C5_02_ICITEE 2016

by Pranowo Pranowo

Optimized A-Star Algorithm in Hexagon-Based Environment using Parallel Bidirectional Search

Pratyaksa Ocsa Nugraha Saian
Masters Student
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
pratyaksa.ocsa@gmail.com

Suyoto
Lecturer
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
suyoto@yahoo.com

Pranowo
Lecturer
Universitas Atma Jaya Yogyakarta
Yogyakarta, Indonesia
pran@mail.uajy.ac.id

***Abstract*—Shortest path algorithm is one of classic IT problems and already used in many aspects. One of well-known shortest path algorithm is A-Star algorithm. Usually A-Star will be implemented to a Non-Playable Character (NPC) in some games. This paper wants to tell how to optimize A-Star algorithm in a hexagon-based environment using Parallel Bidirectional Search (PBS). The result of this paper is PBS A-Star can be accelerating classics A-Star algorithm by 68.8% faster in hexagon-based environment.**

***Keywords*—shortest path; A-Star; Parallel Bidirectional Search;**

I. INTRODUCTION

Shortest path is one of common problems in IT industry [1] [2], especially in game industry [3]. One of them is creating a non-playable character (NPC) who always move in an available path and sometimes can find the player's character position too [4]. The NPC able to find a path although there are many obstacles around. This NPC looks like have brain and can decide where to move, but actually that just an implementation of a shortest path algorithm. There are many kind of well-known shortest path algorithm [5] [6], but in a game industry, A* is a famous algorithm and being used in many kind of games.

A* algorithm is one of heuristic search which have a good reputation in gaming industry. Many games developer using this algorithm to solve a shortest path problem in their games [7]. Although this algorithm had an accurate and fast to find a solution, there is one weakness in A*. A* need to find the path first before the NPC can moved based on this calculation. It will take some time until the NPC actually moving. The time to calculate the path might be different in every computer because it very dependent on how fast the computer can process it.

Central Processing Unit (CPU) this day have a multi-core in one single chip [8]. Intel and AMD is a processors company who compete -in a good way- to create as many cores as possible in one single chip. Many cores mean many instructions can be executed at the same time (parallel processing). This trends makes any researcher want to use this parallel processing in order to calculate shortest path problems faster than to calculate in serial process.

Parallel Bidirectional Search (PBS) is created based on a Bidirectional Search. Bidirectional Search is one example of how to optimize a classic shortest path algorithm. Searching path process will be divided into two part, (1) from start node to goal node, and (2) from goal node to start node. They will stopped if they meet in the halfway [9] [10].

The rest of this paper will organize as follows. Section 2 will discuss about some theory and previous works which related with this paper. Section 3 is a research methodology that being used in this paper. Section 4 is the discussion about how to implementing PBS A* on hexagon-based environment and the result of the tests. Section 5 will tell about conclusion and about future works.

II. THEORY AND PREVIOUS WORKS

A. Hexagon-Based Environment

In a game or robotic industry, a real life environment can be modeled as a square-grid environment. There are two type of a square-grid environment, which is center-based grid environment and corner-based grid environment [11].

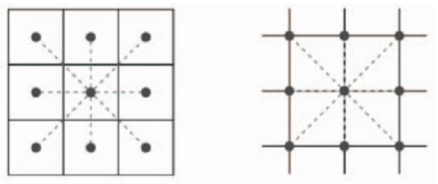
Fig. 1 Center-based (left) and corner-based (right) grid environment

Fig. *1* is showing the difference of center-based and corner-based grid environment. Every node in center-based grid environment will be placed in the middle of the square. It makes the NPC who moved in this kind environment will always stopped in the middle of the square. Corner-based grid environment put every node in the corner of the square. This kind of environment will make NPC always walks in the edge of the square.

NPC in grid-based environment had eight possible movements from where he stands. There are, north, north west, west, south west, south, south east, east, and north east. Game developer usually think that grid-based environment is a perfect way to modeling an environment because it really simple and very intuitive for the players. Although it easier to implement, there is one shortage in a grid-based environment. The NPC will have a strange movements and will not look naturally. When NPC want to move vertical or horizontal it will take p long, but if the NPC move diagonally it will take $p\sqrt{2}$ long. This problem can be solved by changing the environment into a hexagon.

Hexagon-based environment is a popular environment in game who had turn-based gameplay. With a hexagon as an environment, the distance of every node will be constant. Instead of have eight possibility of movement, hexagon-based environment only has six possibilities.

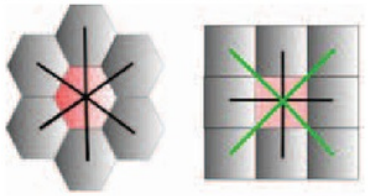
Fig. 2 Hexagon-based (left) and Grid-based (right) environment

Fig. 2 shows the difference of possible movement in a hexagon-based environment and grid-based environment. NPC movement in hexagon-based environment depends in the hexagon itself. If the creator of hexagon-based environment chooses to use "pointy at top" (see Fig. 3), then the possible movement of the NPC is north, north east, south east, south, south west, and north west. If the hexagon-based environment use "pointy at side", then the possible movement will be north west, north east, east, south east, south west, and west.

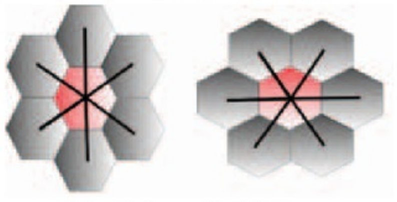
Fig. 3 Pointy at top (left) and pointy at side (right) hexagon

B. *A* Algorithm*

A* algorithm first introduced by Peter Hart, Nils Nilsson, and Bertram Raphael [12]. A* algorithm is an implementation of heuristic search. Heuristic search is a searching process that will give an estimation value from current node to goal node. A* will chooses the smallest value from $f(n) = g(n) + h(n)$ where n is a node, $g(n)$ is a distance from starting node to n node, and $h(n)$ is a heuristic value of estimation distance from node n to finish node.

Fig. 4 shows a general description of how A* looking for a path. A* mainly used an open list and closed list to separate which node had been visited and node that never visited. A* will check every neighbor node and one by one inputted in to open or closed list. Started with find the lowest value of $f(n)$, A* will looking for possibility movement from every neighbor of current node. It will have repeated until the current node to be checked is a goal node. The path will generate from every node which already selected from searching process before.

Bidirectional A* is a modification of A* algorithm. It is a Bidirectional Search used in A* algorithm [13]. The main component in A* is the open list and closed list, so both of them will be stored in a share memory. They can be accessed together by searching process in both directions.

```
function AStar(StartNode, GoalNode)
{
 closed_list = {};
 open_list = {};
 open_list.add(StartNode);
 path = {};
 while open_list.isNotEmpty()
 {
  //Find the node with lowest value of
  //f(n)=g(n)+h(n) from neighbor.
  CurrentNode = FindLowestValue();
  open_list.remove(CurrentNode);
  if(CurrentNode == FinishNode)
  {
   //Path found, stop iteration
   break;
  }
  for(each NeighborNode of CurrentNode)
  {
   current_cost = NeighborNode.getGValue()+
   findDistanceCost(CurrentNode,NeighborNode);
   if(NeighborNode is in open_list)
   {
    if(NeighborNode.getGCost()<=current_cost)
     continue;
   }
   else if (NeighborNode is in closed_list)
   {
    if(NeighborNode.getGValue()<=current_cost)
     continue;
    closed_list.remove(NeighborNode);
    open_list.add(NeighborNode);
   }
   else
   {
    open_list.add(NeighborNode);
    NeighborNode.setHvalue();
   }
   NeighborNode.setGValue(current_cost);
   path.add(CurrentNode);
  }
  closed_list.add(CurrentNode);
 }
}
```
Fig. 4 Pseudocode of A* Algorithm

A* will always find a solution path if at least there is one available. This capability makes another researcher always want to optimize this algorithm more and more every day. Like Lawrence & Bulitko [14] who do some research about A* in *Dragon Games™* and *Counter-Strike: Source* games. Moreover, Lee & Lawrence optimized A* algorithm by reducing its iteration using a database generation [15]. So, when A* start to do its calculation, instead search all possible way, A* will look in database first.

C. *Parallel Bidirectional Search*

Bidirectional Search still use one core to do the job, but it will use a shared memory to store their calculation. It appears that this process can be optimized by use more than one cores to do the calculation. This is the main idea of how Parallel Bidirectional Search implemented. Pathfinding process will handle by two different cores in CPU which make both of them can run at the same time. PBS will be stop when each of searching process meet in the halfway and usually both of them will be meet at the same node [16].

Implementing PBS in a shortest path problem is commonly do by some researchers. Like Rios & Chaimowicz who do some research about PBS. They slightly modify PBS A* who runs in a grid-based environment. They called their algorithm a Parallel New Bidirectional A* (PNBA*). PNBA* will tweak how to use cores in CPU and will be used a share memory allocation [17].

III. Research Methodology

The purpose of this research is to prove that PBS can be a solution to optimize A* to find a shortest path in hexagon-based environment. Not only that, this research will tell how optimized the A* algorithm with PBS compared with A* without PBS is.

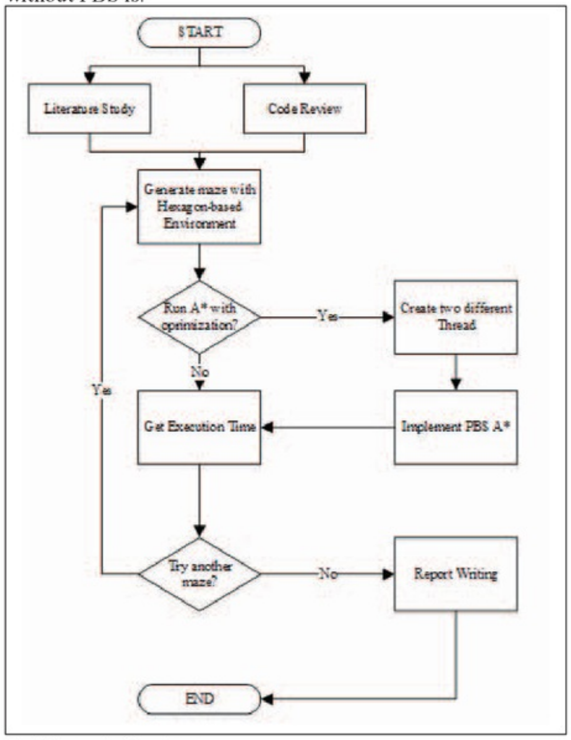

Fig. 5 Research Flowchart

Fig. 5 shows step-by-step of how this research done. It divided in three main step, first is a literature study and some code review to understand more about both A* or parallel processing. Literature study conducted by finding a previous works which related to this paper such as hexagon-based environment, A* algorithm, and PBS. The second step is technical way about how to prove that PBS can be a solution to optimize A* in hexagon-based environment. For second step will be explained more details in the next section. The last step is working in report writing.

IV. Discussion

This sections will tell how to compare an A* algorithm with PBS and without PBS. Based on Fig. *5* to do this testing process, it need four main step which is, (1) generating a maze with a hexagon-based environment, (2) Defining a start node and goal node in the maze, (3) implement A* algorithm with PBS or without PBS, (4) get the execution time.

This PBS A* implemented using C# language and being tested in a PC which used Intel Core i7-6700 3.40 GHz and 16 GB RAM.

A. Generating a maze with a hexagon-based environment

Generating a maze is a common way to test shortest path problems [18]. For testing purpose, the maze that being used is not a random generated maze by computer, but it will be defined piece by piece manually. The size of the maze is 55 hexagon width and 30 hexagon height. The maze will have two different color which symbolized an available path to get through and a path that can't be used. Available path will have white color and unavailable path will have black color. The generated maze can be seen in *Fig. 6* .

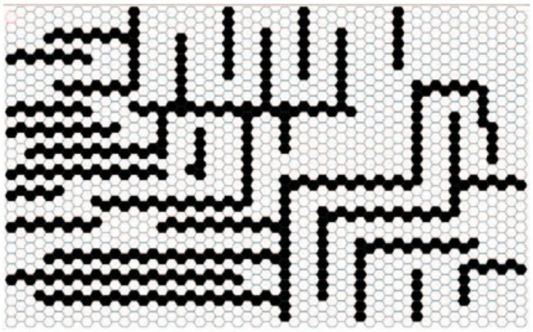

Fig. 6 Generated Maze

B. Defining Start and Goal Node

Generated maze can be looked as a Cartesian coordinate system, which every hexagon position can be mapped to be an *x* and *y*. For example, hexagon tile in top left corner is *(0, 0)* and bottom right corner is *(54, 29*). This position will be used to tell the algorithm which hexagon tile will be the start node and the goal node. To make it a valid test, start node or goal node can't be a "black" hexagon.

C. Implement A Algorithm*

The next step is implement the A* algorithm in the generated maze. As it tells before, there are two shortest path algorithm that being tested in this research. Every start and goal node being chosen and it will be tested with both of this algorithm. The test result will be recorded in the next step.

First, A* without PBS will run and search a shortest path from start node to goal node. There is no modification in the A*, so it will run as it tells in *Fig. 4*.

Second, A* with PBS will be run with a same start node and goal as the first one. PBS will be implemented with a two different threads. This two threads working and running at the same time. To make this two threads can communicate with each other, we need to do a little modification in the A* algorithm pseudocode (see *Fig. 4*) that presented before.

```
function checkFromAnotherThread()
{
 for(each Node in ThisThread.path)
 {
  if(Node is in AnotherThread.path)
  {
   ThisThread.stop();
   AnotherThread.stop();
   return true;
  }
 }
 return false;
}

function PBSAStar(StartNode, GoalNode)
{
 closed_list = {};
 open_list = {};
 open_list.add(StartNode);
 path = {};
 while open_list.isNotempty()
 {
  //Find the node with lowest value of
  //f(n)=g(n)+h(n) from neighbor.
  CurrentNode = FindLowestValue();
  open_list.remove(CurrentNode);
  if(checkFromAnotherThread())
  {
   //Path found, stop iteration
   break;
  }
  . . .
```

Fig. 7 Pseudocode of PBS A* Algorithm

Fig. 7 shows the pseudocode of how PBS A* works. It is almost same what A* and PBS A* do in the search process. The only modification is in the start of iteration (marked with dashed lines). In A* algorithm the iteration will be stopped when it found the goal node, but in PBS A* it will be stopped when the two threads met. Condition that makes the two threads met each other is from node which included as a solution path they found is a same node.

D. Get execution time

The last step of this testing process is to record all of execution time in every test case. Since this testing process use C# as programming language, it will be easy to use any .NET library that already prepared to be used. In .NET, it is possible to check the execution time with their library (see *Fig. 8*).

```
var watch =
System.Diagnostics.Stopwatch.StartNew();
// the code that you want to measure comes here
watch.Stop();
var elapsedMs = watch.ElapsedMilliseconds;
```

Fig. 8 Get execution time implementation

After every test cases were done, the result will be collected and it will be examined later to see the difference between A* without optimization and PBS A*. *Table I* shows the execution time test result on every test cases.

TABLE I TEST RESULT

Test Case	A* implementation	Start Node	Goal Node	Execution Time
1	PBS A*	(0,0)	(54,29)	14ms
	Not optimized A*	(0,0)	(54,29)	45ms
2	PBS A*	(0,0)	(54,0)	8ms
	Not optimized A*	(0,0)	(54,0)	11ms
3	PBS A*	(0,0)	(0,29)	8ms
	Not optimized A*	(0,0)	(0,29)	23ms

Compared to not optimized A*, PBS A* have a faster execution time in hexagon-based environment. Based on some tests in a same maze, same goal node and same finish node, PBS A* have execution time 29ms faster in the first test case, 3ms faster in the second test case, and 15ms faster in the third case than not optimized A*.

From all three test cases, test case number 1 is the longest distance between start node and goal node. More distance means more node need to be expanded to search. It appears PBS A* can accelerate execution time 68.8% faster than not optimized A*.

V. CONCLUSION

This paper proposed that PBS can be an alternative to optimized a shortest path algorithm, especially A*. Two threads needed to run A* from start node and from finish node. Another success to run PBS is a CPU capability to have multiple cores in a single chip. With everything combined, it is possible to run two A* at the same time and find a solution path faster. PBS A* can accelerate path finding process 68.8% faster.

VI. FUTURE WORKS

This paper uses A* algorithm as a main path finding algorithm. In another research, we can find there is a tweak in A* algorithm itself to make it run faster. For future works, PBS can be used in tweaked A* algorithm and see if it makes execution time faster or not.

ACKNOWLEDGMENT

This work was being used as one of college task and it will be used as my Thesis as well.

REFERENCES

[1] S. Abbasi and F. Moosavi, "Finding shortest path in static networks : using a modified algorithm," *International Journal of Finance & Banking Studies,* vol. 1, no. 1, pp. 29-33, 2012.

[2] M. Randour, J.-F. Raskin and O. Sankur, "Variations on the Stochastic Shortest Path Problem," *Verification, Model Checking, and Abstract Interpretation,* vol. 8931, pp. 1-18, 2015.

[3] T. Chatzidimitris, D. Gavalas and V. Kasapakis, "PacMap: Transferring PacMan to the Physical Realm," in *Internet of Things. User-Centric IoT*, Rome, Italy, 2014.

[4] K. E. Merrick and M. L. Maher, "Non-Player Characters in Multiuser Games," in *Motivated Reinforcement Learning*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2009, pp. 3-16.

[5] H. Qu, Z. Zeng, C. Chen, D. Wang and N. Yao, "An Optimization Method of SNNs for Shortest Path Problem," in *Proceedings of the 18th*

Asia Pacific Symposium on Intelligent and Evolutionary Systems, Singapore, 2015.

[6] A. R., "Path Finding Solutions For Grid Based Graph," *Advanced Computing: An International Journal,* vol. 4, no. 2, pp. 51-60, 2013.

[7] P. Cowling, M. Buro, M. Bida, A. Botea, B. Bouzy, M. Butz, P. Hingston, H. Muñoz-Avila, D. Nau and M. Sipper, "Search in Real-Time Video Games," *Artificial and Computational Intelligence in Games,* vol. 6, pp. 1-19, 2014.

[8] B. Venu, "Multi-core processors - An overview," Cornell University Library, Ithaca, New York, 2011.

[9] W. Pijls and H. Post, "Note on "A new bidirectional algorithm for shortest paths"," *European Journal of Operational Research,* vol. 207, no. 2, pp. 1140-1141, 2010.

[10] H. Post and W. Pijls, "Yet another bidirectional algorithm for shortest paths," *Econometric Institute Report EI 2009-10,* pp. 1-9, 2009.

[11] K. Yakovlev, E. Baskin and I. Hramoin, "Grid-Based Angle-Constrained Path Planning Konstantin," *KI 2015: Advances in Artificial Intelligence,* vol. 9324, pp. 623-629, 2015.

[12] P. Hart, N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, no. 2, pp. 100-107, 1968.

[13] A. Goldberg, "Point-to-point shortest path algorithms with preprocessing," *Sofsem,* pp. 9-12, 2007.

[14] R. Lawrence and V. Bulitko, "Database-Driven Real-Time Heuristic Search in Video-Game Pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 5, no. 3, pp. 227-241, 2013.

[15] W. Lee and R. Lawrence, "Fast Grid-Based Path Finding for Video Games," in *Advances in Artificial Intelligence*, Canada, 2013.

[16] S. Brand and R. Bidarra, "Parallel Ripple Search – Scalable and Efficient Pathfinding for Multi-core Architectures," in *Motion in Games*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2011, pp. 290-303.

[17] L. Rios and L. Chaimowicz, "PNBA*: A Parallel Bidirectional Heuristic Search Algorithm," *ENIA VIII Encontro Nacional de Inteligência Artificial,* 2011.

[18] C. Reuter, V. Wendel, S. Göbel and R. Steinmetz, "Towards Puzzle Templates for Multiplayer Adventures," in *7th International Conference, Edutainment 2012 and 3rd International Conference, GameDays 2012*, Darmstadt, Germany, 2012.

C5_02_ICITEE 2016

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

C5_02_ICITEE 2016

ORIGINALITY REPORT

7%	0%	7%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Pijls, W.. "Note on "A new bidirectional algorithm for shortest paths"", European Journal of Operational Research, 20101201 Publication	7%

Exclude quotes	Off	Exclude matches	< 5%
Exclude bibliography	On		