

Improving the Efficiency of Depth-First Search by Cycle Elimination

John F. Dillenburg and Peter C. Nelson*

Department of Electrical Engineering
and Computer Science (M/C 154)
University of Illinois
Chicago, IL 60680 U.S.A.
E-mail: nelson1@uicbert.eecs.uic.edu

Abstract

An experimental evaluation of a technique for improving the efficiency of depth-first search on graphs is presented. The technique, referred to as cycle checking, prevents the generation of duplicate nodes in the depth-first search of a graph by comparing each newly generated node to the nodes already on the search path. Although cycle checking can be applied to any depth-first type search, this paper focuses on its effect with the heuristic depth-first iterative deepening algorithm IDA*. Two types of cycle checking are compared, full cycle checking and parent cycle checking. Full cycle checking compares a new node to all nodes on the search path. Parent cycle checking merely compares a new node to the parent of the node being expanded. Both types of cycle checking are shown to improve the efficiency of IDA* search in a grid search problem and a route planning problem. Simple guidelines are presented showing which type of cycle checking should be used on a given problem.

Keywords: algorithms, heuristic search

*Please send all correspondence to the second author

1. Introduction

The IDA* algorithm was introduced by Richard Korf in 1985 as an optimal heuristic search algorithm. In addition to being a relatively simple algorithm, IDA* is provably optimal in terms of memory usage and time complexity over all best-first searches on a tree [2][3]. Note that the optimality of IDA* applies only to trees and not to graphs in general. There are two major shortcomings of using IDA* on graphs. First, if IDA* is used on directed acyclic graphs with an edge branching factor greater than the node branching factor, then it will waste time exploring alternative paths to the same node. The second shortcoming deals with using IDA* on graphs in which cycles are present. Addressing this second shortcoming will be the focus of this paper.

To see why cycle checking is important, one must first understand how the IDA* algorithm works. The IDA* algorithm is based on the concept of depth-first iterative deepening (DFID). A brute force DFID search uses successive depth-first searches to find the goal. A normal depth-first search stops only when there are no more nodes left to expand. The depth-first search used by DFID stops when a depth limit has been reached. This is necessary because it is not possible to mark every node which has been expanded when the search space is large. Successive depth-first searches are performed to greater and greater depths. After each depth-first search, the depth limit is increased by one. Initially, the depth limit is one. A DFID search terminates when the goal is encountered.

The IDA* algorithm improves upon brute force DFID by using a heuristic, $h(n)$. Successive depth-first searches are performed with the A* cost function $g(n) + h(n)$ used to determine when to stop searching along the current path. Search continues along a path to the next node n only if $g(n) + h(n) \leq cutoff$. Unlike brute force DFID, where *cutoff* is always incremented by one, the value of *cutoff* for the next iteration of IDA* is determined from the minimum of all cost values that exceeded *cutoff* in the current iteration. Initially, *cutoff* is the heuristic estimate of the cost from the start state to the goal state. Note that the path from the start state to the current state must be maintained as part of the depth-first search in order to obtain the path to the goal when the algorithm terminates.

Since IDA* is based on depth-first search, it will blindly wind its way through a cycle until its depth limit is reached. The algorithm can be modified to check for these cycles fairly easily [4, p.39]. Since the current path back to the start must be maintained, a test can be put in place which compares each node being generated to some or all of the nodes on the current path. Two types of cycle checking will be considered here, *parent* and *full* cycle checking. Parent cycle checking discovers cycles of length two by simply comparing a newly generated node to the parent of node being expanded. Full cycle checking discovers cycles of any length by comparing a newly generated node to every node on the current path. Empirical evidence suggests that this $O(d)$ brute force search is faster than using a $O(1)$ hash table lookup because the overhead of adding and deleting nodes from the hash table is quite large and usually negates any benefit obtained from faster cycle detection.

Since cycle checking IDA* cuts off more branches of the search tree, it will have a lower effective branching factor than regular IDA*. A lower branching factor means less time spent doing node expansions. This

is balanced against the extra time needed to check for cycles. When comparing two search spaces with different branching factors, IDA* will be asymptotically faster on the search space with the lower branching factor, even when the extra time for cycle checking is taken into account. The next section will provide experimental evidence which shows that full cycle checking provides a significant reduction in node expansions and search time for two types of search problems.

2. Test Results

Tests were run on a four-connected grid search problem, the 15-puzzle problem [4], and a transportation route planning problem in order to determine the effectiveness of cycle checking. All tests were run on Sun SparcStation IPX workstations. It will be shown that parent cycle checking improves the search time of all problems tested. Full cycle checking is shown to improve the search time of the grid search problem and the route planning problem.

2.1 The Grid Search Problem

The objective of a grid search problem is to find a path from one point on a grid to another point on a grid, with obstacles blocking some grid positions. The results for the grid search problem are summarized in figures 1 and 2. To make things interesting, the tests run in these figures had a single obstacle placed perpendicular to a line connecting the start and the goal. The distance between the start and the goal grid positions was varied and one test was run for each start/goal position. The air-distance heuristic was used in all tests for the grid search problem. Shown in figure 1 is the number of node expansions as a function of the solution length. By fitting an exponential function to each curve, the branching factor was determined to drop from 2.15 with no cycle checking to 1.97 with parent checking and 1.92 with full checking.

Reducing the number of node expansions is only meaningful if there is also a corresponding decrease in the search time. Figure 2 shows the amount of CPU time needed to solve each of the grid search problems versus path length. The amount of time needed correlates very closely with the number of node expansions. The small added cost of cycle checking is more than made up for by the reduction in node expansions. Full cycle checking IDA* is 33% faster than parent cycle checking IDA* when the path length is 28. This graph shows that for the grid search problem, it is worthwhile to use full cycle checking.

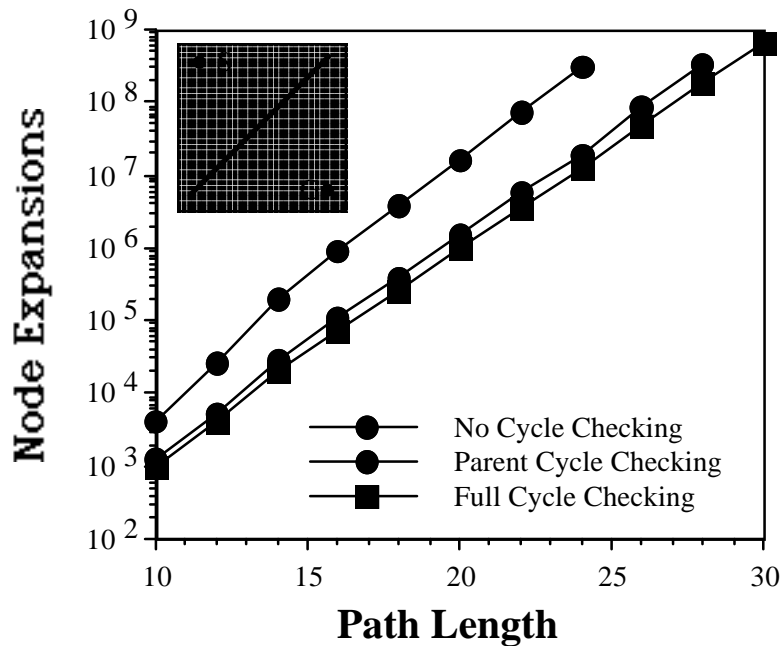


Figure 1. Cycle checking results for four connected grid

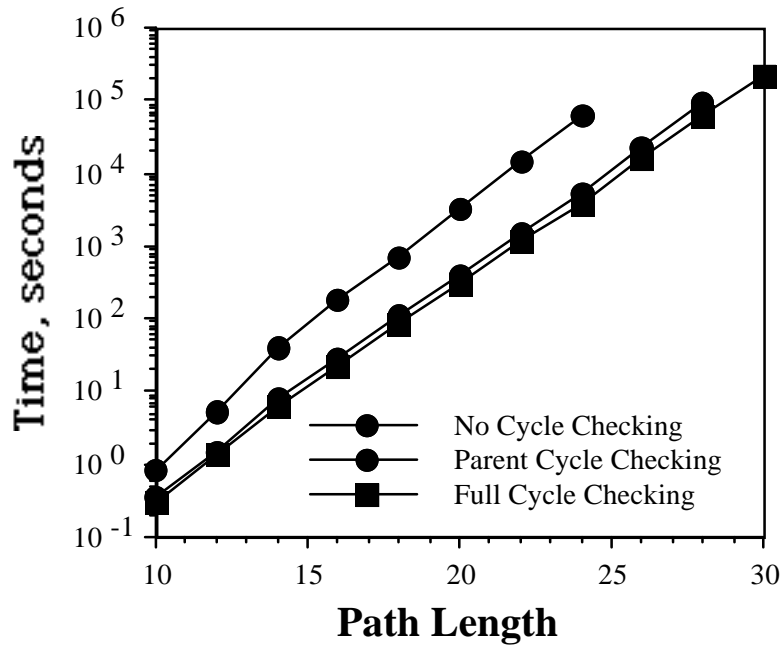


Figure 2. Cycle checking results for four connected grid.

2.2 The 15-Puzzle Problem

Unlike the tests for the grid search problem, the tests for the 15-puzzle problem involved solving twenty random puzzles for each path length rather than solving just one case for each path length. Each problem was solved without cycle checking, with parent cycle checking IDA*, and with full cycle checking IDA*. The Manhattan distance was used as the heuristic. The test results are summarized in figure 3. Branching factors were computed by fitting an exponential curve through the node expansions versus path length data. This is basically equivalent to computing the mean value of the d^{th} root of the number of nodes expanded. The branching factor dropped from 1.9 without cycle checking to 1.44 with cycle checking yielding a corresponding exponential improvement in search time. The graph shows that cycle checking IDA* is about 800 times faster than non-cycle checking IDA* at path length 30. Parent cycle checking IDA* was about 13% faster than the full cycle checking IDA* at the same path length.

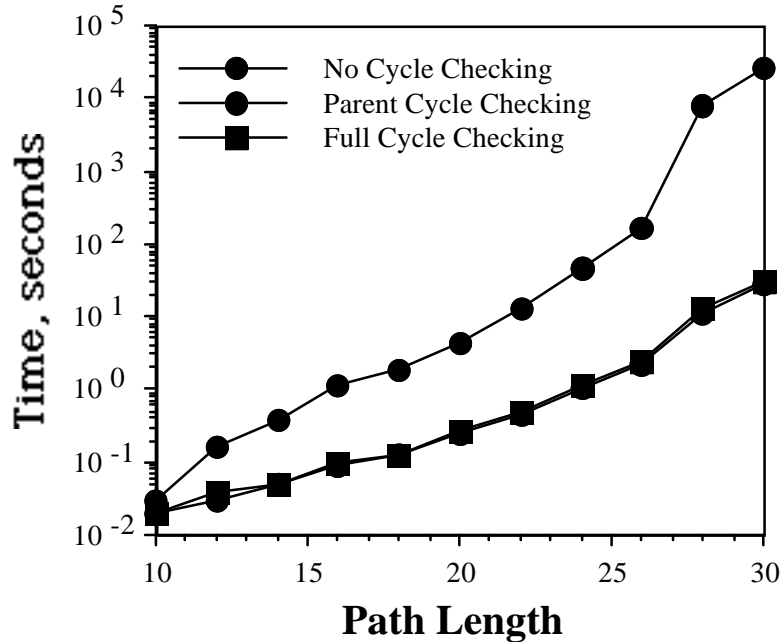


Figure 3. 15-puzzle test results

A second test was run to see if full cycle checking would provide more of a benefit on puzzle instances with longer path lengths. The second test involved solving twenty random puzzle instances of arbitrary path lengths. Random 15-puzzle instances are intractable with non-cycle checking IDA* and hence no puzzles were solved without cycle checking. Once again, the Manhattan distance was used as the heuristic. The results are summarized in Table 1. The table shows that even though full cycle checking IDA* expands less nodes, it is about 17% slower than parent cycle checking IDA*.

Number of Tests = 20 Mean Path Length = 50.25

		Parent Cycle Checking	Full Cycle Checking	$\frac{\text{Parent} - \text{Full}}{\text{Parent}} \times 100$
Nodes Expansions	Mean	92,230,862	92,155,932	0.05
	Deviation	166,398,011	166,201,336	0.06
Elapsed Time, sec	Mean	48,686.0	60,326.2	-16.6
	Deviation	95,248.5	116,412.2	15.8

Table 1. Summary of test results for the 15-puzzle problem.

The 15-puzzle experiments show what happens when full cycle checking is used on a simple search space with a strong heuristic. The search space for the 15-puzzle problem contains cycles of length 2 with the next larger length cycles being of length 12 and then length 16. Parent cycle checking eliminates the cycles of length 2, and the Manhattan heuristic eliminates most of the longer length cycles. For these reasons, full cycle checking IDA* should not be used on this type of problem.

2.3 The Route Planning Problem

The tests for the route planning problem involved finding the fastest travel-time path between two points on a Chicago area map. The map contained over 16,000 miles of roadway within a 60 mile radius of Chicago. The map did not contain any residential streets. The map was represented as a directed graph with 8,847 nodes and 27,836 edges. The average node branching factor was 3.1. The heuristic used was the air-distance divided by 55 miles per hour. This heuristic is admissible since the maximum legal speed limit in the map area is 55 miles per hour. The cost of traversing a road segment was set equal to the length of the segment divided by the speed limit on the segment. This assumes very light traffic conditions and a total absence of any congestion.

Data was collected on a set of 100 pseudo-randomly selected (*start, destination*) pairs. The *start* and *destination* points were restricted to be within 2 to 4 miles of each other in order to reduce the variance in the test results. Parent cycle checking and full cycle checking IDA* were used to solve each of the 100 test cases. The test results are summarized in table 2. The table shows that full cycle checking IDA* expanded 38% fewer nodes than parent cycle checking IDA* and was 31% faster. This is an example of the improvement in efficiency that can be obtained by using full cycle checking in a complex search space with a relatively weak heuristic. Tests run with longer path lengths indicated an even further improvement in the efficiency of full versus parent cycle checking.

Number of Tests = 100

		Parent Cycle Checking	Full Cycle Checking	
Nodes Expansions	Mean	16,263,766	3,695,788	38.8
	Deviation	48,707,689	11,589,785	33.3
Elapsed Time, sec	Mean	5,264	1,481	31.4
	Deviation	15,799	4,678	38.0

Table 2. Summary of test results for the route planning problem.

The effectiveness of full cycle checking was compared to the effectiveness of parent cycle checking by the analyzing the two hypotheses listed in table 3. The analysis was based on the unbiased, non-trivial test cases for the route planning problem domain. The significance of the hypotheses were evaluated using the difference-of-means test [1]. The results of the hypothesis tests are shown in table 4. The quantity δ is the standard deviation and Z represents the standardized variable. The results show that, for a two-tailed test, the difference between the average performance of parent and full cycle checking IDA* is significant at the 0.02 level for both hypotheses. Therefore, we can assert with 98% confidence that full cycle checking IDA* significantly outperforms parent cycle checking IDA* in the route planning domain.

Hyp. number	Hypothesis
1	Full cycle checking IDA* expands fewer nodes than parent cycle checking IDA* in the route planning domain
2	Full cycle checking IDA* is faster than parent cycle checking IDA* in the route planning domain

Table 3. Hypotheses

Hyp. number	δ	Z	Test	Significance level	Confidence level
1	5006757	2.51	two-tailed	0.006	99.4%
2	1647.7	2.30	two-tailed	0.011	98.9%

Table 4. Results of tests of hypotheses

3. Conclusion

Two weaknesses of the IDA* algorithm were discussed. Both weaknesses stem from the fact that IDA* is basically a series of depth-first searches with little knowledge of previous computations. The weakness pointed out here is that each depth-first search will blindly follow a cycle if an unmodified version of the IDA* algorithm is used. A modification to the IDA* algorithm can eliminate these cycles by checking each newly generated node to determine if the node is already present on the current path. This simple modification to the IDA* algorithm was shown to significantly improve the performance of IDA* when a complex state space is searched using a weak heuristic.

Two types of cycle checking were presented. Deciding which type of cycle checking to use depends mainly on two factors: heuristic pruning power and search space complexity. The 15-puzzle problem showed that longer length cycles will be eliminated by a strong heuristic and hence obviate the necessity of doing full cycle checking. Define a simple search space as a search space with few cycles of length greater than two. The 15-puzzle search space is an example of a simple search space. Full cycle checking should not be used in a simple search space since the cost of checking for cycles of length greater than two will negate any benefit obtained by reducing the number of node expansions. Table 5 provides a guide to the type of cycle checking which should be used on a given problem. The entries marked with a "?" indicate that experimentation should be used to determine which type of cycle checking works best.

	Simple Search Space	Complex Search Space
Weak Heuristic	?	Full Cycle Checking
Strong Heuristic	Parent Cycle Checking	?

Table 5. Guide to cycle checking type to use

There are other methods of reducing the complexity of IDA* that are notable. Sen and Bagchi [5] present an algorithm called MREC which uses a fixed amount of memory to store nodes. This not only eliminates cycles, but it also prevents the same node from being generated more than once. When memory runs out, however, MREC reverts back into an IDA* type search and hence the cycle checking modifications presented here can also be applied to MREC. Also note that cycle checking IDA*, unlike MREC, does not require any additional memory usage beyond what IDA* would normally use. Taylor and Korf [6] also present a novel method of eliminating cycles by using a finite state automaton to generate moves. The automaton is constructed by performing a breadth-first search prior to the use of IDA*. The drawback to this approach is that it is only useful for problems which are described implicitly, like the 15-puzzle problem or the Towers of Hanoi problem. The full cycle checking approach presented here, however, can potentially improve the node expansion efficiency of IDA* in any problem domain.

Acknowledgement

We would like to thank the anonymous referees for their contributions to this paper.

References

- [1] S. Dowdy and S. Wearden, *Statistics for Research*, New York, NY, John Wiley & Sons, 1983.
- [2] R.E. Korf, Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, *Artificial Intelligence* 27 (1985) 97-109.
- [3] A. Mahanti, S. Ghosh, D.S. Nau, A.K. Pal and L. Kanal, Performance of IDA* on Trees and Graphs, *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)* 539-544.
- [4] Pearl, *Heuristics*, Reading, MA, Addison-Wesley, 1984.
- [5] A. Sen and A. Bagchi, Fast Recursive Formulations for Best-First Search that Allow Controlled use of Memory, *Proceedings of the 11th International Joint Conference of Artificial Intelligence (IJCAI-89)* 297-302.
- [6] L.A. Taylor and R.E. Korf, Pruning Duplicate Nodes in Depth-First Search, *UCLA Computer Science Technical Report*, University of California at Los Angeles, 1992.