Design and Implementation of a Novel Weighted Shortest Path Algorithm for Maze Solving Robots

Behnam Rahnama (*Senior Member, IEEE*),
Makbule Canan Özdemir, and Yunus Kıran
Computer Engineering Department
European University of Lefke
Gemikonagi, North Cyprus
rahnama@ieee.org

Atilla Elçi
Electrical and Electronics Engineering Department
Aksaray University
Aksaray, Turkey
atilla.elci@gmail.com

***Abstract*—This research presents design and implementation of the shortest path algorithm for labyrinth discovery application in a multi-agent environment. Robot agents are unaware of the maze at the beginning; they learn as they discover it. Each agent solves a part of the maze and updates the shared memory so that other robots also benefit from each other's' discovery. Finding of the destination cell by an agent helps others to interconnect their discovered paths to the one ending with the destination cell. The proposed shortest path algorithm considers the cost for not only coordinate distance but also number of turns and moves required to traverse the path. The Shortest Path algorithm is compared against various available maze solving algorithms including Flood-Fill, Modified Flood-Fill and $ALCK_{EF}$. The presented algorithm can be used also as an additional layer to enhance the available methods at second and subsequent runs.**

Keywords—Labyrinth, Maze Solving, Cooperative Robotics, Shortest Path

I. INTRODUCTION

Cooperative Labyrinth Discovery is where multiple of maze solving agents help each other in solving an uncharted and unknown maze. In this regard, various maze solving algorithms including Flood-Fill [1,2], Modified Flood-Fill [3], Variable Flood-Fill [4], and $ALCK_{EF}$ [5] are used to find the shortest possible path within available solutions from the starting point, typically the entry of the maze, to the destination, typically its exit.

The case of labyrinth discovery is to find the shortest possible path among discovered areas of the labyrinth. Yet it might not be the optimal solution due to the fact that the robot does not have the absolute information on entire platform but it learns as it goes. The aim of this research is finding the shortest path in an initially-unknown labyrinth based on consideration of the timing cost turns and moves, so that it provides the shortest timely possible discovered path. The proposed Shortest Path (SP) algorithm is first emulated then implemented on the target machine.

This task can be more interesting when multi-agents are solving the labyrinth sharing the information among each other. The real life applications of the cooperative labyrinth discovery include Traffic Management System, Multi-Layer PCB Routing, and Transporter Robots in Factories, Network Routing and Trajectory Planning.

II. RELATED WORK

There are many researches available in the literature for finding the best possible path including the work presented by [6]. In their approach, the problems of task allocation and path planning for multiple robots have been discussed in detail. Their proposed method utilizes A* algorithm. A* algorithm acts as the search algorithm among all possible paths and evaluates the path cost. The algorithm calculates the paths at each every step then the robot decides the path and chooses the least costly one. This continuous process provides calculation of the path at each step and updates all other nodes and queues for any calculated changes. However, A* algorithm requires a lot of computational power and large memory to keep the tree data structure and therefore it's not suitable for labyrinth discovery robots with limited resources running on a tiny microcontroller and carrying few kilobytes of RAM.

The other well-known method is to find the destination based on the Flood-Fill (FF) algorithm [1,2]. The FF assigns designation values to each of the maze cells. The destination will be assigned zero, neighboring cells to the destination will be considered as one and so forth the value of each cell represents its distance to the destination cell. If the mouse reaches a cell with the value of 4 it means that mouse is away from destination cell by 4 moves. Yet, the cost of moves and turns are not considered whereas in FF types of algorithms such as Modified FF in Variable FF moves and turns also affect the total cost.

The cooperative labyrinth discovery consists of multi-agents solving maze. All presented methods in [3] are based on sharing discovered information among agents.

978-0-7695-4987-3/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSACW.2013.49

In case an agent finds the destination cell, the rest of them try to apply search algorithms to find the destination cell as well. On the other hand, many researchers provide frameworks and solutions on how to share data and discovered information among agents. A method presented by [7] utilizes the Markov decision process. This structure guaranties sharing motivation and knowledge information within all agents.

The aim of solving labyrinth using multi-agents working cooperatively is to use energy efficiently by spending less time at discovery. All agents share partial information on the discovered locations of the maze. In addition, they are aware of all discovered locations by other colleagues. In this way they do not act individually but cooperatively.

In the next Section, we provide our proposed SP algorithm in two phases, firstly without consideration of cost for turns and secondly with consideration of cost of turns. Section IV presents the implementation of the cost based algorithm; Section V supplies comparative results. This paper concludes with Section VI.

III. PROPOSED WEIGHTED SHORTEST PATH ALGORITHM FOR MULTI-AGENT ENVIRONMENT

The proposed algorithm calls the ShortestPath function recursively to discover all cells which have been discovered at least once by any of the robots. In Multi Agent Maze Solving environment, robots will update the shared memory accessible by all agents.

The algorithm first checks if the current cell has been visited before and adds the current cell to the visited list if it's the first time at this round. Notice that due to the future updates of the visited list, we might add a cell several times for various paths.

The algorithm checks if the current cell is the destination; it returns "true" if so. In this way the true value is passed recursively to the parent caller and it increases the counter presenting the total number of found paths from the caller cell to the destination.

The algorithm considers all four directions (Forward, Right, Backward and Left) around the current cell for possible moves. If there is no wall towards that direction and if the neighboring cell is the destination cell or it has been visited by any agent already, and the neighboring cell has not been visited at the current call, then ShortestPath is called for all possible directions satisfying aforementioned conditions. Successful result entails adding the cell with returning true value to the recommend list and removing trace of all visited cells till the current cell from Visited list.

```
BOOL ShortestPath (Start, End)
BEGIN
   IF (Visited list does not include Start)
      ADD Start to Visited List
   IF (Start is the End)
   RETURN True
   Count = 0
   FOR (All directions as i)
   BEGIN
      IF (there is no wall towards ith direction of
      Start AND (ith Neighbor is End OR Counter of ith
      Neighbor > 0 // at least visited once) AND (ith
      Neighbor is NOT in Visited List))
   BEGIN
      IF (ShortestPath (ith Neighbor, End)
      BEGIN
         INCREASE Count by 1
         ADD ith Neighbor to the head of Recommend list
            REMOVE all elements from the end until ith
         Neighbor including itself from Visited list
         END
      END
   END
   IF (Count > 0) RETURN True
   ELSE RETURN False
END
```

Finally, robot starts moving according to the cells available at the recommend list in case the ShortestPath function returns true value in total meaning that at least one path from starting point to destination is found. The accompanying pseudo code presents the aforesaid shortest path algorithm.

In the shortest path costs for turns are not taken into account. For instance movement from cell (0, 0) to cell (4, 4) as shown in Fig.1 requires 8 moves and 7 turns in ladder mode but 8 moves and 1 turns in straight mode. However, as costs of turns are not considered, therefore, it returns both ways with the same weight. However, the ladder path is much more costly than straight path as robots spend time for about a portion of a second to turn. This adds up to the total cost affecting time and energy.

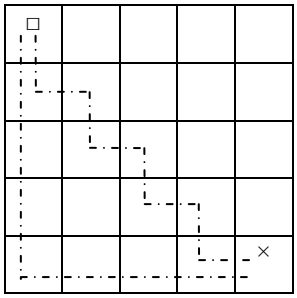
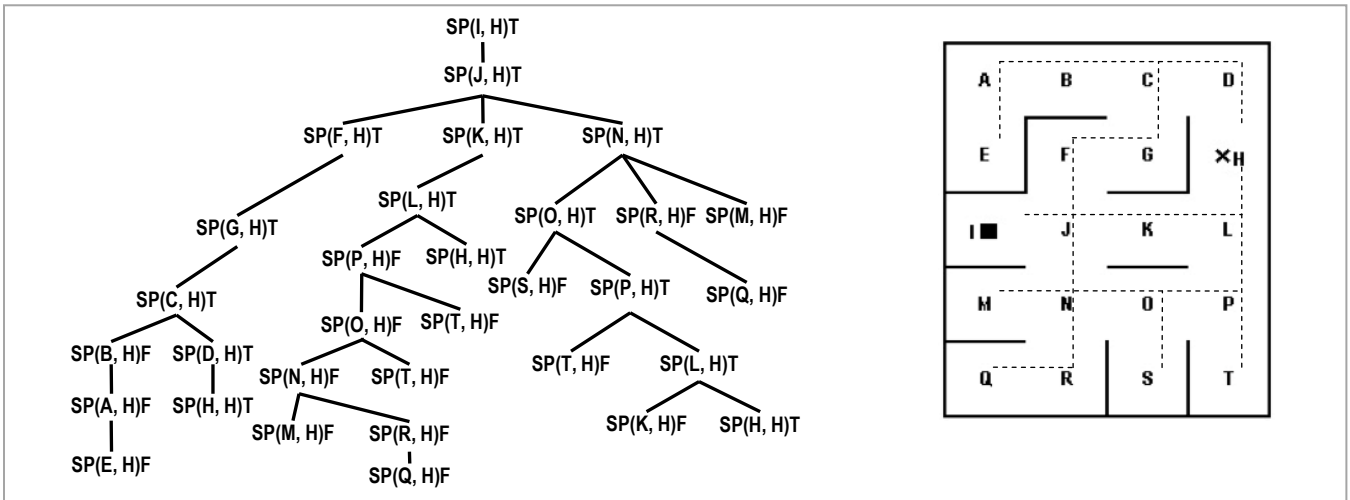


Fig.1. Comparison of the straight and ladder moves and turns

Following is the updated pseudo code for the Shortest Path method considering costs for turns as well as moves. In addition to the old style, now the algorithm checks the cost of turns as well as moves.

In reality a physical robot might take about a second to move from a cell to the next and half of a second to turn 90 degrees. Cost values are considered 3 and 2 accordingly. Robot facing forward requires turning right and then moves one block to reach the right hand side neighbor for instance. This creates the total cost of 5 while it only creates the cost of 2 for a forward movement. Each new call provides a new discovery and it is cancelled if the current cost is more than previous discovery with true answer. This way it eliminates the longer, that is, costlier, paths.



Visited [I, J, F, G, C, B, A, E, D, H, K, L, P, O, N, M, R, Q, T, H, N, O, S, P, T, L, K, H, R, Q, M]
Recommend [H, D, C, G, F, H, L, K, H, L, P, O, N, J]

Fig. 2. Tracing the Weighted Shortest Path Algorithm in Labyrinth Discovery to reach cell H from cell I

```
BOOL ShortestPath (Start, End, Cost, Direction)
BEGIN
   IF(Visited list does not include Start)
      ADD Start to Visited List
   IF(Cost > Minimum)
      RETURN False
   IF(Start is the End)
   BEGIN
   REMOVE all elements from Recommend List
   Minimum = Cost
   RETURN True
   END
   Count = 0
   FOR(All directions as i)
   BEGIN
      IF(there is no wall towards ith direction of
      Start AND(ith Neighbor is End OR Counter of ith
      Neighbor > 0 // at least visited once)AND(ith
      Neighbor is NOT in Visited List))
   BEGIN
      IF(Turn is not necessary)
         CurrentCost = 2
      ELSE
         CurrentCost = 3
      IF(ShortestPath (ith Neighbor, End,
   Cost+CurrentCost, i)
      BEGIN
         INCREASE Count by 1
         ADD ith Neighbor to the head of Recommend list
            REMOVE all elements from the end until ith
         Neighbor including itself from Visited list
         END
      END
   END
   IF(Count > 0)RETURN True
   ELSE RETURN False
END
```

Reference Fig. 2 below and let's suppose the robot wants to traverse from ***I*** to ***H***. There are different paths available so that the algorithm should choose the shortest one. The agent starts from cell ***I***. It moves to cell ***J*** and then it runs the shortest path towards three open ways namely ***K***, ***N*** and ***F***. Of course the cell ***I*** is excluded from new discovery list.

First trace will be towards cell ***F*** it follows cells ***G*** and ***C***. When reaching ***C*** it branches two ShortestPath functions for ***B*** and ***D*** cells. Cell ***B***, reaches the dead end ***E*** without reaching ***H*** and therefore it returns false. The other call towards ***D*** follows ***H***. This returns the first true with updating the calculated cost. The other branch call towards cell ***K***, reaching ***H*** via ***J***, ***K***, and ***L*** provides a lower cost and therefore it will be replaced by previously found trace.

Later, the robot agent calls the function towards ***N*** that is followed by ***O***, ***P*** and ***L*** cells in this way it reaches cell ***H*** with a greater cost and that's why it's bound by algorithm. Other traces end up returning false consequently.

Suppose robot wants to traverse from ***I*** to ***D*** without consideration of cost calculation for turns. The first algorithm does not consider costs for turns so that calculated cost for both traces (***J***, ***F***, ***G***, ***C***, and ***D***) and (***J***, ***K***, ***L***, ***H***, and ***D***) are five steps or the value 10 in the program. But the second one calculates cost for turns as well as moves therefore, total cost for first trace is 14 in comparison with 11 for the second trace. The usual maze solving algorithms only consider moves but in reality it differs in time as robots require time to turn as well.

IV. VC++ Weighted Shortest Path Implementation

The CRobot Class as given below defines all necessary methods for robot movement and robot communication. The SP method searches among all available paths. GetCounter reads the current value of the inquired cell and GetWall checks if there is a wall in the requested direction. Directions are sequentially Forward, Right, Backward and Left. GetNeighbor provides the row and column coordinate of the neighbor at a certain direction.

There are two lists namely, Visited, and Recommend. Visited indicates already processed cells and Recommend indicates the final shortest path from Start to the End.

```
bool CRobot::SP(CPoint Start, CPoint End, int Cost, int
Direction){
   if (Visited.Find(Start)==NULL)
      Visited.AddTail(Start);
   if (Cost>MinCost)
      return false;
   if (Start==End) {
      Recommend.RemoveAll();
      MinCost=Cost;
      return true;
   }
   int Count=0, CurrCost=2, IsWall=0, MazeCounter=0;
   CPoint Neighbor;
   for (int i=0; i<4; i++){
      Neighbor=GetNeighbor(Start, i);
      IsWall=SharedMemory->GetWall(Start, i);
      MazeCounter=SharedMemory->GetCounter(Start);
      if
      ((!IsWall)&&((Neighbor==End)||(MazeCounter>0))
      && (Visited.Find(Neighbor)==NULL)){
         CurrCost=(abs(Direction-i)%2==0)?2:3;
         if (SP(Neighbor, End, Cost+CurrCost, i)){
            Count++;
            Recommend.AddHead(i);
            while(Visited.GetTail()!=Neighbor)
               Visited.RemoveTail();
            Visited.RemoveTail();
         }
      }
   }
   if (Count)
      return true;
   return false;
}
```

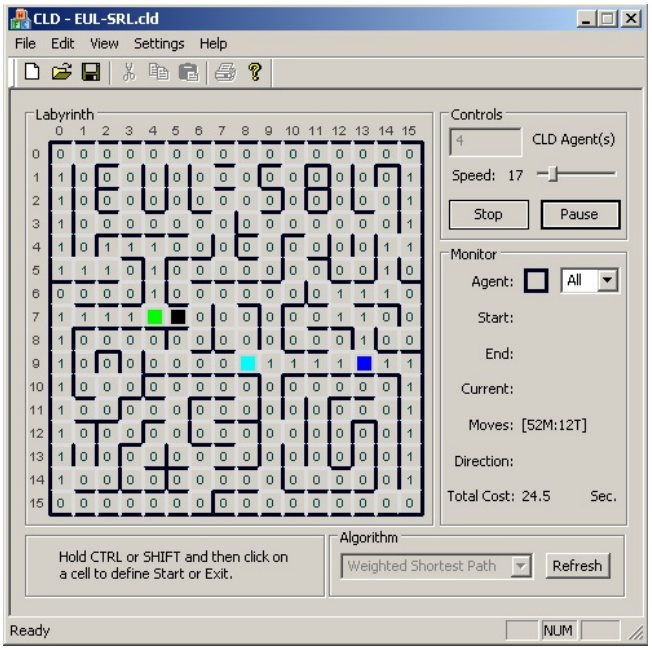


Fig. 3. Implementation of Weighted Shortest Path Algorithm in Cooperative Labyrinth Discovery platform

The following figure, presents the implementation of Weighted Shortest Path algorithm in Cooperative Labyrinth Discovery (CLD) [*8*] platform as four agents try to solve the maze cooperatively. The platform presents the individual cost for each robot as well as total cost based on number of moves and turns. Cell counters are updated when robots enter them.

Next section presents the results of the performance comparison of Weighted Shortest Path, Modified Flood Fill, Ideal Shortest Path, and $ALCK_{EF}$ algorithms in Multi Agent Environment.

V. COMPARISON

Performance evaluation of Flood Fill, Modified Flood Fill, Ideal, and $ALCK_{EF}$ algorithms was already reported in [3]. This time those reported results are compared against our proposed Weighted Shortest Path Algorithm over five different platforms used in international robotic contests. Following figure presents the performance comparison of 16 agents in labyrinth and it guaranties unmatched performance of the proposed algorithm against others.

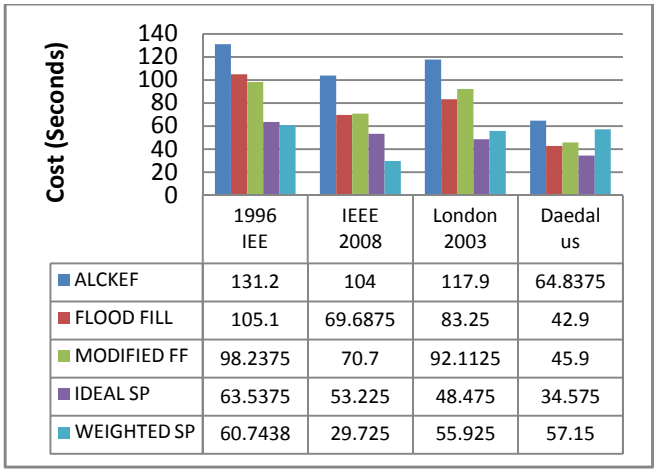


Fig.4. Comparison results of various maze solving algorithms

VI. CONCLUSION

We presented the design and implementation of the weighted shortest path algorithm including cost consideration for turns and moves in the case of cooperative labyrinth discovery robots. Robots are unaware of the maze at initial point and they learn as they start the discovery. Proposed Shortest Path algorithm was compared against various available maze solving algorithms including Flood-Fill, Modified Flood-Fill and $ALCK_{EF}$. Results prove that the new algorithm is quite stronger than other available approaches and it is closer to the ideal case. The presented algorithm can be used also as an additional layer to enhance the available methods at second and subsequent runs.

REFERENCES

[1] Swati Mishra and Pankaj Bande, "Maze Solving Algorithms for Micro Mouse," in *Technology and Internet Based Systems*, Ghaziabad, Dec, 2008, pp. 86-93.

[2] Manoj Sharma and Kaizen Robeonics, "Algorithms for Micro-mouse," in *Future Computer and Communication*, April, 2009, pp. 581-585.

[3] Cankat Özermen, "Design & Implementation of Cooperative Labyrinth Discovery Algorithms in Multi-Agent Environment," Lefke, Master of Science in Computer Engineering 2011.

[4] B.H Kazerouni, M.B. Moradi, and P.H. Kazerouni, "Variable priorities in maze-solving algorithms for robot's movement," , Aug, 2003, pp. 181-186.

[5] Behnam Rahnama, "Extended Open World Assumption as Core for Reasoning in Collaborative Problem Solving on Multiple Autonomous Semantic Robots," Famagusta, North Cyprus, Doctor of Philosophy in Computer Engineering Dec, 2009.

[6] Zheng Taixing and Xiangyang Zhao, "Research on Optimized Multiple Robots Path Planning and Task Allocation Approac," in *Robotics and Biomimetics*, Dec, 2006, pp. 1408-1413.

[7] Ping Xuan, Lesser Victor, and Zilberstein Shlomo, "Communication Decisions in Multi-Agent Cooperation: Model and Experiments," in *nternational Conference on Autonomous Agents and Multiagent Systems*, 2001, pp. 616-623.

[8] Atilla Elçi and Behnam Rahnama, "Human-Robot Interactive Communication Using Semantic Web Tech. in Design and Implementation of Collaboratively Working Robots," in *The 16th IEEE International Symposium of Robot and Human Interactive communication*, Korea, Aug, 2007, pp. 273-278.