NPSCS-91-011

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Persistence Search - A New Search Strategy For
The Dynamic Shortest Path Problem

Man-Tak Shing
Michael M. Mayer

APRIL 1991

Prepared for:

Naval Postgraduate School
Department of Computer Science, Code CS
Monterey, California 93943-5100

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared in conjunction with research funded by the Naval Postgraduate School.

Man-Tak Shing
Associate Professor
of Computer Science

Reviewed by:

ROBERT B. MCGHEE
Chairman
Department of Computer Science

Released by:

Paul J. Marto
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): NPSCS-91-011	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Research Council of Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5100	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: O & MN, Direct Funding
8c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Persistence Search - A New Search Strategy For The Dynamic Shortest Path Problem (U)

12. PERSONAL AUTHOR(S)
M.T. Shing, M.M. Mayer

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Techical	FROM 10/88 TO 9/90	1991 April 17	38

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	persistence search, path planning, maze exploration, dynamic shortest path

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The research reported in this paper deals with the problem of searching through an unknown terrain by a physical agent such as a robot. The unknown terrain over which the agent will travel is represented by an undirected graph. The agent has no prior knowledge of the graph. It can only learn about its environment by physically roaming it. Given a starting location s, the agent tries to reach a target location t using the minimum amount of physical movement. This problem, which is a natural generalization of the classical shortest path problem, will be referred to as the dynamic shortest path problem. Most of the classical shortest path algorithms perform very poorly in the scenario of a physical agent traversing an initially unknown search space. They do not attempt to minimize the amount of physical movement required by the agent to reach the goal location. In order to overcome the failings of these search algorithms in dealing with searches of this particular nature, a new search strategy, called persistence search, is developed and presented in this paper.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Man-Tak Shing	22b. TELEPHONE (Include Area Code): (408) 646-2634	22c. OFFICE SYMBOL: CS/SH

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Persistence Search - A New Search Strategy For The Dynamic Shortest Path Problem

Man-Tak Shing[1] **and** **Michael M. Mayer**[2]

A-1

ABSTRACT

The research reported in this paper deals with the problem of searching through an unknown terrain by a physical agent such as a robot. The unknown terrain over which the agent will travel is represented by an undirected graph. The agent has no prior knowledge of the graph. It can only learn about its environment by physically roaming it. Given a starting location s, the agent tries to reach a target location t using the minimum amount of physical movement. This problem, which is a natural generalization of the classical shortest path problem, will be referred to as the *dynamic shortest path problem.* Most of the classical shortest path algorithms perform very poorly in the scenario of a physical agent traversing an initially unknown search space. They do not attempt to minimize the amount of physical movement required by the agent to reach the goal location. In order to overcome the failings of these search algorithms in dealing with searches of this particular nature, a new search strategy, called *persistence search,* is developed and presented in this paper.

KEYWORDS:

persistence search, path planning, maze exploration, dynamic shortest path

---

(1) Man-Tak Shing is with the Computer Science Department, Naval Postgraduate School, Monterey, CA 93943. This paper was prepared in conjunction with research funded by the Naval Postgraduate School.

(2) LT Michael M. Mayer, USN is with the Software Support Department, Naval Security Group Activity Skaggs Island, Sonoma, CA 95476. Research reported here was done while LT Mayer was a graduate student at the Computer Science Department, Naval Postgraduate School.

1. INTRODUCTION

One of the central problem-solving techniques in AI is the use of *search*. In this paper, we focus our attention on the problem of searching through an unknown terrain by a physical agent such as a robot. The unknown terrain over which the agent will travel is represented by an undirected graph. The agent has no prior knowledge of the graph. It can only learn about its environment by moving through it. Starting at a given vertex s, the agent tries to reach a target vertex t using the minimum amount of physical movement. This problem, which is a natural generalization of the classical shortest path problem, will be referred to as the *dynamic shortest path problem*. If the agent has prior knowledge of the entire graph, it can minimize its physical movement by first computing the shortest path from s to t using the standard shortest path algorithms and then moving towards t along the shortest path. (See for example, [2] and [3] for a survey of the shortest path algorithms.) Without complete information of the entire graph, no optimum algorithm (i.e. one that always produces the shortest path from s to t in any given graph) can exist. The reason is because, without prior knowledge of the entire graph, the agent must physically roam its environment, learning about it by sensing the immediate surroundings. A state of any state-space search algorithm now depends on both the current physical location of the agent and the total amount of physical movement that has been made already. Since the energy or time involved in physical movement cannot be recovered once expended, any optimum algorithm must require the agent to traverse along the shortest path from s to t at all times. This is impossible since the algorithm does not have any prior knowledge of the graph to start with. Hence, any algorithm for solving the dynamic shortest path problem is a heuristic algorithm. Such algorithm is considered to be "good" if it produces paths which are not much longer than the shortest paths most of the time.

Most of the classical shortest path algorithms perform very poorly in the scenario of a physical agent traversing an initially unknown terrain. They do not attempt to minimize the amount of physical movement required by the agent to reach a desired goal location. These algorithms ignore the physical aspects of search, measuring the quality of their solutions only

by the amount of computation required. In order to overcome the failings of these search algorithms in dealing with searches of this particular nature, new search strategies and new measures for evaluating such strategies are needed.

One measure for comparing the performance of the algorithms for dynamic problems is the ratio

$$R = \frac{\text{the total distance traversed by the agent}}{\text{the length of the shortest path from } s \text{ to } t}$$

In [4], Papdimitriou and Yannakakis studied several versions of the dynamic shortest path problem. They have devised search strategies that optimize the ratio R for two special cases: layered graph with bounded width and two-dimensional scenes with unit square obstacles. They further showed that no bounded ratio is possible for slightly more general graphs and scenes, and the computational problem of devising optimal search strategies for the dynamic shortest path is *PSPACE*-Complete. Hence, a new heuristic search strategy, called *persistence search*, is developed and presented in this paper to tackle the dynamic shortest path problem for general graphs.

2. PERSISTENCE SEARCH

As stated in the previous section, the agent begins with no knowledge of the graph. It can learn about the graph by moving from vertex to vertex via the edges incident to the vertices, and it can remember the vertices and edges which it has explored by dropping markers along the way [1]. At each vertex v, the agent has the ability to discover all the edges incident to the vertex v. Additionally, for each unexplored edge incident to v, the agent has some estimation of the distance from the vertex v to the target vertex t via the unexplored edge. Starting at the vertex s, persistence search directs the agent towards the destination vertex t as follows:

Algorithm I

Begin

(1) EXIT := FALSE.

(2) $c := s$; /* c denotes the current position of the agent */

(3) L := empty; /* L denotes the set of edges to be explored, called *frontier edges* */

(4) Repeat

(5) Mark c as explored.

(6) If $c = t$

(7) Then EXIT := TRUE /* search has succeeded */

(8) Else Begin

(9) For each unexplored edge (c, w) do

(10) Begin

(11) If w is explored Then mark the edge (c, w) as explored

(12) Else add the edge (c, w) to the set L.

(13) End.

(14) If L is not empty

(15) Then Begin

(16) Among the frontier edges in L, find the frontier edge e that minimizes the equation

$$f(e) = pf \times g(v) + h(e), \qquad (1)$$

where v is the explored vertex adjacent to e. $g(v)$ is the shortest distance from c to v using only edges in the explored subgraph, $h(e)$ is the estimated future cost of e, which is the lower bound estimate for the distance from v to t via the unexplored edge e, and pf is the *persistence factor*, a real number between 0.0 and 1.0 for discounting the cost of "backtracking" to the frontier edge e.

```
(17)            Move the agent from c to v along the shortest path using only edges
                   in the explored subgraph.
(18)            Mark the edge e explored.
(19)            Let w be the vertex at the other end of e. Move the agent along e to w.
(20)            c := w;
(21)         End
(22)         Else EXIT := TRUE        /* search has failed */
(23)      End
(24)   Until EXIT;
End.
```

While the above algorithm is very close to the algorithm for A^* search [5], one should note the subtle but important differences. The function $g(v)$ is computed from the current vertex c to v, rather than from the initial starting vertex s. This is a logical choice for $g(v)$ since the agent must be moved physically from the current position c to the vertex v before it can resume its exploration from v.

A second difference is in the use of the persistence factor pf to bias the cost function $g(v)$. The persistence factor varies between 0.0 and 1.0 and serves to discount the cost of "backtracking" versus estimated future cost, $h(e)$. By varying the persistence factor, the behavior of the persistence search can be dramatically altered. When the persistence factor is 0, the cost of "backtracking" from one vertex to another becomes zero and the formula for rating the frontier edges reduces to:

$$f(e) = h(e) \tag{2}$$

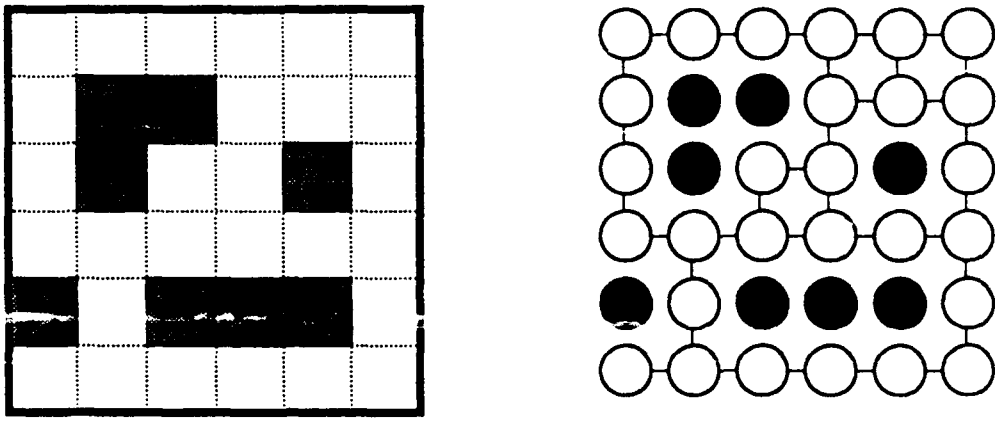
which is equivalent to that used for best-first search [5]. Each frontier edge is ranked only according to its estimated future cost. The agent will move about the explored search space without regard for the amount of physical movement required, traversing to whichever vertex is closest to goal.

When the persistence factor is equal to 1, persistence search behaves more like hill-climbing search [5]. Since the cost of moving from the current vertex to another vertex in the explored subgraph becomes more expensive, the agent tends to be more persistent in continuing the exploration from a vertex nearer to the present location.

3. SEARCHING THROUGH A RECTILINEAR MAZE

As an initial step in evaluating the performance of the proposed algorithm, persistence search is used to tackle a restricted version of the dynamic shortest problem: the case of an agent traversing a random rectilinear maze.

The unknown terrain over which the agent will travel is represented by a rectilinear graph or grid-graph as shown in Figure 1.



(a) a rectilinear maze (b) the corresponding grid-graph

Figure 1

Vertices are arranged in a rectangular fashion with edges connecting vertices immediately above, below, right and left (north, south, east and west respectively), and the vertices are distinguished from one another by their X and Y coordinates. Locations within the maze are either passable or impassable. Impassable locations in the maze are represented by disconnected vertices in the graph, making them unreachable from any direction.

The agent begins with no knowledge of the graph. It can learn about the graph only as it moves from vertex to vertex via the edges connecting them. The agent's ability to sense its surroundings is limited to determining whether the immediate neighbors of the current vertex are passable or impassable. Since the terrain is a grid-graph, the agent is only able to move and survey in the four cardinal directions.

Additionally, the agent knows its position relative to the goal. Several different means could be used to accomplish this. If the agent initially knows the goal's position relative to its own, it can maintain this knowledge through dead reckoning as it moves about the maze from its initial starting point. Using this information, the agent can determine the rectilinear distances from the goal to its current location as well as to any of the passable locations adjacent to its current location. Rectilinear distance is a lower bound for the number of steps necessary to move from the current position to the goal. The rectilinear distance from a vertex v to the goal t can be computed using the formula

$$|X_v - X_t| + |Y_v - Y_t|$$

where (X_v, Y_v) and (X_t, Y_t) are the coordinates of v and t respectively.

We can also direct the agent to the goal using either an active guidance system like radar or a passive guidance system like a beaconing system. While only lines of bearing are required to choose which immediate neighbor is closer to the goal, the goal's exact location is needed to totally order the search space. If the goal is northeast of the current location, then the immediate neighbors to the north and the east are closer to the goal and have the same rectilinear distance to the goal. However, when comparing widely separated locations, a ranking for the locations based on their closeness to the goal cannot be determined without the exact location of the goal. If location P is north of the goal and location Q is south of the goal, there is no static way to tell which one is closer merely by their relative direction to the goal. However, by triangulating lines of bearing taken from different locations, the exact location of the goal can be determined.

Since the edges in the grid-graph all have equal length, we can assume that the agent always knows the cost of traversing an unexplored edge. Hence, instead of keeping track of a set of frontier edges, we can simplify the implementation of the algorithm by keeping track of the set of unexplored vertices that are adjacent to the frontier edges. The algorithm for searching through an unknown rectilinear maze is as follows:

Algorithm II

Begin

(1) EXIT := FALSE;

(2) $c := s$; /* c denotes the current position of the agent */

(3) $L := \{s\}$; /* L denotes the set of vertices to be explored, called *frontier vertices* */

(4) Repeat

(5) Remove c from L.

(6) Mark the vertex c explored.

(7) If $c = t$

(8) Then EXIT := TRUE /* search has succeeded */

(9) Else Begin

(10) $L := L \cup \{$ the set of unexplored immediate neighbors of c $\}$.

(11) If L is not empty

(12) Then Begin

(13) Among the frontier vertices in L, find the frontier vertex v that minimizes the equation

$$f(v) = pf \times g(v) + h(v), \qquad (3)$$

where $g(v)$ is the shortest distance from c to v using only paths with explored vertices as intermediate vertices, $h(v) = |X_v - X_t| + |Y_v - Y_t|$, and pf is the *persistence factor* with value between 0.0 and 1.0.

(14) Move the agent from c to v along the shortest path using only explored vertices as intermediate vertices.

(15) $c := v$;

(16) End

(17) Else EXIT := TRUE /* search has failed */

(18) End

(19) Until EXIT;

End.

In order to reduce the computation required by persistence search, we further modify the algorithm as follows:

Algorithm III

Begin

(1) EXIT := FALSE;

(2) $c := s$; /* c denotes the current position of the agent */

(3) $L := \{s\}$; /* L denotes the set of vertices to be explored, called *frontier vertex* */

(4) Repeat

(5) Remove c from L.

(6) Mark the vertex c explored.

(7) If $c = t$

(8) Then EXIT := TRUE /* search has succeeded */

(9) Else Begin

(10) $L := L \cup \{$ the set of unexplored immediate neighbors of c $\}$.

(11) If c has an unexplored immediate neighbor w such that $h(w) < h(c)$ where $h(w)$ and $h(c)$ denote the rectilinear distances from w and c to the goal location t respectively

(12) Then $c := w$ and move the agent to the vertex w

(13) Else If L is not empty

(14) Then Begin

(15) Among the frontier vertices in L, find the frontier vertex v that minimizes the equation

$$f(v) = pf \times g(v) + h(v),$$

where $g(v)$ is the shortest distance from c to v using only paths with explored vertices as intermediate vertices, $h(v) = |X_v - X_t| + |Y_v - Y_t|$, and pf is the *persistence factor* with value between 0.0 and 1.0.

(16) Move the agent from c to v along the shortest path using only explored vertices as intermediate vertices.

(17) $c := v$;

(18) End

(19) Else EXIT := TRUE /* search has failed */

(20) End

(21) Until EXIT;

End.

Now, the agent always moves to the immediate neighbor w of the current vertex c if $h(w) < h(c)$ (lines 11-12 in Algorithm III). A simple north, east, south, west preference serves as a heuristic to break ties between neighbors of equal estimated future cost. This modification greatly reduces the computation required by the agent since it no longer has to examine the list of frontier vertices whenever c has an immediate neighbor with a cheaper future cost. Algorithm III can be regarded as a hybrid of hill climbing search and persistence search. Note that when $pf = 1$, the physical moves produced by Algorithms II and III are identical for all rectilinear mazes. When $pf < 1$, Algorithms II and III only produced identical moves for rectilinear tree mazes, i.e. mazes without circuits.

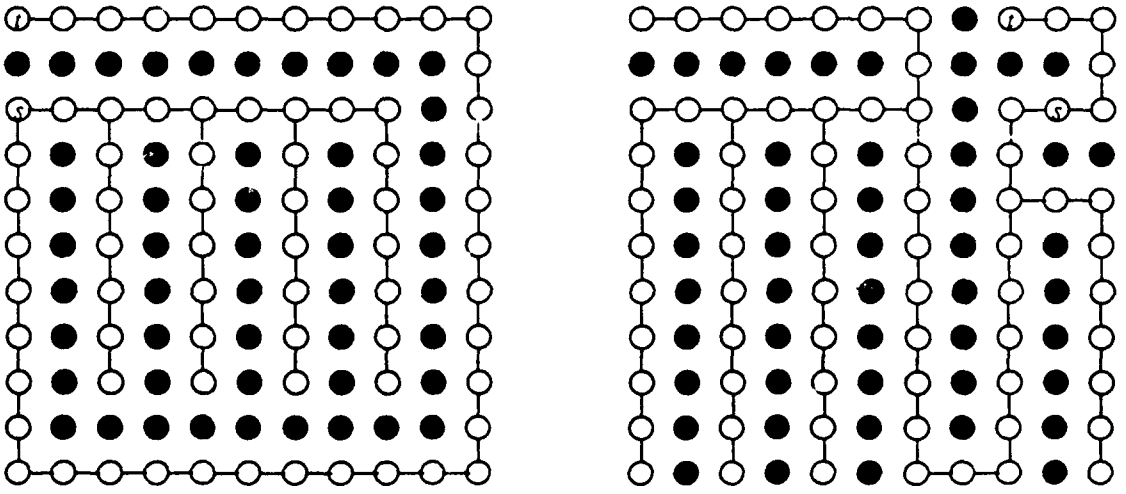
4. EVALUATING THE PERFORMANCE OF THE ALGORITHMS

There are two factors governing the performance of the proposed algorithms: the computational efficiency of the algorithms and the quality of the solutions produced. As mentioned in Section 1, the quality of the solution can be measure by the error ratio R, which is equal to

$$\frac{\text{the total number of edges traversed by the agent to reach the goal}}{\text{the length of the shortest path from } s \text{ to } t \text{ in the entire maze}}$$

The computational efficiency of the algorithms can be measured by counting the total number of current vertices examined plus the total number of vertices examined while computing the shortest paths from a current vertex back to the frontier vertices via the explored subgraph.

Given any integer n and any pf value between 0.0 and 1.0, one can always come up with a worst-case maze with n vertices that requires $O(n^2)$ vertex-examinations and the solution produced by the algorithms has an $O(n)$ error ratio. Figure 2 shows two worst-case examples one for the pf value of 0 and the other for the pf value of 1.



(a) a worst-case example for $pf = 0.0$ (b) a worst-case example for $pf = 1.0$

Figure 2

While the agent may have to visit every vertex in the graph before finding the goal in the worst-case, the total distance traversed by the agent may be substantially lower on the

average. In order to better judge the performance of the persistence search algorithms, we have implemented Algorithms II and III as well as a hill-climbing search algorithm in the C programming language and applied the algorithms to a large set of random mazes generated by the algorithm described in the next section.

4.1 MAZE GENERATION

The random maze generator operates in two modes, creating mazes with or without cycles. The mazes without cycles can be naturally represented as trees and those with cycles can be represented as graphs. We have separated the tree mazes from the nontree mazes because of the way persistence search backtracks. When backtracking, persistence search always travels to the cheapest frontier vertex (i.e. the frontier vertex with the cheapest combined cost) along the shortest path in the explored subgraph. This gives persistence search an extra advantage over hill-climbing search. The hill-climbing search, being a variant of depth-first search, does not take short-cuts and always traces back to the cheapest frontier vertex along the path from which it came. In non-tree mazes, this results in a large amount of back and forth movement by the hill-climbing search algorithm as it winds its way out of large open areas. This inefficient behavior can be seen clearly in Figure 3, where the hill-climbing search requires the agent (which at vertex 13) to visit all blank vertices before backtracking to vertex 14, while persistence search allows the agent to go from vertex 13 to vertex 14 via vertex s in two mechanical steps. If the maze is a tree, there are no circuits and persistence search no longer has such an advantage.

The maze generation algorithm, *makemaze*, uses a depth-first algorithm to create a random path through an initially empty maze, placing obstacles at intervals along its path according to a parameter called *maze density*. Makemaze begins by laying obstacles along the boundary of the rectangular maze. It then picks a random starting point within the maze. The algorithm maintains a trail of the locations in the maze it has explored.

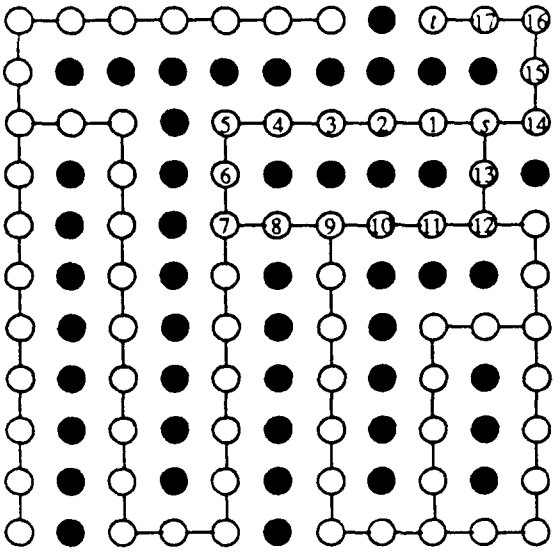


Figure 3

At each step along its path, it randomly determines whether it must turn from its current direction. If so, it places an obstacle in the direction it was going and continues on in a random new direction. The higher the maze density, the more likely it is that the path will turn and an obstacle will be created. If the maze generator becomes "blocked" by its own trail or obstacles, it backtracks along its original trail until it comes to a new location and resumes its exploration.

If makemaze is restricted to creating mazes which are trees, it must also check to see if the next location it is going to explore will create a cycle in the maze. If so, it places an obstacle there instead and chooses a new direction. Hence, there is only one path between any two vertices in a tree maze.

4.2 THE HILL-CLIMBING SEARCH

For comparison with persistence search, a hill-climbing search algorithm was implemented. We chose to compare the proposed algorithm against the hill-climbing search because the hill-climbing search (or some other depth-first search variant) is the only

reasonable alternative to persistence search in guiding a physical agent through an unknown bounded terrain. The other alternatives, like best-first search or some variants of A* search, require too much movement on the part of the physical agent, causing it to physically crisscross known search space to expand new vertices.

The hill-climbing search algorithm is very similar to the algorithm used to create the maze, as they both are variants of the depth-first search algorithm. The hill-climbing search algorithm uses a simple stack to keep track of its current path and a bitmap to mark the vertices it has visited. It places the unexplored immediate neighbors of the current vertex on the stack according to the non-increasing order of their estimated future costs, leaving the one with the cheapest estimated future cost on the top of the stack. It then pops a vertex off the stack, updates the current vertex and moves the physical agent to the new current vertex.

4.3 SPEEDING UP THE PERSISTENCE SEARCH

A major portion of the computation is spent in Line 13 of Algorithm II and Line 15 of Algorithm III for finding the shortest-path from the current vertex to the cheapest frontier vertex in the list L. Since the edges in the grid-graph all have equal length, a simple breadth-first search (BFS) is used to scan the explored subgraph for the cheapest frontier vertex each time the list L has to be examined. In order to reduce the time required to compute these shortest paths, "branch-and-bound" is introduced into the breadth-first search as follows. At the beginning of the BFS, the lower bound for the cheapest combined cost is initialized to the smallest estimated future cost of all the frontier vertices in L and the upper bound for the cheapest combined cost is initialized to positive infinity (or some very large positive number). During the course of the search, the upper bound for the cheapest combined cost is set to the lowest combined cost of the frontier vertices visited so far while the lower bound for the cheapest combined cost is set to the sum of the current depth reached by BFS plus the cheapest estimated future cost among all frontier vertices in L. To keep track of the smallest estimated future cost among all the frontier vertices in L, the vertices in L are maintained in

the form of a min-heap, with the frontier vertex which has the smallest estimated future cost on top. The min-heap is updated whenever vertices are added to or deleted from L. The search terminates when the lower bound is greater than or equal to the upper bound, thus signaling that the frontier vertex with the lowest combined cost has been found. The agent can then traverse from the current vertex c to the frontier vertex v via the search tree constructed by BFS.

4.4 THE EXPERIMENTAL RESULTS

Two experiments were conducted to study the performance of the proposed algorithms. In the first experiment, a total of 8000 runs were performed and analyzed empirically. Input to the first experiment was generated as follows. Two thousand mazes, in groups of 50, were created using the algorithm described in Section 4.1. The mazes in each group were generated based on a unique combination of maze type, maze density and dimension of the underlying rectangular grid. (See Table 1 for a summary of the mazes.) Two pairs of passable locations were chosen from each maze, and two experimental runs were performed for each of the location pair $[a, b]$ chosen, once from a to b and once from b to a.

The results of the first experiment are summarized in Tables 2-7. From the experimental results, one can conclude that persistence search (both Algorithms II and III) outperforms hill-climbing search for both nontree mazes and tree mazes (Tables 2a, 2b, 3a and 3b). The solutions produced by Algorithms II and III are equally good (Tables 4a and 4b) but Algorithm III runs much faster than Algorithm II (Tables 6a, 6b, 7a and 7b).

For nontree mazes, Algorithm III achieves its best results when the persistence factor is between 0.4 and 0.6, producing solutions that are at most 1.45 times as long as the best possible solutions for sparse nontree mazes (mazes with density ≤ 0.5) and 6.5 times as long as the best possible solutions for dense nontree mazes (mazes with density = 0.9). Moreover, these performance ratios grow very slowly when compared to the growth rate of the sizes of the search space. Table 7a shows the computational overhead of Algorithm III for nontree mazes.

For persistence factors between 0.4 and 0.6, Algorithm III takes, on the average, between 15 to 24 computational steps for each physical step produced, which is an acceptable trade-off given today's high-speed computers. (A computational step corresponds to either a vertex examination or a heap operation. Each vertex examination involves marking the vertex as visited and adding the vertex's unvisited neighbors to the agenda, and each heap operation involves comparing a node against its parent and interchanging the positions of the two nodes.) Table 5a shows the comparisons between the solutions produced by Algorithm III and hill-climbing search for nontree mazes. Algorithm III clearly outperforms hill-climbing search in terms of both the frequency of producing better solutions and the average length of the solutions produced.

For tree mazes, Algorithm III achieves its best results when the persistence factor is between 0.2 and 0.4. Although the paths for tree mazes produced by Algorithm III are on the average shorter than those produced by hill-climbing search (Table 3b), Algorithm III only produces better solutions 30% percent of the time (Table 5b). This discrepancy is due to the anomalous behavior of the hill-climbing search. Hill-climbing search actually produces paths that are shorter than the corresponding average path lengths $\bar{R}_h$ over 75% of the time. Unfortunately, there are always five to ten runs in each maze group that cause hill-climbing search to produce paths that are fifty to a hundred times longer than the average lengths. Persistence search, on the other hand, always produces paths with lengths distributed uniformly around the corresponding average path lengths $\bar{R}_{III}$. Hence, persistence search should be used if one wants a good worst-case performance at the expense of longer computational time -- Algorithm III, with persistence factors between 0.2 and 0.4, takes on the average as high as 85 computational steps for each physical step produced.

In order to better understand how persistence factor affects the performance of persistence search for nontree mazes, another 7000 runs were performed and analyzed empirically. (See Table 8 for a summary of the input data.) The results of the second experiment are summarized in Tables 9-11. For all the nontree mazes tested, Algorithm III seems to

perform equally well with different persistence factors between 0.4 and 0.6. More important, the performance ratios remain fairly constant with respect to the increasing sizes of the search space. Hence, one can conclude from the experimental results that persistence search with persistence factor between 0.4 and 0.6 is indeed a very good heuristic algorithm for searching through unknown nontree mazes.

5. CONCLUSIONS

In this paper, a new search strategy is proposed to minimize the total distance traversed by an agent in search of the target location. In light of our experimental results, persistence search is a very good strategy for searching unknown nontree mazes. The success of persistence search comes from its ability to cut across its own path to take advantage of the short circuits in the terrain, as well as the use of the persistence factor to discount the cost of abandoning the current location in order to continue the exploration somewhere else. This is particularly useful in exploring unknown unbounded mazes. Persistence search (with persistence factor less than one) always finds the goal as long as it is reachable from the starting location, while hill-climbing search may make a wrong turn and miss the goal forever. Persistence search is the preferred strategy for searching tree mazes if one wants to minimize the worst-case path length. One way to improve the computational efficiency of persistence search for tree mazes is to modify persistence search so that it behaves like hill-climbing search most of the time and only evaluates the possibility of backtracking to another frontier vertex once every K mechanical steps for some large constant K.

6. ACKNOWLEDGMENT

The authors would like to thank Robert B. McGhee for proposing this problem and Timothy Shimeall for his help in analyzing the experimental results.

7. REFERENCES

[1] G. Dedek, M. Jenkin, E. Milios and D. Wilkes, "Robotic Exploration as Graph Construction," Tech. Report, RBCV-TR-88-23, Department of Computer Science, University of Toronto, Ontario, Canada, 1988.

[2] S.E. Dreyfus, "An Appraisal of Some Shortest Path Algorithms," *Operations Research*, vol. 17, pp. 393-411, 1969.

[3] C.H. Papadimitriou, "Shortest Path Motion," *Proc. of the 1987 FST-TCS Conference*, 1987.

[4] C.H. Papadimitriou and M. Yannakakis, "Shortest Paths Without a Map," *Proc. of the 1989 ICALP Conference*, 1989.

[5] P.H. Winston, *Artificial Intelligence, 2nd Ed.*, Addison Wesley, Reading, MA., 1984.

group number	maze type	number of mazes	maze density	grid dimension	avg. number of passable cells	avg. length of the shortest paths
(1)	nontree	50	0.1	128 x 128	14734.38	84.76
(2)	nontree	50	0.1	192 x 192	33503.52	120.84
(3)	nontree	50	0.1	256 x 256	59866.72	167.34
(4)	nontree	50	0.1	320 x 320	93830.20	215.70
(5)	nontree	50	0.3	128 x 128	13137.48	90.35
(6)	nontree	50	0.3	192 x 192	29875.64	142.60
(7)	nontree	50	0.3	256 x 256	53382.76	175.55
(8)	nontree	50	0.3	320 x 320	83697.34	218.54
(9)	nontree	50	0.5	128 x 128	11821.24	89.10
(10)	nontree	50	0.5	192 x 192	26889.66	138.35
(11)	nontree	50	0.5	256 x 256	48046.98	176.68
(12)	nontree	50	0.5	320 x 320	75283.82	240.48
(13)	nontree	50	0.7	128 x 128	10475.96	106.90
(14)	nontree	50	0.7	192 x 192	23843.44	162.74
(15)	nontree	50	0.7	256 x 256	42594.08	205.43
(16)	nontree	50	0.7	320 x 320	66799.02	249.33
(17)	nontree	50	0.9	128 x 128	7808.36	177.92
(18)	nontree	50	0.9	192 x 192	17914.74	226.88
(19)	nontree	50	0.9	256 x 256	32259.48	300.98
(20)	nontree	50	0.9	320 x 320	50661.42	384.26
(21)	tree	50	0.1	128 x 128	8884.88	720.03
(22)	tree	50	0.1	192 x 192	20156.64	1554.56
(23)	tree	50	0.1	256 x 256	35960.40	2536.39
(24)	tree	50	0.1	320 x 320	56326.42	3259.97
(25)	tree	50	0.3	128 x 128	9077.72	869.04
(26)	tree	50	0.3	192 x 192	20608.66	1932.15
(27)	tree	50	0.3	256 x 256	36778.46	2878.34
(28)	tree	50	0.3	320 x 320	57575.86	4719.78
(29)	tree	50	0.5	128 x 128	8890.08	875.58
(30)	tree	50	0.5	192 x 192	20222.80	1929.14
(31)	tree	50	0.5	256 x 256	36088.92	2961.83
(32)	tree	50	0.5	320 x 320	56561.28	3924.45
(33)	tree	50	0.7	128 x 128	8168.80	745.53
(34)	tree	50	0.7	192 x 192	18670.30	1573.50
(35)	tree	50	0.7	256 x 256	33447.74	2526.46
(36)	tree	50	0.7	320 x 320	51407.10	3821.66
(37)	tree	50	0.9	128 x 128	5397.86	455.80
(38)	tree	50	0.9	192 x 192	12674.86	901.80
(39)	tree	50	0.9	256 x 256	22907.76	1624.02
(40)	tree	50	0.9	320 x 320	36279.30	2340.57

Table 1 : Summary of the 2000 mazes used in the first experiment

maze group	$\bar{R}_h$	$\bar{R}_{II}$					
		$pf = 0.0$	$pf = 0.2$	$pf = 0.4$	$pf = 0.6$	$pf = 0.8$	$pf = 1.0$
(1)	1.105	1.060	1.060	1.060	1.060	1.060	1.071
(2)	1.080	1.061	1.061	1.061	1.061	1.061	1.074
(3)	1.101	1.065	1.065	1.065	1.065	1.065	1.073
(4)	1.116	1.058	1.058	1.058	1.058	1.058	1.075
(5)	1.748	1.193	1.193	1.192	1.195	1.197	1.267
(6)	1.715	1.198	1.198	1.195	1.197	1.198	1.249
(7)	1.432	1.210	1.207	1.207	1.210	1.209	1.248
(8)	2.351	1.211	1.211	1.210	1.211	1.213	1.271
(9)	4.710	1.433	1.422	1.394	1.415	1.422	1.579
(10)	4.940	1.454	1.435	1.429	1.446	1.452	1.583
(11)	3.948	1.435	1.428	1.417	1.425	1.430	1.561
(12)	6.434	1.460	1.438	1.435	1.434	1.487	1.656
(13)	10.725	2.195	2.023	1.985	1.958	2.003	2.488
(14)	11.492	2.213	1.998	1.986	1.971	2.031	2.409
(15)	13.249	2.296	2.055	1.997	2.013	2.031	2.453
(16)	16.514	2.226	2.037	1.971	1.976	1.980	2.348
(17)	17.403	14.515	6.241	5.994	6.089	6.411	7.505
(18)	31.105	12.015	5.686	5.066	5.511	6.144	8.472
(19)	50.371	15.597	6.491	5.991	6.469	6.392	8.663
(20)	68.756	12.093	5.608	5.305	5.216	5.581	7.118

Table 2a : Summary of the results for Algorithm II on the nontree mazes in the first experiment

Note:

$\bar{R}_h$ -- the average value of *h-moves/min-moves* over the 200 runs in each maze group,

$\bar{R}_{II}$ -- = the average value of *II-moves/min-moves* over the 200 runs in each maze group,

where

h-moves -- the total number of physical moves required by the agent to reach the goal under hill-climbing search,

II-moves -- the total number of physical moves required by the agent to reach the goal under Algorithm II,

min-moves -- the length of the shortest path from s to t in the entire maze.

maze group	$\bar{R}_h$	$\bar{R}_{II}$					
		$pf = 0.0$	$pf = 0.2$	$pf = 0.4$	$pf = 0.6$	$pf = 0.8$	$pf = 1.0$
(21)	13.065	38.997	10.895	10.296	11.488	9.673	13.065
(22)	13.666	67.050	14.577	14.979	14.631	13.789	13.666
(23)	29.515	95.562	17.136	16.171	17.489	17.251	29.515
(24)	23.079	110.570	19.272	19.206	21.156	21.031	23.079
(25)	10.059	31.770	10.876	10.571	10.583	9.094	10.059
(26)	12.543	58.243	12.713	11.933	12.484	12.213	12.543
(27)	51.699	75.415	13.650	16.174	18.977	18.357	51.699
(28)	26.052	109.929	16.981	20.367	22.071	18.719	26.052
(29)	11.863	33.949	9.291	10.315	10.824	9.384	11.863
(30)	19.964	54.030	11.058	12.647	13.566	12.528	19.964
(31)	17.367	72.856	15.579	15.141	15.927	15.336	17.367
(32)	22.518	94.564	16.875	18.393	20.809	20.476	22.518
(33)	10.069	32.153	9.023	9.370	9.044	9.736	10.069
(34)	18.723	46.966	12.327	11.305	11.729	12.003	18.723
(35)	20.672	70.423	13.882	15.400	15.399	15.133	20.672
(36)	46.383	90.225	14.504	15.458	18.954	16.293	46.383
(37)	10.989	31.353	7.724	8.086	7.298	8.905	10.989
(38)	30.009	48.389	10.447	10.460	10.028	10.274	30.009
(39)	24.401	70.389	13.050	13.078	12.920	15.196	24.401
(40)	17.953	82.959	13.639	14.820	14.251	13.847	17.953

Table 2b : Summary of the results for Algorithm II on the tree mazes in the first experiment

maze group	$\bar{R}_h$	$\bar{R}_{III}$					
		pf = 0.0	pf = 0.2	pf = 0.4	pf = 0.6	pf = 0.8	pf = 1.0
(1)	1.105	1.060	1.060	1.060	1.060	1.060	1.071
(2)	1.080	1.061	1.061	1.061	1.061	1.061	1.074
(3)	1.101	1.065	1.065	1.065	1.065	1.065	1.073
(4)	1.116	1.058	1.058	1.058	1.058	1.058	1.075
(5)	1.748	1.193	1.193	1.192	1.195	1.197	1.267
(6)	1.715	1.198	1.198	1.195	1.197	1.198	1.249
(7)	1.432	1.210	1.207	1.207	1.210	1.209	1.248
(8)	2.351	1.211	1.211	1.210	1.211	1.213	1.271
(9)	4.710	1.433	1.422	1.394	1.415	1.422	1.579
(10)	4.940	1.454	1.435	1.429	1.446	1.452	1.583
(11)	3.948	1.435	1.428	1.417	1.425	1.430	1.561
(12)	6.434	1.460	1.438	1.435	1.434	1.487	1.656
(13)	10.725	2.195	2.023	1.985	1.958	2.003	2.488
(14)	11.492	2.213	1.998	1.986	1.971	2.030	2.409
(15)	13.249	2.296	2.055	1.996	2.013	2.031	2.453
(16)	16.514	2.226	2.037	1.972	1.976	1.980	2.348
(17)	17.403	14.515	6.242	6.020	6.092	6.411	7.505
(18)	31.105	12.015	5.686	5.064	5.512	6.144	8.472
(19)	50.371	15.597	6.491	5.993	6.471	6.394	8.663
(20)	68.756	12.093	5.608	5.305	5.213	5.581	7.118

Table 3a : Summary of the results for Algorithm III on the nontree mazes in the first experiment

Note:

$\bar{R}_h$ -- the average value of *h-moves/min-moves* over the 200 runs in each maze group,

$\bar{R}_{III}$ -- = the average value of *III-moves/min-moves* over the 200 runs in each maze group,

where

h-moves -- the total number of physical moves required by the agent to reach the goal under hill-climbing search,

III-moves -- the total number of physical moves required by the agent to reach the goal under Algorithm III,

min-moves -- the length of the shortest path from s to t in the entire maze.

maze group	$\bar{R}_A$	$\bar{R}_{III}$					
		pf = 0.0	pf = 0.2	pf = 0.4	pf = 0.6	pf = 0.8	pf = 1.0
(21)	13.065	38.997	10.895	10.296	11.488	9.673	13.065
(22)	13.666	67.050	14.577	14.979	14.631	13.789	13.666
(23)	29.515	95.562	17.136	16.171	17.489	17.251	29.515
(24)	23.079	110.570	19.272	19.206	21.156	21.031	23.079
(25)	10.059	31.770	10.876	10.571	10.583	9.094	10.059
(26)	12.543	58.243	12.713	11.933	12.484	12.213	12.543
(27)	51.699	75.415	13.650	16.174	18.977	18.357	51.699
(28)	26.052	109.929	16.981	20.367	22.071	18.719	26.052
(29)	11.863	33.949	9.291	10.315	10.824	9.384	11.863
(30)	19.964	54.030	11.058	12.647	13.566	12.528	19.964
(31)	17.367	72.856	15.579	15.141	15.927	15.336	17.367
(32)	22.518	94.564	16.875	18.393	20.809	20.476	22.518
(33)	10.069	32.153	9.023	9.370	9.044	9.736	10.069
(34)	18.723	46.966	12.327	11.305	11.729	12.003	18.723
(35)	20.672	70.423	13.882	15.400	15.399	15.133	20.672
(36)	46.383	90.225	14.504	15.458	18.954	16.293	46.383
(37)	10.989	31.353	7.724	8.086	7.298	8.905	10.989
(38)	30.009	48.389	10.447	10.460	10.028	10.274	30.009
(39)	24.401	70.389	13.050	13.078	12.920	15.196	24.401
(40)	17.953	82.959	13.639	14.820	14.251	13.847	17.953

Table 3b : Summary of the results for Algorithm III on the tree mazes in the first experiment

maze	*pf* = 0.0			*pf* = 0.2			*pf* = 0.4			*pf* = 0.6			*pf* = 0.8			*pf* = 1.0		
group	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III
(1)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(2)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(3)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(4)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(5)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(6)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(7)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(8)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(9)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(10)	0	200	0	0	200	0	0	199	1	0	200	0	0	200	0	0	200	0
(11)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(12)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(13)	0	200	0	0	199	1	0	200	0	0	200	0	0	199	1	0	200	0
(14)	0	200	0	0	200	0	0	200	0	0	199	1	0	197	3	0	200	0
(15)	0	200	0	0	200	0	0	197	3	0	200	0	0	199	1	0	200	0
(16)	0	200	0	0	200	0	1	199	0	2	198	0	3	197	0	0	200	0
(17)	0	200	0	1	199	0	3	195	2	10	188	2	3	194	3	0	200	0
(18)	0	200	0	0	198	2	0	197	3	2	197	1	4	190	6	0	200	0
(19)	0	200	0	1	199	0	2	190	8	3	191	6	10	187	3	0	200	0
(20)	0	200	0	1	199	0	2	198	0	3	195	2	7	189	4	0	200	0

Table 4a : Comparisons between Algorithms II and III for the nontree mazes in the first experiment

Note:

II < III -- the number of times *II-moves<III-moves* among the 200 runs in each maze group,

II = III -- the number of times *II-moves=III-moves* among the 200 runs in each maze group,

II > III -- the number of times *II-moves>III-moves* among the 200 runs in each maze group.

maze	*pf* = 0.0			*pf* = 0.2			*pf* = 0.4			*pf* = 0.6			*pf* = 0.8			*pf* = 1.0		
group	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III	II<III	II=III	II>III
(21)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(22)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(23)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(24)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(25)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(26)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(27)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(28)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(29)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(30)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(31)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(32)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(33)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(34)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(35)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(36)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(37)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(38)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(39)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0
(40)	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0	0	200	0

Table 4b : Comparisons between Algorithms II and III for the tree mazes in the first experiment

maze	*pf* = 0.0			*pf* = 0.2			*pf* = 0.4			*pf* = 0.6			*pf* = 0.8			*pf* = 1.0		
group	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h
(1)	21	165	14	21	165	14	21	165	14	21	165	14	21	165	14	4	196	0
(2)	25	160	15	25	161	14	25	162	13	25	162	13	25	162	13	3	197	0
(3)	31	151	18	31	151	18	31	151	18	31	151	18	31	151	18	4	196	0
(4)	40	135	25	40	135	25	40	136	24	39	137	24	39	137	24	12	188	0
(5)	85	60	55	86	60	54	87	62	51	86	63	51	86	63	51	41	155	4
(6)	115	41	44	115	45	40	122	36	42	122	36	42	122	36	42	44	151	5
(7)	106	33	61	106	33	61	108	32	60	108	32	60	108	32	60	53	151	6
(8)	128	14	58	129	15	56	130	18	52	128	18	54	128	18	54	61	126	13
(9)	121	24	55	121	24	55	121	25	53	118	26	56	119	26	55	74	106	20
(10)	135	8	57	138	7	55	138	9	53	135	12	53	135	10	55	96	85	19
(11)	130	8	62	132	8	60	127	9	64	124	9	67	124	9	67	100	84	16
(12)	139	8	53	141	10	49	145	12	43	144	12	44	141	13	46	114	67	19
(13)	139	5	56	147	7	46	148	9	43	147	9	44	148	10	42	114	60	26
(14)	137	4	59	142	7	51	148	4	48	148	5	47	148	5	47	141	30	29
(15)	146	3	51	158	2	40	161	2	37	159	3	38	157	3	40	145	31	24
(16)	150	2	48	155	2	43	161	3	36	157	3	40	160	2	38	156	19	25
(17)	97	3	100	126	3	71	130	3	67	132	3	65	129	4	67	139	19	42
(18)	119	4	77	146	4	50	150	3	47	157	4	39	149	4	47	150	14	36
(19)	103	2	95	139	2	59	144	2	54	149	2	49	143	2	55	147	2	41
(20)	127	0	73	163	0	37	171	0	29	171	0	29	169	0	31	164	1	33

Table 5a : Comparisons between Algorithm III and hill-climbing search for the nontree mazes in the first experiment

Note:

III < h -- the number of times *III-moves<h-moves* among the 200 runs in each maze group,

III = h -- the number of times *III-moves=h-moves* among the 200 runs in each maze group,

III > h -- the number of times *III-moves>h-moves* among the 200 runs in each maze group.

maze	*pf* = 0.0			*pf* = 0.2			*pf* = 0.4			*pf* = 0.6			*pf* = 0.8			*pf* = 1.0		
group	m<h	m=h	m>h	m<h	m=h	m>h	m<h	m=h	m>h	m<h	m=h	m>h	m<h	m=h	m>h	m<h	m=h	m>h
(21)	16	4	180	57	4	139	63	4	133	61	6	133	62	8	130	0	200	0
(22)	18	0	182	52	0	148	49	0	151	55	0	145	63	1	136	0	200	0
(23)	13	0	187	46	0	154	50	0	150	50	0	150	42	1	157	0	200	0
(24)	14	1	185	52	1	147	64	1	135	51	1	148	51	3	146	0	200	0
(25)	22	0	178	49	0	151	43	0	157	59	0	141	58	0	142	0	200	0
(26)	20	0	180	40	0	160	49	1	150	41	1	158	41	1	158	0	200	0
(27)	30	0	170	56	0	144	50	0	150	51	0	149	42	0	158	0	200	0
(28)	9	0	191	42	0	158	37	0	163	42	0	158	37	0	163	0	200	0
(29)	23	0	177	45	0	155	49	0	151	45	0	155	56	0	144	0	200	0
(30)	21	0	179	52	0	148	38	0	162	43	0	157	41	0	159	0	200	0
(31)	19	0	181	54	0	146	52	1	147	45	1	154	48	1	151	0	200	0
(32)	14	0	186	52	0	148	49	0	151	40	0	160	52	0	148	0	200	0
(33)	21	0	179	43	0	157	53	2	145	59	2	139	52	3	145	0	200	0
(34)	26	1	173	53	1	146	52	1	147	65	0	134	67	1	132	0	200	0
(35)	25	0	175	52	1	147	54	0	146	47	0	153	53	0	147	0	200	0
(36)	20	4	176	54	4	152	49	5	146	50	5	145	52	5	143	0	200	0
(37)	32	3	166	63	3	134	69	3	128	82	5	113	87	6	107	0	200	0
(38)	31	3	166	69	3	128	73	4	123	80	4	116	97	5	98	0	200	0
(39)	24	1	175	48	1	151	63	1	136	52	1	147	67	1	132	0	200	0
(40)	20	1	179	62	1	137	64	1	135	70	1	129	70	1	129	0	200	0

Table 5b : Comparisons between persistent search and hill-climbing search for the tree mazes in the first experiment

maze	*pf* = 0.0		*pf* = 0.2		*pf* = 0.4		*pf* = 0.6		*pf* = 0.8		*pf* = 1.0	
group	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*
(1)	4.244	17.637	5.673	17.637	5.843	17.639	5.094	17.639	5.262	17.639	4.505	17.623
(2)	4.274	19.149	5.393	19.153	5.808	19.153	5.158	19.154	5.571	19.154	4.754	19.094
(3)	4.281	20.571	5.171	20.571	5.606	20.573	5.360	20.574	5.792	20.574	4.524	20.552
(4)	4.286	22.251	5.098	22.251	5.520	22.254	5.464	22.252	5.883	22.252	4.889	22.137
(5)	3.997	14.036	5.222	14.042	5.478	14.075	4.830	14.067	5.071	14.066	5.118	14.052
(6)	4.004	15.774	4.925	15.781	5.350	15.823	4.911	15.813	5.329	15.809	4.821	15.821
(7)	4.006	16.540	4.842	16.572	5.284	16.599	5.023	16.588	5.441	16.596	4.925	16.580
(8)	4.015	17.386	4.869	17.400	5.110	17.429	5.181	17.430	5.417	17.425	5.541	17.298
(9)	3.763	10.503	5.004	10.567	5.069	10.679	4.835	10.666	4.897	10.658	5.548	10.579
(10)	3.759	11.812	4.689	11.910	5.041	11.986	4.846	12.025	5.159	12.014	5.711	11.888
(11)	3.771	12.396	4.676	12.475	4.955	12.623	4.859	12.615	5.132	12.624	5.572	12.591
(12)	3.743	13.521	4.577	13.661	4.854	13.758	4.992	13.782	5.415	13.739	6.389	13.661
(13)	3.475	7.159	4.725	7.493	5.189	7.638	5.074	7.706	5.339	7.700	7.660	7.506
(14)	3.461	7.843	4.697	8.351	5.318	8.490	5.325	8.605	5.883	8.588	7.804	8.450
(15)	3.482	8.196	4.707	8.750	5.206	8.988	5.483	9.007	5.798	9.042	8.474	8.812
(16)	3.477	8.818	4.664	9.328	5.026	9.605	5.318	9.694	5.628	9.725	7.292	9.568
(17)	3.245	2.570	11.831	3.643	14.059	3.855	15.616	3.894	17.985	3.939	19.765	3.768
(18)	3.172	2.735	10.635	4.013	12.333	4.290	15.929	4.343	17.183	4.326	21.619	4.130
(19)	3.267	2.630	12.298	4.097	14.536	4.427	18.165	4.507	18.446	4.568	25.852	4.439
(20)	3.223	2.964	10.254	4.668	13.484	5.053	15.465	5.233	17.246	5.212	23.880	4.952

Table 6a : Average number of vertex examinations and heap exchanges required by Algorithm II for the nontree mazes in the first experiment

Note:

v-exam --average number of vertex examinations per mechanical step required by Algorithm II among the 200 runs in each maze group. Each vertex examination involves marking the vertex as visited and adding the vertex's unvisited neighbors to the agenda.

h-op -- average number of heap operations performed in the heap per mechanical step required by Algorithm II among the 200 runs in each maze group. Each heap operation involves comparing a node against its parent and interchanging the positions of the two nodes.

maze	*pf* = 0.0		*pf* = 0.2		*pf* = 0.4		*pf* = 0.6		*pf* = 0.8		*pf* = 1.0	
group	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*
(21)	2.838	0.693	23.123	1.858	27.505	1.994	28.750	1.938	28.098	2.041	38.140	2.402
(22)	3.180	0.551	47.708	1.986	60.338	1.989	57.370	1.939	50.233	1.934	72.450	2.392
(23)	3.517	0.468	82.153	2.014	100.852	1.956	81.770	1.860	73.668	1.777	102.289	2.426
(24)	3.645	0.469	114.477	2.039	114.327	2.011	109.523	1.816	87.854	1.829	136.336	2.373
(25)	2.905	0.805	31.066	2.037	34.360	2.028	31.034	1.997	30.816	2.113	36.962	2.772
(26)	3.193	0.587	59.417	2.015	58.978	2.013	59.763	1.797	47.448	1.775	65.648	2.733
(27)	3.302	0.581	88.955	2.088	93.490	1.993	78.441	1.785	71.208	1.742	97.989	2.735
(28)	3.583	0.422	162.733	1.977	126.012	1.907	105.296	1.706	88.109	1.620	120.056	2.664
(29)	2.972	0.780	29.624	2.009	36.331	2.010	33.599	2.005	33.183	2.016	39.163	2.721
(30)	3.209	0.670	65.914	2.089	66.994	2.040	58.260	1.824	54.575	1.939	65.872	2.864
(31)	3.375	0.557	103.539	2.101	88.804	2.040	80.441	1.801	71.162	1.784	92.799	2.830
(32)	3.465	0.487	148.733	2.108	132.015	1.966	109.083	1.730	90.874	1.715	128.980	2.605
(33)	2.992	0.807	28.408	2.048	32.476	2.110	34.522	2.083	34.738	2.175	34.210	2.746
(34)	3.233	0.716	59.310	2.144	59.317	2.099	58.967	2.031	55.413	2.160	63.458	2.755
(35)	3.388	0.583	100.379	2.088	94.974	2.008	84.351	1.836	78.634	1.897	89.846	2.674
(36)	3.641	0.537	134.305	2.126	128.385	1.990	108.152	1.850	93.005	1.921	118.842	2.699
(37)	2.904	0.816	20.242	1.900	25.515	1.996	27.124	2.112	27.559	2.152	30.440	2.199
(38)	3.188	0.661	38.303	1.915	53.529	2.017	50.565	2.081	53.647	2.140	57.660	2.252
(39)	3.480	0.600	75.368	2.110	83.098	2.195	94.074	2.149	81.345	2.232	78.773	2.563
(40)	3.561	0.475	107.567	2.121	144.533	2.094	118.255	2.060	109.707	2.141	114.442	2.519

Table 6b : Average number of vertex examinations and heap exchanges required by Algorithm II for the tree mazes in the first experiment

maze	*pf* = 0.0		*pf* = 0.2		*pf* = 0.4		*pf* = 0.6		*pf* = 0.8		*pf* = 1.0	
group	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*
(1)	1.087	17.637	1.127	17.637	1.120	17.639	1.125	17.639	1.116	17.639	1.227	17.623
(2)	1.083	19.149	1.124	19.153	1.125	19.153	1.119	19.154	1.119	19.154	1.326	19.094
(3)	1.087	20.571	1.128	20.571	1.139	20.573	1.120	20.574	1.129	20.574	1.218	20.552
(4)	1.076	22.251	1.109	22.251	1.119	22.254	1.108	22.252	1.116	22.252	1.411	22.137
(5)	1.274	14.036	1.382	14.042	1.399	14.075	1.401	14.067	1.407	14.066	1.927	14.052
(6)	1.279	15.774	1.376	15.781	1.387	15.823	1.379	15.813	1.390	15.809	1.761	15.821
(7)	1.296	16.540	1.397	16.572	1.441	16.599	1.428	16.588	1.444	16.596	1.820	16.580
(8)	1.305	17.386	1.397	17.400	1.442	17.429	1.426	17.430	1.464	17.425	2.163	17.298
(9)	1.537	10.503	1.766	10.567	1.769	10.679	1.873	10.666	1.879	10.658	2.622	10.579
(10)	1.539	11.812	1.745	11.910	1.815	11.986	1.864	12.025	1.889	12.014	2.713	11.888
(11)	1.537	12.396	1.733	12.475	1.788	12.623	1.810	12.615	1.860	12.624	2.626	12.591
(12)	1.543	13.521	1.736	13.661	1.817	13.758	1.822	13.782	2.008	13.739	3.117	13.661
(13)	1.843	7.159	2.330	7.492	2.558	7.638	2.624	7.706	2.709	7.700	4.345	7.506
(14)	1.889	7.843	2.440	8.351	2.733	8.490	2.838	8.607	3.083	8.589	4.458	8.450
(15)	1.941	8.196	2.528	8.750	2.743	8.990	2.927	9.008	2.987	9.043	4.894	8.812
(16)	1.878	8.818	2.463	9.328	2.565	9.604	2.718	9.694	2.795	9.724	4.110	9.568
(17)	2.523	2.570	8.574	3.643	9.754	3.855	10.465	3.893	11.343	3.939	11.986	3.768
(18)	2.476	2.735	7.647	4.014	8.350	4.290	10.355	4.343	10.935	4.325	12.880	4.130
(19)	2.670	2.630	9.093	4.097	10.071	4.426	11.857	4.507	11.621	4.567	15.304	4.439
(20)	2.627	2.964	7.625	4.668	9.261	5.054	10.263	5.234	11.007	5.212	14.105	4.952

Table 7a : Average number of vertex examinations and heap exchanges required by Algorithm III for the nontree mazes in the first experiment

Note:

v-exam --average number of vertex examinations per mechanical step required by Algorithm III among the 200 runs in each maze group. Each vertex examination involves marking the vertex as visited and adding the vertex's unvisited neighbors to the agenda.

h-op -- average number of heap operations performed in the heap per mechanical step required by Algorithm III among the 200 runs in each maze group. Each heap operation involves comparing a node against its parent and interchanging the positions of the two nodes.

maze	$pf = 0.0$		$pf = 0.2$		$pf = 0.4$		$pf = 0.6$		$pf = 0.8$		$pf = 1.0$	
group	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*
(21)	2.607	0.693	14.770	1.858	17.069	1.994	17.593	1.938	17.017	2.041	22.208	2.402
(22)	3.064	0.551	28.793	1.986	35.414	1.989	33.738	1.939	29.864	1.934	41.555	2.392
(23)	3.428	0.468	48.196	2.014	57.812	1.956	47.437	1.860	43.081	1.777	58.174	2.426
(24)	3.554	0.469	65.933	2.039	65.555	2.011	62.852	1.816	50.979	1.829	77.461	2.373
(25)	2.730	0.805	19.534	2.037	21.207	2.028	19.265	1.997	19.049	2.113	21.682	2.772
(26)	3.070	0.587	35.712	2.015	35.217	2.013	35.565	1.797	28.848	1.775	37.598	2.733
(27)	3.191	0.581	52.306	2.088	54.400	1.993	46.106	1.785	42.379	1.742	56.275	2.735
(28)	3.513	0.422	93.303	1.977	72.607	1.907	61.000	1.706	51.820	1.620	68.378	2.664
(29)	2.787	0.780	18.805	2.009	22.279	2.010	20.664	2.005	20.343	2.016	22.623	2.721
(30)	3.070	0.670	39.116	2.089	39.525	2.040	34.718	1.824	32.736	1.939	37.553	2.864
(31)	3.269	0.557	59.968	2.101	51.723	2.040	46.971	1.801	42.238	1.784	52.613	2.830
(32)	3.381	0.487	84.852	2.108	75.179	1.966	62.823	1.730	53.163	1.715	72.881	2.605
(33)	2.771	0.807	18.255	2.048	20.183	2.110	21.296	2.083	21.232	2.175	19.840	2.746
(34)	3.059	0.716	35.460	2.144	35.194	2.099	34.917	2.031	32.929	2.160	35.991	2.755
(35)	3.264	0.583	58.122	2.088	54.748	2.008	49.157	1.836	45.996	1.897	50.417	2.674
(36)	3.491	0.537	76.042	2.126	72.648	1.990	61.840	1.850	53.910	1.921	66.242	2.699
(37)	2.571	0.816	13.818	1.900	16.805	1.996	17.227	2.112	17.235	2.152	18.681	2.199
(38)	2.945	0.661	25.064	1.915	32.969	2.017	31.386	2.081	32.266	2.140	33.682	2.252
(39)	3.320	0.600	46.386	2.110	49.935	2.195	56.110	2.149	48.310	2.232	45.138	2.563
(40)	3.448	0.475	63.356	2.121	83.112	2.094	69.040	2.060	64.269	2.141	65.007	2.519

Table 7b : Average number of vertex examinations and heap exchanges required by Algorithm III for the tree mazes in the first experiment

group number	maze type	number of mazes	maze density	grid dimension	avg. number of passable cells	avg. length of the shortest paths
(41)	nontree	50	0.1	128 x 128	14735.60	81.12
(42)	nontree	50	0.1	192 x 192	33498.10	127.92
(43)	nontree	50	0.1	256 x 256	59854.94	178.71
(44)	nontree	50	0.1	320 x 320	93825.78	213.85
(45)	nontree	50	0.1	384 x 384	135424.20	248.73
(46)	nontree	50	0.1	448 x 448	184573.90	283.32
(47)	nontree	50	0.1	512 x 512	241348.26	338.12
(48)	nontree	50	0.3	128 x 128	13135.72	92.65
(49)	nontree	50	0.3	192 x 192	29874.00	132.12
(50)	nontree	50	0.3	256 x 256	53392.28	174.03
(51)	nontree	50	0.3	320 x 320	83700.30	209.69
(52)	nontree	50	0.3	384 x 384	120756.36	231.48
(53)	nontree	50	0.3	448 x 448	164621.76	306.44
(54)	nontree	50	0.3	512 x 512	215232.52	339.97
(55)	nontree	50	0.5	128 x 128	11825.30	84.90
(56)	nontree	50	0.5	192 x 192	26873.38	146.76
(57)	nontree	50	0.5	256 x 256	48046.96	172.79
(58)	nontree	50	0.5	320 x 320	75301.88	230.09
(59)	nontree	50	0.5	384 x 384	108664.68	275.99
(60)	nontree	50	0.5	448 x 448	148092.66	293.89
(61)	nontree	50	0.5	512 x 512	193664.86	336.02
(62)	nontree	50	0.7	128 x 128	10476.06	107.81
(63)	nontree	50	0.7	192 x 192	23821.20	142.65
(64)	nontree	50	0.7	256 x 256	42604.00	186.06
(65)	nontree	50	0.7	320 x 320	66786.08	240.72
(66)	nontree	50	0.7	384 x 384	96425.50	304.05
(67)	nontree	50	0.7	448 x 448	131414.64	346.17
(68)	nontree	50	0.7	512 x 512	171841.42	430.79
(69)	nontree	50	0.9	128 x 128	7800.44	155.87
(70)	nontree	50	0.9	192 x 192	17865.94	249.40
(71)	nontree	50	0.9	256 x 256	32196.04	312.63
(72)	nontree	50	0.9	320 x 320	50536.52	392.20
(73)	nontree	50	0.9	384 x 384	73132.48	429.22
(74)	nontree	50	0.9	448 x 448	99938.46	482.32
(75)	nontree	50	0.9	512 x 512	130727.92	568.87

Table 8 : Summary of the 1750 mazes used in the second experiment

maze group	$\bar{R}_h$	$\bar{R}_{III}$				
		pf = 0.40	pf = 0.45	pf = 0.50	pf = 0.55	pf = 0.60
(41)	1.222	1.066	1.066	1.066	1.066	1.066
(42)	1.438	1.066	1.066	1.067	1.067	1.067
(43)	1.146	1.070	1.070	1.071	1.071	1.071
(44)	1.077	1.062	1.062	1.062	1.062	1.062
(45)	1.069	1.062	1.062	1.062	1.062	1.062
(46)	1.074	1.064	1.064	1.064	1.064	1.064
(47)	1.081	1.063	1.063	1.063	1.063	1.063
(48)	1.781	1.194	1.194	1.193	1.193	1.194
(49)	1.552	1.200	1.200	1.202	1.202	1.202
(50)	2.367	1.207	1.207	1.210	1.210	1.211
(51)	2.764	1.196	1.196	1.198	1.198	1.199
(52)	1.490	1.208	1.207	1.209	1.209	1.210
(53)	1.406	1.222	1.222	1.223	1.223	1.225
(54)	1.390	1.188	1.188	1.189	1.189	1.189
(55)	2.995	1.438	1.438	1.423	1.423	1.432
(56)	5.641	1.412	1.414	1.419	1.418	1.416
(57)	3.812	1.500	1.500	1.508	1.508	1.523
(58)	4.624	1.408	1.411	1.411	1.411	1.413
(59)	11.396	1.428	1.429	1.433	1.433	1.436
(60)	2.542	1.444	1.444	1.445	1.446	1.445
(61)	3.067	1.437	1.438	1.435	1.435	1.438
(62)	10.408	2.062	2.087	2.078	2.078	2.100
(63)	32.736	2.123	2.132	2.115	2.117	2.095
(64)	11.397	1.928	1.934	1.927	1.925	1.934
(65)	36.639	1.990	1.983	1.995	1.999	2.015
(66)	19.107	1.996	2.011	2.011	2.014	2.026
(67)	16.616	2.010	2.011	1.975	1.980	1.970
(68)	32.626	1.966	1.958	1.965	1.967	1.963
(69)	23.815	5.131	5.211	5.285	5.293	5.489
(70)	32.713	5.096	5.031	4.918	5.100	5.306
(71)	53.981	5.141	5.231	5.018	4.969	5.216
(72)	46.527	5.751	5.321	5.699	5.800	5.509
(73)	63.471	5.567	5.555	5.408	5.399	5.621
(74)	67.442	5.106	5.275	5.289	5.263	5.095
(75)	78.456	5.668	5.849	5.895	6.069	5.937

Table 9 : Summary of the results for Algorithm III on the nontree mazes in the second experiment

..me group	pf = 0.40			pf = 0.45			pf = 0.50			pf = 0.55			pf = 0.60		
	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h	III<h	III=h	III>h
(41)	18	173	9	18	173	9	18	173	9	18	173	9	18	173	9
(42)	31	152	17	31	152	17	31	152	17	31	152	17	31	152	17
(43)	42	129	29	42	129	29	42	129	29	42	129	29	42	129	29
(44)	39	132	29	39	132	29	39	132	29	39	132	29	39	132	29
(45)	40	132	28	40	132	28	40	132	28	40	132	28	40	132	28
(46)	51	126	23	51	126	23	51	126	23	51	126	23	51	126	23
(47)	52	118	30	52	118	30	52	118	30	52	118	30	52	118	30
(48)	100	54	46	100	54	46	101	53	46	101	53	46	101	53	46
(49)	100	48	52	100	48	52	100	48	52	100	48	52	100	48	52
(50)	110	28	62	110	28	62	110	28	62	110	28	62	109	29	62
(51)	122	35	43	122	35	43	122	35	43	122	35	43	121	36	43
(52)	124	24	52	124	24	52	122	25	53	122	25	53	120	26	54
(53)	138	17	45	138	17	45	137	16	47	137	16	47	136	15	49
(54)	135	19	46	135	19	46	134	21	45	134	21	45	134	21	45
(55)	119	30	51	119	30	51	119	30	51	119	30	51	119	29	52
(56)	138	11	51	137	11	52	136	11	53	136	11	53	135	10	55
(57)	136	12	52	136	12	52	137	12	51	137	12	51	138	12	50
(58)	153	9	38	152	7	41	152	7	41	152	7	41	151	8	41
(59)	154	3	43	154	2	44	152	3	45	152	3	45	152	3	45
(60)	147	6	47	147	6	47	148	4	48	148	4	48	148	6	46
(61)	147	4	49	145	4	51	148	4	48	148	4	48	147	3	50
(62)	162	3	35	161	4	35	164	3	33	165	3	32	165	4	31
(63)	148	6	46	146	6	48	147	7	46	147	7	46	147	6	47
(64)	152	1	47	152	1	47	150	2	48	150	2	48	150	2	48
(65)	153	2	45	154	3	43	155	2	43	155	2	43	154	4	42
(66)	152	2	46	152	1	47	152	0	48	151	0	49	151	1	48
(67)	160	2	38	164	1	35	160	2	38	160	2	38	162	3	37
(68)	174	0	26	174	1	25	171	0	29	171	0	29	171	0	29
(69)	132	8	60	133	8	59	128	8	64	130	8	62	129	10	61
(70)	154	0	46	156	0	44	155	0	45	152	1	47	151	0	49
(71)	150	2	48	152	2	46	155	2	43	154	2	44	149	2	49
(72)	158	1	41	157	1	42	156	1	43	155	2	43	156	1	43
(73)	154	0	46	157	0	43	161	0	39	157	0	43	159	0	41
(74)	164	0	36	162	0	38	160	0	40	160	0	40	162	0	38
(75)	155	1	44	151	1	48	150	1	49	150	1	49	151	1	48

Table 10 : Comparisons between Algorithm III and hill-climbing search for the nontree mazes in the second experiment

maze	*pf* = 0.40		*pf* = 0.45		*pf* = 0.50		*pf* = 0.55		*pf* = 0.60	
group	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*	*v-exam*	*h-op*
(41)	1.132	17.056	1.135	17.056	1.094	17.056	1.127	17.056	1.130	17.056
(42)	1.146	19.362	1.140	19.362	1.097	19.359	1.134	19.359	1.130	19.358
(43)	1.143	20.990	1.134	20.990	1.097	20.989	1.135	20.989	1.127	20.987
(44)	1.123	22.004	1.112	22.004	1.081	22.004	1.122	22.004	1.110	22.004
(45)	1.127	22.610	1.118	22.610	1.086	22.610	1.129	22.610	1.122	22.608
(46)	1.122	23.432	1.114	23.432	1.084	23.432	1.127	23.432	1.118	23.432
(47)	1.125	24.180	1.112	24.180	1.084	24.181	1.131	24.181	1.118	24.181
(48)	1.389	14.009	1.403	14.009	1.308	14.023	1.371	14.023	1.391	14.023
(49)	1.454	15.349	1.444	15.349	1.367	15.344	1.443	15.344	1.451	15.349
(50)	1.441	16.415	1.426	16.415	1.355	16.416	1.441	16.416	1.445	16.413
(51)	1.391	17.203	1.377	17.204	1.309	17.197	1.394	17.197	1.391	17.192
(52)	1.412	17.823	1.385	17.826	1.318	17.821	1.418	17.821	1.402	17.820
(53)	1.458	18.680	1.440	18.680	1.364	18.682	1.462	18.682	1.467	18.672
(54)	1.367	19.321	1.351	19.321	1.291	19.327	1.386	19.327	1.376	19.324
(55)	1.790	10.165	1.796	10.166	1.672	10.188	1.780	10.188	1.833	10.183
(56)	1.811	12.215	1.805	12.212	1.693	12.214	1.806	12.218	1.801	12.226
(57)	1.902	12.507	1.891	12.517	1.786	12.528	1.913	12.533	1.942	12.535
(58)	1.763	13.635	1.771	13.633	1.641	13.637	1.764	13.637	1.775	13.650
(59)	1.798	14.343	1.766	14.341	1.677	14.328	1.810	14.327	1.811	14.325
(60)	1.853	14.464	1.833	14.468	1.762	14.482	1.899	14.482	1.877	14.476
(61)	1.833	14.810	1.807	14.812	1.716	14.860	1.867	14.860	1.882	14.848
(62)	2.710	7.475	2.817	7.466	2.667	7.504	2.814	7.504	3.024	7.492
(63)	2.797	8.080	2.864	8.079	2.745	8.108	2.891	8.107	2.927	8.166
(64)	2.540	8.992	2.523	8.989	2.447	9.009	2.601	9.017	2.644	9.035
(65)	2.610	9.571	2.597	9.583	2.515	9.587	2.700	9.587	2.810	9.593
(66)	2.625	10.219	2.717	10.203	2.611	10.220	2.779	10.218	2.863	10.212
(67)	2.744	10.239	2.747	10.260	2.555	10.324	2.732	10.316	2.819	10.349
(68)	2.627	11.135	2.588	11.172	2.476	11.203	2.663	11.200	2.740	11.232
(69)	8.163	3.829	9.760	3.843	10.452	3.857	11.030	3.836	10.720	3.858
(70)	8.389	4.442	8.175	4.493	8.563	4.554	9.208	4.551	9.704	4.556
(71)	10.319	4.655	9.844	4.706	9.235	4.749	9.397	4.760	10.196	4.741
(72)	9.711	4.887	9.124	4.943	10.417	4.978	10.919	5.002	10.337	5.074
(73)	10.296	4.979	10.159	5.003	10.356	5.066	10.591	5.101	11.841	5.125
(74)	9.363	5.335	10.528	5.324	10.895	5.382	11.468	5.382	10.284	5.490
(75)	13.050	5.372	13.775	5.397	14.978	5.466	16.762	5.478	15.677	5.526

Table 11 : Average number of vertex examinations and heap exchanges required by Algorithm III for the nontree mazes in the second experiment

Distribution List

Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145	2 copies
Library, Code 0142 Naval Postgraduate School, Monterey, CA 93943	2 copies
Center for Naval Analyses, 4401 Ford Avenue Alexandria, VA 22302-0268	1 copy
Director of Research Administration, Code 012, Naval Postgraduate School, Monterey, CA 93943	1 copy
Professor Robert B. McGhee Code CS Naval Postgraduate School Monterey, California 93943-5100	1 copy
Dr. Man-Tak Shing Code CS Naval Postgraduate School, Monterey, California 93943-5100	20 copies
LT Michael Mayer, USN Software Support Department Naval Security Group Activity Skaggs Island Sonoma, CA 95476	10 copies
Professor Evangelos Milos Department of Computer Science University of Toronto Toronto, Ontario, CANADA M5S 1A4	1 copy