SIMPLE PATH-PLANNING ALGORITHM FOR MOBILE ROBOTS USING QUADTREES

Jozef V ő r ő s

Slovak Technical University
Faculty of Electrical Engineering and Information Technology
Ilkovicova 3, 812 19 Bratislava, Slovakia

Abstract: The paper presents a simple path-planning algorithm that selects a reasonable collision-free path tying start and goal points and using a quadtree representation of the robot workspace. A new method for neighbor finding in binary images, which combines two forms of quadtree representation - matrix and linear quadtrees, is used in a path planning procedure in which the obstacle regions and further every free region smaller than the robot sizes are regarded as forbidden regions. The sequence of free region quadrants includes at least a collision-free path of the mobile robot by connecting all center points in every edge where the free regions are close each other. More illustrative examples are included.

Keywords: Mobile robots; image modelling; path planning; quadtrees.

1. INTRODUCTION

Conventional path-planning algorithms can be divided broadly into two categories (Kambhampati and Davis, 1986). In the first category are the methods which make trivial (if any) changes to the representation of the image map before planning a path (regular grid search, vertex graph methods). The methods in the second category make elaborate representation changes to convert to a representation, which is easier to analyze before planning the path (transform methods, Voronoi method). Though the above two categories by no means exhaust the existing methods, they point out that what mobile robots need may be a compromise between these two categories.

In the previous work more approaches have been developed for mobile robot path planning based on a hierarchical tree representation of the robot's environment. In these cases if there are large areas of free space (or obstacles), then those areas can be represented by a few large blocks in the corresponding quadtree and can be dealt with as units by the planning algorithm (Kambhampati, Davis, 1986), (Noborio et al, 1990). The resulting path is a list of adjacent quadtree nodes connecting the start and goal points. Evidently, the neighbor finding procedure is crucial in all the path planning algorithms using quadtree representations.

Recently a new algorithm has been developed for neighbor finding which combines two forms of quadtree representation - matrix and linear quadtrees, respectively (Vörös, 1995). This can be applied to any node even on the border of the image and can generate neighbors not only of greater and equal size, but also smaller ones. The image of robot workspace with forbidden and free areas is represented by a quadtree in the matrix form, while the generation of neighbors in four main directions is implemented by the corresponding linear (locational) codes. In this paper the above mentioned neighbor finding procedure is used in a simple path planning method in which the obstacle regions and further every free region smaller than the robot size are regarded as forbidden regions. The problem is restricted to two-dimensional path-planning without rotation and a vehicle with circular cross-section. The algorithm can select a reasonable path composed of other free regions from the quadtree. The sequence of free regions includes at least a collision-free path of the mobile robot by connecting all center points in every edge where the corresponding quadtree nodes are close each other.

2. QUADTREE REPRESENTATIONS

2.1 Matrix quadtree

A quadtree representation of binary picture is a tree whose leaves represent square areas or quadrants of the picture and are labeled with the color of corresponding area, i.e. BLACK, WHITE or GRAY (mixed). The quadtree A corresponding to a (2^N x 2^N)-dimensional binary picture can be characterized as an ordered list, actually a four-column matrix, of all the GRAY node descriptions (Vörös, 1994)

$$A = [P_1, P_2, P_3, \dots]^T . \qquad (1)$$

The record describing a GRAY node consists of four entries and for the i-th node at the k-level of quadtree representing (2^k x 2^k)-dimensional region it can be written as an ordered quadruple

$$P_i = (Q_{i1}, Q_{i2}, Q_{i3}, Q_{i4}), \qquad (2)$$

where P_i denotes the parent node and Q_{im}, m = 1, ... , 4, denote its four sons at the (k-1)-level. If Q_{im} is a terminal-leaf node, we can substitute two distinct non-positive integers for the corresponding color into (2), e.g.:

$$Q_{im} = \begin{cases} -1 & \text{for BLACK quadrant,} \\ 0 & \text{for WHITE quadrant.} \end{cases} \qquad (3)$$

If Q_{im} is a nonterminal node, it is identified with a new parent node being at the (k-1)-level, i.e.

$$Q_{im} = P_j \qquad \text{for GRAY quadrant,} \qquad (4)$$

j > i, and it means that for P_j a new quadruple of form (2) is created in the same way as for P_i. Actually, P_j is an integer number pointing to the row of matrix (1) where this GRAY subquadrant is described.

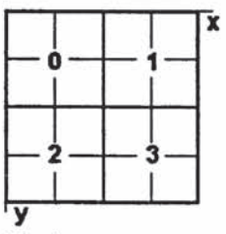

Fig. 1

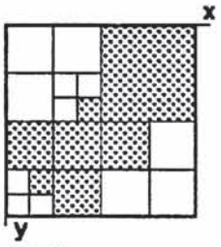

Fig. 2

For example, if the order of quadrants is chosen according to Fig.1, the binary picture in Fig. 2 is described by the matrix quadtree given in the following table.

Table 1

P_i/Q_i	0	1	2	3
1	2	-1	3	4
2	0	0	0	5
3	-1	-1	6	-1
4	-1	0	0	0
5	0	0	0	-1
6	0	-1	0	0

However, for the path planning process it will be more convenient to use another substitution. As the most relevant nodes are those of WHITE color, i.e. the free or nonobstacle nodes, in this paper we will use the following substitutions:

$$Q_{im} = \begin{cases} -m & \text{for free quadrant,} \\ 0 & \text{for forbidden quadrant,} \end{cases} \qquad (3')$$

where m > 0 is any integer number. Generally, the matrix quadtree is proper for storing further relevant data (weights, colors) bound with the chosen type of nodes by using different negative integers.

2.2 Linear quadtree - Locational codes

Pointerless quadtree representations are used because they may lead to significant savings in space as there is no need to store pointers. The linear quadtree technique is the best known representative and has gained increasing use (Gargantini, 1982).

In the linear quadtree, each leaf node can be assigned a unique locational code corresponding to a sequence of directional codes that locate the leaf along a path from the root of the tree. Gargantini's original algorithm computes the value of the locational code for BLACK nodes only. Each BLACK node of (2^N x 2^N)-dimensional binary image is encoded as

$$A = (A_0 A_1 \dots A_N), \qquad (5)$$

with digits 0, 1, 2, 3 for A_i and each successive digit represents the quadrant subdivision from which it originates. The collection of BLACK node descriptions is usually represented as a list whose elements appear in increasing order of locational codes. Such an ordering is useful because it is the same order in which the leaf nodes of a pointer-based quadtree would be visited by a depth-first traversal.

A modified variant of block encoding is shown in Fig. 3. The quadrant locational codes are represented by a variable number of base 4 digits, i.e. n digits for a node at the n-th level of quadtree. The codes are equivalent to a depth-first traversal such that quadrants are visited in the order given by Fig. 1.

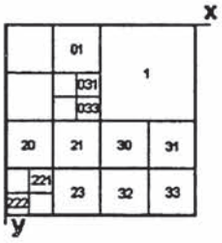

Fig. 3

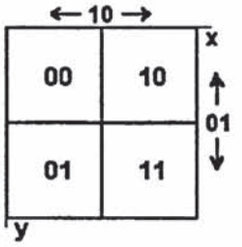

Fig. 4

This method of addressing blocks indicates the size of blocks relative to the image dimensions implicitly, by the number of digits. The base 4 digits of locational code can be

represented in the binary form. A node at the n-th level of quadtree is then given as

$$A = (x_0, y_0 \; x_1, y_1 \; x_2, y_2 \; . \; . \; x_n, y_n) \quad (6)$$

where x_i, y_i are 0 or 1, and $0 \le n \le N$. This form of representation for free quadrants will be used in this paper to simplify the neighbor finding.

3. NEIGHBOR FINDING

3.1 Neighbor generation

Many of the special image operations involve the so-called neighbor finding procedure, i.e. the examination of nodes whose corresponding blocks are spatially "adjacent" to the block corresponding to the node being processed. Note that adjacency in space does not imply any simple relationship among the nodes in the tree.

In the prior work both pointer and pointerless representations of quadtrees were used in neighbor finding algorithms. For linear quadtrees two cases are distinguished (Gargantini, 1982): in the first one, two neighboring nodes belong to the same quadrant relative to the n-th subdivision; in the other: they do not. In the second case, transition from two 'different' quadrants occurs. However, the number of these transitions must be extra investigated. These procedures can be used only to find nodes adjacent to an internal node.

In another known approach, for trees that are implemented using both a pointer and a pointerless representation, a neighbor of greater than or equal size is computed in the given direction. The basic idea is to ascend the tree until a common ancestor is located, and then descend back down the tree in search of the neighboring node. Properly defined table functions are used (Samet, 1989). This procedure ignores the situation in which a neighbor may not exist (e.g. when the node is on the border of the image).

As the neighbor finding is fundamental for the presented path planning method, a new approach was developed using both matrix and linear quadtree representations (Vörös, 1995). This enables to find greater, smaller and equal sized neighbors to all the quadtree nodes.

First we give some definitions and explain our notation. We say that node B is a *neighbor* of node A in direction D if B corresponds to the block adjacent to A in direction D of size greater than, equal to or smaller than the block corresponding to A. A node can have no neighbor, one or more neighbors in a chosen direction. In general, the neighbors need not correspond to blocks of the same size. In the following the neighbors in 4 directions (left, right, up, and down) will be considered only. These will be concentrated in 2 generalized directions, i.e. D = [1,0] for left and right, and D = [0,1] for up and down.

Now we show how to alter the address of a node (i.e. its locational code) by changing the appropriate digits of the address to match the address of the desired neighbors of equal size. Assume that we are trying to find the equal-sized neighbor B of node A in direction D. We distinguish two cases: in the first one, A and B belong to the same quadrant relative to the n-th subdivision, i.e. we speak about an inner neighbor; in the other: A and B do not and hence we speak about an outer neighbor.

Let A be a node at the n-th level of quadtree given by its locational code in the binary form as $\{x_i, y_i\}$, i = 0, 1, .. , n. There exists one *equal-sized inner neighbor* B in the direction $D = [d_0, d_1]$ and its binary locational code $\{\underline{x}_i, \underline{y}_i\}$ is given as follows:

$$\underline{x}_j = x_j, \quad j = 0, 1, \ldots, n\text{-}1, \quad (7)$$
$$\underline{y}_j = y_j, \quad j = 0, 1, \ldots, n\text{-}1, \quad (8)$$
$$\underline{x}_n = x_n + d_0, \quad (9)$$
$$\underline{y}_n = y_n + d_1, \quad (10)$$

where the addition is modulo 2. It means that only the lowest level code is changed in the locational code of node A. Actually, as D can be only [1,0] or [0,1], the node A and its inner neighbor B differ only in one bit. Note that every node has an inner neighbor in both generalized directions.

To find equal-sized outer neighbors the number of needed changes must be determined. Let k > 0 be an integer such that

$$x_n = x_{n-1} = \ldots = x_{n-k+1} \neq x_{n-k}. \quad (11)$$

Then there exists one *equal-sized outer neighbor* B in the direction *D = [1,0]* and its binary locational code $\{\underline{x}_i, \underline{y}_i\}$ is given as follows:

$$\underline{x}_{n-i} = x_{n-i} + 1, \quad i = 0, 1, \ldots, k, \quad (12)$$
$$\underline{x}_i = x_i, \quad i = 0, 1, \ldots, n\text{-}k\text{-}1, \quad (13)$$
$$\underline{y}_i = y_i, \quad i = 0, 1, \ldots, n. \quad (14)$$

where (12) is again modulo 2. Similarly, if k > 0 is an integer such that

$$y_n = y_{n-1} = \ldots = y_{n-k+1} \neq y_{n-k}, \quad (15)$$

then there exists one *equal-sized outer neighbor* B in the direction *D = [0,1]* and its binary locational code $\{\underline{x}_i, \underline{y}_i\}$ is given as follows:

$$\underline{y}_{n-i} = y_{n-i} + 1, \quad i = 0, 1, \ldots, k, \quad (16)$$
$$\underline{y}_i = y_i, \quad i = 0, 1, \ldots, n\text{-}k\text{-}1, \quad (17)$$
$$\underline{x}_i = x_i, \quad i = 0, 1, \ldots, n. \quad (18)$$

The values resulting from (11) and (15) are called the "k-values" of nodes for the corresponding directions.

It means that in both cases always "k+1" bits must be changed. At the same time the k-value determines if it is even possible to find an outer neighbor. Namely, if $k > n$, there exists *no outer neighbor*. In both cases above, the k-value reflects, figuratively speaking, the 'closeness' of quadrants to a common border line parallel with x or y-axis, respectively.

Using the locational code of the equal-sized neighbor B of node A, all the possible neighbors of greater or smaller dimensions can be found. Evidently, a node A can have only outer neighbors of greater dimension than that of A. *Greater-sized outer neighbors* of a node A in the direction D are ${}^{j}B = \{{}^{j}\underline{x}_i, {}^{j}\underline{y}_i\}$ whose binary locational codes are received from the code of the corresponding equal-sized neighbor by successive canceling the lowest level codes up to the (n-k)-th level, i.e.

$$
\begin{aligned}
{}^{j}B = ({}^{j}\underline{x}_0, {}^{j}\underline{y}_0 \\
{}^{j}\underline{x}_1, {}^{j}\underline{y}_1 \\
. \\
. \\
{}^{j}\underline{x}_{n-j}, {}^{j}\underline{y}_{n-j})
\end{aligned}
\qquad (19)
$$

where $j = 1, .. ,k$, and k is given by (11) or (15). Hence the number of greater outer neighbors is equal to the corresponding k-value of node A.

The *smaller-sized neighbors* of quadrant A are the couples of adjacent subquadrants of its equal-sized neighbor. They may be both inner and outer and are determined by the code of corresponding equal-sized neighbor $B = \{\underline{x}_i, \underline{y}_i\}$, $i = 0, 1, .., n$. Their binary locational codes are

$$
\begin{aligned}
{}^{j}B = ({}^{j}\underline{x}_0, {}^{j}\underline{y}_0 \\
{}^{j}\underline{x}_1, {}^{j}\underline{y}_1 \\
: \\
{}^{j}\underline{x}_n, {}^{j}\underline{y}_n \\
{}^{j}\underline{x}_{n+1}, {}^{j}\underline{y}_{n+1} \\
: \\
{}^{j}\underline{x}_N, {}^{j}\underline{y}_N)
\end{aligned}
\qquad (20)
$$

where for D = [0,1] the codes of *outer* neighbors are

$$
{}^{j}\underline{x}_{n+i} = d\,, \qquad (21)
$$

$$
{}^{j}\underline{y}_{n+i} = \underline{y}_n\,, \qquad (22)
$$

and those of *inner* ones are

$$
{}^{j}\underline{x}_{n+i} = d\,, \qquad (23)
$$

$$
{}^{j}\underline{y}_{n+i} = \mathrm{inv}(\underline{y}_n)\,, \qquad (24)
$$

where d is 0 or 1 and $\mathrm{inv}(\underline{y}_n)$ means the inverse of $\underline{y}_n$, while

$$
{}^{j}\underline{x}_i = \underline{x}_i\,, \qquad i = 0, 1, \ldots, n\,, \qquad (25)
$$

$$
{}^{j}\underline{y}_i = \underline{y}_i\,, \qquad i = 0, 1, \ldots, n\,. \qquad (26)
$$

Analogously, the codes for D = [1,0] can be determined.

Finally note that real directions for neighbors can be easily determined from the changes of the lowest level codes of generalized ones. For example, if x changes from 0 to 1 then D = [1,0] means that the inner neighbor is on the right side and the outer one is on the left side and vice versa.

3.2 Quadtree traversal

In practice, computing locational codes of nodes takes only a small fraction of the total execution time required by the neighbor finding. By a linear quadtree-based approach, searching and updating the node list usually takes up the vast majority of execution time (Gargantini, 1982). In the case of pointer-based quadtrees, traversal in two directions is done by ascending father links to the nearest common ancestor of the node and its neighbor, then descending the tree to the neighbor (Samet, 1989).

In our approach the search for neighbors is made in the matrix quadtree representation. By definition the entries (3) of matrix are -1 and 0 for BLACK and WHITE leaf nodes, and positive integers for GRAY nodes, referring to the row where the corresponding subquadrant is described. This simplifies the search for the desired neighbor node given by its locational code. If the columns of matrix are identified with the binary codes according to Fig. 4, it is easy to locate the node in the quadtree and to find out what type of node corresponds to the computed locational code.

The matrix quadtree traversal by neighbor finding is only in one direction. It is restricted to simple operations with binary data of corresponding locational codes and recognizing the respective matrix entries.

4. PATH PLANNING

Given a binary array or raster representation of a robot's environment or workspace, this will be converted into a matrix quadtree representation using substitutions (3'). Then according to the radius of robot's cross section we majorize the quadtree extending the obstacles and shrinking the free space (Vörös, 1992). This is a trivial operation because the matrix quadtree representation is a top-down one. In the resulting 'pruned' quadtree nonobstacle blocks represent free-space nodes and the other blocks represent obstacle nodes.

Now given the start and goal points, we first determine the quadtree leaf nodes S and G, representing the regions of the image containing these points. Next we plan a path between S and G formed by the nonobstacle nodes of the quadtree. This involves always finding an optimal nonobstacle leaf node adjacent to the node being processed and beginning with the node S.

The above described neighbor finding method can be used to accomplish effectively the distance transform of quadtree by marking every free node with its distance from the goal node. The matrix quadtree enables direct storing of these extra data as they can be put into the corresponding matrix entries as integers with minus sign. Note that the marking of quadtree nodes is a trivial operation performed right with the quadtree traversal. In this case all paths from the goal

node back to any start node are considered. Then from any starting node in the workspace, the shortest or distance optimal path to the goal is traced by following the path of steepest descent. The main disadvantage of this planning approach is the necessity of (possible multiple) visiting of all the free nodes.

The neighbor finding strategy described in the preceding section, where only the neighbors in the horizontal and vertical directions are considered, was applied in the following very simple directional-oriented path planning method. First, the optimal neighbor (i.e. inner or outer) is generated in the dominant direction towards the goal, i.e. [0,1] or [1,0]. If there is no such node, other existing neighbors are generated and that of minimum distance in a straight line to the goal is included into the path. The process starts in the node S and finishes after reaching the neighbor identical with the node G. The matrix entries corresponding to the included nodes are marked during the process to prevent back-runs. This can be done by substituting proper negative integers for -1's into the matrix. Also two types of shortcomings are to be overcome in the proposed algorithm - the so-called "dead-end" and "roundabout". The first one may lead to a failure of path planning as no free node can be found, while the second leads to less optimal path with unnecessary (detour) nodes being traveled through.

The result of applying the proposed algorithm to the quadtree representation of workspace is a list of nodes from the quadtree (ordinarily of varying sizes) which defines a path between the start and goal nodes. If desired, an optimal path through these blocks can be computed, or the center points of consecutive blocks on the list can be connected to compute a negotiable path.

To illustrate the feasibility of proposed path planning approach three examples are presented. They are compared with the results of the above mentioned distance transform path planning method.

In Fig. 5a there is a simple example of a path obtained using this algorithm. A binary array is shown with start and goal points marked, along with an indication of the path determined by the algorithm. If compared with the distance optimal path shown in Fig. 5b, resulting from the distance transform of quadtree representation, it can be seen that the difference is minimal. However, there is a significant difference in the processing time. While the execution time for the distance transform method was 1,9230 sec., it was only 0,1099 sec. for the new directional oriented method (on PC 386SX, 20 MHz).

A more complex example is shown in Fig. 6a, where the proposed method had to cope with the dead-end problem. Despite this fact the generated path is not much longer than that of distance transform in Fig. 6b, but its execution time 0,2747 sec. is much smaller than 2,5824 sec. for the path in Fig. 6b.

Even in the workspace full of obstacles in Fig. 7a the difference between the directional oriented path and the optimal one (Fig. 7b) is minimal, while the execution times were 0,1099 sec. to 2,1978 sec.

Note that the same examples are chosen as in (Kambhampati, Davis, 1986) and (Noborio et al, 1990) to enable comparison between the quadtree-based path planning algorithms presented so far and the proposed simple directional-oriented one.

5. CONCLUSION

In this paper a simple method of path planning for mobile robots using quadtree hierarchical data structures has been presented. The matrix/linear quadtree based approach is a good compromise between the above mentioned categories of methods. In addition, the path produced by the proposed algorithm, although not "optimal", is a "negotiable" path which can be computed relatively quickly. Corner-clipping inflexible paths are eliminated by considering only neighbors in the horizontal and vertical directions.

Compared with the similar path-planning algorithms the cost for quadtree-based search will be substantially lower because the number of nodes to be searched in the quadtree is considerably smaller. The changes to the quadtree representation of the robot workspace because of marking the visited nodes are minor and reversible.

Finally it is worthy to note that the proposed neighbor finding algorithm and the path planning approach can be easily extended to 3D case using octree representations in matrix and linear forms and adopting properly the neighbor generation rules.

REFERENCES

Gargantini, I. (1982). An effective way to represent quadtrees. *Communication ACM*, Vol. **25**, No. 12, pp. 905-910.

Kambhampati, S. and L.S. Davis (1986). Multiresolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, Vol. **RA-2**, No. 3, pp. 135-145.

Noborio, H., T. Naniwa, and S. Arimoto (1990). A quadtree-based path-planning algorithm for a mobile robot. *Journal of Robotic Systems*, Vol. **7**(4), pp. 555-574.

Samet, H. (1989). Neighbor finding in images represented by octrees. *Computer Vision, Graphics, and Image Processing*, Vol. **46**, pp. 367-386.

Vörös, J. (1992). Approximate quadtree and octree representations for manufacturing tasks. *Preprints 3rd IFAC Low Cost Automation, Wien*, pp. 391-396.

Vörös, J. (1994). Top-down generation of quadtree representation using color-change-code. *Computers and Artificial Intelligence*, Vol. **13**, No.1, pp. 91-103.

Vörös, J. (1995). Neighbor finding in images using matrix and linear quadtree representations. (submitted to *IEEE Trans. Pattern Analysis and Machine Intelligence*).

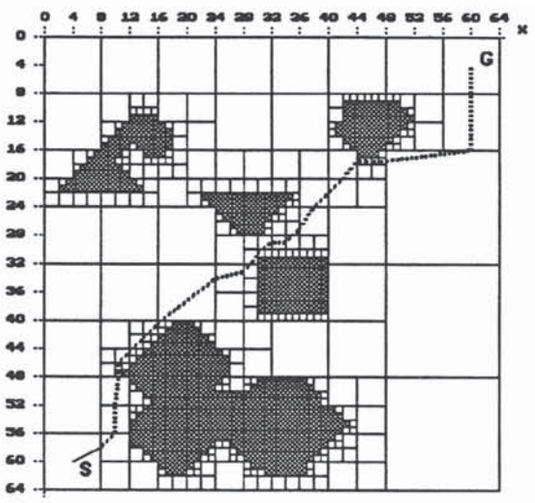

Fig. 5a

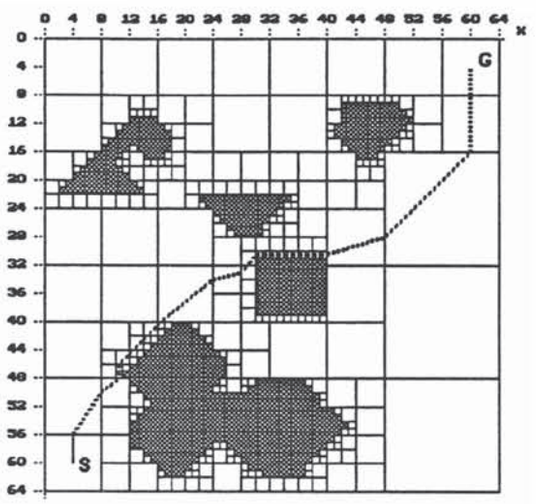

Fig. 5b

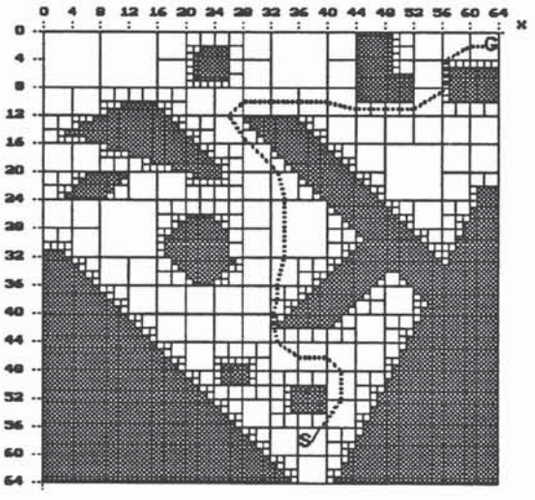

Fig. 6a

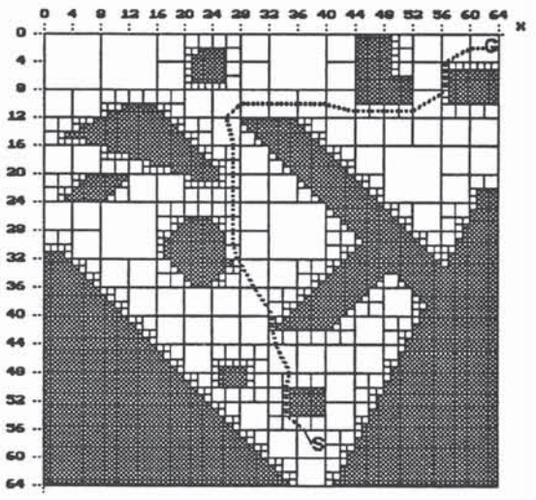

Fig. 6b

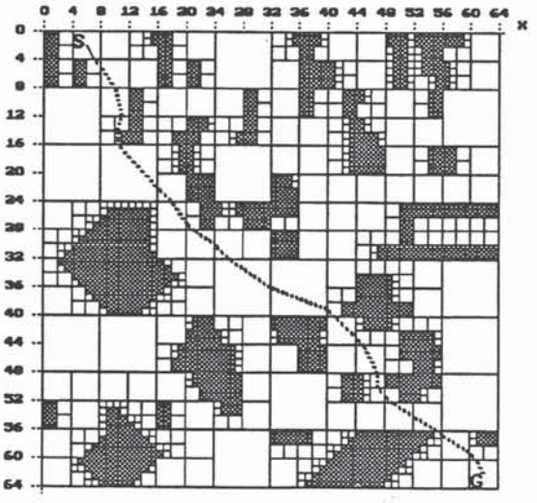

Fig. 7a

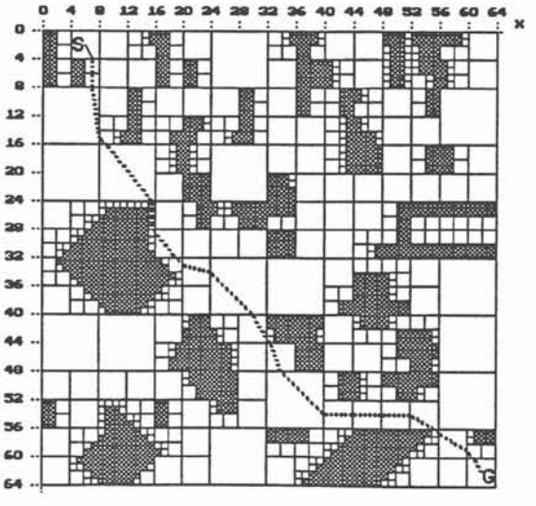

Fig. 7b