# THE VISIT PROBLEM: VISIBILITY GRAPH-BASED SOLUTION †

*N. S. V. Rao, S.S. Iyengar*
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803

*G. deSaussure*
Center for Engineering Systems Advanced Research
Oak Ridge National Laboratory
Oak Ridge, TN 37831

*ABSTRACT*

The *visit problem* is defined as follows: a point body $R$ is located at point $d_0$ in a finite-sized two-dimensional terrain populated by a finite number of polygonal obstacles; each obstacle has a finite number of vertices. The $R$ is equipped with a sensor system capable of detecting all vertices and edges seen from its present location. A computing device with a finite storage capability is also housed on $R$. $R$ is capable of translational motion. Initially, $R$ is located at a point $d_0$, and no terrain model is available to $R$. $R$ is required to *execute* a *navigation mission* which involves visiting a sequence of the destination points, namely $d_1, d_2, \cdots, d_M$, in the specified order. We present the algorithm LNAV that navigates $R$ from $d_i$ to $d_{i+1}$. We then present the algorithm GNAV that executes the navigational mission. It uses LNAV in traversing in between successive destination points in the mission, and builds a global visibility graph of the terrain by integrating the sensor information obtained, over a period of time, as $R$ keeps navigating. We also illustrate that, in a general case of executing a navigation mission, the performance using GNAV is more efficient than the repeated application of LNAV in terms of number of scan operations and the distance traversed by $R$.

**Key-words and Phrases:** visibility graph, algorithm, learning, time and storage complexities.

## 1. INTRODUCTION

The *visit problem* is defined as follows: Consider a finite-sized two-dimensional terrain populated by a finite number of simple stationary polygonal obstacles; each obstacle has a finite number of vertices. The total number of obstacle vertices is given by $N$. We consider a point body $R$, capable of translating to a specified destination point in a straight line path. $R$ takes a finite amount of time to translate through a finite distance. The $R$ is equipped with a computing device with a finite storage capability. It is also equipped with a sensor system which detects all edges and vertices that are seen from the present location of $R$. Each such process is referred to as a *scan* operation.

Initially, the terrain is *unexplored* or *unknown*, i.e. no terrain model is available to $R$. $R$ is initially located at a point $d_0$ and is required to visit a sequence, $d_1, d_2, ..., d_M$, of destination points without colliding with the obstacle polygons. This sequence, $d_1, d_2, ... , d_M$, is called the *navigation mission*, and the process of visiting these points is termed as the *execution* of the navigation mission. Navigation from $d_i$ to $d_{i+1}$ is called a *traversal*, where $d_i$ is called the *source* point and $d_{i+1}$ is called the *destination* point. The $R$ is required to execute the mission. We assume that $d_1, d_2, ..., d_M$ are in obstacle-free region. The obstacle polygons are simple and disjoint, hence there always exists a path for $R$ from a source point to a destination point of any traversal.

Navigating a body through a terrain of known obstacles has been solved in many cases. See Lozano perez and Wesley [4], O'Dunlaing and Yap [6], Reif [10], and Schwartz and Sharir [11] for some of the fundamental approaches. However, the navigational algorithms for unknown terrains have not been developed to the same extent. These algorithms can be classified into two broad categories. The first category, often referred to by the appellation of *non-heuristic algorithms*, deals with precise algorithms whose correctness is guaranteed within a framework of stated assumptions. The second category deals with heuristic types of algorithms whose correctness is not explicitly shown.

We now discuss the first category. The Pledge algorithm discussed in Abelson and diSessa [1] enables a point robot to move out of mazes by utilizing a "touch" type of sensing ability. Lumelsky and Stepanov [5] present two very interesting algorithms - bug1 and bug2 - that enable a point automaton to reach a destination point in two-dimensional terrains populated by simple closed obstacles. The point automaton here uses a "touch" type of sensor for navigational purposes. These algorithms do not, in general, implement learning in the navigation process, and as a consequence the navigation paths remain the same (in terms of computation, sensing, etc.) despite executing the navigation process over a number of times. In Oommen et al [7], a navigation algorithm that navigates a point robot amidst convex polygonal obstacles to a specified destination point is presented. The robot here is equipped with a "touch" type of sensor and a distance probing sensor. The interesting feature of the approach of [7] is the implementation of "learning" by integrating the sensor information into the visibility

† Research Sponsored by Office of Basic Energy Sciences, U.S. Department of Energy under contract number DE-AC05-840R21400 with Martin Marietta Energy Systems.

graph of the terrain (which is the global terrain model). These algorithms are also theoretically validated. The approaches of Chatila [2], Iyengar et al [3], belong to the heuristic category.

Our approach is similar to that of Oommen et al [7] in terms of the global methodology of incorporating learning. In terms of the problem formulation, our work is different in the following ways: (a) the obstacle terrain consists of polygonal obstacles and is not restricted to contain only convex polygons (unlike [7]). As a result obstacles such as mazes, traps, etc. (which are more complicated to deal with than convex obstacles) are included in our set of obstacles. (b) $R$ is equipped with a "see from distance" type of sensor, such as a range finder, etc., in place of the combination of a touch and a distance probe sensor of [7] (we note that the characteristics of the sensors considerably influence the nature the navigational strategies).

In terms of the solution methodology our work differs from that of [7] as follows: In [7] metric based arguments (similar to those of Lumelsky and Stepanov [5]) are employed to validate the algorithms. In particular, the convexity of the obstacles is used as a basis for proving the convergence of the algorithms. In our approach we use the connectivity of the visibility graph to prove the correctness of the algorithm. This approach seems to be new for navigation algorithms in unexplored terrains.

At this point some comments are in order about the usage of learning as an enhancement for navigational algorithms in unexplored terrains. Here we deal with the learning only as it is applied to the navigational aspect, and we do not imply the generic ability to understand and pick new faculties (an ability that seems to be naturally present in humans). Learning has been employed in robot navigation in earlier works by Chatila [2] Iyengar et al [3] and Oommen et al [7].

The paper is organized as follows: In section 2, we present the algorithm LNAV and its analysis. The algorithm GNAV is discussed in section 3, along with its analysis. An illustrative example is presented in section 4. In this paper, we present our theorems without proof and the proofs can be found in our report [9].

## 2. NAVIGATION IN UNEXPLORED TERRAINS

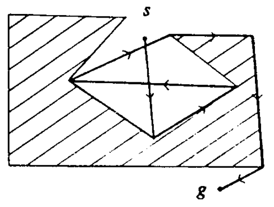
In this section, we consider the problem of navigating $R$ from its present location, at point $s$, to a specified destination point $g$. Both $s$ and $g$ lie in the obstacle-free region.

We first present a few definitions. A point $v$ is *seen* from a point $p$ if and only if the line joining $p$ to $v$ is not intercepted by any obstacle. Note that a point $v$, seen from a point $p$, will be detected by the sensor of $R$ located at $p$. An obstacle vertex is *seen* from $p$ if it is seen as a point from $p$. An obstacle edge is *seen* from $p$ if and only if a point on the edge is seen from $p$. The *seen-part* of an edge, from $p$, is the set of all points of the edge that are seen from $p$. The *seen-part* of the terrain, from $p$, is the union of the all obstacle vertices seen from $p$, and all seen-parts of all edges that are seen from $p$. The sensor operation performed by $R$, located at $p$, precisely obtains the seen-part of the terrain, from $p$.

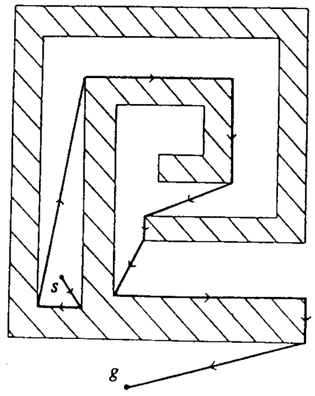
The *visibility graph VG* is defined as the graph $(V,E)$, where $(v_1,v_2)\in E$, for $v_1$, $v_2 \in V$, indicates the fact that the line joining $v_1$ and $v_2$ is either (i) not intersected by any obstacle or, (ii) it corresponds to an edge of an obstacle. The visibility graph of the entire terrain is referred to as the *global* VG, where $V$ is the union of the set of all vertices of all grown obstacle polygons, and the set of all source and destination points. The set $V$ in the visibility graph of [4] contains only the obstacle vertices and in this case the visibility graph is called *local* VG. Here the global *VG* contains additional nodes corresponding to the source and destination points of the navigational mission.

After each scan, from a vertex $v$, the adjacency list of $v$ is computed and stored. From the adjacency list of $v$, a node $s^*$ nearest to the destination point is selected, and then $R$ moves to this node. The path to this node is stored on a stack. With each node stored on the stack, we also store its adjacency list. The vertices from which a scan is performed are labeled as *visited* and are stored in memory. $R$ continues this process of "scan and move forward" until it either reaches the destination point or it reaches a node with all its adjacent nodes visited earlier. If the latter occurs, then $R$ "backtracks"; which involves accesses to the stack and the computation of the node to visit next. The stack is repeatedly popped until a node $r$ with at least one unvisited adjacent node (neighbor) is obtained. We temporarily store the path popped off the stack. The next visit point for $R$ is the unvisited neighbor $s^*$ of $r$ that is nearest to the destination point. Then $R$ moves to $r$ (along the path popped off the stack) and then to $s^*$. The navigation algorithm is recursively applied from $s^*$. The algorithm is presented in detail in LNAV.

```
algorithm LNAV($s$,$g$)
1.  scan and obtain the seen-part, from $s$, of terrain;
2.  obtain adjacency list if $s$ in global VG;
3.  if ($g$ is directly reachable)
4.     move to $g$;
5.  else
6.     obtain the adjacency list $L_s$, of $s$ in the global VG;
7.     if (all nodes of $L_s$ are not visited)
8.        push $s$, and $L_s$ onto stack;
9.        compute $s^*$, an unvisited vertex of $L_s$ nearest to $g$;
10.       mark $s$ as visited;
11.       compute a shortest path to $s^*$ along edges of VG;
12.       move to $s^*$ along the computed path;
13.       LNAV($s^*$,$g$);
14.    else
15.       repeatedly pop the top of the stack till a node $r$ is
          obtained such that $L_r$ contains at least one vertex
          that was not visited earlier;
16.       compute $s^*$ an unvisited vertex in $L_r$ nearest to $g$;
17.       move back to $r$ along the path on stack and then to $s^*$;
18.       LNAV($s^*$, $g$);
       endif;
    endif;
```



(a) Escaping out of a concavity



(b) Escaping out a maze

Fig. 1. Execution of LNAV - no backtracking.

The backtracking is a very important feature of this algorithm. The execution of algorithm LNAV is illustrated in Fig. 1, where two traversals are undertaken without backtracking. In Fig. 2, $R$ starts at $s$ and moves to node 4, where $R$ backtracks to node 5 via node 2.

Now we analyze the algorithm LNAV for its correctness and its performance. We first state a result.

**Result 1** [8]: *The local VG is graph connected and every point in obstacle-free space is seen from some vertex of local VG.* □

It is direct to see that global VG satisfies the same properties stated above. A close look at the algorithm LNAV reveals the following property.

**Proposition 1:** The order in which $R$ visits the new obstacle vertices (while executing algorithm LNAV from $s$ to $g$) is equivalent to performing a depth first search on global VG with $s$ as a starting node.

Depending on the locations of $s$ and $g$, $R$ (executing LNAV) will visit a subset of the vertices of global VG. In the worst-case, $R$ would visit all the nodes. Now, combining the above two results we show that the algorithm LNAV correctly navigates $R$ from the source point $s$ to the destination point $g$.

**Theorem 2.1:** *Algorithm LNAV navigates $R$ from $s$ to $g$ in a finite amount of time.* □

The execution of LNAV by $R$ involves operations such as scans, movements and computation. The time taken for navigating from $s$ to $g$ is a function of the time taken to perform these operations. These parameters are estimated in the following theorem.

**Theorem 2.2:** *In executing algorithm LNAV,* (i) *the number of scan operations is at most* $N+1$, (ii) *the total distance traversed is at most equal to twice the length of the depth first tree, of global VG, rooted at $s$.* □

The computational complexity of the algorithm LNAV is given in the following theorem.

**Theorem 2.3:** *In executing the algorithm LNAV,* (i) *the storage required is $O(N^2)$,* (ii) *complexity of path planning is $O(N^3)$,* (iii) *complexity of stack operations is $O(N^2)$.* □

## 3. LEARNED NAVIGATION

In this section we present the algorithm GNAV which implements a restricted form of "learning". The algorithm LNAV is modified such that the adjacency lists that are generated after each scan are stored in a partially-built global VG called GVG. Note that the GVG contains the visited destination points of the navigation mission as nodes (along with their visibility information) in addition to the nodes corresponding to the obstacle vertices. Let the modified LNAV be called LNAV1. Now the navigation mission is executed as follows: For each traversal from $d_i$ to $d_{i+1}$, a scan is performed from $d_i$ and the GVG is augmented with the adjacency information of $d_i$. Then a node $d^*$ nearest to $d_{i+1}$ is computed. A shortest path to $d^*$ is planned on GVG. Note that $d_i$ is graph node. $R$ moves to $d^*$ along the edges of GVG. From $d^*$ to $d_{i+1}$ the navigation is carried out using LNAV1. The details of algorithm GNAV are given below:

```
algorithm GNAV($d_i$, $d_{i+1}$)
1.  scan and obtain seen-part, from $d_i$, of the terrain;
2.  augment the visibility graph;
3.  if ($d_{i+1}$ is directly reachable)
4.      move to $d_{i+1}$;
5.  else
6.      compute the $d^*$, the vertex nearest to $d_{i+1}$;
7.      compute the shortest path to $d^*$;
8.      move to $d^*$ on GVG;
9.      LNAV1($d^*$, $d_{i+1}$);
10.     if (($i+1) \neq M$)
11.         GNAV($d_{i+1}$,$d_{i+2}$);
        endif
    endif;
```

Consider the traversal from $d_i$ to $d_{i+1}$. The navigation from $d_i$ to $d^*$ is along edges of GVG, and hence is collision-free. The navigation from $d^*$ to $d_{i+1}$ is correctly carried by LNAV1 (theorem 2.1). Hence we have the following theorem.

**Theorem 3.1:** *Algorithm GNAV correctly executes the navigation mission.* □

We note that the GVG at any stage is dependent on the exact nature of the navigation mission. The GVG, at any stage, will be more complete if the destination points are scattered

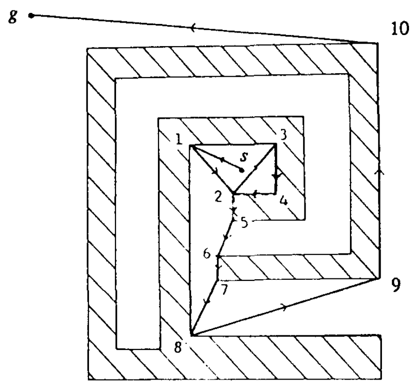


Fig. 2. Execution of LNAV with backtracking (from 4 to 5 along 2).

around the terrain rather than clustered to a small region. Since, our learning is "incidental", i.e., the terrain model of a region is built only in the regions $R$ moves into, we can only make probabilistic statements about the learning.

**Theorem 3.2:** *The terrain model converges to complete local VG with probability of one, if every obstacle vertex and edge has a non-zero probability of being seen during some scan operation while executing the navigational course. The terrain model will be completely built in $N+2M$ scans, then* (i) *execution of each traversal takes two scan operations if $N$ is unknown, no scan operations if $N$ is known,* (ii) *the planned path is optimal from $d_i$ to $d^*$, if $N$ unknown, and the entire path is optimal if $N$ is known.* □

While executing the algorithm GNAV, the global visibility graph *GVG* contains nodes corresponding to the destination points (at most $M$ in number) of the navigational mission in addition to the nodes corresponding obstacle vertices (at most $N$ in number). Hence the number of nodes in the graph used by GNAV is at most $N+M$. A straightforward extension of Theorem 2.2 results in the following theorem.

**Theorem 3.3:** *In executing the navigation mission, using GNAV* (i) *the number of scan operations is at most $N+2M$,* (ii) *the total distance traversed is at most $2\sum_{i=1}^{M}$ (depth first tree rooted at $d_i$ ).*□

The computational complexity of GNAV is estimated in the following theorem.

**Theorem 3.4:** *In executing the navigation mission, using GNAV* (i) *the storage complexity is $O((N+M)^2)$,* (ii) *the complexity of stack operations is $O(MN^2)$,* (iii) *the complexity of path planning is $O((N+M)^3)$.* □

## 4. EXAMPLE

Let us compare the performance of the algorithm GNAV over repeated application of algorithm LNAV. We consider the terrain of Fig. 3. The Fig. 4 through 6 present seven traversals carried out using algorithms GNAV and LNAV. Note that the global information available to GNAV enabled it to navi-

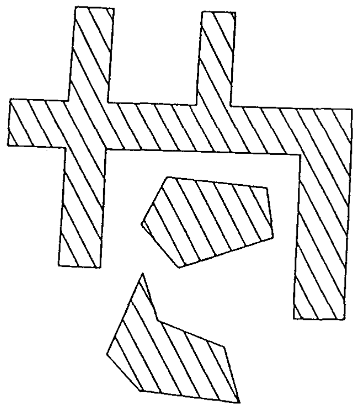
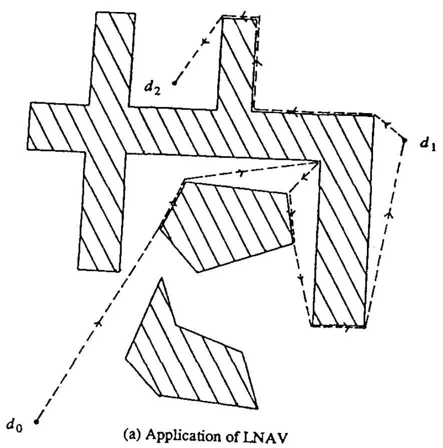
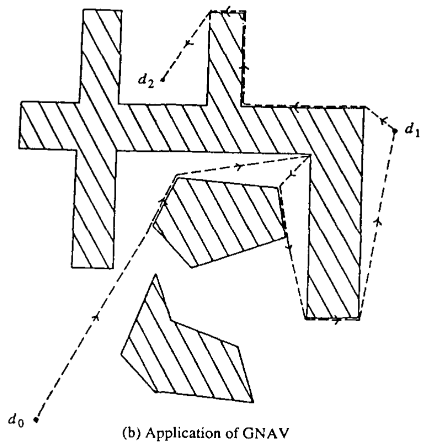


Fig. 3. Terrain

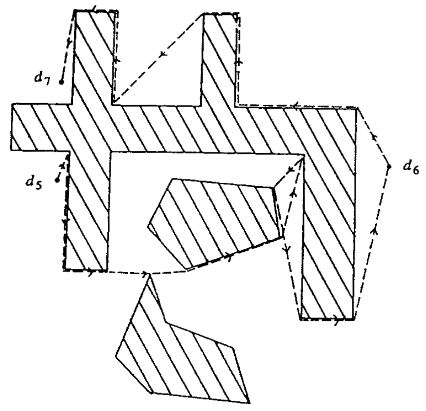


(a) Application of LNAV

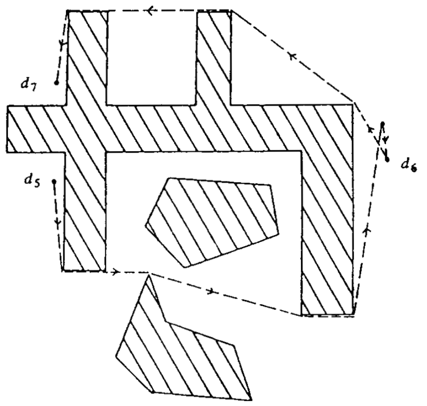


(b) Application of GNAV

Fig. 4. Navigation from $d_0$ to $d_1$ and then to $d_2$.

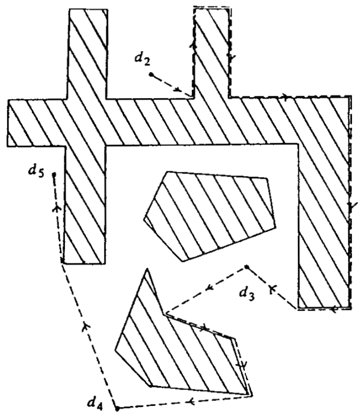


(a) Application of LNAV

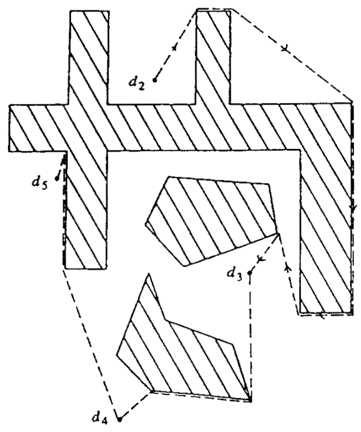


(b) Application of GNAV

Fig. 6. Navigation from $d_5$ to $d_6$ and then to $d_7$.



(a) Application of LNAV



(b) Application of GNAV

Fig. 5. Navigation from $d_2$ to $d_3$ to $d_4$ and then to $d_5$.

gate better compared to LNAV. In Fig. 7 we show the relative performance of these two algorithms in terms of the number of scan operations. Notice the decrease in the number of scans performed by GNAV as $R$ traverses in the terrain. Similar phenomenon is seen in the number of localized concavities entered by $R$ in Fig. 8.

## 5. CONCLUSIONS

In this paper we presented an algorithm to navigate a point robot through a sequence of destination points amidst unknown stationary polygonal obstacles in a two dimensional terrain. The algorithm implements learning in the way of building a global terrain model by integrating the sensor information obtained during the course of navigation. This global model is used in planning the future navigational paths. This approach prevents the robot from getting into localized detours, and also results in better navigational course, in an average case, compared to the algorithms without learning. The proposed algorithms are implemented in language C on a simulator for HERMIES-II robot running on IBM/PC.

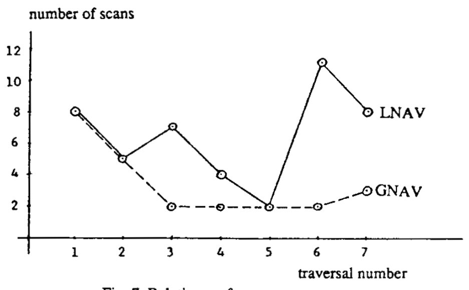


Fig. 7. Relative performance.

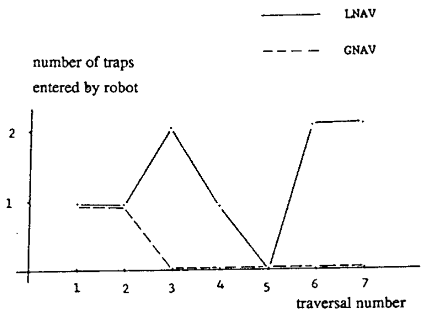


Fig. 8. relative performance

## ACKNOWLEDGEMENTS

We acknowledge the support of C. R. Weisbin of Oak Ridge National Laboratory. We also acknowledge the support of O. Manley of the Office of Basic Energy Sciences and the encouragement of Alex Zucker and F. C. Maienschein.

## REFERENCES:

[1]. Abelson, H., and A. diSessa, *Turtle Geometry*, Cambridge, Ma., MIT Press, 1980.

[2]. Chatila, R., Path planning and environment learning in a mobile robot system, *Proc. European Conf. Artificial Intelligence*, Torsey, France, 1982.

[3]. Iyengar, S.S., C.C. Jorgensen, S.V.N. Rao, and C.R. Weisbin, Robot navigation algorithms using learned spatial graphs, **Robotica**, vol. 4, Jan. 1986, 93-100.

[4]. Lozano-Perez, T. and M. Wesley, An algorithm for planning collision-free paths for an autonomous vehicle, **Commun. ACM**, vol. 22, 1979, 560-570.

[5]. Lumelsky, V.J. and A.A. Stepanov, Dynamic path planning for a mobile automaton with limited information on the environment, **IEEE Trans. on Automatic Control**, vol. AC-31, no.11, Nov. 1986, 1058-1063.

[6]. O'Dunlaing, C. and C.K. Yap, A "retraction" method for planning the motion of a disc, **J. Algorithms**, vol.6, 1985, 104-111.

[7]. Oommen, J.B., S.S. Iyengar, N.S.V. Rao, and R.L. Kashyap, Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case, **IEEE J. Robotics and Automation**, Vol. RA-12, 1987.

[8]. Rao, N.S.V., S.S. Iyengar, J.B. Oommen, and R.L. Kashyap, Terrain acquisition by a point robot amidst polyhedral obstacles, *Proc. 3rd IEEE Conf. on AI Appl.*, Orlando, Fl. Feb 1987 pp. 170-175.

[9]. Rao, N.S.V., S.S. Iyengar, G. deSauaaure, The visit problem: visibility graph based approach, Tech. Rep #ORNL/TM-, Oak Ridge National Laboratory, Oak Ridge (under preparation).

[10]. Reif, J., Complexity of mover's problems and generalizations, *Proc. 20th Ann. Symp. on Found. Comput. Sci.*, 1979, 421-427.

[11]. Schwartz, J.T. and M. Sharir, On the piano movers' problem I: The special case of a rigid polygonal body moving amidst polygonal barriers, **Commun. Pure and Appl. Math.**, vol. 36, 1983, 345-398.