An Efficient Algorithm for Grid-Based Robotic Path Planning Based on Priority Sorting of Direction Vectors

Aolei Yang, Qun Niu, Wanqing Zhao, Kang Li, and George W. Irwin

Intelligent Systems and Control Group
School of Electronics, Electrical Engineering and Computer Science
Queen's University of Belfast, Belfast, BT9 5AH, UK

Abstract. This paper presents an efficient grid-based robotic path planning algorithm. This method is motivated by the engineering requirement in practical embedded systems where the hardware resource is always limited. The main target of this algorithm is to reduce the searching time and to achieve the minimum number of movements. In order to assess the performance, the classical A* algorithm is also developed as a reference point to verify the effectiveness and determine the performance of the proposed algorithm. The comparison results confirm that the proposed approach considerably shortens the searching time by nearly half and produces smoother paths with less jagged segments than A* algorithm.

Keywords: path planning, grid-based map, priority sorting, time-efficient algorithm.

1 Introduction

Path planning is considered as one of the most important tasks in building robotic systems and has been researched for several decades. The path planning problem which has been proved to be a PSPACE hard problem [1] is characterized by the ability to search or find a feasible collision-free path from a start location to a goal location. Many applications rely on path planning such as: intelligent wheelchairs, computer games, computer animals, and robot guidance [2, 3].

Many published paper have addressed the path planning problem. Potential-field algorithms are efficient for high-dimensional systems under complex constraints. Its main idea is to construct an attractive potential at the goal location and repulsive potentials on the obstacles. The path is then generated by following the gradient of a weighted sum of potentials [4]. Sampling-based algorithms are currently considered as good choice for motion planning in high-dimensional spaces. These are based on uniform sampling which considers the whole map environment as uniformly complex and thus the overall sampling density will be equivalent to the density needed by the most complex region. The result is that every region in the space has the same computational complexity [5]. Low-dimensional path planning problems can be solved with grid-based algorithms that overlay a grid on top of the map. Several approaches to such grid-based path planning have been proposed. Breadth-first search (BFS), a graph search algorithm, can be used to solve grid-based path planning problems. This method begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, it explores their unexplored neighbour nodes, and so on, until it finds the goal [6, 7]. The A* algorithm is a classical method and along with its variants has been widely applied. It uses a heuristic idea to focus the search towards the goal position. Using an edge cost and a heuristic based on the Euclidean distance, the A* algorithm can search the shortest paths [8].

Although there are many algorithms available to solve the path planning problem, these methods are not always suitable in practical engineering applications because of the limitations on runtime and on the resources available in embedded systems. For the classical A* algorithm, its scalability is limited by its memory requirements. It stores all explored nodes of a search graph in the memory, using an Open List to store nodes on the search frontier and a Closed List to store already-expanded nodes. One of the slowest parts of the A* algorithm is to find the grid square on Open List because it needs to check each item on the list to make sure the lowest F cost item is found. Depending on the size of the grid map, dozens, hundreds or even thousands of nodes may have to be searched at any given time. Needless to say, repeatedly searching through these lists can slow down the application significantly. Therefore, from the perspective of a real-time engineering application, the algorithm designed should be simple, efficient, consume minimal resources and be easy to implement.

This paper mainly addresses the grid-based robotic path planning problem discussed above. A practical and efficient approach is proposed for generating a path with the minimum number of steps. It is termed as the Direction Priority Sequential Selection (DPSS) method as it is based on the Goal Direction Vector (GDV). In order to assess the algorithm performance, the classical A* algorithm is also included to verify the effectiveness of the proposed technique and to determine its performance. Results show that the proposed approach considerably reduces the search time by nearly 50% and produces smoother paths with less jagged segments than the A* alternative.

This paper is organised as follows: Section II presents a detailed description of the direction priority sequential selection algorithm. Section III reports results from comparative simulation studies. Section IV contains the conclusion and future work.

2 Direction Priority Sequential Selection Algorithm

2.1 Preliminaries

Consider the three 7×7 grid-based maps shown in Fig.1, a “0” means free space and a “1” for an obstacle. This paper assumes that nodes are placed in centres of grid squares and each node is an eight-connected mapping. For simplicity, each square of the map is considered as a node and this node can be represented using the coordinates of the cell, for example, Node (1, 1).

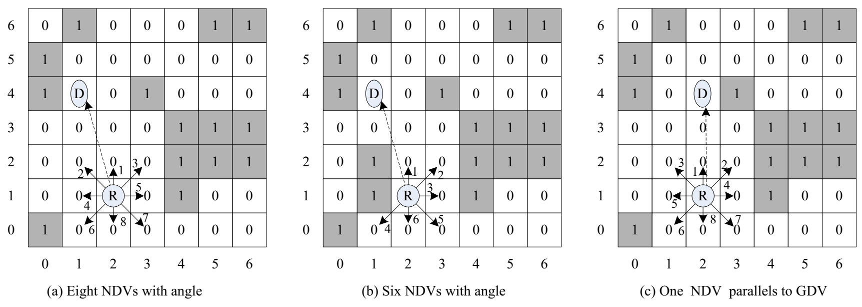


Fig. 1. Three examples for illustrating the process of sorting neighbour node

As for the algorithm, there are a few considerations which should be taken into account. The first is the rule about cutting corners movement. In the implementation of algorithm, a diagonal movement is not allowed if any of two adjacent diagonal nodes are filled with obstacles. The second consideration is about the "Length" between adjacent nodes and it can also be considered as a "Cost" of moving from one node to another. There is an assumption that the length between diagonal adjacent nodes is of $\sqrt{2}$ units and the length of vertical and horizontal adjacent nodes is of 1 unit. The third consideration is to introduce a few definitions: "steps", "current node", "tail node", "Goal Direction Vector (GDV)" and "Neighbour Direction Vector (NDV)". A single movement of the robot between neighbouring nodes is called as one step. In the next of this paper, "Steps" stands for the number of step in a path. A "current node" is one which is currently under search. In the user-defined data structure (Link List) introduced in this paper, a current node is pointed to by a head pointer. As for the "tail node", it is the last node of the link list which is pointed to by a tail pointer. The Goal Direction Vector(GDV) and the Neighbour Direction Vector(NDV) are shown in Fig. 1. The GDV is represented by a dashed arrow which is the direction vector from the current robot location to the destination node. The NDV is shown by a solid arrow line which is the direction vector from the current location to every valid neighbouring node.

Here, a few rules for sorting neighbouring nodes for the new DPSS method are given below:

1. The angles between the GDV and the NDV are used to define the priority of every NDV for the current node. The closer the angle between the GDV and the NDV, the higher the priority of the corresponding neighbouring node.
2. If the GDV of the current node is parallel to the horizontal or vertical directions, the NDV which parallels the GDV holds the highest priority and the reverse direction of the GDV is assigned the lowest priority. The remaining NDVs of the nodes are also allocated according to the angle between the GDV and the NDV. However, there are always three pairs of NDVs having the same angles, the solution is to randomly specify the priority of NDVs.

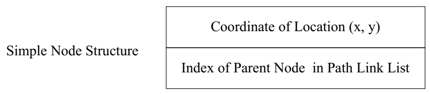

Simple Node Structure
Coordinate of Location (x, y)
Index of Parent Node in Path Link List

Fig. 2. User-defined data structure

To illustrate clearly the DPSS method, Fig. 1 shows the application of the rules introduced above. Robots (Circle R stands for current node) traverse the grid map and reach the goal location (Oval D stands for goal node). Note that the number near NDV represents the sorting priority, the smaller the value, the higher its priority. In Fig. 1(a), eight neighbouring nodes of the current node are labelled “0”, which mean these are valid nodes for the next search process. Fig. 1(b) shows that six neighbouring nodes of the current node are valid. The two cases in Fig. 1(a), (b) can adopt rule 1 to get the priority for all valid NDVs. Actually every NDV corresponds to a neighbouring node, so these nodes can be sorted according to NDV priority and can be stored into a user-defined link list. For the case of Fig. 1(c), rule 2 can be used to achieve the sorting priority.

Considering Fig. 2, a simple node data structure is designed to store the search path information sequentially. The “coordinate of location” field is filled with the coordinate position in the map, and the “index of parent node” field is filled by the index of the parent node of the current one in user-defined link list. Fig. 4 displays the using of this data structure.

2.2 Introduction of DPSS Algorithm

The DPSS algorithm consists primarily of two functions: a Node Sorting Search (NSS) and Path Fin Cutting (PFC). The Node Sorting Search function is based on a direction priority sorting and can be used to achieve a feasible path with minimum steps which is called as raw path. Path Fin Cutting is used to optimize the raw path. Since the focus of the NSS function is on the steps rather than the length/cost, it is necessary to develop some measures to generate smoother paths with less jagged segments and to improves the quality of the raw path.

Consider the problem of traversing the grid map as shown in Fig. 3(a). The NSS function would search and identify all nodes that surround the current one. After a node is searched, it is considered as a visited node. All these visited nodes are recorded in order into a user-defined link list according to the rules of sorting mentioned above. The question arises as to how to know a node is a visited node in the next search operation? A widely used method that is used by the A* algorithm is to search an “Open List” or a “Closed List”. However, repeatedly searching through a list is a very inefficient method if thousands of nodes are stored there. The method used in the implementation of the new DPSS algorithm is to build a mirror image of the grid-based map. If a node in the map has been visited, a corresponding position in the mirror image can be recorded. When judging the state of a node, it is then just necessary to read the value in the corresponding position of the mirror image.

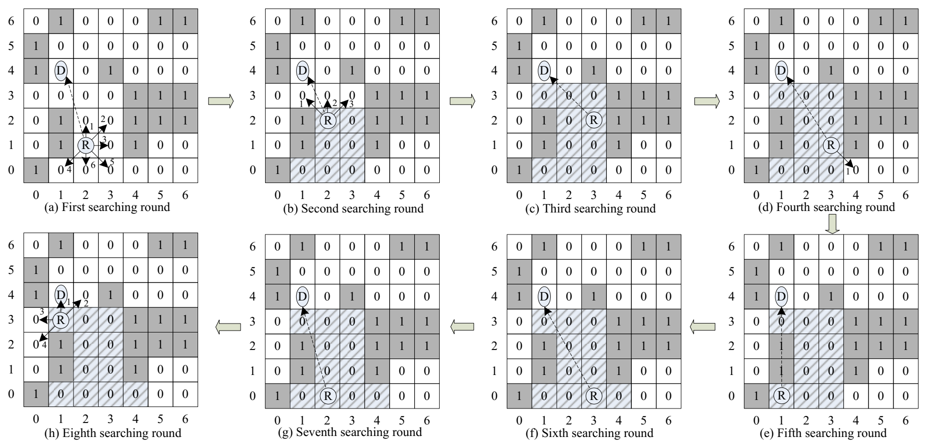

(a) First searching round
(b) Second searching round
(c) Third searching round
(d) Fourth searching round
(e) Fifth searching round
(f) Sixth searching round
(g) Seventh searching round
(h) Eighth searching round

Fig. 3. The process of operating of NSS function

Fig. 3 illustrates the detailed operation of the NSS algorithm for solving a simple path planning problem. Node(2, 1) is the start location and Node(1, 4) is the destination. The grey squares indicate the obstacles on the grid map and the light grey squares represent nodes which have already been visited in a previous searching operation. The mirror image keeps track of previous visited nodes by marking them temporally in the workspace, as if they were obstacles.

Consider the search process shown in Fig. 3, where the NSS function has performed eight searches. Fig. 4 illustrates the process of evolution about the link list structure corresponding to Fig. 3. The head and tail arrows in Fig. 4 indicate the user-defined pointers which are used to indicate the head and tail of the link list. The head pointer is always considered as the current robot location which can also be called the focus node. The tail pointer points to the last node of the link list as the NSS function runs continuously. The method always searches the node pointed by head pointer and stores the valid traversal node in the position pointed by the tail pointer.

To simplify matters, the search operation is stopped when the goal node is found for the first time or when finding the target node fails. If the goal node is found, the method just starts here, and works backwards moving from one node to its parent, following the parent pointers. This will eventually track back to the start node, and produce the resulting path. As depicted in Fig. 4(h), a feasible path can then be easily achieved as follows: $Node(2,1) \rightarrow Node(2,2) \rightarrow Node(1,3) \rightarrow Node(1,4)$.

Since the main target of the DPSS algorithm is to decrease the search time and to achieve the minimal step number of a path, it is possible that there are different path lengths with the same step number. So it is then necessary to design the Path Fin Cutting function to optimize the raw path in order to produce smoother paths with less jagged path segments.

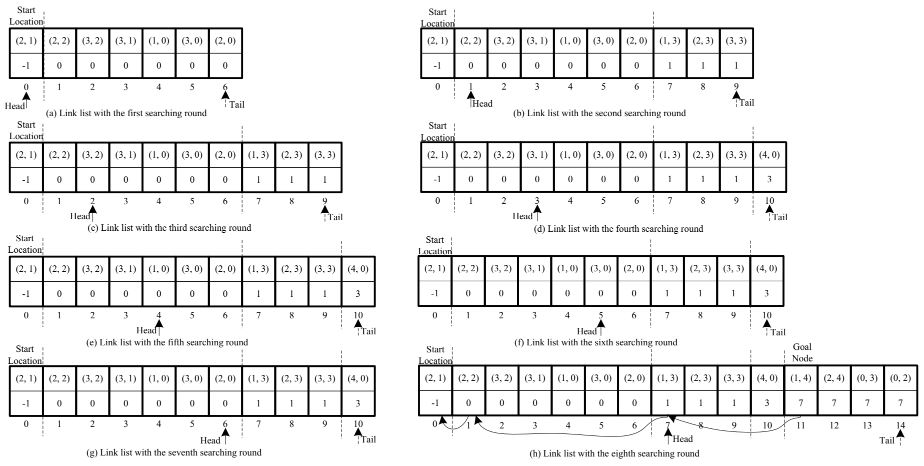


Fig. 4. Running process of link list

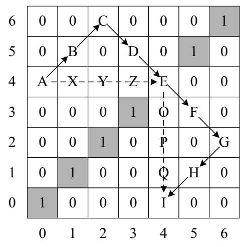


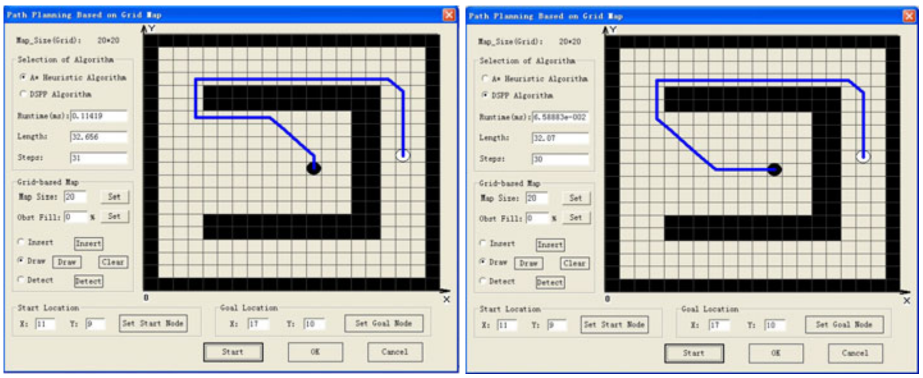
Fig. 5. Optimize the raw path

Fig. 5 show a segment of a feasible path. Consider the path from Node A to Node E. The step number is equal to 4, the length is $4 \times \sqrt{2}$ units, and further there are no obstacles on the straight line $Node\ A \rightarrow Node\ X \rightarrow Node\ Y \rightarrow Node\ Z \rightarrow Node\ E$. It is obvious that the path $Node\ A \rightarrow Node\ B \rightarrow Node\ C \rightarrow Node\ D \rightarrow Node\ E$ can be replaced by the path $Node\ A \rightarrow Node\ X \rightarrow Node\ Y \rightarrow Node\ Z \rightarrow Node\ E$ which has smaller length (4×1 units) but the same number steps. Similarly, the path $Node\ E \rightarrow Node\ F \rightarrow Node\ G \rightarrow Node\ H \rightarrow Node\ I$ can also be replaced by a path $Node\ E \rightarrow Node\ O \rightarrow Node\ P \rightarrow Node\ Q \rightarrow Node\ I$.

3 Software Implementation and Comparison

To implement the algorithm, C++ is used to develop the Graphical User Interface (GUI) to display the trajectory. All simulation tests were performed on Intel Centrino Due2300 processor with 1G RAM and Windows XP.

In order to assess the performance, the A* algorithm was used for comparison purposes. The performances of the two algorithms were compared with respect

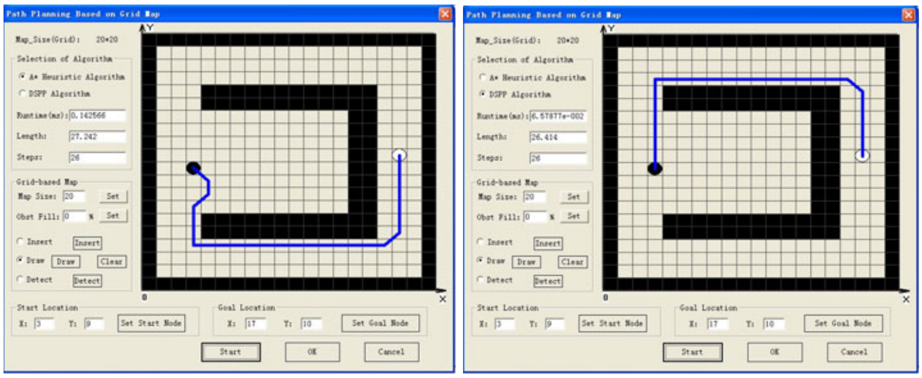


Path Planning Based on Grid Map
Map_Size(Grid): 20*20
Selection of Algorithm: [✓ A* Heuristic Algorithm] DSPP Algorithm
Runtime(ms): 0.11419
Length: 32.656
Steps: 31
Grid-based Map — Map Size: 20; Obst Fill: 0 %
Insert [✓ Draw] Detect
Start Location — X: 11 Y: 9
Goal Location — X: 17 Y: 10

Path Planning Based on Grid Map
Map_Size(Grid): 20*20
Selection of Algorithm: A* Heuristic Algorithm [✓ DSPP Algorithm]
Runtime(ms): 6.58883e-002
Length: 32.07
Steps: 30
Grid-based Map — Map Size: 20; Obst Fill: 0 %
Insert [✓ Draw] Detect
Start Location — X: 11 Y: 9
Goal Location — X: 17 Y: 10

(a) A* algorithm (b) DPSS Algorithm

Fig. 6. Comparison on manually generated maps for Test 1



Path Planning Based on Grid Map
Map_Size(Grid): 20*20
Selection of Algorithm: [✓ A* Heuristic Algorithm] DSPP Algorithm
Runtime(ms): 0.142566
Length: 27.242
Steps: 26
Grid-based Map — Map Size: 20; Obst Fill: 0 %
Insert [✓ Draw] Detect
Start Location — X: 3 Y: 9
Goal Location — X: 17 Y: 10

Path Planning Based on Grid Map
Map_Size(Grid): 20*20
Selection of Algorithm: A* Heuristic Algorithm [✓ DSPP Algorithm]
Runtime(ms): 6.57877e-002
Length: 26.414
Steps: 26
Grid-based Map — Map Size: 20; Obst Fill: 0 %
Insert [✓ Draw] Detect
Start Location — X: 3 Y: 9
Goal Location — X: 17 Y: 10

(a) A* algorithm (b) DPSS Algorithm

Fig. 7. Comparison on manually generated maps for Test 2

to run times, length and steps of the generated path. Note that the "runtime" displayed on the GUI is the average time spent for 100 tests running in the same environment.

In the simulation, various types of tests were carried out on manually generated maps, random maps with a uniform distribution, and small to large maps. In all cases, a black circle stands for the start node, a white circle stands for the goal node, and white squares were using for free spaces and black squares for obstacles.

3.1 Comparison of Solutions

In the manually generated maps which consisted of a 20×20 grid of cells, Fig. 6 (Test 1) and Fig. 7 (Test 2) clearly compare the DPSS and A* algorithms under the following conditions: the same map, the same goal node, but with different start nodes. As depicted in Fig. 6 (a) and the Fig. 7 (a), the A* is not optimal and the DPSS gave better results.

Table 1 compares the time consumption, and the path lengths and steps. The Fig. 6, Fig. 7 and Table 1 together confirm that the performance of the DPSS technique is better than the A* algorithm. The time spent by the DPSS algorithm was nearly 50% less that from the A* method in all tests. Considering Test 1, the DPSS method achieved smaller steps and shorter lengths than A*, while in Test 2 the DPSS length was shorter than that of the A* for the same number of steps.

Table 1. Comparison of two algorithms

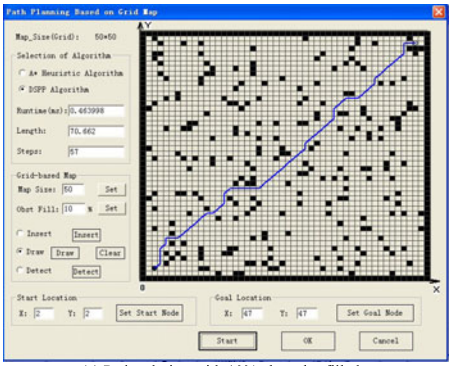
Map Type	Case	Start Node	Goal Node	Parameter	DPSS Algorithm	A*Algorithm
The same map generated manually	Test 1	(11, 9)	(17, 10)	Time (ms)	0.066	0.114
				Length	32.07	32.656
				Steps	30	31
	Test 2	(3, 9)	(17, 10)	Time (ms)	0.066	0.143
				Length	26.414	27.242
				Steps	26	26

3.2 Comparison of Search Times

Similarly to [9, 10], several different densities of obstacles can be chosen to assess the performance of the new algorithm in different environments. In this paper, nine different densities of obstacles were selected, viz. 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40% of filled squares in the whole grid. Through many tests, it was found that if the density of obstacles was greater than 40%, there was almost no chance of a feasible path being found. Fig. 8(a), (b) present the path solutions from the DPSS algorithm under the following conditions: 10% and 25% density of obstacles, the same map of size (50×50), and the same start and goal location. The obstacles were filled randomly in the map space. A detailed comparison is provided in Table 2.

As shown in Table 2, the two algorithms produced exactly the same paths under the same density of filled obstacles. This is a precondition for comparing the times taken. Some valuable conclusions can be reached:

1. The average time spent by the new DPSS is almost 50.4% shorter than that of the A* algorithm in the tests performed.
2. The time spent by the DPSS first increases, and then decreases as the density of obstacles increases. By contrast, the time spent by the A* algorithm decreases monotonously with increasing density of obstacles.
3. The density of obstacles has less impact on the performance of the new DPSS technique than that of A* method.


Path Planning Based on Grid Map

Map_Size(Grid): 50*50

Selection of Algorithm: A* Heuristic Algorithm [✓ DSPP Algorithm]

Runtime(ms): 0.463998

Length: 70.662

Steps: 57

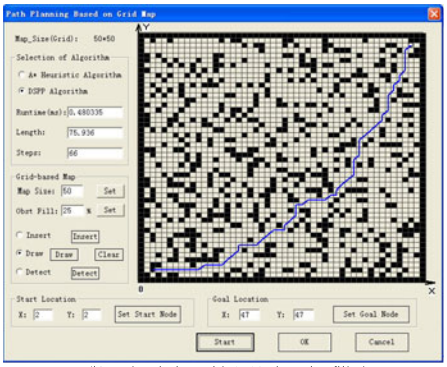
Grid-based Map — Map Size: 50 — Obst Fill: 10 %

Insert [✓ Draw] Detect

Start Location — X: 2 Y: 2

Goal Location — X: 47 Y: 47

(a) Path solution with 10% obstacles filled


Path Planning Based on Grid Map

Map_Size(Grid): 50*50

Selection of Algorithm: A* Heuristic Algorithm [✓ DSPP Algorithm]

Runtime(ms): 0.480335

Length: 75.936

Steps: 66

Grid-based Map — Map Size: 50 — Obst Fill: 25 %

Insert [✓ Draw] Detect

Start Location — X: 2 Y: 2

Goal Location — X: 47 Y: 47

(b) Path solution with 25% obstacles filled

Fig. 8. Path solution of DPSS algorithm with 10% and 25% density of obstacles filled randomly

Table 2. Algorithm comparison with the gradual change of density of obstacle filled

Test environment	Density	DPSS Algorithm			A* Algorithm			Same path	Shorten time	Average shorten
		Time(ms)	Length	Steps	Time(ms)	Length	Steps			
Map size: 50*50 Start node: (2,2) Goal node: (47,47)	0%	0.38	63.63	45	1.01	63.63	45	Yes	62.4%	50.4%
	5%	0.42	67.146	51	0.98	67.146	51	Yes	57.1%	
	10%	0.46	70.662	57	0.96	70.662	57	Yes	52.1%	
	15%	0.46	73.248	60	0.94	73.248	60	Yes	51.1%	
	20%	0.47	75.592	64	0.92	75.592	64	Yes	48.9%	
	25%	0.48	75.936	66	0.88	75.936	66	Yes	45.5%	
	30%	0.44	79.694	71	0.83	79.694	71	Yes	47.0%	
	35%	0.39	94.866	87	0.72	94.866	87	Yes	45.8%	
	40%	0.26	106.038	99	0.46	106.038	99	Yes	43.5%	

Table 3. Algorithm comparison with the gradual change of map size

Map Dimension	Start node	Goal node	Density	DPSS Algorithm			A* Algorithm			Same path
				Time(ms)	Length	Steps	Time(ms)	Length	Steps	
20*20	(2,2)	(17,17)	10%	0.065	22.382	17	0.15	22.382	17	Yes
			25%	0.075	27.898	25	0.13	27.898	25	Yes
40*40	(2,2)	(37,37)	10%	0.29	54.178	43	0.61	54.178	43	No
			25%	**0.3**	**62.28**	**54**	**0.57**	**62.038**	**55**	**No**
60*60	(2,2)	(57,57)	10%	0.66	83.63	65	1.43	83.63	65	Yes
			25%	0.67	88.904	74	1.28	88.904	74	No
80*80	(2,2)	(77,77)	10%	1.17	114.49	88	2.63	113.66	88	No
			25%	1.22	124.216	106	2.56	124.216	106	No
100*100	(2,2)	(97,97)	10%	1.84	142.534	109	4.56	142.534	109	No
			25%	1.91	157.184	134	3.77	157.184	134	Yes

3.3 Comparison of Performances on Large Maps

In addition to assessing the performance with small and medium sized maps, the large maps were also examined. Table 3 illustrates the change in performance with increasing map size from 20×20 to 100×100. Several conclusions can be drawn:

1. The DPSS algorithm shortens the running time by nearly half in the same environment (the same map, same start and goal locations).
2. On large maps (100×100), the time consumption is shorteded by 59.6% and 49.3% corresponding to 10% and 25% density of obstacles respectively.
3. The DPSS algorithm can always achieve the smallest number of steps in a path for all the tests, although the path length for the DPSS is larger than that of the A* algorithm under certain conditions, as shown in bold in Table 3 (map of size 40×40 and 25% density).

4 Conclusions and Future Work

A novel DPSS algorithm has been proposed and applied to grid-based path planning problems in this paper. It was compared with the well known A* algorithm which is extensively used in practical systems to verify its effectiveness and to determine its performance. A number of simulation tests showed that the proposed approach considerably reduces the search time and generates smoother paths with less jagged segments. The execution time of the DPSS was shorter by nearly 50% in all tests. The path step number was always smaller than that of A* alternative, although sometimes the smallest path length was not also achieved. Compared with the A* algorithm, the DPSS one only needs to design one Link List, whose size is proportional to the number of grid squares. Moreover, there is no necessity to search "Open List" and "Close List" frequently. These are the key factors in ensuring that the DPSS algorithm is efficient and can also reduce the memory requirements during search.

The proposed algorithm can be further improved in the future. Here, to improve the search speed, the search was stopped when the goal node was found for the first time. If achieving the minimum path length is the main target, it is essential to continue the search operation until a path with minimum length is found. Further, this algorithm can be further extended to dynamically adapt to the environment by the changing maps and changing the goal location.

Acknowledgments. This work was supported by the China Scholarship Council and EPSRC under UK-China Science Bridge grant EP/G042594/1.

References

[1] Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proceeding of the 20th IEEE Symposium on Foundations of Computer Science, pp. 421–427. IEEE, New York (1979)

[2] Burgard, W., Cremers, A.B., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S.: The interactive museum tourguide robot. In: Proceeding of the Fifteenth National Conference on Artificial Intelligence, Madison, Wisconsin, United States, pp. 11–18 (1998)

[3] Arai, T., Pagello, E., Parker, L.E.: Advanced in multi-robot systems. IEEE Transactions on Robotics and Automation 18(5), 655–661 (2002)

[4] Al-Sultanl, K.S., Aliyul, M.D.S.: A new potential field-based algorithm for path planning. Journal of Intelligent and Robotic Systems (17), 265–282 (1996)

[5] Taha, T., Valls Miro, J., Dissanayake, G.: Sampling based time efficient path planning algorithm for mobile platforms. In: Proceeding of the 2006 International Conference on Man-Machine Systems (ICoMMS 2006), Langkawi, Malaysia (2006)

[6] Zhou, R., Hansen, E.A.: Breadth-first heuristic search. Artificial Intelligence 170(4), 385–408 (2006)

[7] Vacariu, L., Roman, F., Timar, M., Stanciu, T., Banabic, R., Cret, O.: Mobile robot path planning implementation in software and hardware. In: Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation, pp. 140–145 (2007)

[8] Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics (2), 100–107 (1968)

[9] Nash, A., Daniel, K., Koenig, S., Felner, A.: Any-angle path planning on grids. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 1177–1183 (2007)

[10] Šišlák, D., Volf, P., Pěchouček, M.: Accelerated a* trajectory planning: Grid-based path planning comparison. In: Proceedings of ICAPS 2009 Workshop on Planning and Plan Execution for Real-World Systems, Greece, pp. 74–81 (2009)