ISSN 1691-5402
Environment. Technology. Resources
Proceedings of the 9th International Scientific and Practical Conference. Volume II

Pathfinding Algorithm Efficiency Analysis in 2D Grid

Imants Zarembo, Sergejs Kodors
Rezekne Higher Education Institution

Abstract. **The main goal of this paper is to collect information about pathfinding algorithms A*, BFS, Dijkstra's algorithm, HPA* and LPA*, and compare them on different criteria, including execution time and memory requirements. Work has two parts, the first being theoretical and the second practical. The theoretical part details the comparison of pathfinding algorithms. The practical part includes implementation of specific algorithms and series of experiments using algorithms implemented.**

Such factors as various size two dimensional grids and choice of heuristics were taken into account while conducting experiments.

Keywords **– A*, BFS, Dijkstra's algorithm, HPA*, LPA*, shortest path problem.**

I Introduction

Pathfinding theory describes a process of finding a path between two points in a certain environment. In the most cases the goal is to find the specific shortest path, which would be optimal, i.e., the shortest, the cheapest or the simplest. Criteria such as, a path, which imitates path chosen by a person, a path, that requires the lowest amount of fuel, or a path from point A to point B through point C is often found relevant in many pathfinding tasks.

The shortest path problem is a pressing issue in many fields, starting with navigation systems, artificial intelligence and ending with computer simulations and games. Although all of these fields have their own specific algorithms, there are many general purpose pathfinding algorithms which can be successfully applied. But it is not always clear what advantages certain algorithm has in comparison to its alternatives.

As a part of this paper pathfinding algorithms A*, BFS, Dijkstra's algorithm, HPA* and LPA* were implemented to analyze their efficiency in an environment based on two dimensional grid. Such factors as grid size, traversed node count and execution time were taken into consideration conducting series of experiments.

II Materials and methods

To assess algorithm efficiency in two dimensional grids experiments were conducted using A*, BFS, Dijkstra's algorithm, HPA* and LPA* to find the shortest path between two randomly placed nodes. Algorithm execution times and traversed node count were measured.

Two dimensional grids used in experiments contained two types of nodes: passable and blocked. 20% of grid was randomly filled with blocked nodes. To assess algorithm efficiency experiments were conducted on various size grids: 64x64, 128x128, 256x256, 512x512 and 1024x1024 nodes.

In case of HPA* three level hierarchy was chosen and initial grid was divided into 4x4 clusters. These parameters were chosen because any smaller division of base grid (64x64 in this case) would lead to incorrect search results while executing preprocessing phase.

Manhattan distance was chosen as heuristic function, because it is strictly grid based distance:

$$H = |x_1 - x_2| + |y_1 - y_2|. \quad (1)$$

Every experiment was repeated 100 times to reduce the amount of random errors. Algorithms were implemented assuming that pathfinding may only occur horizontally or vertically, with no diagonal movement. Every transition between two neighboring nodes costs 1.

All experiments were conducted on the computer with CPU running at a frequency of 2.8 GHz.

III Algorithm A*

A* is a pathfinding algorithm used for finding optimal path between two points called nodes. Algorithm A* uses best-first search to find the lowest cost path between start and goal nodes. Algorithm uses heuristic function, to determine the order in which to traverse nodes. This heuristic is sum of two functions:

G — exact cost of the path from initial node to the current node;

H — admissible (not overestimated) cost of reaching the goal from current node;

$F = G + H$ — cost to reach goal, if the current node is chosen as next node in the path.

Estimated heuristic cost is considered admissible, if it does not overestimate the cost to reach the goal [3].

Selection of heuristic function is an important part of ensuring the best A* performance. Ideally H is equal to the cost necessary to reach the goal node. In this case A* would always follow perfect path, and would not waste time traversing unnecessary nodes. If

VII ALGORITHM LIFELONG PLANNING A*

Lifelong Planning A* (LPA*) is an algorithm intended for solving the shortest-path problems on known finite graphs whose edge cost increase or decrease over time [5]. S denotes the finite set of nodes of the graph. $succ(s) \subseteq S$ denotes the set of successors of node $s \in S$. Similarly, $pred(s) \subseteq S$ denotes the set of predecessors of node $s \in S$. $0 < c(s, s') \leq \infty$ denotes the cost of moving from node s to node $s' \in succ(s)$. LPA* always determines the shortest path from a given start node s_{start} to a given goal node $s_{goal} \in S$, knowing both the topology of the graph and the current edge costs. The start distances satisfy the following relationship:

$$g^*(s) = \begin{cases} 0 & if\ s = s_{start} \\ min_{s' \in pred(s)}\left(g^*(s') + c(s', s)\right) & otherwise. \end{cases} \quad (2)$$

$g^*(s)$ denotes the start distance to node $s \in S$, i.e., the cost of the shortest path from s_{start} to s.

LPA* is an incremental version of A* that applies to the same finite path-planning problems as A*. It shares with A* the fact that it uses nonnegative and consistent heuristics $h(s)$ that approximate the goal distance of the node s to focus its search. Consistent heuristics obey the triangle inequality $h(s_{goal}) = 0$ and $h(s) \leq c(s, s') + h(s')$ for all nodes $s \in S$ and $s' \in succ(s)$ with $s \neq s_{goal}$. LPA* reduces to a version of A* that breaks ties among vertices with the same F value in favor of smaller G values when LPA* is used to search from scratch and to a version of DynamicsSWSF-FP that applies to path-planning problems and terminates earlier than the original version of DynamicsSWSF-FP when LPA* is used with uninformed heuristics [6].

VIII RESULTS AND DISCUSSION

Algorithm execution time

Breadth-first search is brute-force search algorithm; its results differ noticeably in comparison with informed search algorithms. Table I shows, that the algorithm execution time increases exponentially with search area size increase.

TABLE I
ALGORITHM BFS EXECUTION TIME

Grid size (nodes)	Execution time (ms)
64x64	150
128x128	2803
256x256	48313
512x512	821598
1024x1024	13962457

To find the shortest path in 512x512 node grid, the algorithm took 821 seconds and in 1024x1024 node grid — 13962 seconds. This considerable execution time shows that the algorithm is the most likely not applicable to real-time search problems in large grids.

Increasing the search problem size Dijkstra's algorithm execution time increases linearly. On average in 1024x1024 grid the algorithm finds the shortest path in 2,3 seconds. Table II shows the algorithm execution times for different grid sizes.

TABLE II
DIJKSTRA'S ALGORITHM EXECUTION TIME

Grid size (nodes)	Execution time (ms)
64x64	6
128x128	25
256x256	120
512x512	515
1024x1024	2362

Algorithm A* performance was greater in comparison with Dijkstra's algorithm in every grid size selected for experiments. The algorithms execution time increases linearly with grid size. Table III shows the algorithm execution times for different grid sizes.

TABLE III
ALGORITHM A* EXECUTION TIME

Grid size (nodes)	Execution time (ms)
64x64	4
128x128	16
256x256	77
512x512	265
1024x1024	1148

Lifelong Planning A* performance is higher than Dijkstra's algorithms in all grid sizes, but it is lower than A* performance in 512x512 and 1024x1024 node grids. The algorithms execution time increases linearly with grid size. Table IV shows the algorithm execution times for different grid sizes.

TABLE IV
ALGORITHM LPA* EXECUTION TIME

Grid size (nodes)	Execution time (ms)
64x64	4
128x128	11
256x256	57
512x512	319
1024x1024	1490

Algorithm HPA* execution time, using 4x4 clusters and 3 level hierarchy, is lower than any other algorithm in this experiment. The algorithms execution time increases linearly with grid size. Table V shows the algorithm execution times for different grid sizes.