# Parallel Algorithms for One and Two-Vehicle Navigation

Eitan Gurewitz*, Geoffrey Fox, Yiu-fai Wong

Caltech Concurrent Computation Program
California Institute of Technology
Mail Code 206-49, Pasadena, CA 91125

## Abstract

A two vehicle navigator on a descrete space is analyzed. The concept of linking time maps as source to optimal path planning is discussed. The rules for constructing these maps are given in a cellular automata mode. The implementation of these rules on a parallel computer is presented.

## 1. Introduction.

In this study navigation means determination of a path on a navigation surface [NS] from an origin point to a destination point. A cost function is defined on the NS, measures the cost of traveling a length segment. The cost can be: time, length, hazard of traveling the segment etc. An optimal path on the NS is a path along which the integration of the cost function from origin to destination is minimum. The objective of a navigator is to find the optimal path under the constraints set by the NS. The problem of an optimal path for a single vehicle on a continuous surface [1] as well as a discrete surfaces [2] were solved. This study analyses the two vehicle navigator and presents the linking time maps as a tool to deal with these problems.

A discrete solution for navigation on a continuous space requires mapping of the space into a finite graph. This is done by choosing a finite number of points $\{v_i\}$ on the surface as the nodes of the graph. Each node is connected by an edge to all the nodes which can be reached, without traversing another node. The set of all the nodes $\{v_j\}$ having a common edge with $v_i$ is the set of $v_i$ nearest neighbors [nn(i)]. The value $w_{ij}$ of the cost of traveling along the directed edge $[v_i, v_j]$ is assigned to this directed edge. This procedure maps the surface onto a directed graph, Fig. 1. Mapping the NS onto a directed graph transfered the search for an optimal path to the the search for an optimal path on a directed graph. This search is solved by a dynamic programing approach [3], where a "signal" is initialized at a source point and propagates from a node to all its nn along the edges joining them. The time the signal travels along an edge is the weight of the directed edge. Every node $v_i$ records the first time $t_i$ it was hit by the signal. The graph in which all the nodes $v_i$ have their correct time values $t_i$ is called: the *linking time map* [LTM] with respect to the generating node.

In fact the linking time $t_i$ at the node $v_i$ is the cost of an optimal path from the source to this node and it depends only on the weights of the edges and the generating node. The linking times $t_i$ and $t_j$ of two sequential nodes $v_i$ and $v_j$ on an optimal path, where $v_i$ proceeds $v_j$ are related by:

$$t_j = t_i + w_{ij} \qquad (*)$$

An optimal path from the origin to any point on the graph is traced from that point back to the origin. Every step is from a node $v_j(t_j)$ to a node $v_i(t_i)$ where $t_i$ and $t_j$ satisfy (*). Tracing back ensures that one stays on an optimal path initialized at the origin.

Let us call the traveling object a vehicle and consider the case of two vehicles traveling on the same NS. If the path of each vehicle introduces restrictions to the path of the other vehicle (e.g. collision avoidance) then a search for an optimal solution is much more complicated.

The layout of this study is : In section 2 we discuss navigation of an autonomous vehicle on an NS which is updated while traveling. In section 3 we intoduce time and deal with conflicts between vehicles. The resolution of a conflict by imposing a delay on a vehicle is discussed and the paths solving a two vehicle navigator are analysed. Section 4 outlines briefly the algorithm for two vehicle navigator. Section 5 presents the cellular automata rules for constructing linking maps. Section 6 deals with the



0-8186-2113-3/90/0000/0140$01.00 © 1990 IEEE

actual parallel implementations of the construction of linking maps, and section 7 presents the simulation results.

## 2. Autonomous vehicle in uncertain environment

An autonomous vehicle in an uncertain environment start with an estimate of the edges weights. The estimate reflects the prior knowledge or model it has for the terrain to be transversed. The estimate is improved as more information is obtained. The vehicle knows its position and destination and at each instance of time the vehicle is doing the following: 1) updates the database of the weights $\{w_{ij}\}$. 2) Based on the updated data it determines the optimal path from its current position to the destination. 3) Moves on the chosen optimal path. 4) Collects data.

Updated weights $\{w_{ij}\}$ change the LTM, but a change in the linking time of a node may effect the linking times of only part of the other nodes. In section 5 we show how to update the LTM in a cellular automata fashion, based on local decisions of each node.

The navigator for an autonomous vehicle is based on the *reversed linking time map* [RLTM]. The construction of the RLTM is similar to the construction of the LTM. Except that in constructing the RLTM the signal is initializing at the destination point and propagates from $v_i$ to $v_j$ with traveling time of $w_{ji}$. The path is traced from the vehicle position toward the destination, from $v_i$ with reversed linking time $\theta_i$ to its nearest neighbour $v_j$ with reversed linking time $\theta_j$ which satisfies:

$$\theta_j = \theta_i - w_{ij}$$

Whenever the vehicle gets new information it updates the $\{w_{ij}\}$ database and its RLTM, and determines an optimal path, Fig. 2.

## 3. Navigation in Space-Time, and non conflicting paths for two vehicles.

Assume that the cost function is time, i.e. the weights $\{w_{ij}\}$ are the time of travel along the corresponding edges. Then a navigator for two vehicles aims to find two paths, one for each vehicle, which yield the minimum time of travel. Assuming the two vehicles start at the same time, then time of travel is the time it takes until both of them have arrived. This optimum is restricted to non conflicting paths.

*A conflict between two paths occurs when the two vehicles are at the same site at the same time.* The set of points on the graph edges is partitioned into sites as follows. Each point is associated to the nearest of the two nodes terminating the edge. A conflict can occur either: a) inside this site or b) a swap conflict on the boundary between two sites. In the second case the vehicles are going in opposite directions. Let $v_i^0, v_j^0, v_k^0$ and $v_{i'}^1, v_j^1, v_{k'}^1$ be three sequential nodes on the paths of vehicle 0 and vehicle 1 respectively. The node $v_j$ is on the two paths. A conflict of type (a) at $v_j$ occur if and only if

$$t_j^1 - \frac{w_{i'j}}{2} < t_k^0 - \frac{w_{jk}}{2}$$

and

$$t_j^0 - \frac{w_{ij}}{2} < t_{k'}^1 - \frac{w_{jk'}}{2}$$

A conflict of type (b) at the boundary between $v_j$ and $v_k$ occur if and only if:

$$i' = k$$

and

$$t_k^0 - \frac{w_{jk}}{2} = t_j^1 - \frac{w_{i'j}}{2}$$

To resolve the conflict at $v_j$ one vehicle cannot enter into the site until the other clears the site of $v_i$. In the graph representation this is done by imposing a *delay* $w$ at $v_i$ on either one of the two vehicles:

$$w = t_{k'}^1 - \frac{w_{jk'}}{2} - t_j^0 + \frac{w_{ij}}{2}$$

on vehicle 0, or

$$w = t_k^0 - \frac{w_{jk}}{2} - t_j^1 + \frac{w_{i'j}}{2}$$

on vehicle 1. Imposing a *delay* $w$ at $v_i$ on vehicle k means that $t_i^k$ is set to $t_i^k = t_i^k + w$ and $LTM^k$ is accordingly updated. *imposing a delay on a vehicle and updating its LTM preserves the characteristic of the LTM to yield, by the tracing back procedure, the optimal paths under the imposed restriction.*

If the optimal paths of vehicles 0 and 1 have more than one conflicting nodes then: 1) their path segments from the first to the last conflict have exactly the same time of travel. 2) On these equivalent segments they are traveling in the same direction. When the two paths have

more than one conflict, the resolution of each conflict requires the minimal *delay* given above. Therefore, imposing the maximal *delay* of these waits on the first node of conflict resolves all the conflicts between these two paths. However, the path with the *delay* on it may not be an optimal path anymore.

Consider the case where an optimal path of one vehicle conflicts, at $v_i$ with the optimal path of the other vehicle. Assume that the required *delay* at $v_i$ was imposed on one of the vehicles, its LTM was updated and a new optimal path was traced. Then one of the following will occur:

1. The new path does not conflict with the path of the other vehicle, and the are candidates for an optimal solution.
2. The new path conflicts with the path of the other vehicle, but it does not pass through $v_i$.
3. The new path passes through $v_i$ and it conflicts with the path of the other vehicle. In this case the new conflict is a swap conflict at the boundary between $v_i$ and its proceeding node on the other vehicle path.

In an optimal solution of the two vehicle navigator there cannot be an instant when the two vehicles are waiting. Therefore, the paths solving this problem can be of three types:

1. Neither of the vehicles waits.
2. One of the vehicles waits.
3. The two vehicles have to wait. The last case happens resolving a swap conflict when vehicle k has to wait for vehicle l to step aside letting k to path and then looping or detouring.

Let us extend the NS by adding to it the time dimension Fig. 3. The graph $\{v_i, e_{i,j}(w_{i,j})\}$ on the navigation plane is the projection of the extended graph on the $t = 0$ plane. The linking time value $t_i$ of a node $v_i$ is its t-coordinate in the extended space. The linking times $t_i$ and $t_j$ of two sequential nodes, $v_i$ and $v_j$, on a legal path in the extended space are restricted to the condition (1). In the extended graph *delay* means that two sequential nodes on a path have the same NS coordinates but different time coordinate. A loop means that two non sequential nodes on the path have the same NS coordinates but different time coordinate. A detour means that two sequential nodes on the path do not obey the path rule, i.e. $\theta_i + w_{ji} > \theta_j$.

The space-time representation of the paths depicts the difference between this problem and the K-disjoint[4] problem. In this problem we do not know the t-coordinates of the destination points. These points are subjected to the searching process.

The complexity of a search for an optimal solution for multiple vehicles grows fast with the number of vehicles. For this reason, other suboptimal methods are investigated, such as neural networks [5,6].

## 4. Algorithm for the two vehicle navigator.

The algorithm for the two vehicle navigator is based on the concepts discussed in the previous section using the cellular automata rules of the next section. The idea is to hold LTM and RLTM for each vehicle and to update them wenever a restriction is set. The need for a RLTM arises whenever a swap conflict occur, and a search for a loop or a detour is regarded.

As was already stated: the two vehicles cannot wait at the same time, and a solution which imposes *delays* on the two vehicles is obtained only when one of the paths is a loop or a detour. Therefore, the algorithm finds two separate solutions. A solution when the delays are imposed on vehicle 1 only and a solution where the delays are imposed only on vehicle 0. When imposing a *delay* to resolve a swap conflict the algorithm checks for loop or a detour. The best of these solutions is the optimal solution. In practice the algorithm will not construct those two solutions, but to minimize computations, it will prune the search by always adjusting the path of the vehicle with the sorter time.

On a binary speed NS the speed of the vehicle at each point is either 1 or 0. The two vehicle navigation problem on this NS is much easier as the rules get simpler form. on this NS a conflict of type (a) is at the node itself and it needs $w = 1$ to be resolved. The swap conflict (of type (b)) needs w=2 to be resolved. Fig. 4

presents the two vehicle navigator solution for a conflict imposing NS.

## 5. Cellular automata instruction for the navigation algorithm

Rule 1: *The linking time of the generating point is always* $t_o = 0$.

Rule 2: *The linking time* $t_i$ *of every node* $v_i i \neq o$ *is:*

$$t_i = Min\{t_i, t_j + w_{ji} | \forall j \in nn(i)\}$$

*,where nn(i) are all* $v_i$ *nearest neighbors.*

Rule 2': *The reversed linking time* $\theta_i$ *of every node* $v_i i \neq o$ *is:*

$$\theta_i = Min\{\theta_i, \theta_j + w_{ji} | \forall j \in nn(i)\}$$

Rule 3: *If a node other than the generator does not have a source, it set its linking time to infinity.* Namely, if $i \neq 0$, and $t_i > t_j \forall j \in nn(i)$ then $t_i = \infty$.

Rule 3': *If* $i \neq o, and \theta_i > \theta_j \forall j \in nn(i)$ then $\theta_i = \infty$.

**Algorithm for constructing the LTM or RLTM:**

1. Initialize the linking times of all the nodes to "infinity".
2. Set the generator linking time to 0.
3. Apply rule 2 or 2'.
4. When there is not a node which update its value the LTM or RLTM is done.

**Algorithm for updating the LTM or RLTM where a *delay W* is imposed on $v_i$:**

1. Set $t_i = t_i + W$ / $\theta_i = \theta_i + W$
2. Apply rule 3 / 3'.
3. Apply rule 2 / 2'.

## 6. Parallel implementation of the time-linking map

The cellular automata mode of constructing the LTM is asynchronous but the linking process has a propagating nature. The wave front of the propagating linking signal depend on the data and the location of the generating node. Therefore, the scattered decomposition[7] would be the most appropriate decomposition approach. The mapping in this approach is as follows: The NS is tessellated into $N_x$x$N_y$ congruent templet. Each templet is tessellated again to K equal tiles, where K is the number of processors. Each processor is assigned to the same tile of the templet over all the templets, Fig. 5. As the computational graph in our case is very irregular and time dependent, the scattered decomposition will hopefully balance the work done in each processor.

As the information propagates from a node to its neighbors the smaller the tiles in each templet are the greater the number of nodes propagating the correct linking time is. On the other hand the smaller the tile is the greater the number of nodes on the boundary is. Therefore, for given number of processors and dimension of the descrete NS there is an optimal size of tile. The bigger the number of processors is the smaller the size of the tile.

In planning the broadcast of the information one has to decide how many informing nodes to accumulate before transmitting the new data. On the one hand accumulating the information saves transmition time. On the other hand getting the information as soon as possible save updating and enables more templets to participate in the propagation process. In our simulation, we adopt the strategy of broadcasting the new information to the neighboring processor whenever a node on the boundary was updated. When the communication overhead is not too large, as in the case of the Meiko transputer board, the number of updates are kept to a minimum since the information delay is very small.

## 7. Simulation results

Extensive simulations have been carried out on an NS which was tessellated into 145 by 145 nodes. Fig. 6, is a plot of the speedup versus number of processors under different tile sizes. This plot shows that the 4 processors are the natural choice for the two-dimensional NS. For a given patch size, the speedup decreases with the number of processors. This is expected because of the propagated nature of the problem at

hand. The simulation shows, as depicted in Fig. 6, the optimal sizes of the example simulation. These sizes are: $19x19$ for 4 processors, $13x13$ for 8 processors, and approximatly $9x9$ for 16 processors. It shows the general trend that increasing the number of processors decreases the optimal size of the tile.

## 7. References


[1] Jones S. T, "Solving Problems involving Variable Terrain. Part 1: A General Algorithm", Byte, Vol. 5, No. 2, February 1980.

[2] Mitchell J. S. B and Papadimitriou C. H., " The weighted region problem", Tech. Rep., Department op Operations Research, Stanford University, Stanford, CA, 1985.

[3] Bellman, R., *Dynamic Progrramming* Princeton University Press, Princeton, New Jersey, 1957.

[4] Seymour, P. D., "Disjoint paths in graphs", Discrete Math. 29, pp. 293-309, 1980.

[5] Wong, Y. F., and Fox, G. C., " Use of Neural Network for Path Planning," Technical Report C3P-784, 1989.

[6] Fox, G. C., Gurewitz, E., and Wong, Y. F., "A neural network approach to multivehicle navigation", SPIE Vol. 1196, pp. 164-169, 1989.

[7] Salmon J., and Goldsmith J., " A hypercube Ray-Tracer", 3ed Conf. on Hypercube Concurrent Computers an Applications", pp. 1194-1206, Pasadena, CA 1988.


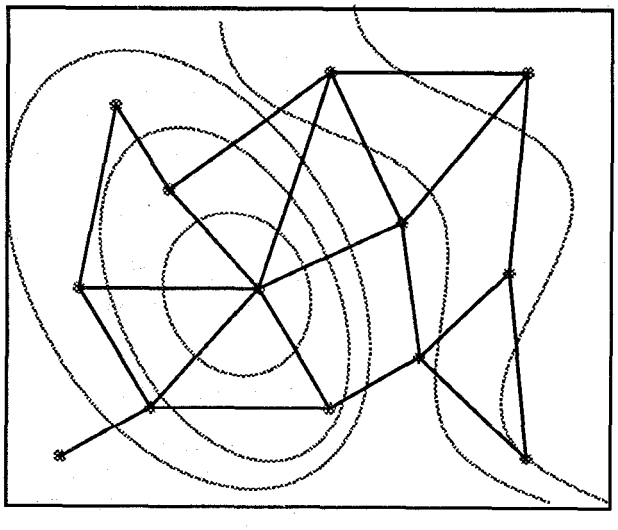

Figure 1. mapping of a terrain onto a graph

* On a leave of absence from NRCN Israel.

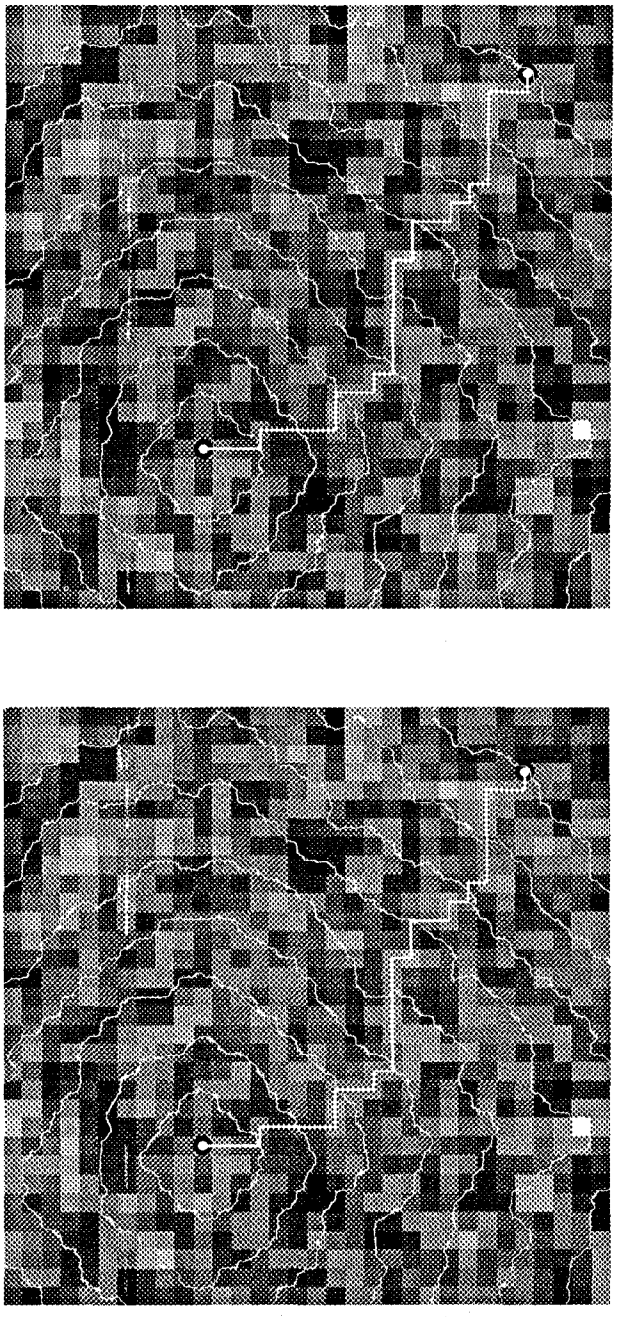

Figure 2. Autonomous vehicle in uncertain environment. The gray level of an area is proportinal to its cost. The white lines are the equi-cost contours After a short travel along the optimal path (a) the vehicle updated its data and determined a new optimal path (b).

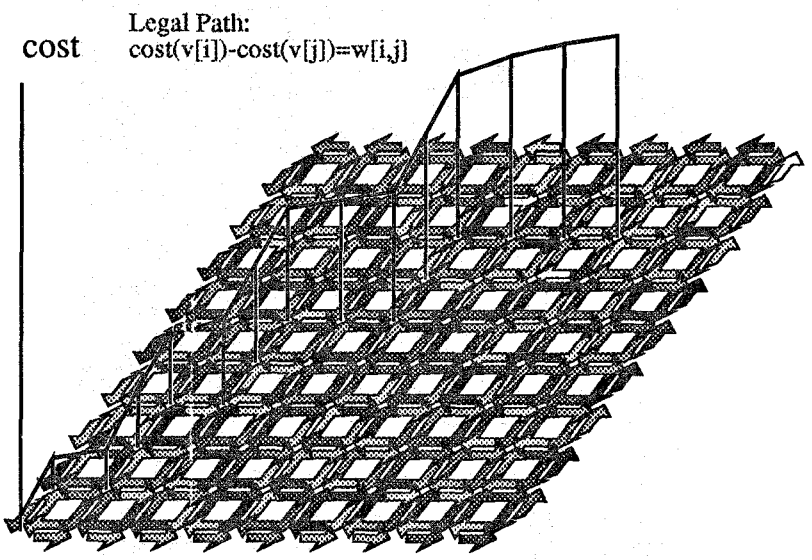


Figure 3. Nodes, terrain's directional values (gray level arrows) and a path in the Cost-Terrain space.

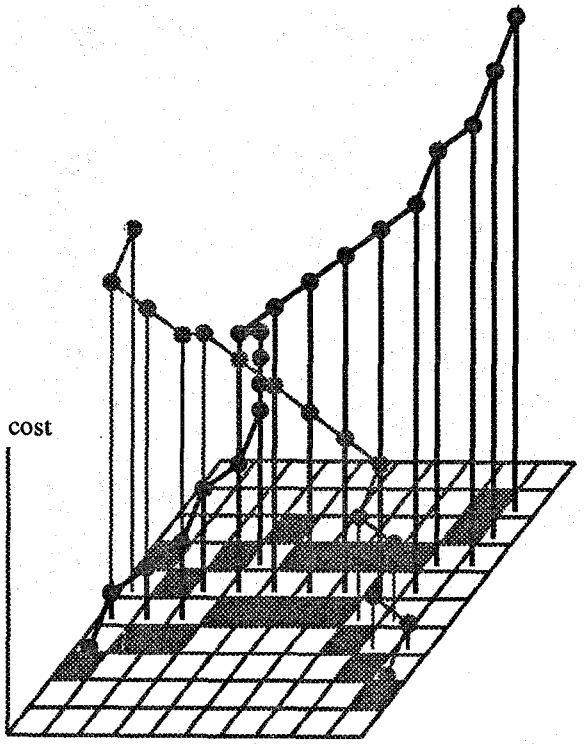


Figure 4. The two-vehicle navigator solution for a conflict imposing terrain and a path in the Cost-Terrain space.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 |

Figure 5. Scattered decomposition, the basic template of 4 processors is repeated over the terrain.

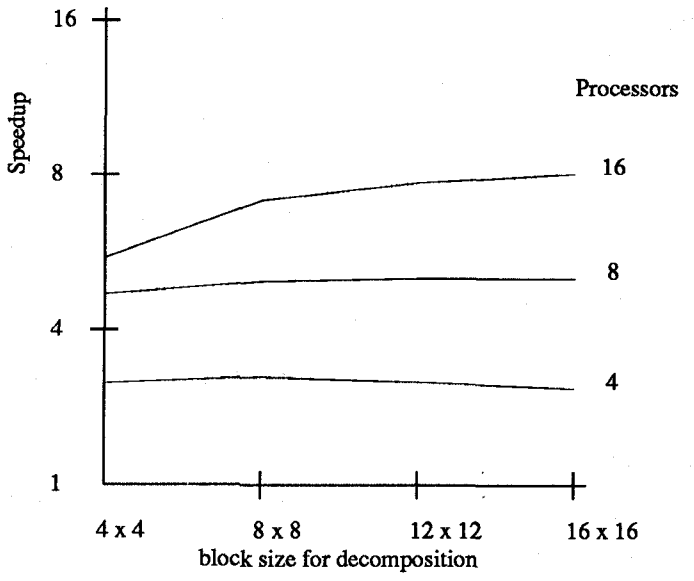


Figure 6. Speedup for decomposition scheme for different block sizes on 16-node Meiko Computing Surface