A Quadtree-Based Path-Planning Algorithm for a Mobile Robot

Hiroshi Noborio
Department of Precision Engineering
Osaka Electro-Communication University
Neyagawa, Osaka 572, Japan

Tomohide Naniwa
Department of Mechanical Engineering
Osaka University
Toyonaka, Osaka 560, Japan

Suguru Arimoto
Department of Mathematical Engineering and Information Physics
University of Tokyo
Bunkyo-ku, Tokyo 113, Japan
Received May 19, 1989; accepted March 21, 1990

To enable a mobile robot to select automatically a collision-free path in a given workspace, design of a path-planning algorithm which must work efficiently in real-time is crucial. This article proposes a path-planning algorithm that selects a reasonable collision-free path tying start and goal points out of a quadtree representation of the robot workspace. The quadtree is obtained from fast conversion of a real image taken through a camera on the ceiling. It represents obstacles and their allocation in the workspace in good time and hence the algorithm is able to find a collision-free path while following the change of obstacles and their allocation. The algorithm is designed on the basis of "small-is-quick" principle. That is, the smaller a search space of the algorithm is, the faster the algorithm selects the shortest path out of the search space. To put the principle in practice, the algorithm investigates a path graph instead of the quadtree while spreading the path graph on the quadtree as small as possible, and selects fast the shortest collision-free path out of the path graph as a reasonable collision-free path. Thus the algorithm fulfils its function fast even in a workspace that has a number of obstacles with complicated shape. In comparison with several conventional path-planning algorithms presented so far, it is shown from experimental results that the proposed algorithm selects faster a reasonable collision-free robot path than others.

クワッドツリー上で機能する移動ロボットの障害物回避経路生成アルゴリズムを提案する。作業空間を上から撮影した実画像からクワッドツリーは高速に作成されることから、アルゴリズムは障害物の配置の変化に柔軟に対応できる。クワッドツリーにおいて移動ロ

Journal of Robotic Systems 7(4), 555–574 (1990).
 CCC 0741-2223/90/040555-20$4.00

ポットより小さな領域を探索しないようにすると、ロボットと障害物の干渉チェックが省かれアルゴリズムが高速化される。

また本研究では、膨大な個数のノードを保持するクワッドツリーを直接的には探索せず、その上にノード数を小さく抑えて展開されたパスグラフを探索する。このことから、アルゴリズムは複雑な形状の障害物が多数存在する作業空間において高速に機能する。

INTRODUCTION

In a not-too-distant future, a mobile robot may take action autonomously in an arbitrary workspace without guidance of human beings. Then the autonomous mobile robot must be able to find a reasonable path between given start and goal points without colliding with obstacles in the workspace. From this point of view, such a path-planning problem for the autonomous mobile robot has been studied intensively, and a good many of path-planning algorithms have been presented so far.[1–4]

However, most algorithms do not work well in the workspace where shape and allocation of obstacles change flexibly time after time. The reason is as follows: (1) Some of them run on fixed data constructed from a set of obstacle polygons. However, data structure of the polygon is not suitable for automatic registration of obstacle shapes into computer memory via any sensor information. A more suitable data structure should be found, which can be constructed directly from some sensor information, for example, a real image obtained through a camera; and (2) Conversion from the set of obstacle polygons to the fixed data requires an enormous amount of calculation time. For example, the algorithm in Brooks[1] runs on the set of generalized cones converted from the polygon set. The order of calculation time for the conversion is $O(n^3)$ (n—the number of polygon vertices). The algorithm in Lozano-Perez and Wesley[2] also must investigate the visibility graph (VGRAPH) on the configuration space constructed from the polygon set. Construction of the complete configuration space based on the polygon set[3] and of the VGRAPH on the configuration space needs an enormous calculation time.[2] Construction of the configuration space is indeed time consuming in any data structure, for example, the quadtree representation.[4]

In addition, these algorithms investigate directly their fixed data (the set of generalized cone,[1] VGRAPH,[2] the configuration space quadtree[4]) with a huge number of nodes as the search space, and consequently they require much calculation time to select a reasonable collision-free path from these data. For example, the algorithm[4] selects even a non-shortest collision-free path in bad time since it searches directly the configuration space quadtree with a great number of nodes.

In view of these points, we propose a feasible path-planning algorithm running on a workspace whose obstacle shape or allocation changes flexibly. The algorithm runs on a quadtree representation, but it differs from that used in Kambhampati and Davis[4] in the following sense. The former quadtree expresses

the robot workspace itself and the latter quadtree expresses the configuration space of the workspace constructed by taking into account the scale of a mobile robot. Thus, the proposed quadtree can be fast constructed from a real image obtained through a camera on the ceiling of the robot workspace. Since the quadtree always represents the obstacles and their allocation in a real time manner, the proposed algorithm can work well even though the obstacles or their allocation changes in the workspace time after time. Another difference is that the former is prepared for the robot workspace itself but the latter must be prepared for each mobile robot. Thus the proposed algorithm saves spending memory capacity and calculation time for the registration of the quadtree into computer when it deals with multiple robots. The algorithm controls the multiple robots in the same quadtree at the same time and can be applied to a path-planning of coordinative multiple robots.[5]

One defective point of the proposed algorithm is that some collision check between the robot movement space and the obstacles is required in order to select a collision-free path out of the quadtree. To overcome this defect, the algorithm selects a sequence of free regions as the collision-free path, whose size is equal to or larger than the robot size. The quadtree controls obstacle and free regions with several sizes according to tree levels. Based on the tree level, all obstacle regions and free regions smaller than the robot are easily regarded as forbidden regions, and other regions are easily regarded as free regions. In this situation, the robot shape must be of simple so as to be approximated and included by a square region of the quadtree. In accordance with this regard, the algorithm can select fast the free sequence out of the quadtree without checking the collision.

As mentioned previously, the quadtree consists of a lot of nodes for representing the free and forbidden regions and hence the algorithm running on the quadtree is time consuming if it considers the quadtree as the search space. To get rid of this, the algorithm must keep the search space as small as possible, which follows the "small-is-quick" principle.[6] To put the principle in practice, the proposed algorithm investigates only a part of the quadtree via a path graph with small size. Starting from an arc with start and goal point nodes, the algorithm expands the path graph on the quadtree step by step. It always selects the shortest path connecting the start and goal nodes out of the path graph and checks whether the path intersects the forbidden regions or not. If it is so, the algorithm expands a part of the graph so as not to collide with the forbidden regions along the shortest path. Otherwise, the algorithm ends successfully with the selected path (its intersecting sequence of free regions) as a reasonable collision-free path. With the help of the hierarchical structure of the quadtree, the intersection check and the expansion of the graph are processed efficiently. As a result, the proposed algorithm selects a reasonable collision-free path fast enough to be used practically even in any cluttered workspace.

Compared with the configuration space quadtree and the VGRAPH, the path graph is exceptionally smaller in size and thus the proposed algorithm is faster than conventional algorithms running on the configuration space quadtree and the VGRAPH.

TWO DIFFERENT DATA STRUCTURES

In the proposed algorithm, selection of the shortest path out of the path graph and expansion of the graph are processed synchronously. To attain this synchronism, the algorithm consists of the following two phases. The first phase selects the shortest path out of the present path graph without minding whether the path intersects the forbidden regions. The second phase checks if the path is to be collision-free. If it is so, the proposed algorithm ends in success with the shortest collision-free path. Otherwise, the second phase expands a part of the graph in which the selected path collides with the forbidden regions, and returns to the first phase. The first phase is processed in the path graph, and the second one is processed in the quadtree. In what follows, we explain two different data structures.

Quadtree Representation

The quadtree arranges obstacle regions and other free regions in the workspace with a hierarchical structure in positioning.[7,8] In the quadtree, each node implies one of regions and is labeled according to the position of its corresponding region with respect to a set of obstacle regions: exterior (called a "white" node), interior (called a "black" node), and intersection (called a "mix" node) illustrated in Figure 1. The mix node corresponding to an intersection region has four child nodes corresponding to four subregions in the intersection region. Here, the relation of a subregion position with a registered number of the child node is given in Figure 1. Finally, the quadtree has several levels due to its hierarchical structure, and the levels are numbered increasingly one by one from top to down illustrated in Figure 1.

The quadtree with obstacle and free regions can be obtained from fast conversion of a real image of the whole workspace. The image is taken through a camera located on a ceiling position such that image blurring and geometrical distortion are reduced as small as possible. In an indoor workspace, the floor color can be generally distinguished from the obstacle color. Thus, the real image is efficiently processed to the binary image, whose pixel is classified into the free (floor) region or the obstacle region by using color information.[9] The binary image is fast converted into the quadtree by merging recursively four regions neighboring each other with the same label.[10] Thus the quadtree can represent allocation of obstacle and free regions in the workspace in good time (Fig. 2).

The quadtree fills a similar role of the pre-processing quadtree in Kambhampati and Davis[4] in the following way. The obstacle regions and further every free region smaller than the robot size are regarded as forbidden regions (Fig. 3). In this regard, 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 free regions are close each other with horizontal and vertical edges and quadrant arcs. As a result, the sequence of the free regions is surely considered to be the collision-free path.

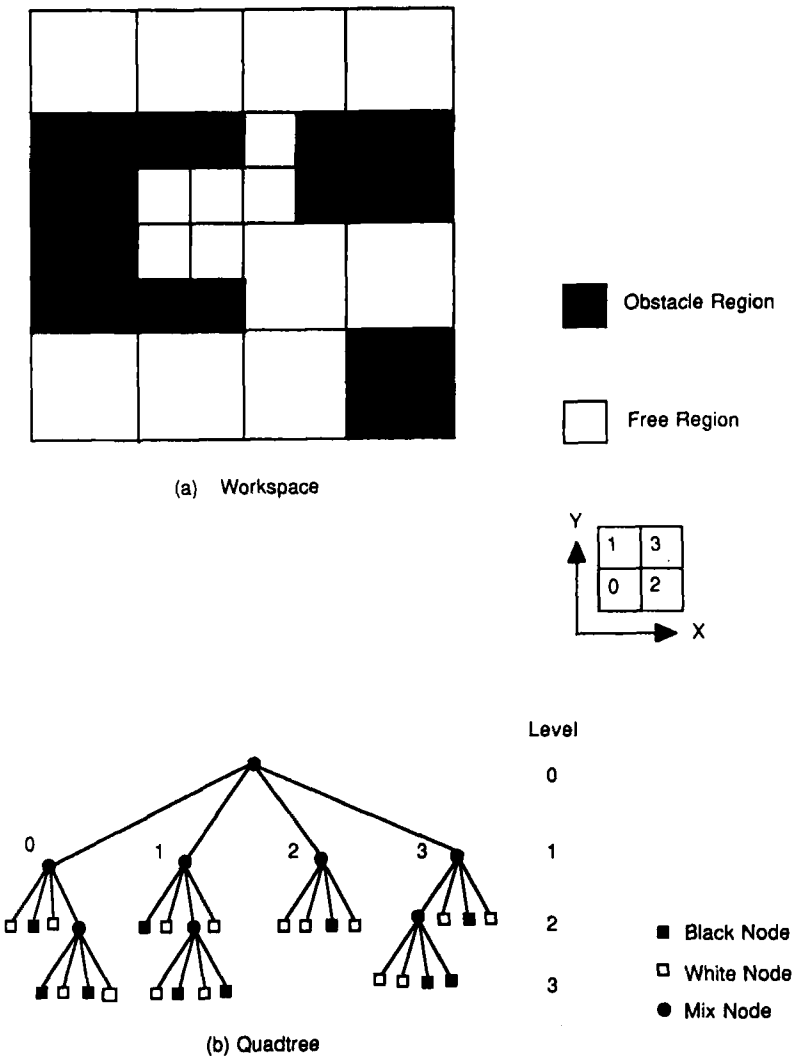


Figure 1. A workspace (a) and its corresponding quadtree (b).

Path Graph

Up to now, we pay attention to selection of the reasonable collision-free path between start and goal points out of the quadtree. The quadtree always has a great number of nodes, and hence it is not trivial to select the collision-free path out of the quadtree in economical calculation time. To accomplish this, we introduce a path graph built as small as possible on the quadtree (Fig. 4). In the path graph, each node corresponds to a position in the workspace, and each arc connecting two nodes corresponds to the line segment between the node positions in the workspace. For example, start and goal nodes in the graph

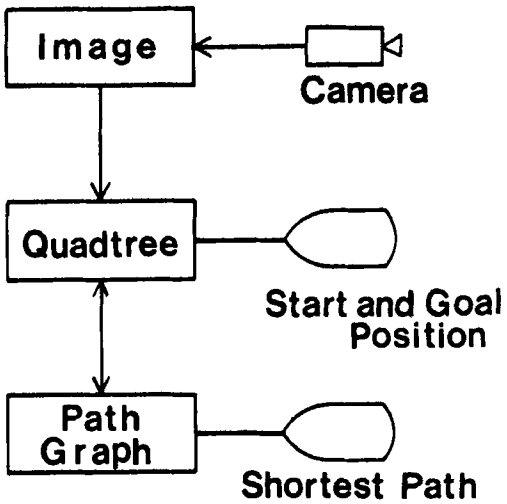


Figure 2. Process diagram.

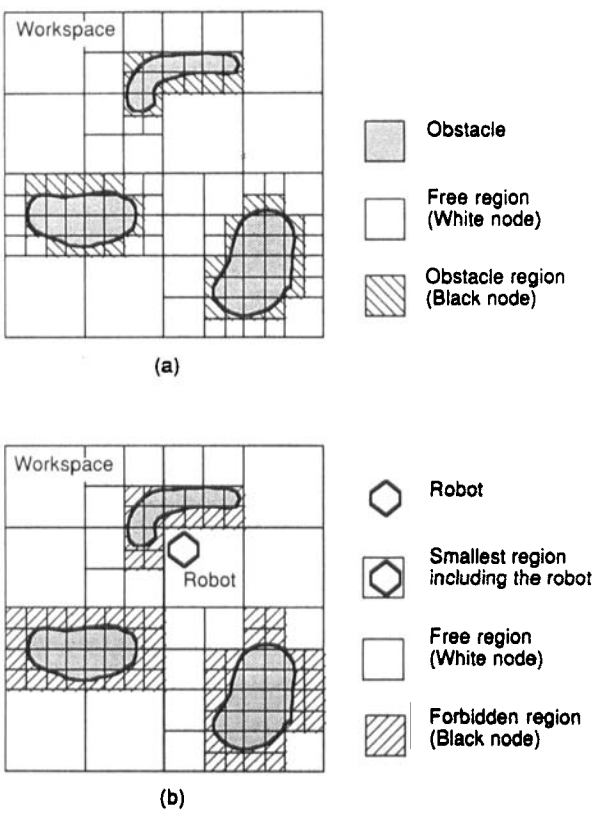


Figure 3. Obstacle and free regions in a quadtree (a), and forbidden and free regions for a robot in the quadtree (b).

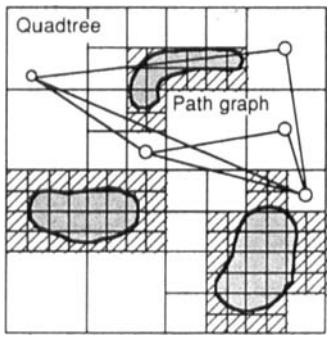


Figure 4. A path graph on the quadtree.

represent the start and goal positions, respectively, and the arc connecting the nodes represents the line segment between the positions.

To keep the path graph smaller, the algorithm constructs the graph without minding if the arc segment intersects the forbidden regions, and then modifies gradually a part of the path graph whose segment intersects them. Thus, the path graph may have several arcs whose segments intersect the forbidden regions in the workspace. In the path graph, when the line segment L corresponding to an arc does not intersect any forbidden region in the workspace, the following information is assigned to the arc: (1) a sign zero (nonintersection) and (2) a cost defined by the Euclidean distance of the segment L. Otherwise, the following information is assigned to the arc: (1) a sign one (intersection), (2) one or two intermediate points for avoiding each intersecting set of forbidden regions, and (3) a cost defined by the minimum sum of length of two line segments connecting the endpoints of the segment L via one of the intermediate points (Fig. 5).

The cost can be easily calculated if intermediate points are selected, and determination of the sign and selection of the intermediate points are processed fast in the quadtree representation.

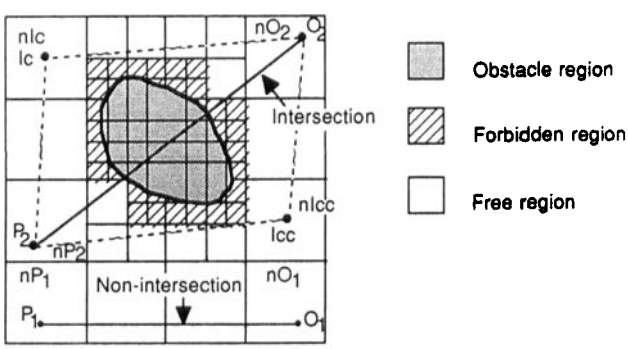


Figure 5. Assignment of a binary sign (0: Nonintersection, 1: Intersection) and a cost for each arc.

A PATH-PLANNING ALGORITHM ON THE QUADTREE VIA THE PATH GRAPH

As discussed previously, the algorithm runs on the path graph principally though it checks an intersection between the path and the forbidden regions and selects intermediate points to avoid each set of the forbidden regions on the quadtree. Now we describe the intersection check and the selection of intermediate points on the quadtree in the following first two paragraphs, and then describe the main frame of the proposed algorithm on the path graph in the last paragraph. Before the algorithm investigates the quadtree, its search level on the quadtree must be set. The search level is defined as the maximum level of a node whose region size is equal to or larger than the robot size. The algorithm regards mix nodes (intersection regions) with the search level as black nodes (forbidden regions), and consequently does not investigate such nodes further.

Intersection Check Between a Line Segment and the Forbidden Regions

When an arc is appended to the path graph anew, it is necessary to check whether its corresponding line segment L intersects the forbidden regions or

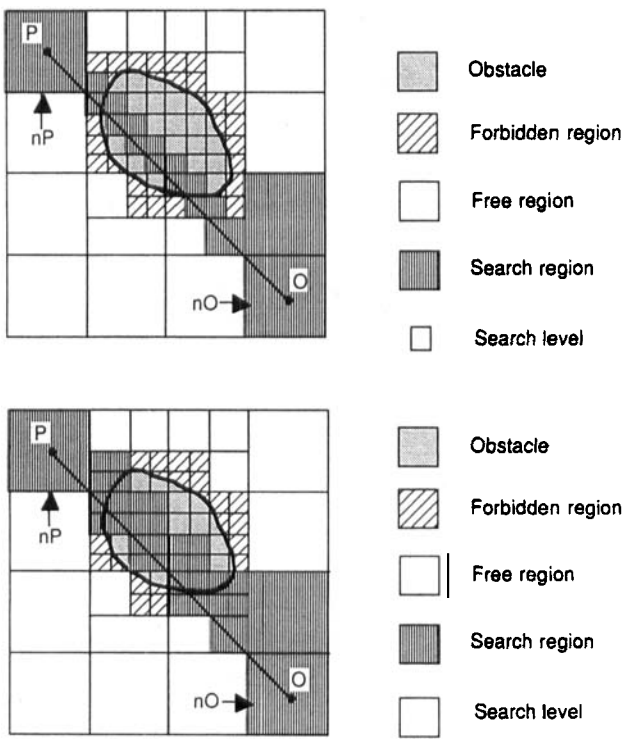


Figure 6. Interference check between the segment L and the forbidden region in the quadtree.

not and then to assign a sign (0: Nonintersection, 1: Intersection) to the arc. In this paragraph, we explain how to detect an intersection between the line segment L and the forbidden regions. The intersection is efficiently detected in the quadtree in the following way:

[Step 1] In the quadtree, select nodes nP and nO corresponding to the endpoints P and O of the line segment L, and set $n = nP$.
[Step 2] Find the neighbor node n' of the node n along the vector from one point P to another point O.[11]
[Step 3] Register the node n into a sequence S, and then set the node $n = n'$. Here note that the node n is registered into the sequence S one by one.
[Step 4] If the node $n = nO$, register the node n into the sequence S and go to [Step 5]. Otherwise, return to [Step 2].
[Step 5] The intersection occurs if the sequence S includes a black node, and in this case the arc sign is set "one." Otherwise, the sign is set "zero" (Fig. 6).

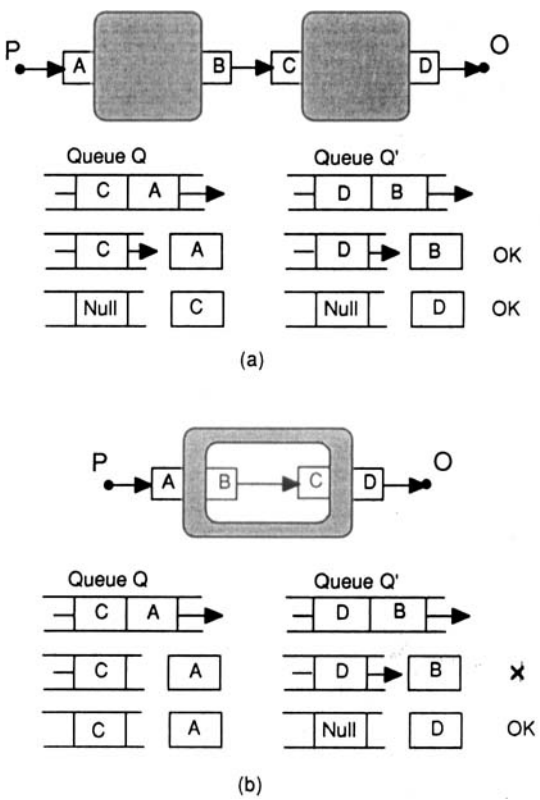


Figure 7. Determination of a pair of two white nodes in which the segment L enters into and goes out of a set of forbidden regions.

If the line segment L intersects a set of forbidden regions, some intermediate points should be selected around the set to avoid it. To select such intermediate points, the algorithm traces free regions (white nodes) between particular two free regions around the forbidden regions (black nodes) in both clockwise and counter-clockwise orders. One of the particular regions corresponds to the free region where the line segment enters into the forbidden set, and another corresponds to the free region where the segment goes out of the forbidden set. Candidates of the former region and these of the latter region are picked up from the sequence S in the following way. A white node (free region) in the sequence S is registered into a queue Q when a black node is close on the heels of the white node, and a white node (free region) in it is registered into a queue Q' when the white node is close on the heels of a black one. The queues Q and Q' integrate the former and latter candidates, respectively (Figs. 6 and 7).

Selection of Intermediate Points Avoiding the Forbidden Regions

In this paragraph, we explain how to select several intermediate points for avoiding a set of forbidden regions intersecting the line segment L of the arc.

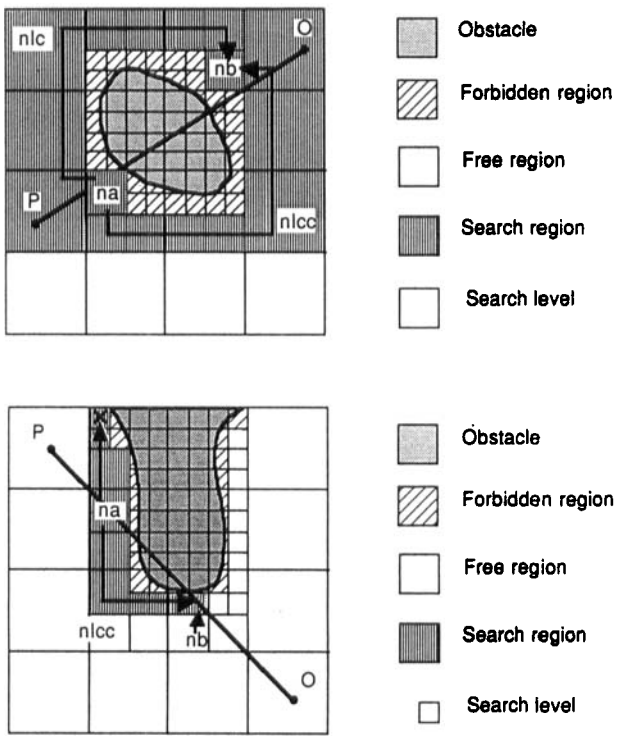


Figure 8. Selection of intermediate nodes by tracing white nodes around a set of forbidden regions (black nodes).

First, remove the topmost nodes from the queues Q and Q', and denote them by *na* and *nb*, respectively. Here note that all nodes in each queue are registered in the order from the node *nP* to the node *nO*. Then, starting from the node *na*, all white nodes (free regions) are traced around the intersecting set of black nodes (forbidden regions) in both clockwise and counter-clockwise orders with the aid of the algorithm of Samet.[12] If the node *nb* is picked up in this trace, a white node with the longest distance from the line segment L is selected out of the traced white nodes as an intermediate point node, and next the present topmost nodes of the queues Q and Q' are removed and set the nodes *na* and *nb* to carry out the next trace. If the node *nb* cannot be picked up, any node is not selected as the intermediate node and next the topmost node of the queue Q' is removed and set the node *nb* to carry out the next trace. Finally, if each topmost node cannot be removed since either the queue Q or Q' is empty, the selection ends with several intermediate point nodes, and the center points of regions corresponding to the selected intermediate nodes are appended to the arc as the intermediate points (Fig. 7).

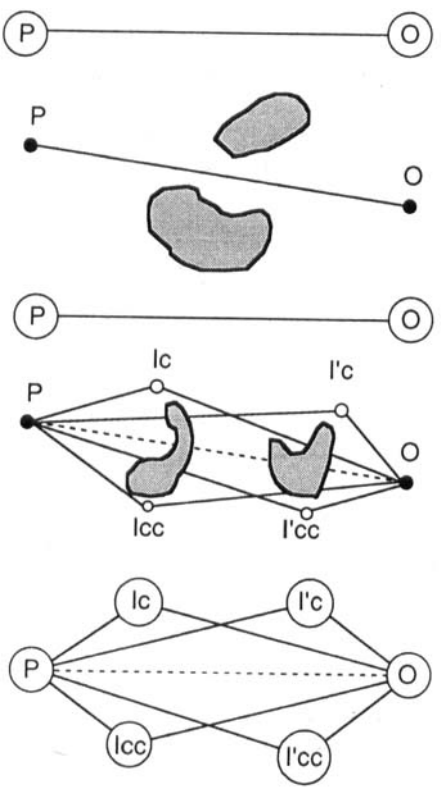


Figure 9. Selection of intermediate nodes when the segment L collides with several sets of forbidden regions.

[Step 1] Remove the topmost nodes of the queues Q and Q' to set them as nodes *na* and *nb*, i.e., $a \leftarrow 0$ and $b \leftarrow 0$.
[Step 2] Trace all white nodes from the node *na* to the node *nb* around the intersecting set of forbidden regions (black nodes), and select one or two white nodes (nodes *nIc* and *nIcc*) with the longest distance from the line segment L in both clockwise and counter-clockwise orders (Fig. 8). The node is to be an intermediate point node and the center of the node region is to be an intermediate point. If the node *nb* is not caught even in both orders, any intermediate node cannot be obtained. Thus, remove the topmost node of the queue Q' to set it as the node *nb*, i.e., $a \leftarrow a$ and $b \leftarrow b + 1$, and process [Step 2] again.
[Step 3] If the queue Q or Q' is empty, exit. Otherwise, remove the topmost node of the queue Q' to set it as the node *nb*, i.e., $b \leftarrow b + 1$, and remove the upper part of the queue Q to set the node *na* ($a \leftarrow b$), and also return to [Step 2].

The procedure can select one or two intermediate points (points *Ic* and *Icc*) for each intersecting set of forbidden regions even though the line segment L intersects several forbidden sets as shown in Figure 9.

Selection of the Shortest Path out of the Path Graph

This procedure is the main part of the proposed algorithm. In the procedure, the shortest path from the start node to the goal node is first selected out of the path graph by using the algorithm of Dantzig.[13] Secondly, the procedure ends successfully with the shortest path if it consists of all arcs with the sign zero. Otherwise, the procedure expands the arc with the sign one and the maximum cost along the selected path. That is, it appends intermediate point nodes prepared in the arc to the path graph. Then the arc is pruned from the path graph, and each intermediate node is connected to two endpoint nodes of the pruned arc by two arcs. Further the intermediate node is connected to the start and goal nodes in the path graph, respectively, so as not to overlook shorter paths.

[Step 1] Create a path graph initially by the arc tying start and goal point nodes. The previous two procedures deal with the arc to determine its information.
[Step 2] Select the optimal (shortest) path out of the path graph constructed so far by using the algorithm of Dantzig.[13] Here a path cost is normally understood to be the sum of costs of all arcs along the path, and the optimal path connecting the start and goal point nodes is defined to be the path with the minimum cost between them. The computational complexity of the Dantzig algorithm has been evaluated by $O(nd)$, where n is the number of nodes in the graph, and d is the average number of branches in each node.
[Step 3] If the selected shortest path consists of all arcs with the sign zero,

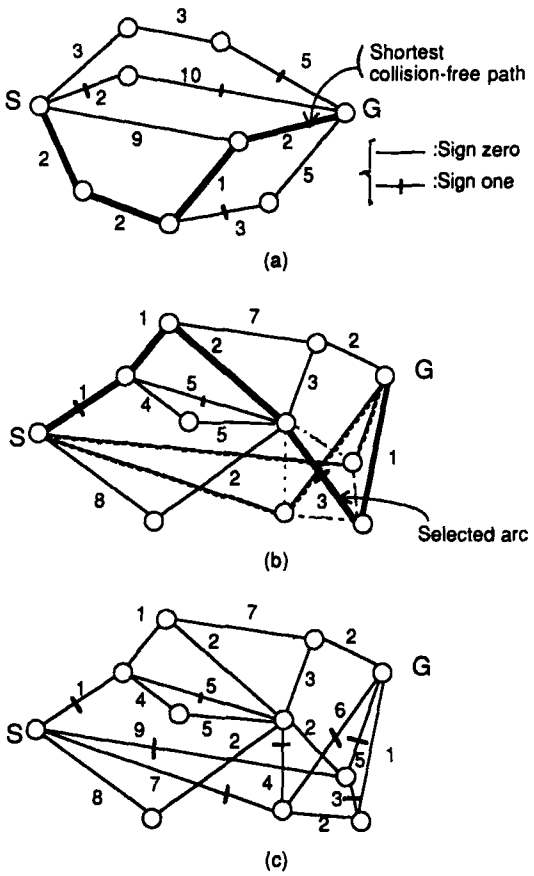


Figure 10. Expansion of the path graph.

the path is to be collision-free and thus the procedure (the algorithm) ends (Fig. 10(a)).

[Step 4] Otherwise, we select an arc with the sign one and also the maximum cost from the shortest path, and append intermediate nodes prepared for the arc to the path graph (Fig. 10(b)).

[Step 5] The selected arc is pruned from the path graph and several new arcs are appended to the graph. The new arcs are appended so as to connect each intermediate node to two endpoint nodes of the pruned arc and also connect it to the start and goal nodes. Here the previous two procedures deal with each new arc to determine its information before the arc is appended to the path graph. As a result, the path graph is renewed (Fig. 10(c)), and return to [Step 2].

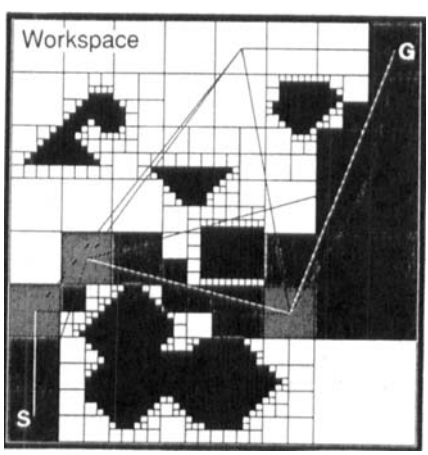
EXPERIMENTAL RESULTS

In this section, we first evaluate calculation times of the proposed algorithm in several workspaces, and second compare these evaluations with calculation times of some other algorithms presented in previous literature. According to several experimental results, it is shown that the proposed algorithm is faster than the algorithms presented so far.[2,4] Moreover, we indicate that the algorithm is good to generate a feasible collision-free path in some workspaces with free shaped obstacles. On the assumption that all quadtrees are already constructed from their workspace images obtained through a camera on the ceiling, all experiments are processed. Every quadtree has seven levels, and therefore the search level of the algorithm is chosen within seven levels. The algorithms are programmed in the C language and run on Apollo DN 590T with a floating point accelerator under the AEGIS operating system.

Comparison of Quadtree Algorithms in Respect of Calculation Time and Discussion about our Algorithm's Character

At first, we present some results showing that the proposed algorithm is faster than the algorithm of Kambhampati and Davis.[4] We use the same workspaces as Kambhampati et al. did, as shown in Figure 11. Table I compares calculation times of these algorithms for selecting similar collision-free paths. Strictly speaking, the calculation times cannot be straightforwardly compared because Kambhampati et al.'s algorithm is implemented in the Franz-Lisp language on Vax 11/785. However, the superiority of our algorithm to their algorithm can be recognized by both the results in Figure 11(a), (d) and Table I and calculating performance of the two computers in several benchmark tests.

In our work, when the search level of the algorithm is changed according to



(a)

Figure 11a.

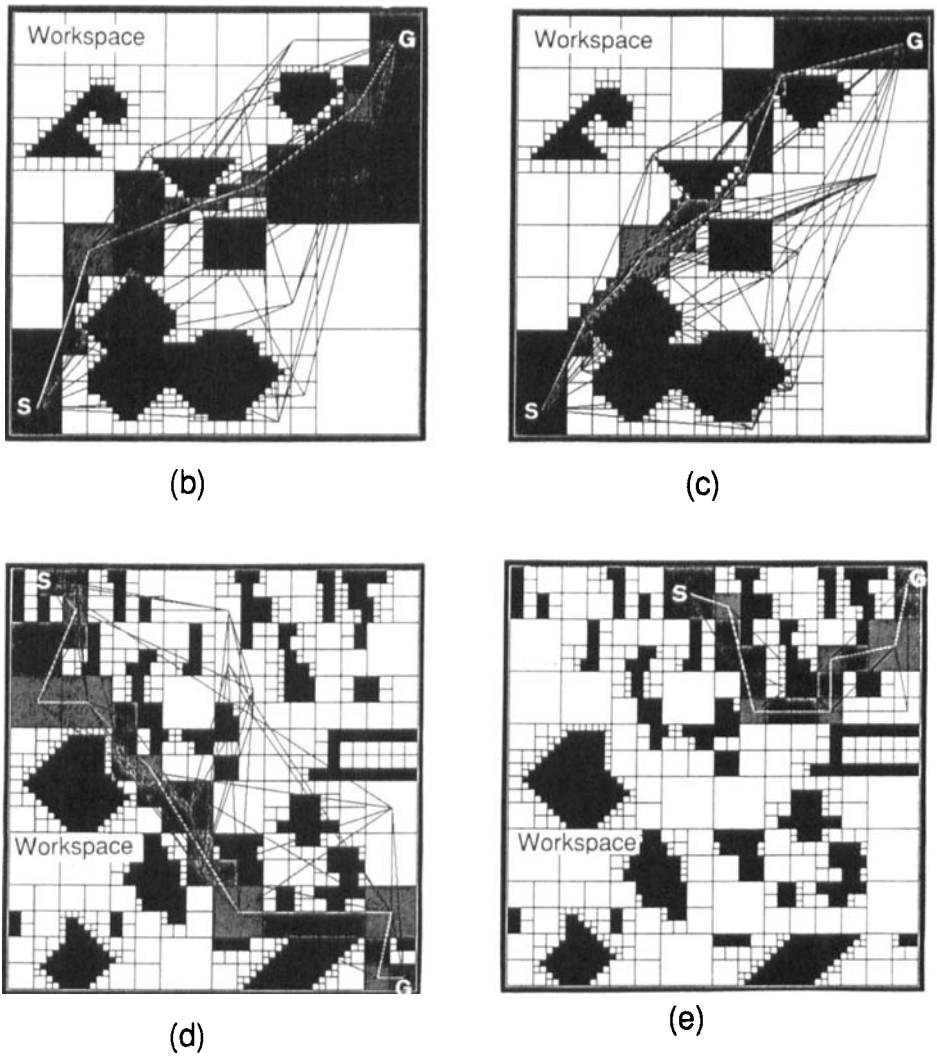


Figure 11. Several paths for five experiments, (a), (b), (c), (d), and (e) in two quadtree workspaces.

Table I. Calculation times for the five experiments (a), (b), (c), (d), and (e).

Experiment	Level	Calculation time (second)		
		Proposed algorithm	Kambhampati's one	
			Pure A*	Modified A*
a	4	0.233	28	11
b	5	1.383	–	–
c	6	6.183	–	–
d	4	2.883	58	13
e	4	0.250	–	–

size of the mobile robot, a different reasonable collision-free path is selected. When the search level is changed from four to six in the same workspace, the algorithm determines different reasonable paths shown in Figure 11(a), (b), (c), and their calculation times are shown in Table I

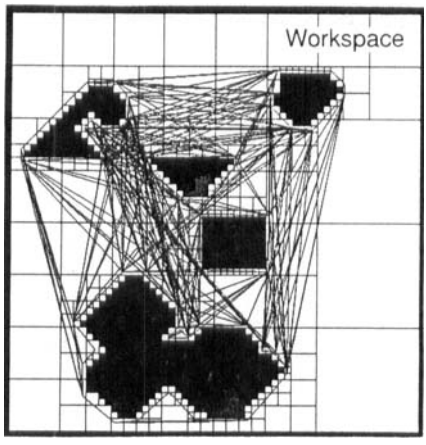
Finally, the closer the given start and goal points are, the smaller the path graph is and the faster our algorithm runs. As illustrated in Figure 11(d), (e) and Table I, the proposed algorithm determines a collision-free path faster in the same workspace when start and goal points are closer.

Comparison of the Proposed Algorithm with the Algorithm Based on the VGRAPH in Respect of Calculation Time

To show that the proposed algorithm is even fast in comparison with the algorithm based on the VGRAPH, we present several results in this paragraph.[2] In this comparison, an isolated set of black (forbidden) regions in the quadtree is defined as a polygon by a rough approximation, and then all vertices of the polygons are converted into a fundamental VGRAPH by connecting two vertices whose line segment does not intersect any black region (Fig. 12(a)).

The algorithm running on the VGRAPH builds a complete VGRAPH by appending a set of arcs spanned from start and goal point nodes to the fundamental VGRAPH (Fig. 12(a)), and then selects the shortest path out of the complete VGRAPH. When the arcs are appended to the fundamental VGRAPH, the intersection between their line segments and the forbidden regions is efficiently checked in the quadtree. However, its calculation time increases proportionally with increase of the number of nodes in the fundamental VGRAPH. Further, calculation time for selection of the shortest path on the complete VGRAPH increases exponentially with increase of the node number.

Table II shows the above comparison when these algorithms generate similar



(a)

Figure 12a.

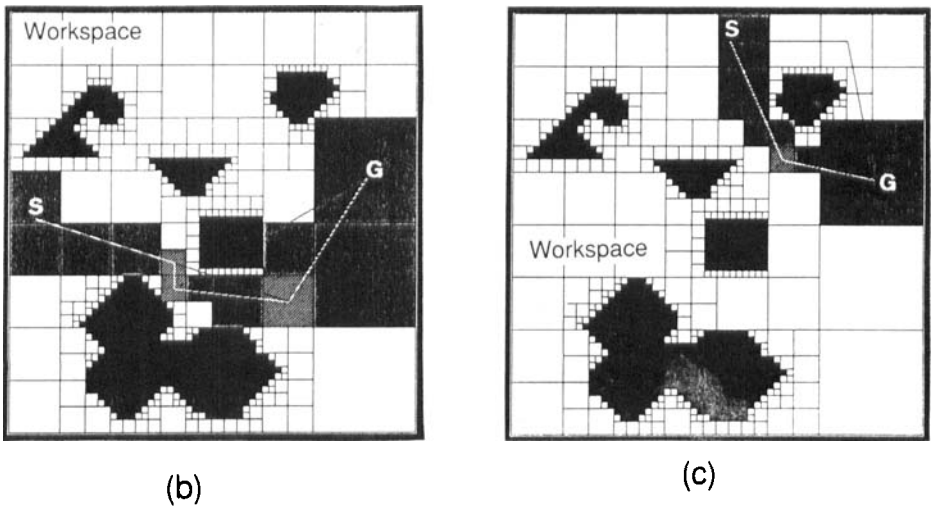


Figure 12. The VGRAPH on the quadtree (a), and two reasonable collision-free paths (b) and (c).

collision-free paths (Fig. 12(b) and (c)). As shown in Table II, the performance of our algorithm is superior to that of the algorithm on the VGRAPH in all cases. Furthermore, the former performance does not depend directly on the number of obstacles and shape complexity of each obstacle in the workspace though the latter performance depends directly on them. It is seen from these results that our algorithm is better than the algorithm based on the VGRAPH in some cluttered workspace with respect to calculation time.

Selection of a Feasible Collision-Free Path in Some Workspace with Free Shaped Obstacles

In previous works, data structures of the search space, i.e., the quadtree workspace and the VGRAPH, have a great number of nodes to avoid each set of forbidden regions for all directions. In this case, the algorithms are time consuming since they oblige us to expand a lot of extra nodes for finding the shortest collision-free path from the search space. On the other hand, our

Table II. Comparison of the algorithms running on the path graph and the VGRAPH with respect to calculation time.

Experiment	Position		Calculation time (second)	
	Start	Goal	Path graph (level 4)	VGRAPH (54 nodes)
1	(32,250)	(450,200)	0.033	0.188
2	(270,32)	(450,200)	0.025	0.183

algorithm needs at most several nodes to avoid each intersecting set of forbidden regions. Consequently the algorithm builds the path graph with small size as the search space and can select fast the shortest collision-free path from the path graph. This tendency is remarkable for several workspaces with free shaped obstacles (Fig. 13 and Table III). For example, if we use the VGRAPH in these workspaces to select the collision-free path, we are easily sure that the VGRAPH has an enormous number of nodes around all free shaped obstacles and in result the algorithm requires much calculation cost to select the collision-

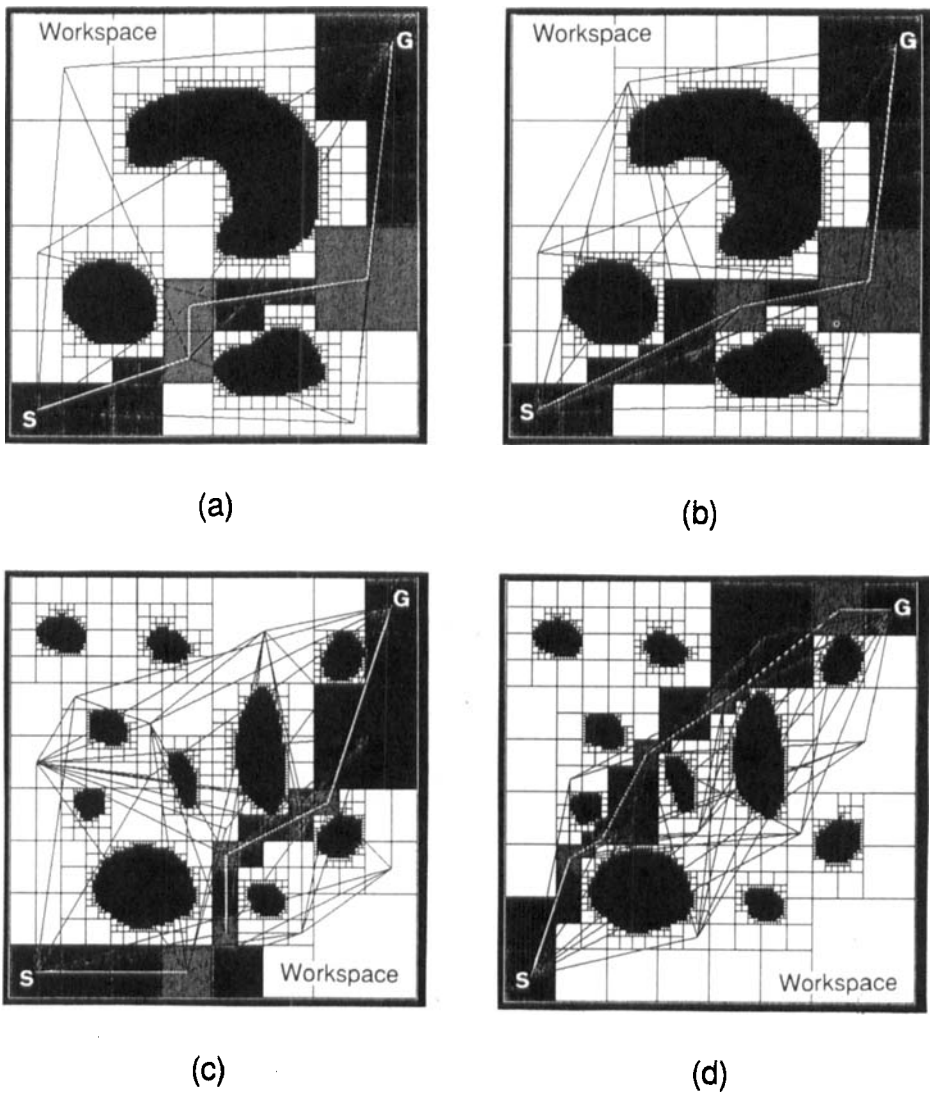


Figure 13. Small path graphs and their feasible collision-free paths in quadtree workspaces with free shaped obstacles.

Table III. Calculation times to generate collision-free paths in free shaped workspaces

Experiment	Workspace	Level	Calculation time (second)
a	1	4	0.183
b	1	5	0.383
c	2	4	0.816
d	2	5	0.783

free path from the VGRAPH. Needless to say, the shortest collision-free path (its intersecting sequence of free regions) is not always coincident with the shortest collision-free path in the VGRAPH, but they are to be much the same from the global point of view in the workspace. With respect to collision safety, the selected path from the path graph may be better than that from the VGRAPH because the former does not come close to the forbidden regions.

CONCLUSIONS

Most of path-planning algorithms investigate the fixed data constructed by a hard preprocessing on the basis of a given robot and its obstacles. Hence their capability of selecting a reasonable collision-free path is restricted when the robot, the obstacles, or their allocation is changed. In addition, since the fixed data must be prepared per mobile robot, it cannot be used for path-planning of coordinative multiple robots. Motivated by this point of view, we proposed a path-planning algorithm running on the quadtree. It can be easily constructed by conversion of an image obtained through a camera on the ceiling of the robot workspace. Since the quadtree expresses the obstacles in the workspace in less than no time, our path-planning algorithm fulfills its function even in a workspace whose obstacle shape or allocation is flexibly altered. Moreover, since the quadtree represents the robot workspace itself without regard of the mobile robot, the algorithm copes with a change of the mobile robot easily. The algorithm can further deal with multiple mobile robots simultaneously in the quadtree and be used for path-planning of coordinative multiple robots.

In the fixed data (the search spaces) with a great number of nodes, most of previous algorithms select a collision-free path on the basis of the A* strategy with bad heuristic information for leading the search to the goal node. Therefore the conventional algorithms must expand most of such nodes to select a shortest collision-free path and consequently are time consuming. This tendency becomes remarkable in a cluttered workspace where the number of obstacles is large and shape of each obstacle is complex. To overcome this difficulty, our algorithm keeps the path graph (the search space) as small as possible while investigating a reasonable collision-free path. Consequently the algorithm runs fast enough to be used practically. Finally, we present several experimental results and show that the proposed algorithm is superior to conventional algorithms in respect of calculation time.

References

1. R. A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-13**, 190–197 (1983).
2. T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles,"*Commun. ACM*, **22**, 560–570 (1979).
3. T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," *IEEE Journal of Robotics and Automation*, **RA-3**, 224–238 (1987).
4. S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, **RA-2**, 135–145 (1986).
5. Y. H. Liu, S. Kuroda, T. Naniwa, H. Noborio, and S. Arimoto, "A practical algorithm for planning collision-free coordinated motion of multiple mobile robots," in *Proc. IEEE Conf. Robotics and Automation*, Scottsdale, Arizona, 1989, pp. 1427–1432.
6. J. Pearl, *Heuristics—Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company: Massachusetts, Ch. 1, 1985.
7. S. L. Tanimoto and T. Pavlidis, "A hierarchical data structure for picture processing," *Computer Vision, Graphics and Image Processing*, **4**, 104–119 (1975).
8. A. Klinger and C. R. Dyer, "Experiments in picture representation using regular decomposition," *Computer Vision, Graphics and Image Processing*, **5**, 68–105 (1976).
9. D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ, 1982, Ch. 4, pp. 149–165.
10. H. Samet, "An algorithm for converting rasters to quadtrees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **PAMI-3**, 93–95 (1981).
11. H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Vision, Graphics and Image Processing*, **18**, 37–57 (1982).
12. H. Samet, "Region representation: Boundary codes from quadtrees," *Commun. ACM*, **23**, 171–179 (1980).
13. G. B. Dantzig, *Linear Programming and Extensions*, The Rand Corporation, Princeton University Press, 1963, Section 17-3.