IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 6, Issue 12, December 2018

Algorithm Design for Navigation of Smart Floor Cleaner Robot

Soujanya Bhowmick[1], Raya Adhikari[2], Sayani Dutta[3], Shiladitya Pal[4], Snigdha Mondal[5], Trishita Banerjee[6]

Student, Department of ECE, Techno International New Town, Kolkata, India[1-6]

Abstract: The design and implementation of autonomous and semi-autonomous robots for accomplishment of different types of domestic tasks and goals, from comparatively simple work such as cleaning to complicated tasks such as interacting with humans to provide companionship, is one of the major thrust areas for the modern age. In the present paper, an effort has been made by the group members to perform a comparatively simple task in a sophisticated manner, through the design of a navigation algorithm for a smart floor cleaner robot. An algorithm for effective cleaning of the floor has been developed through modification of the well-known algorithm by Dijkstra, to allow for minimal backtracking to complement the hill-climbing approach, enhancing the energy efficiency of the system by optimization of robotic motion.

Keywords: Smart Floor Cleaner, Algorithm Design, Dijkstra's Algorithm, Hill-Climbing, Backtracking, Optimization, Energy Efficiency

I. INTRODUCTION

Floor cleaning robots have been in consumer markets since the 1990s, and are generally well-appreciated by many busy individuals who find their time to clean their homes increasingly limited. Floor cleaning robots generally tend to vacuum or sweep out areas of the floor which they identify as dirty, till they finish the task of cleaning the whole floor of a designated area. The earliest models had obstacle-avoidance problems, which were overcome by most of the later models. The newer models usually surmounted the problem of obstacles encountered by reversing their direction appreciably. This has led to many of the models functioning in a non-optimal manner, wasting energy and increasing the time taken to perform the tasks. Smart robots are an elegant solution to this problem since they generally find better solutions to problems such as obstacles or hazards, leading to more time and energy-optimal accomplishment of the task assigned to the robot. Some of these are also able to operate in a partially human-monitored manner, which allows for quicker real-time cleaning.

The current paper deals with finding optimal solutions to the obstacle avoidance problem. This leads to quicker and more energy-efficient cleaning being accomplished by a smart floor cleaner robot. The paper is organized in the following manner. Section II surveys different graph search algorithms commonly used for path traversal solutions. It then discusses Dijkstra's algorithm in the context of graph search and its applicability in robotic motion. Section III describes the proposed modification as a combination of the hill-climbing and backtracking techniques. Section IV compares the simulation results obtained by using the standard Dijkstra's algorithm without backtracking with the proposed algorithm combining hill climbing and backtracking. Section V concludes the paper with a discussion on the scope for research in this domain.

II. PATH TRAVERSAL ALGORITHM OVERVIEW

Graph Search for Path Traversal

Any path traversal is analogous to the traversal of a graph with specific points on the path where branching occurs designated as nodes of the graph. Hence for efficient path traversal, graph search or graph traversal algorithms may in certain cases help to find the best practical solutions to the problems at hand. Some of the most-used graph-traversal algorithms include Breadth-First Search, Depth-First Search, Best-First Search, the Bellman-Ford algorithm, Dijkstra's Algorithm and Backtracking. Breadth-first search is an exhaustive search where the neighbor vertices of a vertex in a graph are examined before moving to its child vertices. It is an exhaustive method that is complete but extremely time-consuming in the worst-case. Depth-first search functions just opposite to breadth-first search and is therefore an incomplete algorithm in certain cases. Best-first search searches the most promising node among a set of neighbours and then expands it. Thus it is a breadth-first search of the graph in a depth-first manner. The Bellman-Ford algorithm

Copyright to IJIREEICE

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 6, Issue 12, December 2018

computes the shortest path from a single source vertex to all other vertices in a weighed directed graph. It is slower than Dijkstra's algorithm for any given problem. Dijkstra's algorithm helps to determine the shortest paths between nodes in a directed weighed graph and is most often used for path traversal. The basic form of Dijkstra's algorithm does not provide for backtracking, which allows the return to and further search from a previous node when the traversal algorithm reaches a dead end. Backtracking can significantly speed up the performance of Dijkstra's algorithm in worst-case scenarios. Hence, the present work starts from an investigation of Dijkstra's algorithm.

Dijkstra's Algorithm and its Implementation

The flowchart for Dijkstra's Algorithm is given in Figure 1 which follows. The implementation of this algorithm in its general form is discussed next.

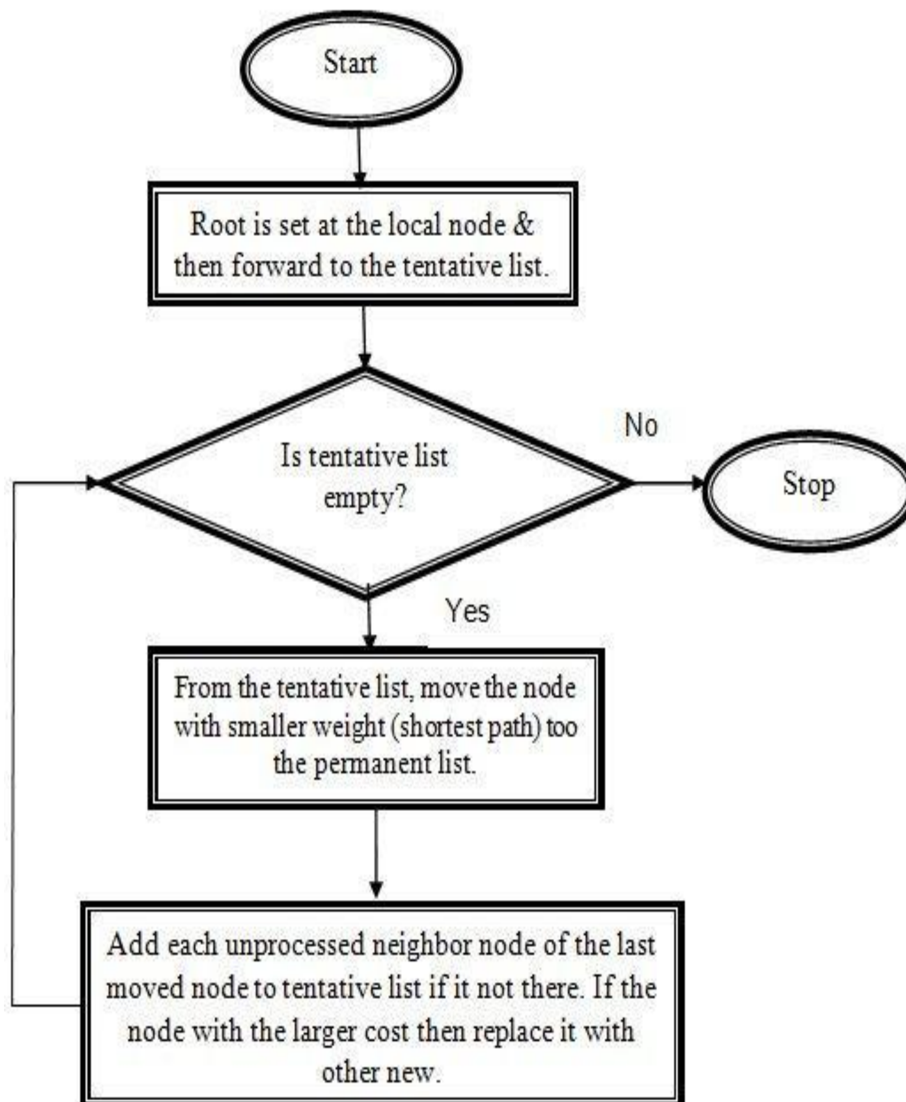

Figure 1: Flowchart of Dijkstra's Algorithm

In the following algorithm, the code u ← vertex in Q with min dist[u], searches for the vertex u in the vertex set Q that has the least dist[u] value. length (u, v) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes u and v. The variable alt on line 18 is the length of the path from the root node to the neighbor node v if it were to go through u. If this path is shorter than the current shortest path recorded for v, that current path is replaced with this alt path. The prev array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 6, Issue 12, December 2018

Pseudocode 1: Forward Traversal

```
1  function Dijkstra(Graph, source):
2
3     create vertex set Q
4
5     for each vertex v in Graph:            // Initialization
6        dist[v] ← INFINITY                  // Unknown distance from source to v
7        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
8        add v to Q// All nodes initially in Q (unvisited nodes)
9
10    dist[source] ← 0                       // Distance from source to source
11
12    while Q is not empty:
13       u ← vertex in Q with min dist[u]   // Node with the least distance
14                                           // will be selected first
15       remove u from Q
16
17       for each neighbour v of u:          // where v is still in Q.
18          alt ← dist[u] + length(u, v)
19          if alt<dist[v]:                  // A shorter path to v has been found
20             dist[v] ← alt
21             prev[v] ← u
22    return dist[], prev[]
```

If we are only interested in a shortest path between vertices source and target, we can terminate the search after line 15 if u = target. Now we can read the shortest path from source to target by reverse iteration.

Pseudocode 2: Reverse Iteration

```
1  S ← empty sequence
2  u ← target
3  if prev[u] is defined or u = source:      // Do something only if the vertex is reachable
4     while u is defined:                    // Construct the shortest path with a stack S
5        insert u at the beginning of S// Push the vertex onto the stack
6        u ← prev[u]   // Traverse from target to source
```

Now sequence S is the list of vertices constituting one of the shortest paths from source to target, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between source and target considering several different paths with same path lengths. On completion of execution of the algorithm, prev[] data structure will actually describe a graph that is a subset of the original graph with some edges removed with every path from any starting node to any other node in the new graph being the shortest path between those nodes in the original graph, and all paths of that length from the original graph being present in the new graph. Then a path finding algorithm can be used to actually find all these shortest paths between two given nodes on the new graph.

Dijkstra's Algorithm has been extensively investigated for the provision of obstacle avoidance based traversal solutions. Data mining based on the algorithm has been used by some researchers to identify efficient path traversal patterns for test environments [1]. Indoor path planning using the standard algorithm, along with dictionaries in one case, for a robotic vehicle has also been investigated by researchers [2] [3]. The algorithm has also been particularly used for the planning of shortest paths for static environments [4]. However, if the environment is dynamic, the standard algorithm suffers greatly in terms of performance. Thus, a modified version of the algorithm is proposed and simulated to compare with the standard algorithm in case of random environments. This is discussed in the next section.

IJIREEICE

ISSN (Online) 2321-2004
ISSN (Print) 2321-5526

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

Vol. 6, Issue 12, December 2018

III. MODIFIED ALGORITHM WITH HILL CLIMBING AND BACKTRACKING

For a dynamic environment, the reduced graph generated analogous to a real-world path by Dijkstra's Algorithm changes continuously, hence it does not provide a good solution on its own. To improve the solution for a dynamic environment, we consider the additional technique of hill-climbing, where the best (or nearest) node among a set of neighbour nodes is chosen for expansion to implement shortest path traversal. Additionally, if the child nodes of the selected node are worse in terms of the distance parameter, backtracking allows a return to the most recent previous node for re-evaluation of path. This circumvents the problem of a robot reversing a long way to avoid an obstacle. The pseudocode for hill-climbing is shown below

Pseudocode 3: Hill Climbing

1 **Set** node u[0] as start node
2 **Set** node t as end node
3 **while** u[0]≠ t
4 **calculate** d(u[0]) as distance between u and t
5 Examine the nodes adjacent to u[0]
6 **if** at a node u[j] adjacent to u[0], d(u[0])-d(u[j]) is maximum
7 **Set** u[0]=u[j]
8 **return** to step 3

In the scenario considered above, it may happen that the shortest path between two nodes may lead to an obstacle in which case it is better to consider the next best node and repeat the process. This is the technique of backtracking. The pseudocode snippet for backtracking is illustrated below.

Pseudocode 4: Backtracking

1 **Set** node u[0] as start node
2 **Set** node t as end node
3 **while** u[0]≠ t
4 **calculate** d(u[0]) as distance between u and t
5 Examine the nodes adjacent to u[0]
6 **if** at a node u[j] adjacent to u[0], d(u[0])-d(u[j]) is maximum
7 **Set** u[1]=u[j]
8 Examine the nodes adjacent to u[1]
9 **calculate** d(u[1]) as distance between u and t
10 **if** at a node u[k] adjacent to u[1], d(u[1])-d(u[k]) is maximum
11 **if** current d(u[1])-d(u[k]> previous d(u[0])-d(u[j])
12 **return** to step 3
13 **else** discard node corresponding to u[1], **set** u[0]as initial node and **return** to step 3

IV. RESULTS

The algorithms were simulated in a dynamic random environment. The results of the simulation are shown in Table 1 below.

Table 1: Comparison of Times for Traversal Dijkstra's Algorithm and Algorithm with Hill-Climbing and Backtracking

Environment	Algorithm	Traversal Time (s)
A	Dijkstra's Algorithm	535
	Hill-Climbing with Backtracking	415
B	Dijkstra's Algorithm	211
	Hill-Climbing with Backtracking	180
C	Dijkstra's Algorithm	255
	Hill-Climbing with Backtracking	191

IJIREEICE
ISSN (Online) 2321-2004
ISSN (Print) 2321-5526
International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering
Vol. 6, Issue 12, December 2018

The comparative results are graphically plotted in Figure 2 which follows.

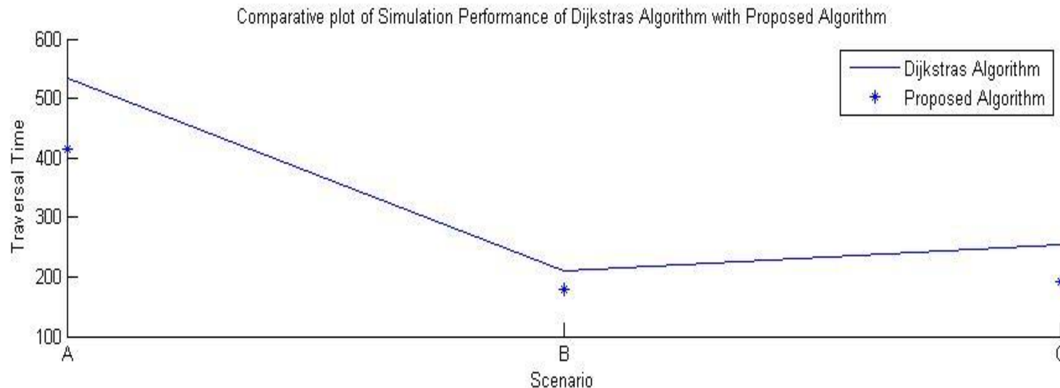

Figure 2: Performance comparison of Dijkstra's Algorithm with Proposed Algorithm

From the results obtained it is clear that in dynamic scenarios, the proposed algorithm gives better results than the traditional Dijkstra's algorithm.

CONCLUSION

The current work accomplished in this paper comprised of a proposed algorithm combining hill-climbing and backtracking which has been seen to outperform Dijkstra's algorithm for randomly changing environments. Future work in this direction shall therefore involve the implementation of better heuristics to shorten best-path search time, involving avenues such as machine learning and pattern recognition for a dynamic heuristic development adapting to the surroundings to give quickest path traversals with best possible search times, leading to minimum energy wastage by the smart robot programmed with an implementation of the algorithm.

REFERENCES

[1]. Ming-Syan Chen, Jong Soo Park and P. S. Yu, "Efficient data mining for path traversal patterns," in IEEE Transactions on Knowledge and Data Engineering, vol. 10, no. 2, pp. 209-221, March-April 1998.
[2]. S. A. Fadzli, S. I. Abdulkadir, M. Makhtar and A. A. Jamal, "Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries," 2015 2nd International Conference on Information Science and Security (ICISS), Seoul, 2015, pp. 1-4.
[3]. Huijuan Wang, Yuan Yu and Quanbo Yuan, "Application of Dijkstra algorithm in robot path-planning," 2011 Second International Conference on Mechanic Automation and Control Engineering, Hohhot, 2011, pp. 1067-1069.
[4]. C. Yin and H. Wang, "Developed Dijkstra shortest path search algorithm and simulation," 2010 International Conference On Computer Design and Applications, Qinhuangdao, 2010, pp. V1-116-V1-119.