A Comparison of Pathfinding Algorithm for Code Optimization on Grid Maps

Azyan Yusra Kapi[1], Mohd Shahrizal Sunar[2*], Zeyad Abd Algfoor[3]
Computing Sciences Studies-College of Computing-Informatics and Media, Universiti Teknologi MARA,
Johor Branch, Pasir Gudang Campus, 81700 Johor, Malaysia[1]
Department of Emergent Computing-Faculty of Computing, Universiti Teknologi Malaysia,
81310 Johor Bahru, Johor, Malaysia[1, 2]
Media and Game Innovation Centre of Excellence-Institute of Human Centered Engineering,
Universiti Teknologi Malaysia, 81310 Johor Bahru, Johor, Malaysia[1, 2]
Department of Computer Science-College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq[3]

***Abstract*—There have been various pathfinding algorithms created and developed over the past few decades to assist in finding the best path between two points. This paper presents a comparison of several algorithms for pathfinding on 2D grid maps. As a result, this study identified Jump Point Search Block Based Jumping (JPS (B)) as a potential algorithm in term of five evaluation metrics including search time. The comparisons pointed out the potential algorithm and code optimization was performed on the selected JPS(B) algorithm, and it was named JPS(BCO). This paper also explores issues regarding the JPS(B) and how to resolve them in order to optimize access to the index pointer. The presented enhance JPS(BCO) is capable to search optimal path quicker than the original JPS(B) as demonstrated by experimental findings. An experiment of different size grid maps is conducted to validate the ability of the proposed algorithm in term of search time. The comparative study with original JPS (B) exhibits the enhancement that has more benefits on grid maps of different size in terms of search time.**

Keywords—Comparative; jump point search; optimization; pathfinding; path planning

I. Introduction

Pathfinding algorithm has become one of the popular techniques to search for a path while avoiding obstacles at the same time. There have been numerous applications of pathfinding such as robotics, virtual reality [1], and commercial games [2]. In the past, pathfinding has traditionally focused on finding the shortest route. However, nowadays, it also focuses on finding the safest, cheapest, or most convenient route to avoid tolls, roads, or other obstacles [3].

Maps and graphs are common methods for representing environments in pathfinding. Most pathfinding algorithms consider the pathfinding environment as a key attribute to determine navigation performance [4]. Navigation meshes, grid maps, and waypoint graphs are three popular pathfinding environments. The superiority of one pathfinding environment over another has been debated for many years. A few advantages and limitations of the most common pathfinding environments used in games are summarized in [5]. Grid maps are the most popular pathfinding environment due to the simplicity and ease of use [6]. Furthermore, it is also relatively fast to generate a grid map and it only includes two categories of cells: passable and block able [7]. The movement of an agent in grid map is limited to eight possible directions. Every vertical and horizontal movement in the grid has one cost unit, while the diagonal movement has a cost of 1.4 units.

Over the past decades, the performance of the pathfinding has been evaluated through various comparisons and analysis. Recently, [8] has compared traditional pathfinding which is A* algorithm and Depth First Search (DFS) with state-of-the-art algorithms, Jump Point Search (JPS) and Subgoal Graph. By testing the four algorithms on eight different grid maps, the author discussed several of the advantages and disadvantages of each algorithm according to the grid maps. As another example, [9] also compared two well-known algorithms: A* and Iterative Deepening A∗ (IDA*). They concluded that when there are no obstacles on the map, IDA* generally performs better than A*. However, when it comes to memory and time usages, IDA* may perform worse if opponent characters are in parallel positions and blocked by obstacles.

On the other hand, [10] conducted an analysis to compare performance of A∗ and Basic Theta∗ algorithm. Results from this study indicate that the A* and Basic Theta* algorithms have both similar completeness and time taken, but the A* has the benefit of searching less nodes, while the Basic Theta* algorithm returns shorter results.

Despite being one of the oldest algorithms, A* is still being favored to be included in the various researches for comparison. Although there are numerous researches in the literature on the comparative analysis of several pathfinding algorithms, however, to date, there have been very few research on the comparative analysis of Jump Point Search algorithm. This paper aims to compare A* and four versions of Jump Point Search using source code which are made available by the author [6]. The total five algorithms which included in this study are A*, JPS, JPS(B), JPS+, and JPS+ (P).

One of the evaluation metrics that is commonly used in comparisons and analyses is the amount of time taken during search. In video games [11], search and rescue (SAR) [12], and unmanned aerial vehicle (UAV) navigation [13], the search time is one of the most critical aspects. Therefore, the key motivation for this study is to reduce the search time.

*Corresponding Author

Ministry of Education Malaysia and Universiti Teknologi MARA (sponsors).

Apart from comparing the pathfinding algorithms, this paper will also discuss code optimizations on the potential algorithm JPS(B) which are derived from the preliminary analysis. The code optimization yields a slight increase in terms of the search time. Thus, the fundamental objective and contribution of the current paper aim to present a comparative analysis on several pathfinding algorithm and a slight code optimization to the potential algorithm. This comparative analysis is important to other researcher for evaluating the performance of pathfinding algorithm and the code optimization performs on it.

In the remainder of the paper, the sections are arranged as follows. In Sections II, reviews of related works on JPS algorithms and descriptions of the JPS are provided. The methodological approach for the experimental setup was explained in Section III. Experimental result is presented in Section IV, while JPS(BCO) optimization and result is discussed in Sections V and VI, respectively. Conclusions and recommendations for future work are presented in Section VII.

II. Related Works

It was found that most pathfinding algorithms are based on A*, regardless of whether they are single or multi-agent pathfinding. A* algorithm was successful in solving pathfinding problems, and since that, numerous studies have concentrated on improving and optimizing A* algorithm. Path scoring is used by A* algorithm to determine the optimal path from the initial node to the end node [14]. Another prominent pathfinding algorithm is the Jump Point Search (JPS) which is the successor of A* variants introduced by [15]. They affirmed that the JPS accelerated more than A* and JPS has gain many attentions from other researcher after that.

Using JPS, an undirected and eight-connected grid map can be identified and eliminated from many path symmetries through combination of A* search and pruning rules. JPS algorithm utilizes pruned neighbor rules to determine which nodes should be searched while jump points are determined by their location relative to forced neighbor. Later in 2014, enhanced JPS is presented in [6] which includes four varieties of enhancement which includes jumping by block, new pruning rules and adds a pre-processing step prior to searching process.

The existing literature on JPS is extensive and focuses particularly on describing some of the more recent developments and enhancements. A summary of the research on JPS's improvement can be found in Table I.

Based on Table I, it is shown how JPS has been enhanced and applied to several areas, including home service robots and even in logistics for AGV. Most of the improvements were focused on reducing search times, which are considered valuable even for a few seconds. Apart from that, the aim is to minimize or shorten the path length as shown in (Ma et al., 2019). The related works conclude that the trends in optimizing the JPS algorithm focus on faster search times, which is also the motivation for this study.

TABLE I. Summary of JPS's Enhancement

Ref.	**Explanation**		
	Objectivity	***Description***	***Application***
[16]	Improve waiting steps and movement steps	IJPS combines JPS with Congestion control in two stages: online and offline mode.	Autonomous Ground Vehicle (AGV)
[17]	Decrease search time and secure distance for robot and the barrier	SD-JPS merge the idea of a jump point with the node domain matrix to suggest a different jump point and limitation that satisfy the JPS 's quick search results and achieve the secure distance between the robot and the barriers	Robotic
[18]	In addition to identifying the direction of the next path point relative to the current path point, it uses vector cross product to verify connectivity between the previous and last points of each original point.	Enhancements are presented in this paper to minimize redundant path points and optimize path length by shortening paths. In order for a robot to move, its pose must be adjusted in points, so the vector cross product and the vector dot product are applied.	Home Service Robot
[19]	Using grid signage, the grid environment is pre-processed, and a valuation function is used to determine the optimal path through the grid.	To improve the path efficiency, this algorithm optimizes the open list with the minimum binary tree, and enlarges the weight coefficient to choose the appropriate valuation function.	Radar simulation system
[20]	The InvJPS algorithm attempts to resolve the problem of inventory-driven pathfinding in the literature	When used in inventory-based variants of game maps, InvJPS maintains JPS's optimality guarantees and its symmetry breaking benefits.	Video Games

III. Experimental Setup

Original source code from [6] can be downloaded from https://bitbucket.org/dharabor/pathfinding. The given source code consists of several pathfinding algorithm in the same program. In Table II, an overview of the five pathfinding algorithms selected for comparison is presented in detail.

Table II shows that all algorithms were unweighted, except for the A* algorithm. A weighted version of A* algorithm is also available as used in [21] for their enhancements. However, for the purpose of simplification and standardization, only the unweighted version of the algorithms has been considered in this study, and the weighted version has been discarded.

While the given source code is in C++, [6] developed the code on Linux 20.04. Computer with Windows 10 operating system were used for the purpose of this study. Therefore, an Oracle VM VirtualBox 6.1.30 and Ubuntu 20.04 platform was used to match the original platform.

TABLE II. OVERVIEW OF PATHFINDING ALGORITHM USED IN THIS STUDY

Algorithm	***Algorithm name (source code)***	***Description***	***Unweighted***	***Weighted***
A*	astar	Traditional algorithm which utilizes an open list and close list to store each visited and unvisited nodes.	✓	✓
JPS	jps	Enhance A* by utilizing a jump point to skip unnecessary nodes by following two set of pruning rules.	✓	X
JPS (B)	jps2	In order to boost the original JPS's performance, it uses block-based jumping	✓	X
JPS+	jps+	This algorithm differs from original JPS as it adds pre-processing to the search method	✓	X
JPS+ (P)	jps2+	This algorithm similar to JPS+ which use pre-processing and also enhance pruning rules	✓	X

The computer has a minimum specification of 3.40GHz Intel Core i7-6700 processor with 4MB of RAM and 8MB of L3 cache. The first test data consists of four benchmark problem sets which were generated by [22]. Maps are taken from commercial games that are standards for 2D benchmarking. Only four maps and problem sets were selected to be included in the study. A description of the four selected benchmark files is presented in Table III.

From Table III, it is apparent that this selection of benchmark file was made to represent different dimension and problem sets complexity. For example, arena file represents smallest dimension which is 49x49 while orz700d contain 3880 problem sets with 1260x1104 dimension.

The comparison analysis relies on a five-evaluation metrics; the descriptions of these metrics can be found in Table IV.

The evaluation metrics used in this analysis are similar to those in [6] as shown in Table IV.

TABLE III. DESCRIPTION OF BENCHMARK FILES USED IN THIS STUDY

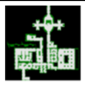
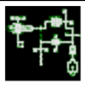
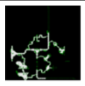
Map's name	***Preview***	***Dimension***	***Number of problem set***	***Maximum length in scenario***
arena		49 x 49	130	51.84
den501d		338 x 320	1170	466.90
brc202d		481 x 530	2550	1019.05
orz700d		1260 x 1104	3880	1551.98

TABLE IV. EVALUATION METRICS USED IN THIS STUDY

Metrics	***Description***
Total of expanded nodes	Since time is usually a hardware and software dependent factor, it is recommended to also determine the total number of expanded nodes in the search.
Total of generated nodes	An indicator of how many nodes are added to an open list after they are generated.
Total of touched nodes	Number of nodes undergoing evaluation, which may result in an update to priority queue.
Search time	Search time is measured in microseconds (wall clock time) which represents how long it takes the algorithm to find solution to the problem. Milliseconds are equal to microseconds multiplied by 1,000.
Memory cost	Metric that measures how much memory in bytes the algorithm uses in finding the end result.

IV. EXPERIMENTAL RESULT ON ANALYSIS OF PATHFINDING ALGORITHMS

Using the experimental procedure described by [23], all experiments were repeated 10 times and the average results calculated for each evaluation metric. A complete disconnect from the internet is kept throughout the experiment to prevent any disruptions caused by unwanted activity. Based on the five-evaluation metrics described previously, the following section discusses the results of the analysis. Results for the expanded nodes (average) are depicted in Table V.

To measure algorithm performance during pathfinding, the average of expanded nodes is an important criterion. In general, as the number of nodes grows, it takes longer time to find the path. Based on Table V, out of five algorithms, there is a tie between JPS (B) and JPS+ (P), whereas JPS+ (P) wins the remaining three benchmark files. Table IV and Table VII compare the result in term of generated nodes and touched nodes, respectively.

TABLE V. COMPARISON RESULT IN TERMS OF EXPANDED NODES

Algorithm	**Expanded nodes (average)**			
	arena	***den501d***	***brc202d***	***orz700d***
A*	31.42	5862.18	15997.63	29104.21
JPS	4.87	196.80	578.16	919.00
JPS (B)	**1.55**	113.43	342.81	479.53
JPS+	3.90	195.86	577.22	918.09
JPS+(P)	**1.55**	**113.43**	**342.81**	**479.53**

TABLE VI. COMPARISON RESULT IN TERMS OF GENERATED NODES

Algorithm	**Generated nodes (average)**			
	arena	***den501d***	***brc202d***	***orz700d***
A*	99.05	6100.47	16294.91	29497.70
JPS	11.11	224.03	606.61	951.14
JPS (B)	10.94	**151.64**	**386.26**	**527.28**
JPS+	**9.46**	222.27	604.85	949.36
JPS+(P)	11.05	151.80	386.45	527.44

TABLE VII. COMPARISON RESULT IN TERM OF TOUCHED NODES

Algorithm	Touched Nodes (Average)			
	arena	*den501d*	*brc202d*	*orz700d*
A*	274.07	48830.13	133876.84	247457.73
JPS	17.05	465.26	1336.50	2177.84
JPS (B)	**13.69**	**334.51**	**1022.47**	**1565.88**
JPS+	14.45	462.73	1333.97	2175.27
JPS+(P)	13.92	335.51	1023.80	1567.10

In Table VI, the average generated nodes that are injected into the open list show JPS+ to be the winner for the arena file, while JPS (B) is the winner for the rest of the files. In Table VII, in term touched nodes, JPS (B) is the most excellent for all four benchmarks file as it exhibits the smallest number of touched nodes in average. In terms of search time, the obtained result is presented in Table VIII.

In Table VIII, in terms of search time in microseconds, JPS+ is the winner for arena, while JPS+ (P) for the rest of three benchmark files. This situation is because the JPS+ and JPS+ (P) is based on pre-processing enhancement, thus, it will speed up the search time.

For the final evaluation metrics, the algorithms were compared in terms of memory in bytes. The comparison result is depicted in Table IX.

Based on Table IX, the lowest memory consumption for arena file is A*, while JPS (B) is the top algorithm for the rest of the files.

As a continuation of the analysis described previously, this study includes further calculations to find the potential and superior algorithm for further code optimization. To identify the superior algorithm among five previously tested algorithms, a scoring method was used.

TABLE VIII. COMPARISON RESULT IN TERMS OF SEARCH TIME

Algorithm	Search Time (microseconds)			
	arena	*den501d*	*brc202d*	*orz700d*
A*	18.86	3510.18	12111.38	24633.38
JPS	7.01	86.00	238.84	414.97
JPS (B)	3.36	69.91	206.85	319.55
JPS+	**2.37**	73.52	220.77	332.63
JPS+(P)	2.57	**57.88**	**171.90**	**268.09**

TABLE IX. COMPARISON RESULT IN TERMS OF MEMORY

Algorithm	Memory (bytes)			
	arena	*den501d*	*brc202d*	*orz700d*
A*	6182968	6551496	7215324	8400304
JPS	6183768	6301944	6721476	7513048
JPS (B)	**6183728**	**6301904**	**6721436**	**7248744**
JPS+	6239296	8224656	10925668	30021816
JPS+(P)	6239256	9017384	12775476	40856424

TABLE X. OVERALL SCORE FOR THE SUPERIOR ALGORITHM

Algorithm	The Superior Algorithm (Overall)				
	arena	*den501d*	*brc202d*	*orz700d*	*Total score*
A*	1	0	0	0	1
JPS	0	0	0	0	0
JPS (B)	**0.5**	**3**	**3**	**3**	**9.5**
JPS+	2	0	0	0	2
JPS+(P)	2	2	2	2	8

For this experiment, each winner algorithm receives a score of 1 and a score of 0.5 for a tie situation. Scoring calculation has been summarized and presented in Table X.

Table X shows that according to five evaluation metrics as explained previously, JPS (B) is the superior and potential algorithm among the five algorithms tested in the given source code. Thus, JPS(B) is the chosen algorithm for further testing and code optimization which will be explained in the following section.

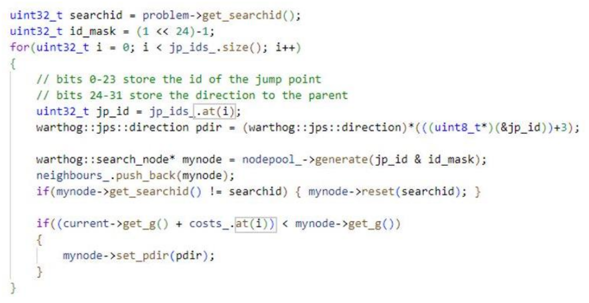
V. JPS(B) CODE OPTIMIZATION

For the implementation of JPS(B), the use of vector is widely utilized in the program's code as highlighted in Fig. 1.

As depicted in Fig. 1, JPS (B) in its implementation spends numerous of its time accessing the vector, thus, impeding, and slow down the overall process to search the path. For example, there are 130 scenarios in arena.scen benchmarking file, and it requires almost 1448 method invocation by the syntax "jp_ids_at(i)" in the loop. Every time the program runs the syntax "at(i)" to access the vector's element, compiler will do a range checking. When the program trying to access an element that does not exist in the vector, it throws an exception. Errors will be more easily found using this checking procedure through the syntax "at(i)". However, as mentioned earlier, the syntax "at(i)" will cause overhead to the overall program in terms of search time.

In this code optimization for JPS(B), the original implementation access values from the "std::vect" class by using syntax "at(i)". The implementation is changed from syntax "at(i)" to syntax "[i]" to access the vector as highlighted in Fig. 2.

The two differ significantly: "at()" checks boundaries while operator "[]" does not. Thus, this will result in a reduction of overhead for the program's code since the syntax for accessing the vector has been changed.

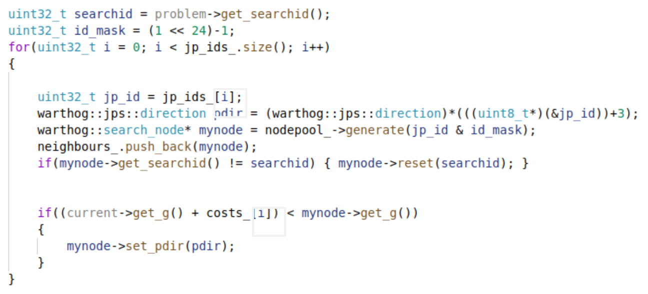


```
uint32_t searchid = problem->get_searchid();
uint32_t id_mask = (1 << 24)-1;
for(uint32_t i = 0; i < jp_ids_.size(); i++)
{
    // bits 0-23 store the id of the jump point
    // bits 24-31 store the direction to the parent
    uint32_t jp_id = jp_ids_.at(i);
    warthog::jps::direction pdir = (warthog::jps::direction)*(((uint8_t*)(&jp_id))+3);

    warthog::search_node* mynode = nodepool_->generate(jp_id & id_mask);
    neighbours_.push_back(mynode);
    if(mynode->get_searchid() != searchid) { mynode->reset(searchid); }

    if((current->get_g() + costs_.at(i)) < mynode->get_g())
    {
        mynode->set_pdir(pdir);
    }
}
```

Fig. 1. Snippet for JPS(B) source file.



```
uint32_t searchid = problem->get_searchid();
uint32_t id_mask = (1 << 24)-1;
for(uint32_t i = 0; i < jp_ids_.size(); i++)
{

    uint32_t jp_id = jp_ids_[i];
    warthog::jps::direction pdir = (warthog::jps::direction)*(((uint8_t*)(&jp_id))+3);
    warthog::search_node* mynode = nodepool_->generate(jp_id & id_mask);
    neighbours_.push_back(mynode);
    if(mynode->get_searchid() != searchid) { mynode->reset(searchid); }


    if((current->get_g() + costs_[i]) < mynode->get_g())
    {
        mynode->set_pdir(pdir);
    }
}
```

Fig. 2. Snippet for JPS (BCO) source file.

VI. Result and Discussion for Code Optimization of JPS (BCO)

Based on the results of previous preliminary tests, it is necessary to gain a good understanding of the JPS (B) algorithm in order to optimize it.

In the given source code implementation, JPS (B) were noticed to make good use of vectors and it can be manipulated to take advantage to shorten the search time. In JPS (B) source code, the way vectors are accessed has been modified, and the optimized code version is named JPS (BCO). The following Table XI is the comparison between JPS (B) and JPS (BCO) in terms of search time. The same four benchmark files were selected as used in previous preliminary testing. The experiment is also repeated 10 times and the average is calculated as shown in Table XI.

Based on Table XI, for the comparison, in terms of expanded, generated, and touched nodes, there are no differences between the JPS (B) and JPS (BCO). This is also the same with memory consumption. It is because the changes made to the vector did not affect any nodes. However, the result was placed in the same table as search time, to demonstrate the differences of overall performance.

TABLE XI. Comparison of JPS (B) and JPS (BCO)

Evaluation metrics	***Algorithm***	***arena***	***den501d***	***brc202d***	***orz700d***
Expanded nodes	JPS (B)	1.554	113.429	342.806	479.533
	JPS (BCO)	1.554	113.429	342.806	479.533
Generated nodes	JPS (B)	10.938	151.638	386.259	527.278
	JPS (BCO)	10.938	151.638	386.259	527.278
Touched nodes	JPS (B)	13.692	334.507	1022.475	1565.875
	JPS (BCO)	13.692	334.507	1022.475	1565.875
Search time (micro seconds)	**JPS (B)**	**4.987**	**98.240**	**273.451**	**430.474**
	JPS (BCO)	**4.021**	**90.212**	**264.906**	**416.608**
Memory (bytes)	JPS (B)	6185888	6304568	6719204	7245720
	JPS (BCO)	6185888	6304568	6719204	7245720

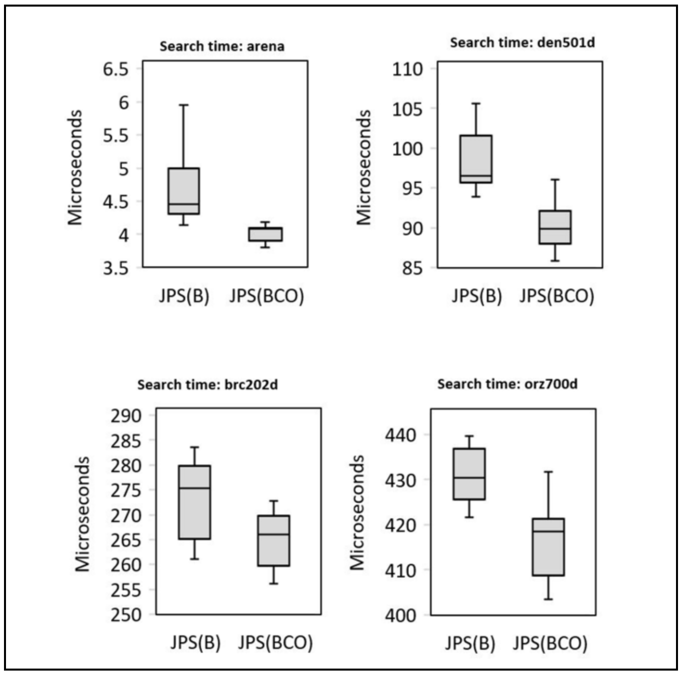


Fig. 3. Boxplot of result of JPS (B) and JPS (BCO).

In terms of search time, the average result is shown in microseconds which is much smaller unit than second and milliseconds. For arena file, the average search time using JPS (BCO) is only a microsecond shorter than JPS(BCO) performance. For the rest of the three benchmark files, each represents 8.0, 8.5 and 13.9 microseconds of faster acceleration in terms of search time. In order to provide a clear understanding of the code optimization results, a comparison of JPS(B) and JPS(BCO) search times is shown in Fig. 3.

From Fig. 3, in this boxplot, average time for JPS(B) take longer time in microseconds than JPS(BCO). In this modification of vector access's syntax, switching from "at()" to operator "[]" was a straightforward fix, but it resulted in a speed boost for the average of search time.

VII. Conclusion

Several pathfinding algorithms were compared including the well-known and traditional algorithm A*. The preliminary comparisons identify JPS(B) as a potential and the superior algorithms among the five tested algorithm. Thus, this paper performs a code optimization to the JPS(B) and called it as JPS(BCO). JPS(BCO) enhance JPS(B) performance by slightly shorten the search time.

In future, several modifications to the code implementation need to be studied in order to improve and enhance the performance of the JPS(B) significantly. The optimization should aim to improve not only the search time, but other evaluation metric such as total expanded nodes. Generally, it is not necessary to record time across platforms or machines for total expanded nodes to prove their performance, since the average is constant regardless of the platform.

In conclusion, this study has provided insight into potential and superior algorithms among JPS family members. As a result, this proposed work presents a comparison of existing algorithms. Apart from that, this study improved the JPS(B) search time through code optimization. The direction of future research can be explored in other areas, such as SAR and UAV navigation for pathfinding optimization.

ACKNOWLEDGMENT

This research was conducted during the main author's study leave under the 2019 Academic Training Scheme for Bumiputera (SLAB) scholarship scheme. A special thanks is extended to the Ministry of Education Malaysia and Universiti Teknologi MARA for the generous funding and support. It is with great appreciation that this research is being conducted with the assistance of postgraduate students from the Media and Game Innovation Centre of Excellence (MaGICX) at Universiti Teknologi Malaysia (UTM).

REFERENCES

[1] N. Nor, M. Sunar, and A. Kapi, "A review of gamification in virtual reality (VR) sport," EAI Endorsed Trans. Creat. Technol., vol. 6, no. 21, 2019.

[2] A. Y. Kapi, M. S. Sunar, and M. N. Zamri, "A review on informed search algorithms for video games pathfinding," Int. J. Adv. Trends Comput. Sci. Eng., vol. 9, no. 3, pp. 2756–2764, 2020, doi: 10.30534/ijatcse/2020/42932020.

[3] A. Y. Kapi, M. S. Sunar, and Z. A. Algfoor, "Summary of Pathfinding in Off-Road Environment," in 2020 6th International Conference on Interactive Digital Media (ICIDM), 2020, pp. 1–4, doi: 10.1109/ICIDM51048.2020.9339639.

[4] Z. Abd Algfoor, M. S. Sunar, and H. Kolivand, "A comprehensive study on pathfinding techniques for robotics and video games," Int. J. Comput. Games Technol., vol. 2015, 2015, doi: 10.1155/2015/736138.

[5] S. Rabin and N. R. Sturtevant, "Choosing a Search Space Representation," Game AI Pro 360, vol. 1, no. c, pp. 13–18, 2019, doi: 10.1201/9780429055096-2.

[6] D. Harabor and A. Grastien, "Improving jump point search," in Proceedings of the International Conference on Automated Planning and Scheduling, 2014, vol. 24, pp. 128–135.

[7] A. N. Sabri, N. Haizan, M. Radzi, and H. Hassan, "The State of Art Heuristic Pathfinding in Games," vol. 24, no. 2, pp. 1273–1278, 2018, doi: 10.1166/asl.2018.10731.

[8] X. Wei and D. Lu, "A Comprehensive Study on Pathfinding Algorithm for Static 2D Square Grid," Proc. - 2022 2nd Asia Conf. Inf. Eng. ACIE 2022, pp. 77–80, 2022, doi: 10.1109/ACIE55485.2022.00024.

[9] A. Primanita, R. Effendi, and W. Hidayat, "Comparison of A∗ and Iterative Deepening A∗ algorithms for non-player character in Role Playing Game," ICECOS 2017 - Proceeding 2017 Int. Conf. Electr. Eng. Comput. Sci. Sustain. Cult. Herit. Towar. Smart Environ. Better Futur., pp. 202–205, 2017, doi: 10.1109/ICECOS.2017.8167134.

[10] E. R. Firmansyah, S. U. Masruroh, and F. Fahrianto, "Comparative analysis Of A∗ and basic theta∗ algorithm in android-based pathfmding games," Proc. - 6th Int. Conf. Inf. Commun. Technol. Muslim World, ICT4M 2016, pp. 275–280, 2017, doi: 10.1109/ICT4M.2016.56.

[11] A. Rafiq, T. A. A. Kadir, and S. N. Ihsan, "Pathfinding Algorithms in game development," in IOP Conference Series Materials Science and Engineering, 2020, vol. 769, no. 1, p. 12021.

[12] A. Goyal, P. Mogha, R. Luthra, and N. Sangwan, "Path finding: A* or dijkstra's?," Int. J. IT Eng., vol. 2, no. 1, pp. 1–15, 2014.

[13] Z. Wu, Z. Meng, W. Zhao, and Z. Wu, "Fast-RRT: A RRT-Based Optimal Path Finding Method," Appl. Sci., vol. 11, no. 24, p. 11777, 2021.

[14] J.-Y. Wang and Y.-B. Lin, "Game AI: Simulating Car Racing Game by Applying Pathfinding Algorithms," Int. J. Mach. Learn. Comput., vol. 2, no. 1, pp. 13–18, 2012, doi: 10.7763/ijmlc.2012.v2.82.

[15] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," Proc. Natl. Conf. Artif. Intell., vol. 2, pp. 1114–1119, 2011.

[16] Y. Zhang and H. Huang, "Multi-AGVs Pathfinding Based on Improved Jump Point Search in Logistic Center," in Algorithmic Aspects in Information and Management, 2020, pp. 358–368.

[17] X. Zheng, X. Tu, and Q. Yang, "Improved JPS Algorithm Using New Jump Point for Path Planning of Mobile Robot," in 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Aug. 2019, pp. 2463–2468, doi: 10.1109/ICMA.2019.8816410.

[18] L. Ma, X. Gao, Y. Fu, and D. Ma, "An Improved Jump Point Search Algorithm for Home Service Robot Path Planning," Proc. 31st Chinese Control Decis. Conf. CCDC 2019, pp. 2477–2482, 2019, doi: 10.1109/CCDC.2019.8833422.

[19] J. Wang and J. Jiang, "Jump point search plus algorithm based on radar simulation target path planning," Proc. - 2017 Int. Conf. Comput. Technol. Electron. Commun. ICCTEC 2017, pp. 480–483, 2017, doi: 10.1109/ICCTEC.2017.00110.

[20] D. Aversa, S. Sardina, and S. Vassos, "Path planning with Inventory-driven Jump-Point-Search," no. September, 2016, [Online]. Available: http://arxiv.org/abs/1607.00715.

[21] Z. A. Algfoor, M. S. Sunar, and A. Abdullah, "A new weighted pathfinding algorithms to reduce the search time on grid maps," Expert Syst. Appl., vol. 71, pp. 319–331, 2017, doi: 10.1016/j.eswa.2016.12.003.

[22] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," IEEE Trans. Comput. Intell. AI Games, vol. 4, no. 2, pp. 144–148, 2012.

[23] C. McMillan, E. Hart, and K. Chalmers, "Collaborative diffusion on the gpu for path-finding in games," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9028, pp. 418–429, 2015, doi: 10.1007/978-3-319-16549-3_34.