Solving the 8-Puzzle Problem Using Genetic Programming

Kevin Igwe
School of Mathematics, Statistics &Computer Science
University of KwaZulu-Natal
Pietermaritzburg Campus
+27 33 2605644
kelvlilsky@yahoo.com

Nelishia Pillay
School of Mathematics, Statistics &Computer Science
University of KwaZulu-Natal
Pietermaritzburg Campus
+27 33 2605644
pillayn32@ukzn.ac.za

Christopher Rae
School of Mathematics, Statistics &Computer Science
University of KwaZulu-Natal
Pietermaritzburg Campus
+27 33 2605644
stingrae789@gmail.com

ABSTRACT

The 8-puzzle problem is a classic artificial intelligence problem which has been well-researched. The research in this domain has focused on evaluating traditional search methods such as the breadth-first search and the A* algorithm and deriving and testing various heuristics for use with informed searches to solve the 8-puzzle problem. The study presented in this paper evaluates a machine learning technique, namely genetic programming, as means of solving the 8-puzzle problem. The genetic programming algorithm uses the grow method to create an initial population which is iteratively refined using tournament selection to choose parents which the reproduction, mutation and crossover operators are applied to, thereby producing successive generations. The edit operator has been used to exert parsimony pressure in order to reduce the size of solution trees and hence the number of moves to solve a problem instance. The genetic programming system was successfully applied to 20 problem instances of differing difficulty, producing solutions to all 20 problems. Furthermore, for a majority of the problems the solutions produced solve the problem instance using the known minimum number of moves.

Categories and Subject Descriptors

I.2. [**Computing Methodologies**]: Artificial Intelligence.

General Terms

Algorithms, Performance, Experimentation.

Keywords

Genetic programming, 8-puzzle problem.

1. INTRODUCTION

The 8-puzzle problem is commonly used as a benchmark problem to test the performance of heuristic search algorithms [1]. This is an NP hard problem [2] and is defined in terms of an initial state and final state.

SAICSIT '13, October 07 - 09 2013, East London, South Africa
Copyright 2013 ACM 978-1-4503-2112-9/13/10…$15.00.
http://dx.doi.org/10.1145/2513456.2513492

Solving the problem involves determining the moves needed to get from the initial state to the goal state. An example is illustrated in Figure 1. The puzzle itself is a board with numbered tiles and a space (indicated by a blank in Figure 1). A single tile can be moved into a space at a time. The moves can also be viewed as moving the space rather than the tiles. The space can be moved left, right, up and down depending on its location on the board. A further challenge in some versions of the problem is to get from the initial state to the goal state using a minimum number of moves.

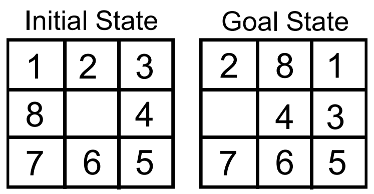

Figure 1. 8-puzzle problem

Majority of the research in this domain has examined the derivation of different heuristics for use with informed searches for solving the 8-puzzle problem, with the A* algorithm proving to be the most effective [3, 4]. These include the number of tiles out of place and the number of tile reversals needed [4]. The Manhattan distance was found to be the most effective. In the context of the 8-puzzle problem this heuristic is the sum of the distances of each tile from its goal position. The Manhattan distance of the initial state in Figure 1 is 7. All three heuristics are admissible.

There has not been much work evaluating population-based or machine learning techniques, like genetic programming, for solving this problem. Bhasin et al. [3] and Shaban et al. [5] implement a genetic algorithm to solve the 8-puzzle problem. Given an initial state the GA produces the goal state.

Genetic programming (GP) is a variation of genetic algorithms which searches a program space instead of a solution space to identify an "optimal program" to solve the problem [6]. Each element of the population is a program, which when executed will produce a solution to the problem. A parse tree is usually used to represent each program. In the context of the 8-puzzle problem this means that genetic programming will produce a sequence of moves that produce a solution rather than a solution board as in the case of genetic algorithms. Previous studies using genetic programming to solve the 8-puzzle problem have been unsuccessful. Barnes et al. [7] use genetic programming to induce an algorithm that will produce a solution for a particular problem instance. The GP system was tested on 10 problem instances and was not able to evolve a solution for any of the problems. Marshall [8] uses genetic programming to evolve a heuristic that can be used with an informed search to solve the problem. The approach was tested on an instance of the 8-puzzle problem, which the known heuristics in the field were used to solve, and was not able to evolve a heuristic that produced a solution to the problem instance.

It is evident from the literature surveyed that there has not been much research into using GP to solve the 8-puzzle problem and the two studies that have been conducted were not successful. This paper revisits the application of GP to this domain. The approach taken is similar to that implemented by Barnes et al. [7] and uses GP to evolve a program to solve a problem instance. The GP algorithm implemented was tested on 20 8-puzzle problems of varying difficulty. The performance of genetic programming was also compared to the traditional search approaches used to solve this problem, namely depth-first search, breadth-first search, and the A* algorithm. GP was able to produce solutions to all 20 problems including those that some of the traditional searches were unable to find solutions to. Furthermore, for a majority of the problems, GP evolved programs which produced a solution in the known minimum number of moves.

The following section presents the genetic programming algorithm implemented to solve the problem. The experimental setup used to evaluate the approach is outlined in section 3. Section 4 discusses the performance of genetic programming in solving the 8-puzzle problem and compares it to that of traditional searches.

2. GENETIC PROGRAMMING SYSTEM

A generational genetic programming algorithm was implemented to solve the 8-puzzle. The algorithm begins by creating an initial population which is iteratively refined via the processes of evaluation, selection and regeneration. If a solution is found the algorithm is terminated. If a solution is not found the algorithm continues until a preset number of generations is completed.

Each element of the population is a program representing moves that must be performed to get from the initial state to the goal state. The moves are defined in terms of moving the space. Thus the terminal set contains up (U), down (D), left (L) and right (R). The function set consists of two elements that are used to combine the moves, namely, PROG2 and PROG3 which take two and three arguments respectively. The grow method [7] is used to create each individual. The use of iteration and move sequences were not included in the primitives as these did not appear to work well in the study conducted by Barnes et al. [9]. Figure 2 illustrates an example of an individual and the resulting state when applying the individual to the initial state in Figure 1. Please note that the moves are in terms of the space and not the tiles, e.g. U means moving the space up, which in the initial state in Figure 1 will result in the tile with a 2 being moved down.

The fitness of each program is calculated by comparing the state produced by the program when applied to the initial state, to the goal state. The Manhattan distance is used to calculate the difference in both states. Thus, a lower fitness value is indicative of a fitter individual. The tournament selection method [7] is used to select parents. The standard genetic operators presented in [7], namely, reproduction, crossover and mutation are used for regeneration.

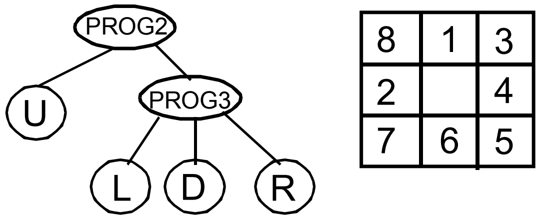

Figure 2. An example of the application of a program

Programs induced by genetic programming often contain redundant code referred to as introns [6]. The amount of redundant code increases over the generations and the exponential growth of introns is known as bloat. In the context of the 8-puzzle problem examples of introns would be PROG2 L R and PROG2 D U. In both these cases the second move will undo the first move. In this domain introns can result in a solution program being evolved with a larger number of moves than that is necessary to find a solution. For example, suppose that the equivalent sequence of moves depicted by a program is LRLRDDURD. This can be reduced to DRD by removing the introns. Thus, in order to prevent the growth of introns, which would result in the solution programs containing a larger number of moves then is necessary to find a solution, the edit operator defined by Koza [7] is applied to each solution program evolved. Two phases of editing is performed sequentially. In both phases each individual is converted into a sequence of moves. In the first phase, the odd-even position editing phase, each odd position is located and a check is performed to determine if it cancels out the move before it. If so, both moves are removed. The same process is followed in the second editing phase, namely the even-odd position editing phase, but each move in an even position is located. Suppose that the sequence of moves corresponding to a program is LRLRDDURD. At the end of the first editing phase this will be reduced to DDURD. This sequence then forms input to the second editing phase which reduces it further to DRD.

3. EXPERIMENTAL SETUP

The genetic programming approach was tested on the 20 8-puzzle problem instances listed in Table 1 which were obtained from the literature and online assignments. These instances differ in difficulty and the Manhattan distance of the initial state is used as a measure of problem difficulty. Table 1 also lists this value for each problem instance. A higher Manhattan distance is indicative of a more difficult problem.

Table 2 lists the values used for the genetic parameters. These values were chosen based on the experiments reported on in [7] and by conducting trial runs. These values are by no means optimal and could possibly be improved. The GP algorithm was implemented in Java. Simulations were run on an Intel Core 3.1 GHz machine with 8192 MB of RAM. Due to the stochastic nature of genetic programming ten runs were performed for each problem [7].

4. RESULTS AND DISCUSSION

This section discusses the performance of genetic programming in solving the 8-puzzle problem. The first section looks at how well GP faired in solving the 20 problems. Section 4.2 compares the performance of GP to traditional searches used to solve the 8-puzzle problem.

Table 1. 8-puzzle problem instances. A-puzzle number, B-initial state, C-goal state, D-Manhattan distance, E- known optimum (minimum number of moves)

A	B	C	D	E
1	123804765	134862705	5	5
2	123804765	281043765	7	9
3	123804765	281463075	10	12
4	134805726	123804765	6	6
5	231708654	123804765	10	14
6	231804765	123804765	4	16
7	123804765	231804765	4	16
8	283104765	123804765	4	4
9	876105234	123804765	16	-
10	867254301	123456780	19	31
11	647850321	123456780	21	31
12	123804765	567408321	24	-
13	806547231	012345678	21	-
14	641302758	012345678	8	14
15	158327064	012345678	12	12
16	328451670	012345678	8	12
17	035428617	012345678	10	10
18	725310648	012345678	7	15
19	412087635	123456780	15	17
20	162573048	123456780	10	10

Table 2. Parameter values

Parameter	Value
Population size	500
Maximum number of generations	50
Initial program depth limit	4
Tournament size	5
Maximum offspring depth	2
Mutation depth	2
Reproduction rate	3%
Crossover rate	40%
Mutation rate	57%

4.1 GP Performance

Genetic programming was able to evolve solution programs using the parameter values in Table 2 for all problem instances except Puzzle 7. A change in the application rates of the reproduction, crossover and mutation operators to 0%, 30% and 70% produced solutions for Puzzle 7. Table 3 lists the success rate, the minimum number of moves in the sequence produced by the solution program evolved, and the minimum and maximum generations by which a solution program was found. The success rate is the percentage of the ten runs producing solutions. Genetic programming was able to evolve programs that produced solutions using the known minimum number of moves.

Table 3. GP performance

Problem	Success Rates	Minimum Number of Moves	Minimum Number of Genera-tions	Maxi-mum Number of Genera-tions
1	100%	5	0	2
2	100%	9	3	7
3	100%	12	5	11
4	100%	6	0	3
5	40%	14	3	38
6	10%	16	23	23
7	20%	16	35	50
8	100%	4	0	0
9	10%	40	49	49
10	100%	31	13	44
11	100%	31	12	35
12	100%	32	14	39
13	90%	31	11	37
14	50%	14	7	38
15	100%	12	3	6
16	100%	12	3	15
17	100%	10	2	5
18	20%	15	19	32
19	90%	17	9	17
20	100%	10	3	5

In those cases where the minimum is unknown the number of moves will be compared to that used by the breadth-first search and A* algorithm in the next section as these algorithms are admissible. For a majority of the problems genetic programming has a 100% or close to 100% success rate (i.e. over the ten runs in each case). For those problem instances with a success rate lower than 50% the problem difficulty is 4, 6 and 11 which is not high. Furthermore, GP was able to produce solutions with high success rates for problems of difficulty levels of 21 and 24. Given this it was suspected that the lower success rates could possibly be attributed to the parameter values used. This was tested and it was found that the GP algorithm is sensitive to the genetic operator application rates for this domain. For example, changing the crossover and mutation application rates to 77% and 20% respectively resulted in an increase in the success rate for Puzzle 9 to 30% and a decrease in the number of moves to 32. These changes to the genetic operator application rates also resulted in a decrease in the minimum number of moves to 30 for Puzzle 12. This will be investigated further as part of future work. All solution programs were found in under a minute. For some problem instances a different solution program was evolved for different seeds.

4.2 Performance Comparison

The performance of genetic programming was compared to that of searches which have generally been used to solve the 8-puzzle, namely, the depth-first search (DFS), breadth-first search (BFS) and the A* algorithm. Table 4 lists the minimum number of moves that are needed to find a solution for each of the searches and GP.

Table 4. Performance comparison

Prob -lem	**DFS**	**BFS**	**A* Algorithm**	**GP**
1	30	5	5	5
2	-	9	9	9
3		12	12	12
4	7	6	6	6
5	12093	14	14	14
6	-	16	16	16
7	11249	16	16	16
8	321	4	4	4
9	-	-	28	32
10	-	-	31	31
11	-	-	31	31
12	4578	-	30	30
13	-	-	30	31
14	-	14	14	14
15	-	12	12	12
16	-	12	12	12
17	-	10	10	10
18	-	15	15	15
19	-	17	17	17
20	42262	10	10	10

Note a hyphen indicates that a solution could not be found. The depth-first search did not perform well on the problem set and was not able to find a solution for a majority of the problems. For those problem instances for which a solution was found, the number of moves exceed the known optimum by far. The breadth-first search was not able to produce solutions for 5 of the problems, namely those of the highest level of difficulty ranging from 16 to 24.

The A* algorithm produced solutions for all 20 problems. For two of the problems GP did not produce a program with the minimum number of moves. However, a change in the genetic operator application rates could result in smaller solution programs being produced as was the case for Puzzle 9 and Puzzle 12. This will be investigated as part of future work, including the investigation of mechanisms for exerting parsimony pressure during the evolutionary process.

5. CONCLUSION AND FUTURE WORK

The study evaluates genetic programming as a means of solving the 8-puzzle problem. While previous studies in this area have not been successful at finding a genetic programming solution to the 8-puzzle problem, the GP approach presented in the paper was able to evolve solution programs for a set of 20 problem instances of varying difficulty. The performance of the approach was also found to be better than both the depth-first search and breadth-first search in solving the 8-puzzle problem. An edit operator was used to remove introns from solution programs. The GP approach was able to produce programs with the minimum number of moves for all except two of the problems. However, it was found that the GP algorithm was very sensitive to the genetic operator application rates used and changes in rates could result in the minimum size solution program being found or not. Furthermore, the application rates that worked well for one problem instance did not necessarily produce optimal results for the others. Future work will investigate how best to fine tune application rates for this domain. Mechanisms for exerting parsimony pressure during evolution will also be examined. In addition to this a linear representation in which each program will be represented directly as a sequence of moves instead of a parse tree will also be investigated.

6. REFERENCES

[1] Reinefeld, A. 1993. Complete Solution of the Eight-Puzzle Problem and the Benefit of Node Ordering in IDA*. In Proceedings of the 13th Joint Conference on Artificial Intelligence (IJCAI '93), Vol. 1, 248-253, Morgan Kaufmann Publishers, Inc., USA.

[2] Kunkle, D. R. 2001. Solving the 8 Puzzle Problem in a Minimum Number of Moves: An Application of the A* Algorithm. http://web.mit.edu/6.034/wwwbob/EightPuzzle.pdf. Accessed 31 May 2013.

[3] Bhasin, H., Singla, N. 2012. Genetic Based Algorithms for N-Puzzle Problem. International Journal of Computer Applications, Vol. 51, No. 22, 44-50.

[4] Luger, G. F., Stubblefield, W. 1998. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Addison-Wesley Longman.

[5] Shaban, R.Z., Alkallak, I. N., Sulaiman, M. M. 2010. Genetic Programming to Solve Sliding Tile 8-Puzzle Problem. Journal of Education and Science, Vol. 23, No. 3, 145-157.

[6] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. 1998. Genetic Programming - An Introduction - On the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann Publishers, Inc.

[7] Barnes, J. M., Hasan, S. H., Lee, S. 2006. Solving the 8-Puzzle Problem: A Genetic Programming Approach. http://sha.ddih.org/f/Barnes-Hasan-Lee-Project-3.pdf. Accessed 31 May 2013.

[8] Marshall, C. 2007. Heuristics for Solving the Eight Puzzle Problem. http://www.soe. ucsc.edu/~csm/ 240/Report.pdf. Accessed 31 May 2013.

[9] Koza, J. R. 1992. Genetic Programming I: On the Programming of Computers by Means of Natural Selection, MIT Press.