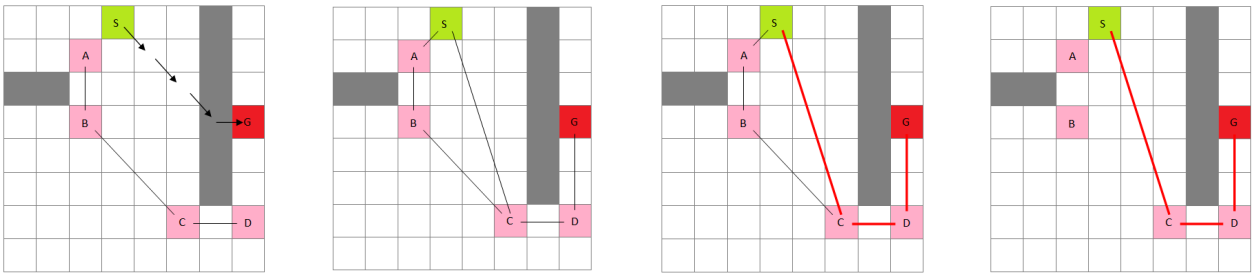


(a) The first step (b) The second step (c) The third step (d) The fourth step

Figure 8: The steps of the Simple Subgoal algorithm.

For the experiments with these algorithms, a simulation environment was created using the *Python* [35, 25] programming language. The user interface was created using the *Pygame* library and the *NetworkX* [11, 20] library was used for graph construction. The node-to-node transition cost in the vertical and horizontal directions was set to 1 and the node-to-node transition cost in the diagonal direction was set to $\sqrt{2}$. The metric thus corresponded to the planar unit Euclidean metric (also referred to as the octile metric in this case).

The monitored values were the number of expanded nodes, shortest path length and time. The resulting values of the achieved times are the medians of the hundred searches performed. These resulting times are the basis for the presented speed up comparisons.

It is also important to extend the selected methods with the so-called bounding box technique, which is based on a simple idea. First, a bounding box is computed around the start and destination nodes. These nodes are then treated as obstacles. The algorithm then search for the shortest path within this box and if it does not find a path inside the box, it expands the box. The initial size of the bounding box, as well as the number of nodes it expands by, should be chosen appropriately with respect to the shape of the map. Since expanding the bounding box laterally by one or more nodes slows down the algorithm.

Experiment 1 (Tab. 1) – A map containing simple obstacles was chosen as the first map, as shown in Fig. 9. The results from finding the shortest path on this type of map are in favor of the Simple Subgoal and JPS algorithms. In the case of Simple Subgoal, the optimal path was not found, it is longer than the other methods due to the limitations of this algorithm. The median preprocessing time for this method was 45 ms.

Experiment 2 (Tab. 2) – Another class of maps in the test was the maze map with obstacle density 45 %, as shown in Fig. 10. The results of the second experiment show that in terms of time and in case of JPS also in number of expanded nodes, the SS and JPS algorithms performed the best. Preprocessing for the SS algorithm took 39 ms. This value ranked the SS algorithm first in speed again. Note that the optimal route was not guaranteed by the principle of the algorithm and in this case was not found.

Table 1: Experiment 1, simple map

Method	Path length	Expand nodes	Speed-Up
Dijkstra	36.38	506	1
A*	36.38	264	1.75
A*+BB	36.38	253	1.45
JPS	36.38	39	2.66
JPS+BB	36.38	18	2.23
SS	38.14	69	4.21
Dijkstra 16.39 ms			

Table 2: Experiment 2, maze, 45% obstacle

Method	Path length	Expand nodes	Speed-Up
Dijkstra	30.31	360	1
A*	30.31	68	2.01
A*+BB	30.31	67	1.68
JPS	30.31	20	2.20
JPS+BB	30.31	20	1.88
SS	34.41	103	2.94
Dijkstra 8.72 ms			

Experiment 3 (Tab. 3) – This set of experiments was performed on two randomly generated maps of 50x50 and 100x100 nodes. The results are again in the spirit of the previous experiments. The JPS and SS algorithms are the fastest, even in terms of the number of expanded nodes. In the case of the 50x50 map, JPS augmented with a bounding box performed the best. Preprocessing for Simple Subgoal took 280 ms for the 50x50 map and 330 ms for the 100x100 map. The Simple Subgoal algorithm was the fastest but by a very small margin, not optimal.

Table 3: Experiment 3, map size $n \times n$

Method, for $n = 50$	Path length	Expand nodes	Speed-Up
Dijkstra	41.14	2124	1
A*	41.14	221	2.67
A*+BB	41.14	221	2.43
JPS	41.14	83	3.07
JPS+BB	41.14	83	3.36
SS	42.31	740	3.06
Dijkstra 55.73 ms			
Method, for $n = 100$	**Path length**	**Expand nodes**	**Speed-Up**
Dijkstra	169.30	10646	1
A*	169.30	7633	1.14
A*+BB	169.30	7633	1.10
JPS	169.30	10	3.23
JPS+BB	169.30	10	2.21
SS	169.88	7	9.28
Dijkstra 234.45 ms			

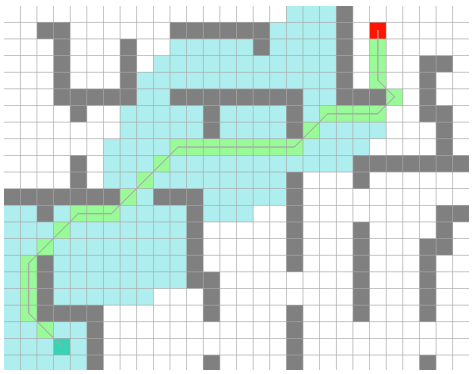


Figure 9: Experiment 1 – test map with simple obstacles. The optimal solution obtained by the A^* algorithm is shown.

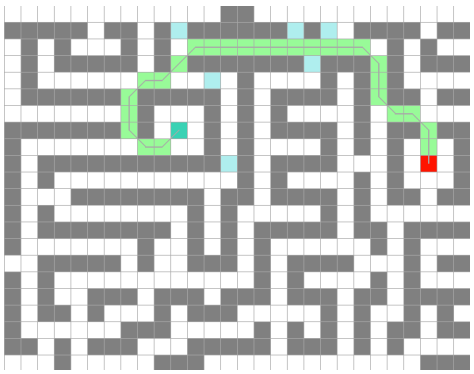


Figure 10: Experiment 2 – test map as a maze with 45% obstacles. The solution achieved by the JPS algorithm is shown.

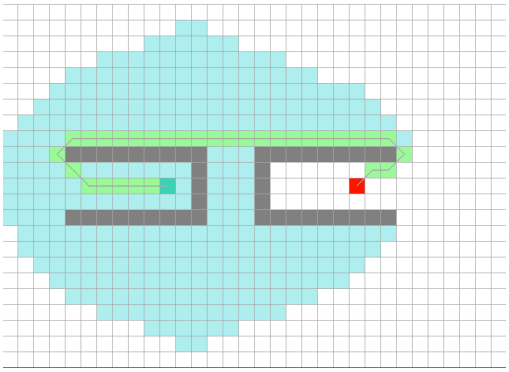


Figure 11: Experiment 4 – test map with deceptive problem and solution using A^*. The turquoise area represents the search space (expanded nodes).

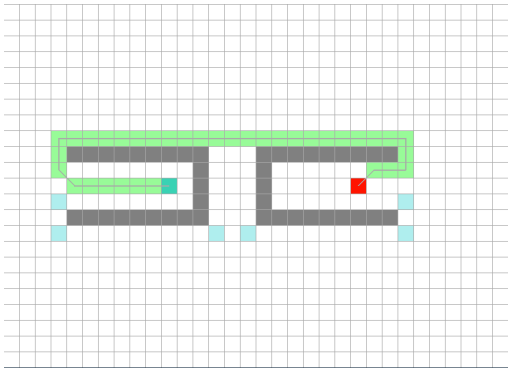


Figure 12: Experiment 4 – deceptive problem and a solution using the SS algorithm. The turquoise area represents the searched space (expanded nodes).

Experiment 4 (Tab. 4) – This experiment was performed on a map containing two identical obstacles. It is a simple deception problem of a known type. Of the results obtained, the Simple Subgoal and JPS algorithms performed best again. Both in terms of number of expanded nodes and computation time. The speed up of the Simple Subgoal algorithm was considerable in this case.

Table 4: Experiment 4, simple deception problem

Method	Path length	Expand nodes	Speed-Up
Dijkstra	34.49	698	1
A*	34.49	336	1.34
A*+BB	34.49	320	1.11
JPS	34.49	12	2.06
JPS+BB	34.49	13	1.43
SS	36.83	12	7.58

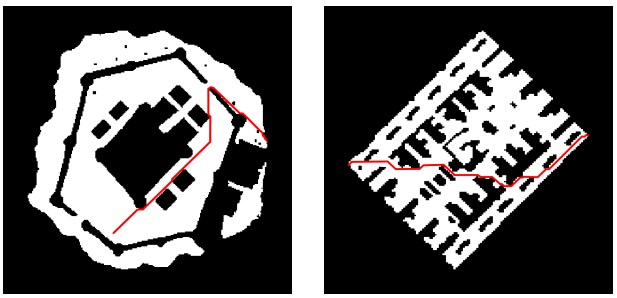
Dijkstra 18.13 ms

Experiment 5 (Tab. 5) – This class of experiments was conducted on larger maps from the MovingAI [30] test corpus. Specifically, the maps were AR0011SR and AR0013SR from the Baldur's Gate 2 game. Both of these maps are 512 x 512 cells in size. As can be seen from the results, methods using the bounding box are significantly slower than methods without this extension. The reason for this slowdown is due to the design of the bounding box itself and its extension. The map preprocessing time for the Simple Subgoal algorithm was 19 758 ms for the AR0011SR map and 7 945 ms for the AR0013SR map.

Table 5: Experiment 5, MovingAI corpus, 512 × 512

Method, for AR0011SR	Path length	Expand nodes	Speed-Up
Dijkstra	553.95	115706	1
A*	553.95	30442	1.72
A*+BB	553.95	30442	0.62
JPS	553.95	62	3.04
JPS+BB	553.95	62	1.15
SS	556.29	1133	9.43
Dijkstra 2 923 ms			
Method, for AR0013SR	**Path length**	**Expand nodes**	**Speed-Up**
Dijkstra	525.24	70791	1
A*	525.24	56261	1.04
A*+BB	525.24	55484	0.33
JPS	525.24	147	2.93
JPS+BB	525.24	147	0.60
SS	525.83	1133	7.91
Dijkstra 1 923 ms			

For reference, the experiments were run on a computer with an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz and 16GB of RAM.



(a) (b)

Figure 13: Games gridworld domains as benchmarks, Moving AI Lab [30], where (a) is AR0011SR map, (b) is AR0013SR map.

4 Conclusion

The aim of this paper was to present and evaluate two very interesting and relatively new algorithms for path planning on the chosen test problems. All tasks were implemented on an orthogonal two-dimensional mesh. This lattice with a defined Moore's neighborhood for each point is a good approximation for mobile object motion planning tasks and path planning tasks, respectively. The advanced JPS and Simple Subgoal algorithms streamline the search process by reducing the number of state space nodes explored. To compare them, representatives of classical path planning methods have also been implemented, these are the A^* algorithm and the Dijkstra algorithm. Five experiments were performed to compare the implemented algorithms. In these experiments, the length of the path found, the number of expanded nodes, and the time required to find the shortest path were evaluated. In addition, a simple search space reduction method was also implemented for the experiments, in the form of a bounding box for the JPS and A^* methods. The benefit of adding a bounding box depends on its setting and the form of the map. Comparative experiments have shown that the use of the JPS and Simple Subgoal methods leads to better shortest path search results with respect to the number of expanded nodes and also to lower time consumption. For the Simple Subgoal algorithm, it is important to note that it solves the problem in an engineering-optimal manner, but in best time. The JPS method is the second best in terms of speed, with a significant bonus of optimal solution. The very frequent and optimal A^* method has also proven its excellent applicability despite its age. All of the above methods except SS are optimal for the path length found. Given the presented detail and illustrative description of the JPS and SS algorithms, the authors hope to benefit from the good understanding of other authors and the creation of new implementations of these advanced path planning algorithms. The experiments were performed on implementations of the algorithms according to the authors' sources. The dependence of algorithms' speedup according to the implementation of the data structure is obvious.

Acknowledgement: This work was supported by IGA of BUT: FME-S-20- 6538 "Industry 4.0 and AI methods", and FEKT/FSI-J-22-7968 "Optimization of the cloud resources for the cyber security games.".

References

[1] Abd Algfoor, Z., Sunar, M. S., and Kolivand, H. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology 2015* (2015).

[2] Björnsson, Y., and Halldórsson, K. Improved heuristics for optimal path-finding on game maps. *AIIDE 6* (2006), 9–14.

[3] Botea, A. Ultra-fast optimal pathfinding without runtime search. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2011), vol. 6.

[4] Botea, A., Müller, M., and Schaeffer, J. Near optimal hierarchical path-finding. *J. Game Dev. 1*, 1 (2004), 1–30.

[5] Brewka, G. Artificial intelligence—a modern approach by stuart russell and peter norvig, prentice hall. series in artificial intelligence, englewood cliffs, nj. *The Knowledge Engineering Review 11*, 1 (1996), 78–79.

[6] Bulitko, V., Björnsson, Y., and Lawrence, R. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research 39* (2010), 269–300.

[7] Cazenave, T. Optimizations of data structures, heuristics and algorithms for path-finding on maps. In *2006 IEEE symposium on computational intelligence and games* (2006), IEEE, pp. 27–33.

[8] Costa, M. M., and Silva, M. F. A survey on path planning algorithms for mobile robots. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)* (2019), IEEE, pp. 1–7.

[9] Dijkstra, E. W., et al. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[10] Galceran, E., and Carreras, M. A survey on coverage path planning for robotics. *Robotics and Autonomous systems 61*, 12 (2013), 1258–1276.

[11] Hagberg, A., Swart, P., and S Chult, D. Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[12] Hamed, O., and Hamlich, M. Hybrid formation control for multi-robot hunters based on multi-agent deep deterministic policy gradient. *MENDEL 27*, 2 (Dec. 2021), 23–29.

[13] Harabor, D., and Botea, A. Breaking path symmetries on 4-connected grid maps. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference* (2010).

[14] Harabor, D., and Grastien, A. Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2011), vol. 25.

[15] Harabor, D., Hechenberger, R., and Jahn, T. Benchmarks for pathfinding search: Iron harvest. In *Proceedings of the International Symposium on Combinatorial Search* (2022), vol. 15, pp. 218–222.

[16] Harabor, D. D., and Grastien, A. The jps pathfinding system. In *SOCS* (2012).

[17] Harabor, D. D., Uras, T., Stuckey, P. J., and Koenig, S. Regarding jump point search and subgoal graphs. In *IJCAI* (2019), pp. 1241–1248.

[18] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[19] Hernández, C., and Baier, J. A. Fast subgoaling for pathfinding via real-time search. In *Twenty-First International Conference on Automated Planning and Scheduling* (2011).

[20] Klobušníková, Z. Mobile robot path planning. Master thesis (in czech), Brno University of Technology, Brno, 2018.

[21] Korf, R. E. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence 27*, 1 (1985), 97–109.

[22] LaValle, S. M. *Planning algorithms*. Cambridge university press, 2006.

[23] Lee, J.-Y., and Yu, W. A coarse-to-fine approach for fast path finding for mobile robots. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2009), IEEE, pp. 5414–5419.

[24] Lozano-Pérez, T., and Wesley, M. A. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM 22*, 10 (1979), 560–570.

[25] Maňáková, L. Advanced methods of mobile robot path planning. Master thesis (in czech), Brno University of Technology, Brno, 2020.

[26] Nobes, T. K., Harabor, D., Wybrow, M., and Walsh, S. D. The jps pathfinding system in 3d. In *Proceedings of the International Symposium on Combinatorial Search* (2022), vol. 15, pp. 145–152.

[27] Pochter, N., Zohar, A., Rosenschein, J. S., and Felner, A. Search space reduction using swamp hierarchies. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010).

[28] Rabin, S., and Silva, F. Jps+: An extreme a* speed optimization for static uniform cost grids. In *Game AI Pro 360*. CRC Press, 2019, pp. 95–108.

[29] Sankaranarayanan, J., Alborzi, H., and Samet, H. Efficient query processing on spatial networks. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems* (2005), pp. 200–209.

[30] Sturtevant, N. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games 4*, 2 (2012), 144 – 148.

[31] Sturtevant, N., and Buro, M. Partial pathfinding using map abstraction and refinement. In *AAAI* (2005), vol. 5, pp. 1392–1397.

[32] Sturtevant, N. R., Felner, A., Barrer, M., Schaeffer, J., and Burch, N. Memory-based heuristics for explicit state spaces. In *Twenty-First International Joint Conference on Artificial Intelligence* (2009).

[33] Uras, T., and Koenig, S. Subgoal graphs for fast optimal pathfinding. In *Game AI Pro 360*. CRC Press, 2019, pp. 109–124.

[34] Uras, T., Koenig, S., and Hernández, C. Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Twenty-Third International Conference on Automated Planning and Scheduling* (2013).

[35] VanRossum, G., and Drake, F. L. *The python language reference*. Python Software Foundation Amsterdam, Netherlands, 2010.