IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 10, No. 2, June 2021, pp. 265~272
ISSN: 2252-8938, DOI: 10.11591/ijai.v10.i2.pp265-272

Empirical study prove that breadth-first search is more effective memory usage than depth-first search in frontier boundary cyclic graph

Al Refai Mohammed N.[1], Jamhawi Zeyad[2]
[1]Department of Software Engineering, Zarqa University, Jordan
[2]Department of Computer Science, Zarqa University, Jordan

Article Info

Article history:

Received Oct 5, 2020
Revised Feb 2, 2021
Accepted Feb 12, 2021

Keywords:

Breadth-first search
Depth first search
Frontier boundary
Queue
Slide tiles puzzle
Stack

ABSTRACT

Memory consumption, of opened and closed lists in graph searching algorithms, affect in finding the solution. Using frontier boundary will reduce the memory usage for a closed list, and improve graph size expansion. The blind algorithms, depth-first frontier Searches, and breadth-first frontier Searches were used to compare the memory usage in slide tile puzzles as an example of the cyclic graph. This paper aims to prove that breadth-first frontier search is better than depth-first frontier search in memory usage. Both opened and closed lists in the cyclic graph are used. The level number and nodes count at each level for slide tile puzzles are changed when starting from different empty tile location. Eventually, the unorganized spiral path in depth-first search appears clearly through moving inside the graph to find goals.

Corresponding Author:

Refai Mohammed
Department of Software Engineering
Zarqa University, Jordan
Email: refai@zu.edu.jo

1. INTRODUCTION

Various search algorithms complexities were considered in terms of time, space, and cost of optimal path solution. Most of the previous work focused on heuristic search at A* [1] and IDA* [2]. The complexity in the worst case in space required for breadth-first search (BFS) is O(bd), and depth-first search (DFS) is only O(d) [3]. For acyclic graph as DFS need linear space in maximum search depth and expand more nodes [4], but One limitation of BFS and DFS algorithms is that they expand and store all possible intermediate nodes [5]. BFS uses too much space, where DFS, in general, uses too much time without guaranteed to find the shortest path in solutions [3].

Frontier, Sparse, [6], [7] and Divide-and-Conquer [8] will reduce the space for a close list. It save memory to increase the expand graph. When a problem is not fit in memory at cyclic graph or only contain a small number of cycles in breadth first search algorthem [9], [10]. Divide-and-Conquer frontier search save memory by storing only the open list and not the closed list as save in slow magnetic disk files to speed up internal memory search. This is done by using delayed duplication detection to remove nodes duplication [11]-[13]. At the same time, frontier reduces the size as a boundary in the search graph, that prevents previously closed nodes from being revisited [14]. To breaks down behaviour of algorithm exploits problem decomposition using search spaces [15] to improve search for breadth-first search and depth-first search.

This research will compare total memory usage at breadth first search [16], [17], [3] and depth-first search [18], [17], [3] algorithms using frontier for an opening list and active closing list which use to detect duplicate nodes. The useless close list nodes for detect duplicate will be removed from memory to magnetic disk files because they only used to extract path when reaching the goal. The slide tile puzzles [5], [19]-[26]

sizes 2X4, 2X5, 3X3 used to compare the memory usage at the two algorithms with frontier search as cyclic graph.

The main contribution for this research is to prove that breadth first frontier search (BFFS) consumes less memory for an open list and close list than depth first frontier search (DFFS) in cyclic graph. This fact for all nodes lists in all case study even when changing the empty tile location. The better description for DFFS when moving in the domain as unorganized spiral way, at last, the graph shape is changing, when the empty tile location change in the case study especially when the branch factor number change.

2. RESEARCH METHOD

All the calculations at this paper consider the start node located at level one.

2.1. Generate BFFS scenario

The scenario to generate breadth first search use current list (CL), future list (FL), previous list (PL) and final close list (NL) to save nodes. The first use list is CL to generate from each node all its related nodes. If the new node is not in PL and FL, then it will be added to the FL in a unique order directly. When finish passes through CL, the PL moves to NL, then CL moves to PL, then FL moves to CL, and FL became empty to be filled in the next step. In the final level, the FL will be empty so PL and CL will move to NL.

As the CL sorted, the siblings visit in order at this paper, but this is not important as the results related to levels. Table 1 display the BFFS lists and there uses. The using of NL is important to get all the domain nodes and the path nodes if required. Otherwise, it can be neglected as the result of Zhou and Hansen discovered the fact to use two levels for check is enough because it suffices to store a subpart of nodes that forms a boundary between the frontier and interior of the explicit search graph [18]. The Algorithm-1 is the pseudocode to generate all BFFS nodes as shown in Figure 1.

Table 1. BFFS lists and there uses

List Name	List Description
FL	Used to check duplication and used to add new unique nodes that not found here or in the previous list
CL	Use its node to Generate new nodes that will add to the future list if not found on the previous list and future list previously
PL	Used to check duplication and move to the final list when raising to a new level
NL	Save all list of the domain not used to detect duplicate nodes at search

```
1: #Generate all BFS Domain from Start Node
2: FL.Append(Start Node)
3: while FL is not empty
4. NL.Append(All PL)
5: PL = CL
6: CL = FL
7: FL = []
8: for each Node in CL
9: Generate Related Nodes(Node)
10: for each Related Node not in PL and FL
11: FL.Append(Related Node)
12: end for
13: end for
14: end while
```

Figure 1. The pseudocode to generate all BFFS nodes

2.2. Generate DFFS scenario

The scenario for DFFS nodes generating is more straightforward but less time-efficient compare to BFFS, only use the open stack (OS), previous list (PL) and final list (NL). First pop the node from the stack then generate its related nodes, any new node generated will be checked if not found in OS, and PL will be added at the top of nodes in the OS; the popped node will be added to PL.

The pop from OS will be repeated until OS became empty, after all, available nodes popped, and no new node added. The NL list will be explained in detail when discussing the removing useless nodes at search from PL for DFFS. Table 2 displays DFFS lists and stack. The Algorithm-2 is the pseudocode to generate all DFFS nodes as shown in Figure 2.

Table 2. DFFS lists and stack

List Name	List Description
OS	Used to pop nodes to generate its related new nodes and push the new unique nodes to it which not found at the open stack and the previous list
PL	The popped nodes from the open stack added to the list to check duplication when new node generated.
NL	Save all useless nodes to detect duplicate nodes from the previous list when adding popped nodes.

```
1: #Generate all DFFS Domain from Start Node
2: OS.Push(Start Node)
3: while OS is not empty do
4: Node = OS.Pop(Node)
5: Generate Related Nodes (Node)
6: for each Related Node not in PL and OS
7: OS.push(Related Node)
8: end for
9: PL.Append(Node)
10: Optimize PL for Node and its related
11: end while
```

Figure 2. Pseudocode to generate all DFFS nodes

2.3. Remove useless duplication detection nodes at DFFS

The discovery of the Zhou and Hansen to form a boundary between frontier and interior of the explicit search graph [14] that used at BFS and then expand on BFFS by Korf and Zhong [7], is done on DFFS at this research. Any node shall have an extra parameter for the count of all its related node. That appears when generating related nodes named related count(RC). When the new corresponding nodes for the popped node generated and added to the OS, if not found in it and the PL; the work will start at PL to extract useless nodes to detect duplications. If the popped node and its new related node found at PL, the RC parameter is reduced by one. At the same time RC for popped node reduced by one. If RC equal zero for the popped node or its related nodes, that found at the previous list, then they can be removed from the previous list and added to the final list.

Figure 3 shows the popped node E, if all the related node (D), (B), (H), and (F) are in the PL, then RC will equal 0. So it can be moved to NL safely because no node at OS will reach to it directly and all its related (D), (B), (H), and (F) found at the PL, that will not found or added to OS. The algorithm-4 is the pseudocode to optimize PL for the popped node and its related as shown in Figure 4.

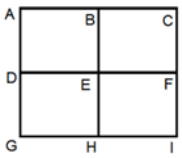

Figure 3. popped node E

```
1: #Remove Useless Nodes from Previous list
2: for each Node at Generate Related Nodes
3. if Node in PL
4: Node RC = Node RC - 1
5: Popped node RC = popped node RC - 1
6: if Node RC = 0
7: NL.add(Node)
8: PL.remove(Node)
9. end if
10. end if
11: end for
12: if popped node RC != 0
13: PL.Append(popped node)
14: Else
15: NL.add(popped node)
16. end if
```

Figure 4. pseudocode to optimize PL for the popped node and its related

3. RESULTS AND DISCUSSION

3.1. Depth-first search path

The node that reaches from two paths at slide tile puzzle appears at level 7 for the first time; then it appears many times. When all nodes at the domain for size 3x3 checked, it found that all nodes generate scale from 12 nodes around, This fact leads to the way that depth-first search passes through the domain in a non-organized spiral way which cut by local maxima and domain edges. Figure 5 displays Manhattan distance that is similar to slide tile puzzle as they are cycle graph with different polygon which is four scales for Manhatten distance and 12 scales for slide tile puzzles, so it is more straightforward. The local maxima neglected to show the step of extract the domain in a spiral way. The priority to choose new node are left, down, right and at last up.

The steps start from node A. Nodes B, D, F, and H pushed to stack. Then node H popped from the stack for the next step, the nodes I, G, and W pushed to stack. Then W popped from the stack for the next step and so on. The Table 3 shows the serial number of visit node, stack size and pushed nodes. It is noted that the path moves to one of the edges, then starts rotating around the domain from left to right. Then it tried to move inside the subdomain again. These steps repeated many times. The local maxima and the number of generated nodes will complicate the moving path for slide tile puzzles. The maximum memory use for the sample is nine nodes.

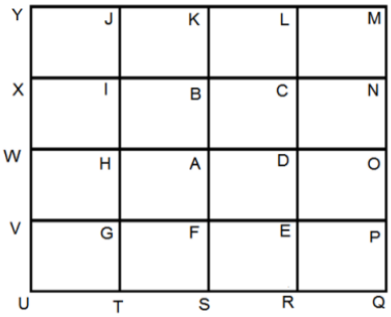

Figure 5. Manhattan distance

Table 3. Visit node serial and stack node

Serial	Node	Stack Status	Stack Size
1	A	B, D, F, H	4
2	H	B, D, F, I, G, W	6
3	W	B, D, F, I, G, X, V	7
4	V	B, D, F, I, G, X, U	7
5	U	B, D, F, I, G, X, T	7
6	T	B, D, F, I, G, X, S	7
7	S	B, D, F, I, G, X, R	7
8	R	B, D, F, I, G, X, E, Q	8
9	Q	B, D, F, I, G, X, E, P	8
10	P	B, D, F, I, G, X, E, O	8
11	O	B, D, F, I, G, X, E, N	8
12	N	B, D, F, I, G, X, E, M, C	9
13	C	B, D, F, I, G, X, E, M, L	9
14	L	B, D, F, I, G, X, E, M, K	9
15	K	B, D, F, I, G, X, E, M, J	9
16	J	B, D, F, I, G, X, E, M, Y	9
17	Y	B, D, F, I, G, X, E, M	8
18	M	B, D, F, I, G, X, E	7
19	E	B, D, F, I, G, X	6
20	X	B, D, F, I, G	5
21	G	B, D, F, I	4
22	I	B, D, F	3
23	F	B, D	2
24	D	B	1
25	B		0

3.2. The case study nodes count and levels

If the domain calculated by permutation and remove the unreached node from the beginning node for the case study, it will be the factorial of the node size divide by 2 [20]. The domain size and level for all the case study when the empty space appears at all allowed locations displayed at Table 4. It's noted that domain size will not be affected by changing the location of empty space but it will affect the domain levels as in some case decrease the maximum level by one especially when empty space at location with extra branch factor; also count of nodes at maximum level is change for each case study as the empty space location is change and maximum count appear when the empty space at location with four available move locations.

Table 4. DFS & BFS sizes and levels

Slide tile puzzle	Domain levels	Domain count	Max. level nodes count
Domain 2X4	36-37	20,160	1-4
Domain 3X3	31-32	181,440	2-148
Domain 2X5	55-56	1,814,400	1-12

The order of all nodes generated on levels for slide tile puzzle 2*4 on BFS as displayed in Figure 6 increase by time as CL size increase, until reach to the maximum size. Then start reduced until reach the maximum level. This case appears for all slide tile puzzles chosen for the case study in this paper. The DFS generation levels order display in Figure 7 is similar for various case studies as DFS reached quickly to one global or local maximum nodes and then return back. So some deep nodes in the domain reached quickly and, others may be reached at the end of the search. DFS passing through the domain in the non-organized spiral way and when reaching local maxima or one of the global maxima return to the low level, then moving up if the node not found at PL until finish the domain or find the solution.

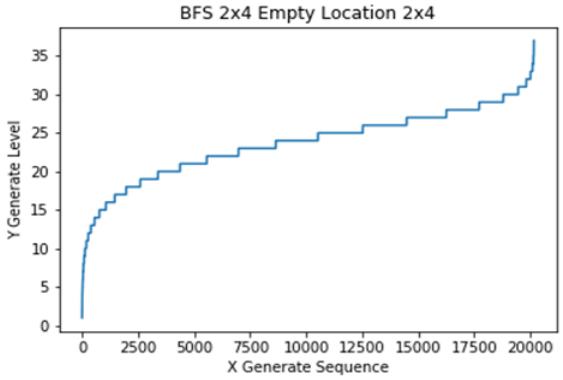

Figure 6. BFS generation levels order

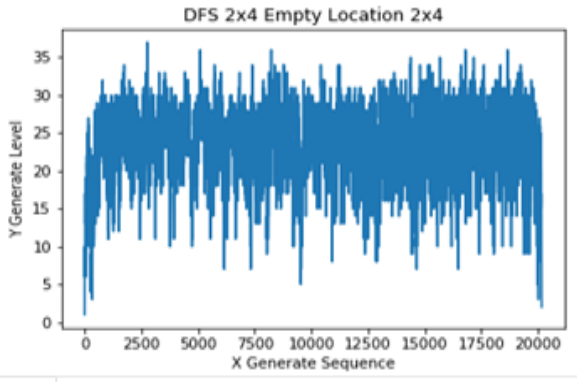

Figure 7. DFS generation levels order

3.3. The related count for each node

Each slide tile puzzle in the case study has tiles in different locations that have 2,3, and 4 related nodes allow for movement. When the generation of all the nodes and count nodes with 2, 3 and 4 nodes, then the results appear at Table 5.

Table 5. Related nodescounts

Slide tile puzzle	Two related nodes	Three related nodes	Four related nodes
Domain 2X4	10,080	10,080	
Domain 3X3	80,640	80,460	20,160
Domain 2X5	725,760	1,088,640	

It appears that values calculated as In formula 1, Where C is related nodes counts, (A) is all domain counts, (T) is all location at slide tile puzzle and N is the count of locations with specifically related nodes.

$$C=(A/T)*N \quad (1)$$

3.4. The edge of slide tile domain

When taking any path from the start node to all last level nodes and check the domain edges to the right and left from the path, the results will appear as in Table 6. It found that the domain edge is more than half of the maximum level in the normal breadth-first search, and the maximum level count ratio compare to all domain nodes decrease as the domain increase. It is 13.3 % for 3*3, 10.6 % in 2*4, and 9% in 2*5.

Table 6. Domain edge

Slide tile puzzle	BFS edge level range	Max. level count range	Number of paths
Domain 2X4	26-28	2,135-2,185	20
Domain 3X3	26-28	24,095-25,956	164
Domain 2X5	40-41	163,345-170,499	36

3.5. The maximum open counts and levels

The open nodes at BFS as the previous scenario will be the size of pending CL nodes plus the FL at any time, and the maximum level will be the future level. The max open nodes at DFS will be the size of OS, and the maximum level will be the popped node from the OS. The Table 7 display the domain maximum open nodes range and maximum level range for DFS and BFS when the Empty space location at all allow places; Figure 8 displays the maximum open count for BFS and DFS as it appears in the table. When the empty space location change then maximum open nodes in DFS and BFS levels change. The clear fact that appears at the table is the open nodes at BFS for all the study cases are less than the open node at DFS at worst cases for each of them. The maximum open nodes at DFS increase compare to BFS also increase when the domain size increase. The maximum open nodes level appears at a different location for the BFS and DFS.

Table 7. DFS and BFS max. open count and level

Slide tile puzzle	BFS Max. Open		DFS Max. Open	
	Count Range	Levels	Count Ranges	Levels
Domain 2X4	2,011-2,063	25-26	4,391-4,440	22-25
Domain 3X3	24,049-25,134	23-25	42,682-42,923	17-29
Domain 2X5	133,477-135,599	36-38	375,210-375,728	26-37

3.6. The maximum close counts and levels

The close nodes at BFS as the previous scenario will be the size of used nodes at CL plus PL at any time, and the maximum level will be the current level. The max open nodes at DFS will be the size of PL, and the maximum level will be the popped node from the OS. Table 8 displays the domain maximum close nodes ranges and the maximum level ranges for DFS and BFS when the Empty space location at all places allow at the case study. Figure 9 displays the maximum close count for BFS and DFS.

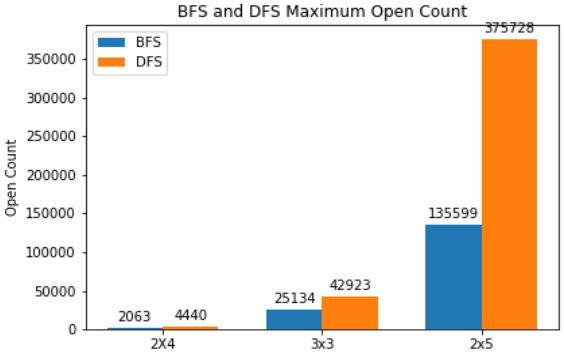

Figure 8. Maximum open count for BFS and DFS

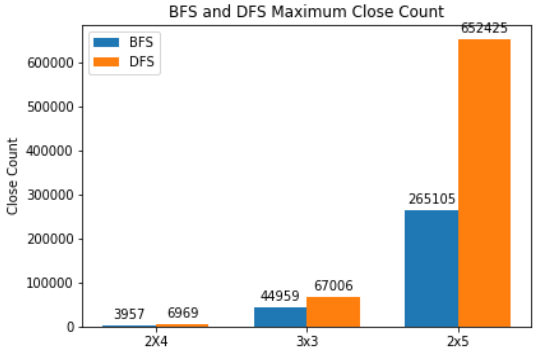

Figure 9. Maximum close count for BFS and DFS

Table 8. DFS and BFS max. close counts and levels

Slide tile puzzle	BFS Max. Close		DFS Max. Close	
	Count Range	Levels	Count Ranges	Levels
Domain 2X4	3,875-3,957	26-27	6,917-6,969	10-21
Domain 3X3	44,271-44,959	25-26	66,774-67,006	21-28
Domain 2X5	262,640-265,105	38	650,907-652,425	23-42

As its appearance in the table when the empty space location change, the maximum close nodes at DFS and BFS levels change. The clear fact that appears in the table is the closed nodes at BFS for all study cases are less than the close node of DFS. The maximum close nodes increase the percentage at DFS compare to BFS when the domain size increase. The maximum close nodes level appears at different location for the BFS and DFS.

4. CONCLUSION

This research concentrated on the memory usage in DFS and BFS strategies with frontier boundaries DFFS and BFFS. It clarified that presented count of nodes saved in memory, in the worst case for DFFS at open list and previous list, consumed more memory than BFFS. In the open list, it consumed 38.478%, and for the close list it consumed 43.229% in the case study. This allows expanding more nodes in the BFFS. The DFFS has the advantage to pass through the different levels in the unsorted path compare to BFFS, especially when the goal falls in deep location. The BFFS has an easy way to sort nodes and path optimality than DFFS, which give it an extra advantage. DFFS pass in the domain in a non-organized spiral path. It will be better than BFFS when the local maxima is reduced or eliminated in the domain as in acycle graph because it saves expanded nodes for each level in the path. This model finds a fact of domain shape change for the number of levels, and the number of nodes in each level when the empty space appears at a different location for the start node. The recommendations for future work to apply these results in depth-first search with iterative model and check expected results on other domains.

ACKNOWLEDGEMENT

This research is funded by the Deanship of Research in Zarqad University/Jordan

REFERENCES

[1] I. Pohl, "Practical and theoretical considerations in heuristic search algorithms," in W. Elcock, D. Michie (Eds.), *Machine Intelligence*, Vol. 8, Ellis Horwood, Chichester, pp. 55-72, 1977, Available: https://www.researchgate.net/publication/243656112_Practical_and_theoretical_considerations_in_heuristic_search_algorithms.

[2] R. E. Korf, M. Reid, and S. Edelkamp, "Time complexity of iterative-deepening-A*," *Artificial Intelligence*, vol. 129, no. 1–2, pp. 199–218, 2001, doi: 10.1016/S0004-3702(01)00094-7.

[3] R. E. Korf, "Depth-first iterative-deepening," Artif*icial Intelligence*, vol. 27, no. 1, pp. 97–109, 1985, doi: 10.1016/0004-3702(85)90084-0.

[4] W. Zhang and R. E. Korf, "Depth-first versus best-first search: New results," in *Proceedings* 11th AAAI, pp. 769-775, 1993, Available: https://www.aaai.org/Papers/AAAI/1993/AAAI93-115.pdf.

[5] G. Wang and R. Li, "DSolving: a novel and efficient intelligent algorithm for large-scale sliding puzzles," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 29, no. 4, pp. 809–822, 2017, doi: 10.1080/0952813X.2016.1259270.

[6] R. Zhou and E. A. Hansen, "Memory-bounded A* graph search," in *15th Int. Florida Artificial Intelligence Research Soc. Conf*., 2002, pp. 203–209, 2002, Available: https://www.aaai.org/Papers/FLAIRS/2002/FLAIRS02-041.pdf.

[7] R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald, "Frontier search," in *Journal of the ACM*, vol. 52, no. 5, pp. 715–748, 2005, doi: 10.1145/1089023.1089024.

[8] R.E. Korf, W. Zhang, "Divide-and-conquer frontier search applied to optimal sequence alignment," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 910-916, 2000, Available: https://www.aaai.org/Papers/AAAI/2000/AAAI00-140.pdf.

[9] A. Auer and H. Kaindl, "A case study of revisiting best-first vs. depth-first search," in *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 141–145, 2004, Available: https://dl.acm.org/doi/10.5555/3000001.3000032.

[10] D. E. Martin and G. Estrin, "Models of computational systems-cyclic to acyclic graph transformations," *IEEE trans. electron. comput*., vol. EC-16, no. 1, pp. 70–79, 1967, doi: 10.1109/PGEC.1967.264607.

[11] R. E. Korf, "Best-first frontier search with delayed duplicate detection," In *Proceedings of the National Conference on Artificial Intelligence (AAAI-2004)*, pp. 650-657, 2004, Available: https://www.aaai.org/Papers/AAAI/2004/AAAI04-103.pdf.

[12] R. Zhou, E. A. Hansen, "Structured duplicate detection in external-memory graph search," in: *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, San Jose, CA, 2004, pp. 683-688, 2004, Available: https://www.aaai.org/Papers/AAAI/2004/AAAI04-108.pdf.

[13] Z. Zhang, J. X. Yu, L. Qin, and Z. Shang, "Divide & conquer: I/O efficient depth-first search," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, doi: 10.1145/2723372.2723740.

[14] R. Zhou and E. A. Hansen, "Sparse-memory graph search," in *Proceedings of the 18th international joint conference on Artificial intelligence*, 2003, pp. 1259–1266, doi:10.5555/1630659.1630839.

[15] L. Otten and R. Dechter, "Anytime AND/OR depth-first search for combinatorial optimization," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2014, pp. 933–937, doi:10.1007/978-3-319-10428-7_70.

[16] S. S. Skiena, The algorithm design manual, 2nd ed. London, England: Springer London, 2009, p 480.

[17] R. E. Korf, "Linear-time disk-based implicit graph search," *J. ACM*, vol. 55, no. 6, pp. 1–40, 2008, doi:10.1145/1455248.1455250.

[18] R. E. Korf. "Artificial intelligence search algorithms," in Algorithms Theory Computation Handbook, CRC Press, 1999, Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.1135&rep=rep1&type=pdf.

[19] K. B. Irani and S. I. Yoo, "A methodology for solving problems: problem modeling and heuristic generation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 5, pp. 676-686, Sept. 1988, doi:10.1109/34.6776.
[20] Slocum. J and Sonneveld. D, "The 15 Puzzle," *Slocum Puzzle Foundation*, 2006, ISBN-13: 9781890980153.
[21] D. Ratner and M. Warmuth, "The (n2−1)-puzzle and related relocation problems," *Journal of Symbolic Computation*, vol. 10, no. 2, pp. 111–137, 1990, doi:10.1016/S0747-7171(08)80001-6.
[22] R. E. Korf and L. A. Taylor, "Finding optimal solutions to the twenty-four puzzle," in Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2, 1996, pp. 1202–1207, Available: https://dl.acm.org/doi/10.5555/1864519.1864565.
[23] R. E. Korf and A. Felner, "Disjoint pattern database heuristics," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 9–22, 2002, doi:10.1016/S0004-3702(01)00092-3.
[24] W. W. Johnson, Notes on the '15' Puzzle," Amer. J. Math., vol. 2, no. 4, pp. 397-399, 1879, doi:10.2307/2369492.
[25] W. E. Story, "Notes on the '15' Puzzle," Amer. J. Math., vol. 2, no. 4, pp. 397-404, 1879, doi:10.2307/2369492.
[26] R. E. Korf and P. Schultze, "Large-scale parallel breadth-first search," in Proceedings of the 20th national conference on Artificial intelligence - Volume 3, 2005, pp. 1380–1385, Available: https://dl.acm.org/doi/10.5555/1619499.1619555.

BIOGRAPHIES OF AUTHORS

Dr. Mohammed N. Al-Refai Chairman of Software Engineering Department in Zarqa University Ph.D in Computer Science (Distributed Systems) Amman Arab University for Graduate Studies 2007 Master Degree in computer science, al-albayt Univercity, almafraq, Jordan, 1999- 2002. BS in Computer science, Mu'ta University, Alkarak, Jordan, 1988-92.

Dr. Zeyad M. Jamhawi Ph.D in Computer Information Systems (Artificial Intelligence), Omdurman Islamic University, Sudan 2013-2016. Master degree in Computer Information System, Arab academy for banking and finance, Amman, Jordan 2009-2012. Professional Diploma specialization in E-Government. Jordanian University, Jordan 2007-2008. BS in Computer science, Mu'ta University, Alkarak, Jordan, 1988-92.