instant, we have a $2k$-dimensional instead of two-dimensional solution space, which is prohibitively large. To reduce the dimensionality of the solution space, we use the configuration constraint and treat the group of agents as a line segment of a fixed length. Now, a path point can be represented by three parameters: the X and Y coordinates of the midpoint, and the orientation. With the additional time-axis, we obtain a four-dimensional parameter space. We can apply our framework in this four-dimensional search space, but it is still too large to be practical. So we make a further reduction by applying a decomposition.

First we represent the locations of the agents by only one cell on the terrain, which is the center of the group, and find a good path for the center, where "good" means a high likelihood that a segment centered at this cell will be safe and intervisible, no matter what orientation the segment is in. After finding this corridor, we can allocate the agents within the corridor using some simple heuristics. Since the corridor has a statistically good evaluation for all the constraints, we should be able to find an acceptable allocation.

To find a path for the center of the group, we can still model the motion constraint by dilation since the range of speed for the center point is the same as for the individual agent by the assumption that all agents have the same speed constraint. Therefore we can apply the method of finding a relatively safe path by propagating a path credit defined on both safety and intervisibility for each cell to be the center of the line segment.

In this case, safety is measured as the number of safe cells within the distance of half of the segment length from each potential group center, i.e., a circle with a diameter equal to the segment length. This information has to be computed for each time step, and the complexity is $O(\log L)$ if we approximate the circle by a square, where L is the length of the line segment. Since L is much less than the side length of the terrain, this computation will not increase the complexity of the entire algorithm.

Intervisibility is measured by sampling several line segments centered at a grid cell, and counting the number of pairs of cells that can see each other along these line segments. Using a variation of the ray-visibility algorithm, which shifts the entire DTM along the direction of each sampled line segment, we can count this number for all cells in $O(M \times L)$ time, where M is the number of samples we take, and L is the length of the line segment. Since both M and L should be fairly small, and this computation needs to be done only once, this computation will not increase the overall complexity of the entire algorithm.

D. Optimization on the Objective Function

In the previous subsections we have dealt with different extensions by moving constraints into the objective functions, dilating path credit instead of trimming, and reducing the dimensionality of the problem variables. There are also many variations on the objective function itself. With small modifications to the dilation process, mainly on the updating of path credit, we can achieve different types of optimization. Here are some examples.

1) Find the path with highest minimal cell credit on all path points. Cell credit refers to the evaluation of each grid cell at each step, e.g., safety, intervisibility, or their combination. Trimming should not be performed during each dilation for this criterion. Each cell first propagates its path credit to its neighbors and sets its path credit to the maximum among the received values and its own path credit. Then each grid cell updates its path credit to the minimum between the result obtained from the above operation and the new evaluation.
2) Find the path with highest sum of cell credits along all path points. In this case, the path credit is the largest partial sum of cell credits. Each cell updates the value by adding its new cell credit to the maximal partial sum received.
3) Highest total cell credit with a threshold on minimal cell credit. We can use the dilation scheme in the previous case, and trim the region by the threshold on its new evaluation.

Besides the dilation process, we can also make modifications to the evaluation of cell credit. For example, we can compute the level of safety by considering the number of adversaries each cell is exposed to, and also the distance to these adversaries.

Usually, it is important to ensure that the final goal will be reached within a reasonable amount of time. For this consideration, we can increase the weight on the distance to the final goal in the subgoal criteria after each iteration of the subplanning process.

VI. Terrain Traversability

So far we have modeled the traversal cost only on the Euclidean distance between points. This is not realistic, since it takes much more time and energy to traverse uphill than downhill. In general, the traversal costs should depend not only on the Euclidean distance, but also on the slope, the terrain type, the ground cover, and other factors that may affect the mobility. In this section, we assume that there is a way to assign traversal costs, and we present a method of taking these costs into consideration.

There are two different interpretations of the traversal cost: one is to model how far the agent can travel within a certain amount of time. This is the interpretation used in the previous sections. Under this interpretation, the agent must be able to travel through several grid cells in one time step in order to distinguish between different traversal costs effectively. Therefore, either the time step has to be very long or the terrain resolution has to be very fine. Both options are usually undesirable since a long time step would reduce the quality of the path, and the size of the solution space grows quadratically in the spatial resolution. Thus, we take a different interpretation, which considers how many time units it will take to travel through a certain distance. In order to distinguish different traversal costs, a finer scale of time is adopted such that it always takes more than one unit of time to move through a cell.

In order to reflect the effect of the terrain, modifications must be made to the dilation process. Dilation is the process that implements the dynamic programming technique to construct the RS structure. The RS structure can be viewed as a three-dimensional table in which each element is identified by its corresponding terrain cell and a time index. Each element stores a value called the path credit, which represents the quality of the best path to the corresponding grid cell at that moment. Under the first interpretation, dilation is performed by propagating the most recent path credit to all cells within a certain distance. Under the second interpretation, we first discretize the directions in which the agent can move through a grid cell and associate a traversal cost for each of them as the amount of time needed to pass through the cell in that given direction. Then dilation is performed by propagating delayed path credits to the neighboring cells. The length of the delay depends on the traversal cost assigned to the corresponding direction of the cell. After each cell receives the values from its neighboring cells, its path credit is updated in the same way as discussed in the previous section. For example, if we are planning a relatively safe path using the number of safe cells along the path as the path credit, the credit value in the RS structure for a cell P at time t is defined by the recurrence relation shown at the bottom of the page, where Q is any neighbor of P, and

$$\begin{aligned}
&\mathrm{Tcost}(Q,P) = \text{the traversal cost through } Q \text{ in the direction to } P\\
&\mathrm{Safety}(P,t) = \begin{cases} 1, & \text{if cell } P \text{ is safe at time } t\\ 0, & \text{otherwise}\end{cases}\\
&\mathrm{Safety_count}(P,t_1,t_2) = \sum_{t=t_1}^{t_2} \mathrm{Safety}(P,t).
\end{aligned}$$

See Fig. 10 for an illustration of this formula, in which four directions are considered for traversing through a cell. Each path credit is delayed by an amount of time equal to the traversal cost before it is propagated to the corresponding neighbor and it is updated with safety information during the delayed period. After receiving the path credits from its neighbors, each cell decides its current path credit by choosing the maximum among the received values and its own previous path credit which is also updated by adding the current safety to it. Using this formula, the RS structure can be constructed by iterating on each unit of time through the subplan interval. Then we can apply the same principles in choosing the subgoal and finding the path backward as discussed before.

There are several issues concerning implementation that we should mention here. First is memory requirements. The size of the RS structure grows linearly in the number of temporal samples as we use a finer scale of time to distinguish between different traversal costs. As the whole RS structure is stored only for finding the path using backward dilation, one possibility for reducing the memory requirement is to store pointers to the predecessors during the construction of the RS structure instead of maintaining the entire RS structure. Since the predecessor of a cell is either itself or a neighboring cell, the number of bits needed for the pointer is usually much less than that for the path credit. Using pointers also simplifies the extraction of the path, because the backward dilation is complicated with the modeling of terrain traversability. It is still necessary to maintain a few slices of the RS structure for the forward dilation, but the number is reduced from the number of temporal samples in the entire subplan interval to the maximal traversal cost, which is usually only a small percentage of the former.

Another issue is the number of visibility analyses needed under this modification. Since the time scale is finer in this case, we do not have to compute the visibility map at each unit of time if we just want to obtain the same accuracy as in the previous versions. We can define a sampling period as several units of time, compute the visibility only once every sampling period and use the result for the entire period. Since the visibility analysis dominates the running time of the entire algorithm, the sampling period is an important factor for the trade-off between accuracy and running time.

$$\begin{aligned}
\mathrm{Path_}\mathit{credit}(P,t) = \mathrm{Max}\{&\mathrm{Path_credit}(P,t-1) + \mathrm{Safety}(P,t),\\
&\mathrm{Path_credit}(Q,t-T\mathrm{cost}(Q,P)) + \mathrm{Safety_count}(Q,t-T\mathrm{cost}(Q,P)+1,t)\}
\end{aligned}$$

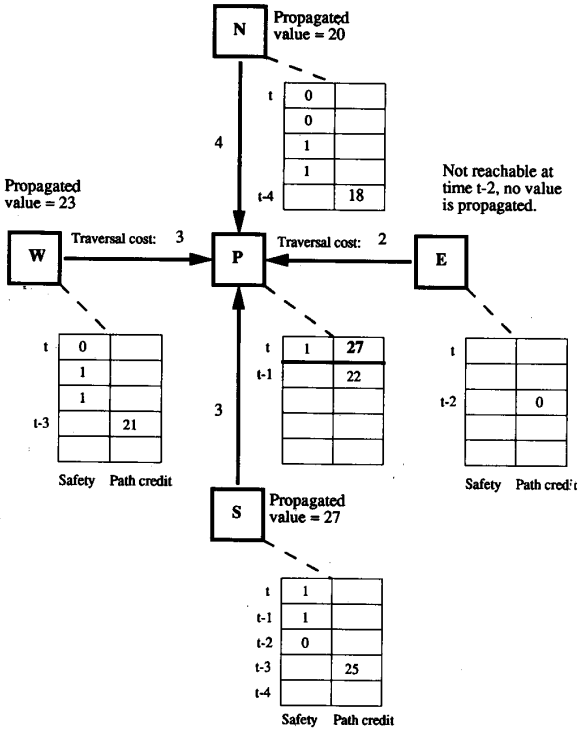


Fig. 10. Dilation with the consideration of terrain traversability. Cell P determines its path credit at time t by choosing the maximum among all received values and the value of staying at P from $t-1$ to t, which is 22+1 in this case. For each neighbor of P, the propagated value is the sum of all values shown in the two column table associated with each cell.

The algorithm can be made more flexible by setting different sampling periods for each adversary depending on its speed of motion to obtain even better efficiency.

VII. Experimental Results

In this section, we present the results from our implementation of the basic example and the extensions. All experiments were conducted on a Connection Machine CM-2 using 8K processors. The terrain size is 512 × 512 grid cells. We allocate one virtual processor to each grid cell, so the virtual processor ratio is 32.

A. Safe Path for a Single Agent

Figs. 11–14 illustrate an experiment of the basic planning algorithm, which returns an absolutely safe path to a subgoal. Fig. 11 shows the terrain and the initial locations. The friendly agent is located to the left of the center of the terrain, which is marked by a black dot in a framed square. The final goal, marked by a solid white square, is at the top of a hill near the bottom of the figure. There are four adversaries moving on the terrain. Their positions are marked by crosses and their predicted trajectories for this subplan interval are shown by the white line segments originating from their initial locations. For simplicity, we assume that each adversary moves along a straight line. The height of the terrain is represented by a gray level; lighter pixels are higher. Since most events occur in the lower part of the terrain, we will only show a 360 × 360 portion in the subsequent figures.

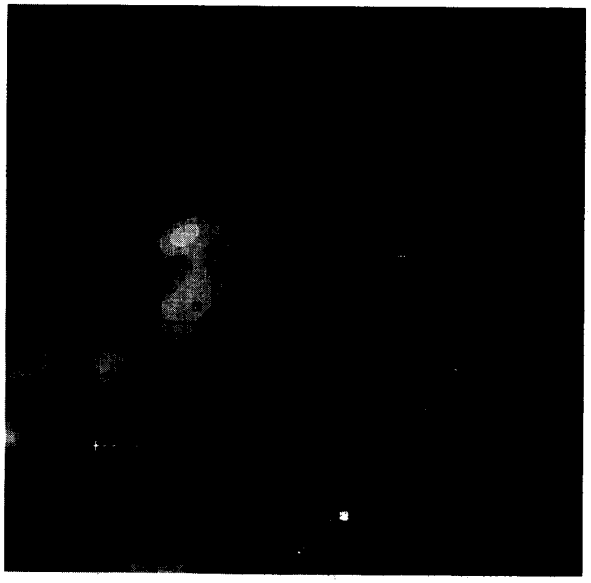


Fig. 11. Initial locations of the experiment.

Fig. 12. The situation at the 29th unit of time.

In this experiment, we plan for an interval of 32 time units, with a two-unit thinking time. During each unit of time, the friendly agent is able to move through two grid points, so the radius for each dilation is two. Our planner analyzes the visibility of each adversary at each time instant and constructs the RS structure. Fig. 12 shows the visibility analysis from one of the predicted adversary positions to the RS region at the 29th unit of time. The RS region is shown by the dark gray area around the initial location of the friendly agent,

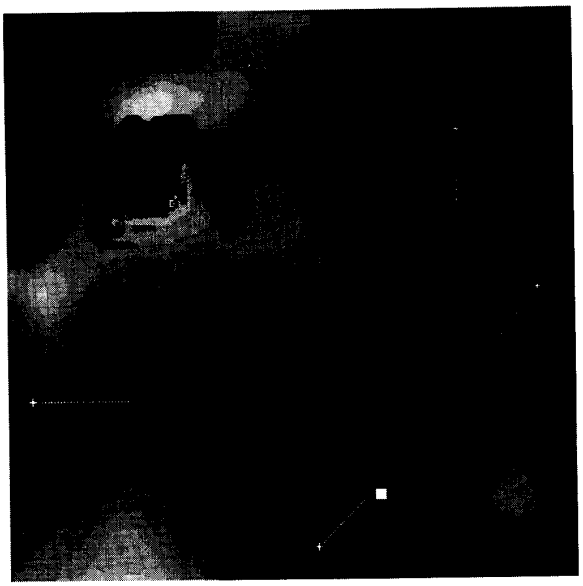


Fig. 13. The situation at the 30th unit of time.

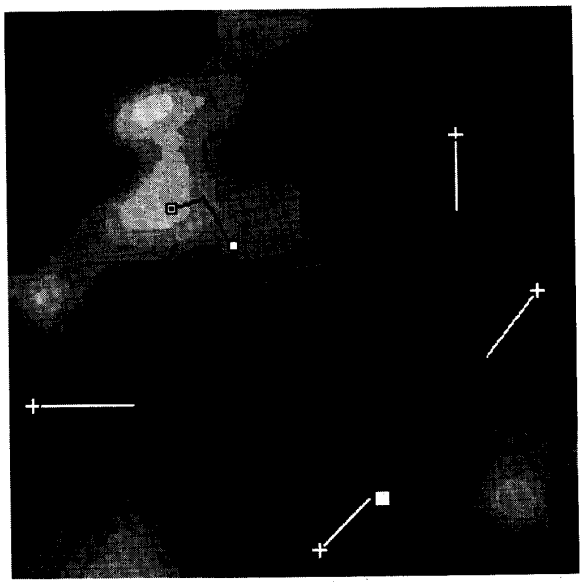


Fig. 14. At the end of the planning interval, a subgoal is chosen a safe and path to the subgoal is found.

which is now marked in white due to the dark background. The highlighted irregular area between the adversary position and the RS region indicates the area that is visible to this adversary. The intersection of the RS region and the visible region, shown by the black area between the two, is removed by the trimming operation and Fig. 13 shows the resulting RS region. If no safety constraint is applied, the RS region would be an octagon since we are using an octagonal dilation algorithm.

After the complete RS structure is built, a subgoal is chosen and a safe path to the subgoal is returned by the planner. See Fig. 14. Due to the safety constraint, the path has a detour

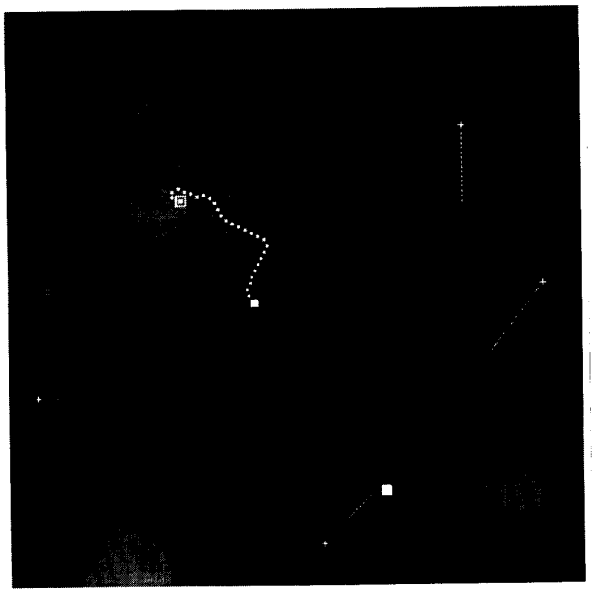


Fig. 15. The path for a group of agents.

toward the upper-right corner at the beginning of the interval, then makes a sharp turn back to the direction toward the final goal.

The overall running time for this experiment is about 100 s, including all graphic display and text messages. Without all these display operations, it can be shortened to 75 s. Visibility analysis is the dominating part of the running time. In our algorithm, visibility analysis is performed for each adversary at each time step, so it is performed 120 times in this experiment.

B. Planning for a Group of Agents

Fig. 15 shows the path returned by our planner for a group of three agents. The path shown in the figure is the path for the center of the group. All initial settings are the same as the previous example except that the friendly agents have a faster speed for a better display of the agent locations. After finding the path for the center, the locations of the other agents are determined by the direction of motion and the spacing between agents. Fig. 16 enlarges the region around the path and shows the configuration of the group as line segments. The three agents are located on both end points and the midpoint of each line segment. The safety and intervisibility of each agent is indicated by the brightness and the shape of the marks showing the agent locations. A white $\perp$ means a path point at which the agent is safe and visible to at least one of its partners. A black $\perp$ means a path point at which the agent is neither safe nor visible to any of its partners. A white dot indicates an agent location that is safe but not visible to its partners, while a black dot indicates the opposite situation.

C. Planning a Relatively Safe Path with Modeling of Terrain Traversability

Figs. 17–19 show three constituent subplans of a complete path from the initial location to the final goal. These paths are

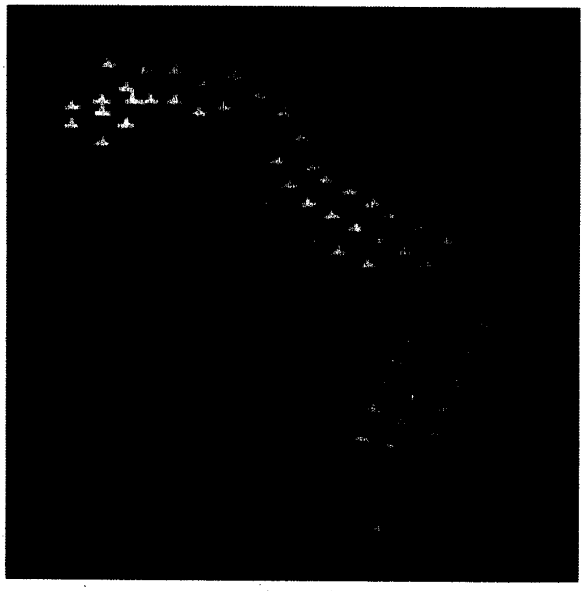


Fig. 16. The configuration of the group and the actual location of each agent.

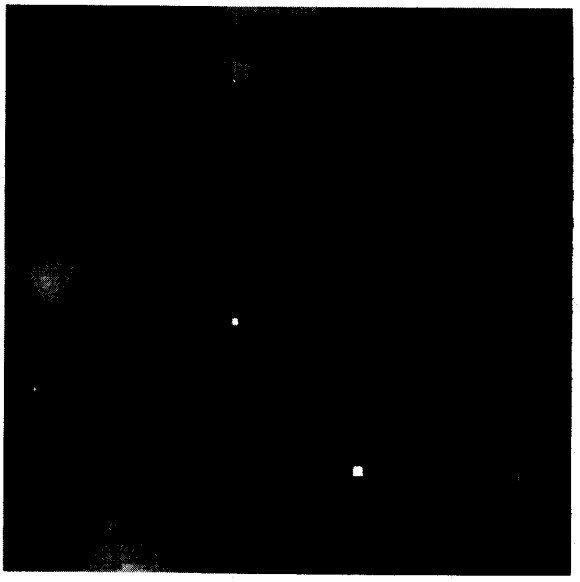


Fig. 17. The path of stage one.

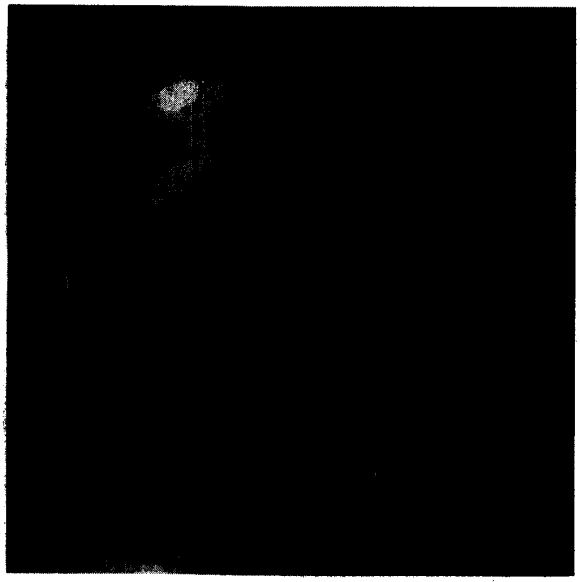


Fig. 18. The path of stage two.

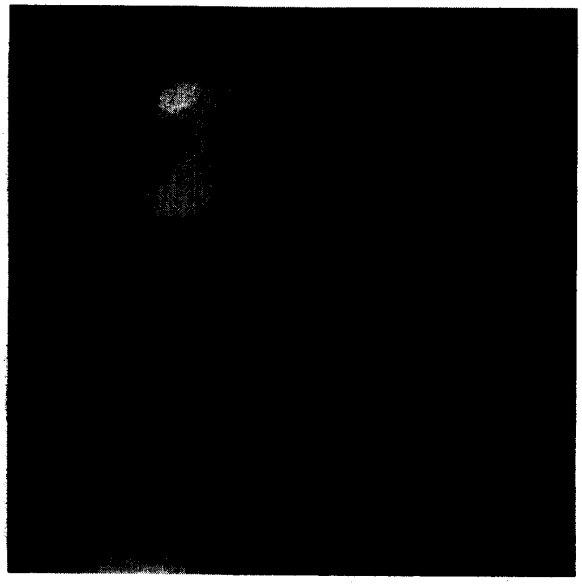


Fig. 19. The path of stage three.

relatively safe and take into consideration terrain traversability. The motions of the adversaries are assumed linear throughout the planned interval. We use the same notations for marking the positions except that the dark area around the friendly agent indicates the reachable region. The subgoal is chosen by a weighted function based on several criteria, so it may not be at the edge of the reachable region. Since we are using a finer scale of time and a four-neighbor propagation scheme, the path is four-connected on the terrain. The sizes and the shapes of the reachable regions vary according to the traversal cost. If all costs are the same, it should be in the shape of a diamond due to the four-neighbor propagation scheme.

VIII. Conclusion

In this paper, we have proposed methods for solving visibility-based terrain path planning problem using massively parallel hypercube machines. Since this kind of problem is known to be hard, our algorithms use approximations based on both temporal and spatial sampling. We first show how to solve a typical example, which plans for a single agent a path that is hidden from several moving adversaries. Next, we show that with certain modifications, we can solve a variety of path planning problems, including optimization among safety and other criteria, planning for a group of

agents under an intervisibility criterion, and consideration of terrain traversability. The time complexity is in general $O(T \times E \times \log N)$ using $O(N)$ processors, where T is the number of temporal samples, E is the number of adversary agents, and N is the number of grid cells on the terrain. Expected communication time is included in this complexity, and we also discussed practical methods for solving communication bottleneck. All algorithms are implemented on a Connection Machine CM-2.

In the future, we plan to develop algorithms that perform more complicated planning for a group of agents. We also plan to develop algorithms on polyhedral terrain models in order to plan a path on a larger scale of terrain with a limited number of processors. We may combine the results in a hierarchical manner so that the coarse-grained planner based on the polyhedral model generates a coarse plan that indicates where the subgoal should be for each stage and the fine-grained planner based on the grid model refines the coarse plan and generates an exact path.

REFERENCES

[1] F. Bitz and H. T. Kung, "Path planning on the Warp computer: Using a linear systolic array in dynamic programming," Tech. Rep. CMU-CS-87-180, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Aug. 1987.

[2] G. E. Blelloch and J. J. Little, "Parallel solution to geometric problems on the scan model of computation," in *Proc. 1988 Int. Conf. Parallel Processing*, 1988, pp. 218–222.

[3] G. Borgefors, "Distance transformations in digital images," *Comput. Vision, Graphics, Image Processing*, vol. 34, pp. 344–371, 1986.

[4] A. Borodin and J. E. Hopcroft, "Routing, merging, and sorting on parallel models of computation," in *Proc. Fourteenth Annu. ACM Symp. Theory of Computing*, 1982, pp. 338–344.

[5] J. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[6] R. Cole and M. Sharir, "Visibility problems for polyhedral terrains," *J. Symbolic Computation*, vol. 7, pp. 11–30, 1989.

[7] F. Dehne and Q. T. Pham, "Visibility algorithms for binary images on the hypercube and the perfect-shuffle computer," in *Parallel Processing*, M. Cosnard et al., Eds. Amsterdam, The Netherlands: North-Holland, 1988, pp. 117–124.

[8] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.

[9] L. Dorst and A. W. M. Smeulders, "Length estimators for digitized contours," *Comput. Vision, Graphic, Image Processing*, vol. 40, pp.311–333, 1987.

[10] L. Dorst and K. Trovato, "Optimal path planning by cost wave propagation in metric configuration space," in *Mobile Robots III, Proc. SPIE 1007*, W. J. Wolfe, Ed., 1989, pp. 186–197.

[11] M. Erdmann and T. Lozano-Pérez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–522, 1987.

[12] K. Fujimura and H. Samet, "Accessibility: A new approach to path planning among moving obstacles," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1988, pp. 803–807.

[13] ——, "Path planning among moving obstacles using spatial indexing," in *Proc. Int. Conf. Robotics and Automation*, 1988, pp. 1662–1667.

[14] ——, "Time-minimal paths among moving obstacles," in *Proc. Int. Conf. Robotics and Automation*, 1989, pp. 1110–1115.

[15] D. Hearn and M. P. Baker, *Computer Graphics*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[16] D.-M. Jung, "Comparisons of algorithms for terrain visibility," Master's thesis, Rensselaer Polytechnic Inst., Troy, NY, 1989.

[17] K. Kant and S. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. Robotics Res.*, vol. 5, pp. 72–89, 1987.

[18] ——, "Planning collision-free trajectories in time-varying environments: A two level hierarchy," in *Proc. Int. Conf. Robotics and Automation*, 1988, pp. 1644–1649.

[19] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, 1983.

[20] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, pp. 560–570, 1979.

[21] M. B. Metea and J. J.-P. Tsai, "Route planning for intelligent autonomous land vehicles using hierarchical terrain representation," in *Proc. Int. Conf. Robotics and Automation*, 1987, pp. 1947–1952.

[22] J. S. B. Mitchell, "An algorithmic approach to some problems in terrain navigation," *Artificial Intell.*, vol. 37, pp. 171–201, 1988.

[23] J. S. B. Mitchell and D. M. Keirsey, "Planning strategic paths through variable terrain data," in *Applications of Artificial Intelligence, Proc. SPIE 485*, 1984, pp. 172–179.

[24] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[25] J. Reif and S. Sen, "An efficient output-sensitive hidden-surface removal algorithm and its parallelization," in *Proc. 4th ACM Symp. Computational Geometry*, 1988, pp. 193–200.

[26] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," in *Proc. 26th Symp. Foundations of Computer Science*, 1985, pp. 144–154.

[27] A. Rosenfeld and J. Pfaltz, "Distance functions on digital picutres," *Pattern Recognition*, vol. 1, pp. 33–61, 1968.

[28] M. Sharir, "The shortest watchtower and related problems for polyhedral terrains," *Inform. Processing Lett.*, vol. 29, pp. 265–270, 1988.

[29] T. C. Smith *et. al.*, "AUV control using geometric constraint-based reasoning," in *Proc. Symp. Autonomous Underwater Vehicle Technology*, 1990, pp. 150–155.

[30] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A characterization of ten hidden-surface algorithms," *Comput. Surveys*, vol. 6, pp. 1–55, 1974.

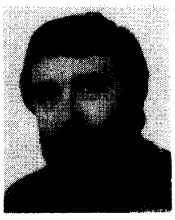
[31] L. Tychonievich *et. al.*, "A maneuvering-board approach to path planning with moving obstacles," in *Proc. 11th Int. Joint Conf. Artificial Intell.*, 1989, pp. 1017–1021.

[32] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proc. Thirteenth Annu. ACM Symp. Theory of Computing*, 1981, pp. 263–277.

[33] P. Verbeek, L. Dorst, B. Verwer, and F. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space," in *Proc. Int. Intelligent Autonomous Systems Conf.*, L. Herzberger and F. Groen, Eds., 1986, pp. 627–634.

Y. Ansel Teng was born in Tainan, Taiwan, on March 3, 1964. He received the B.S. Degree in electrical engineering from National Taiwan University, Taipei, in 1986. He is currently a doctoral candidate in the Department of Computer Science at the University of Maryland, College Park, and was awarded the Departmental Graduate Fellow in 1992. Since 1989, he has been a research assistant in the Center for Automation Research at UMD. His research interests include terrain navigation, parallel processing, geographical data processing, computational geometry, and visualization.

Mr. Teng is a student member of the Association for Computing Machinery.



Daniel DeMenthon received the graduate engineering degree from Ecole Centrale de Lyon, France, the M.S. degree in applied mathematics from Université Claude Bernard, France, and the M.S. degree in engineering from the University of California at Berkeley.

He is currently completing the Ph.D. degree in computer science at the Université Joseph Fourier, Grenoble, France. For the past seven years he has been a senior research engineer at the Computer Vision Laboratory of the Center for Automation Research at the University of Maryland, College Park. His research interests are in the areas of object recognition, planning, and 3-D user interfaces.

Larry S. Davis (S'74–M'77) received the Ph.D. degree in computer science from the University of Maryland, College Park, in 1976.

He is a tenured Professor in the Department of Computer Science and the Director of the University of Maryland Institute for Advanced Computer Studies. He has published over 50 articles on topics in image processing and computer vision.

Prof. Davis is on the editorial board of both *Computer Vision, Graphics, and Image Processing* and the *International Journal of Computer Vision*. He was Program Chairman of the 1988 Computer Vision and Pattern Recognition Conference, and General Chairman of the 1990 Frontiers 90 Conference on Massively Parallel Computing.