Genetic Algorithms with Heuristic Knight's Tour Problem

Jafar Al-Gharaibeh
Computer Department
University of Idaho
Moscow, Idaho, USA

Zakariya Qawagneh
Computer Department
Jordan University for Science and Technology
Irbid, Jordan

Hiba Al-Zahawi
School of Computing
University of Utah
Salt Lake City, Utah, USA

Abstract - *Genetic algorithms are good search techniques for finding solutions to NP problems. However, their high degree of randomness sometimes fails to guide the search towards finding solutions within reasonable costs. The Knight's tour problem is an example of where pure GAs fail (practically) to find solutions. Combining GAs with other approaches on the other hand can highly improve their efficiency. In this paper, we present a new approach for improving the performance and effectiveness of GAs by applying heuristic.*

In our method, while the population is evolving using GA operators, a heuristic is applied to evolve the population either to full solutions or to better partial solutions to be used in generating the next population. The performance of this approach is compared against standard depth-first search with backtracking, heuristic alone, repair alone and also with genetic algorithms with repair. We applied our method to the Knight's tour problem using binary encoding. Results showed that our method generated more solutions compared to other approaches, and that solutions were found early in the search stages with less probability of failing to find a solution.

Keywords: Heuristic, Knight's tour, Repair, Path Search, Crossover, Chromosome Encoding.

1 Introduction and Previous Work

1.1 The Knight's Tour Problem

The knight's tour problem is a classical chess problem that has been studied for centuries. The problem can be described as follows: starting from a square on the chessboard, find a sequence of legal moves for the knight such that the knight visits all the other 63 squares in exactly 63 moves (i.e. touches each square exactly once).

The first definition for the knight tour problem was introduced by Ar–Rumi in Baghdad around 840 [1]. The first 8x8 knight problem solution was described in a manuscript by Al-Hakim in 1350 [1] [2] and the proposed tour is shown by the two diagrams in Figure 1. The image on the left shows the squares labeled in order of the moves and on the right is the geometrical representation of the tour [1] [2].

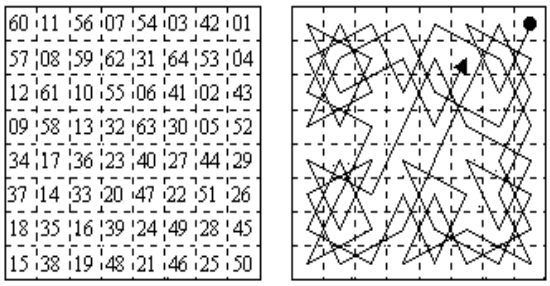

Figure1. Order of tour-moves (left) , Geometrical diagram of the tour(right).

The mathematician Leonhard Euler was the first to introduce a mathematical analysis of the knight's tour problem [1][5]. His approach was to develop tours by deriving new tours from existing tours. Suppose we have a tour with legal moves from a...m...f…n, we can replace the sequence from (m...f) with another tour, from the original one [1].

How many tours are there?

- Robert Willis in 1821 gave 20 tours for 8x8 board.
- C. Flye Sainte- Marie in 1877 calculated the numbers of the tours for the 4x8 board 7772, but he did not draw any actual tours.
- In 1880 E.M. Laqulere extended the 4x8 which designed by Sainte-Marie (1877) to find the 8x8 reentrant tours, his calculations gave 31054144 tours.
- Plank Charles in 1408 calculated 1728 tours for 5x5 board.
- Colone Ugo Papa in 1420 tried to provide a method to enumerate tours on small boards, but he can not.
- In 1995 Martin Lobbing and Lngo ran 20 sun workstations for four months to calculate the possible 8x8 tours, they said, they obtained 33,439,123,484,294 tours. But in 1997 Brendin Mckay got 13,267,364,410,532 tours[6].

1.2 Previous Work

Depth-first search with backtracking is shown to find solutions to the Knights tour problem but is considerably slow and the success rate is s highly dependent on the choice of the starting square for the tour. Warnsdorff in 1823 used a heuristic method to determine the next move rather than using random selection. The Warnsdroff's method says that the next square to be selected is the one that has the fewest number of available moves. The current approach to solving the knight's tour problem is to use a genetic algorithm. Genetic algorithms were developed by John Holland and colleagues in 1975 and are based on biological theories of evolution [5]. The algorithm begins with a given set of solutions represented by chromosomes known as a population. The algorithm then applies selection, crossover, mutation, and fitness functions to choose the best chromosomes to generate a new population. Chromosomes are selected based on their fitness level, with better chromosomes having a better chance of reproduction. The process is repeated until a condition is satisfied. However the pure form of genetic algorithms suffers from high failure rate and slow runtime.

Gordon and Slocum in 2004 introduced genetic algorithm with repair to find solutions for the knight's tour problem. The repair process is applied on the population during the fitness calculation stage. The evaluation of a string doesn't stop when an impasse is reached. Instead, the evaluation of the string continues if the impasse can be replaced by another legal move. Therefore, in such case a repair takes place and alters the population [8].

2 Implementation

2.1 Encoding

The binary encoding that we applied for this problem was the same encoding technique used by Gordon and Slocum in their genetic algorithm with repair approach [8]. According to the knight's position on the board, there exists between two and eight possible moves to choose from (Figure2). Each move can be represented as a 3-bit binary string.

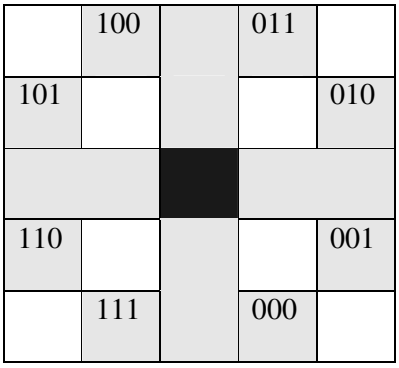

Figure2. The eight different moves for the knight and their binary representation

Each chromosome represents a sequence of 63 moves plus the start square with a total of length 64*3 = 192 bits. The start state can be set to any square on the board. Each 3-bit substring represents the next move on the board according to the current state (knight's current position?). For example the chromosome

111000101100111 …

Can be divided into

111 – 000 – 101 – 100 – 111 …

And this is the sequence of moves represented by this chromosome.

We compute the fitness for each chromosome by counting the number of legal moves represented in each chromosome. The fitness value ranges from 1 to 63. An illegal move can be made by a knight when it jumps off the board or returns to a previously visited square. For example, given the following chromosome and a start at square 50:

010100101000 …
010 – 100 – 101 – 000 …

This string can be translated to the following sequence of square moves on the chess board: (starting from square 50)

50 – 44 – 27 – 33 - 50

This chromosome has a fitness value of 3. The knight moves back to a visited square in the fourth move, leaving a total of three legal moves.

2.2 Crossover and Mutation

In our approach we used a one point crossover that is selected randomly, the crossover rate is between 85% and 95% with 0.5% mutation rate.

2.3 Selection

Chromosomes are selected using Russian Roulette wheel selection, where the better chromosomes have a greater chance of being selected. Using this selection method, a maximum of a quarter of the chromosomes (worst chromosomes in generation) are replaced by better chromosomes in the generation. This gives a greater chance for the best chromosomes to participate in the crossover to produce the next generation, and reduces the chances of the worst chromosomes to produce the next generation.

2.4 Extending partial tours using heuristic

A pure genetic algorithm did not give any solutions for a knight tour. Lee made this argument after trying to solve the Knight's tour problem using a genetic algorithm and limiting his work to small boards less than 8x8 [5].

Gordon and Slocum made similar argument in their paper, evolutionary VS depth-first search [8]. In our approach, we instead extend the partial tour using heuristic.

For each partial solution (chromosome in a generation), when a knight jumps off the board or returns to a previously visited square, a modification with heuristic is applied. We try to replace the 3-bit string that represents the illegal move with another 3-bit string that allows the knight to proceed. Before every move, we examine the squares that can be reached with legal moves from the current square. Then for each of those possible next squares, we count the number of legal moves at that square. . The knight then moves to the square with the lowest number of new choices. We repeat for each chromosome until a substitution cannot be made, and the evaluation of that chromosome then ends.

In the earlier example, the tour contained an illegal move when square 50 was visited twice

010 – 100 – 101 – 000 …

(50 – 44 – 27 – 33 - 50)

Therefore, the right-most move (000) would be replaced with another legal move. According to our approach heuristic must find the legal moves from the current square (33 in this case). Which are the squares, 43 (through move 0) and 18 (through move number 3), and choose the square with the lowest number of legal moves. Square 43 has 7 legal moves, while square 18 has 5 legal moves. Therefore, the move to the square 18 is chosen (move 3) and the string 000 (move 0) is substituted by the string 011 (move 3). (Moves codes were shown in Figure2)

010 – 100 – 101 – 011 …

Each chromosome is evaluated when an illegal move is encountered in the same manner until the knight can no longer make a legal move. Whitley was the first to alter the population during the fitness calculations in genetic algorithms [9]. This process called adaptation.

3 Tests and Results

3.1 GA with heuristic

For testing purposes we set the GA parameters to the following values, which are common in the literature. Higher values for crossover rate were used sometimes to produce more different values to be tested heuristically.

Population size = 50

Crossover percentage = 85-95%

Mutation rate = 0.5%

Number of generations = 50

The GA was applied five times for each square (i.e. 64x5 = 320 runs were made, 800000 chromosome were evaluated for the full board). In each run the results were recorded. So the averages were calculated for each square (5 runs) and also the averages for all of the squares (320 run, i.e. the average performance of the algorithm). Results that were considered in the test include the following:

- Total number of distinct tours that were found and their ratio of the overall chromosomes that were tested.
- Number of tours found in each run (average, maximum, minimum)
- The maximum and minimum number of tours found in one generation
- Number of generations needed to produce the first tour. (average, maximum, minimum)
- Number of runs that produce tours in the first generation and their ratio.
- Number of runs that failed to find any solution and their ratio

Figure3 is a sample result of GA-with heuristic. The first value in each square represents the average number of generation needed to produce the first tour starting from that square (average for five runs). The second value represents the total number of distinct solutions that were found in the five runs for that square.

2 : 294	5 : 226	4 : 217	6 : 183	5 : 181	4 : 269	4 : 166	3 : 184
10 : 135	4 : 109	6 : 173	3 : 184	5 : 212	6 : 118	3 : 193	5 : 204
5 : 133	7 : 247	3 : 45	6 : 153	3 : 204	7 : 233	7 : 140	2 : 233
4 : 317	3 : 138	5 : 98	3 : 208	6 : 191	6 : 154	3 : 214	8 : 204
3 : 251	6 : 126	6 : 205	3 : 122	4 : 218	4 : 270	5 : 137	4 : 159
3 : 257	2 : 108	8 : 120	5 : 121	5 : 213	1 : 144	7 : 49	5 : 178
1 : 274	6 : 127	6 : 112	5 : 171	5 : 123	9 : 208	4 : 147	2 : 223
1 : 492	3 : 210	6 : 179	3 : 258	3 : 237	2 : 352	5 : 151	3 : 182

Figure3. GA with Heuristic, left value: first generation that produce the first tour (average), right value: # of tours found in 5 runs for the square. (16000 Generations)

As an example of the overall performance, Figure3 test results were explored in further details. The total number of distinct complete tours was 12084, which compose 1.51% of the total number of the chromosomes that have been evaluated. On average 4.9 generations/run were needed to produce the first tour. In the worst case 25 generations were needed before any tour appears (excluding failed runs). And in its best case the first tour was found in the first generation (heuristic do this, GA still did nothing in the first generation). This was the case for 79 runs out of the 320 runs. So 25% of the times the first tour was found in the first generation. In every run there were 37.76 solutions on average (188.8/square). In the best case there were 22 solutions in only one generation but some generations might fail to find any solution in the worst case. For a complete run (50 generations) the algorithm failed to find any solutions at all generations only 3 times (0.94% of the total runs).

3.2 Heuristic only

To find out the performance of the heuristic only without GA and to know if the heuristic do all the job we ran the algorithm in the same manner for GA-with heuristic (population=50, 50 generations, 5 runs for each square) but with GA and the chromosomes were generated randomly each generation.

Figure4 shows the results of applying heuristic only. The total number of distinct complete tours was 1979 (0.247% of chromosomes). On average 8.22 generations/run were needed to produce the first tour. In the worst case 47 generations were needed before any tour appears(excluding failed runs). And in its best case the first tour was found in the first generation. This was the case for 81 runs out of the 320 runs (25% of runs). In every run there were 6.18 solutions on average (30.9 solution/square). In the best case there were 10 solutions in only one generation. 65 Runs failed to find any solutions (20.3% of runs).

12 : 41	6 : 64	5 : 31	5 : 94	5 : 20	10 : 12	9 : 61	2 : 38
14 : 25	16 : 20	3 : 24	8 : 29	19 : 20	2 : 6	1 : 23	9 : 36
3 : 32	5 : 13	0 : 15	7 : 21	11 : 6	4 : 20	9 : 17	1 : 42
3 : 50	5 : 41	15 : 29	5 : 18	17 : 26	7 : 19	9 : 14	16 : 42
6 : 36	6 : 31	13 : 12	4 : 43	2 : 33	1 : 19	7 : 22	13 : 25
3 : 30	8 : 31	3 : 26	5 : 12	12 : 40	12 : 38	17 : 20	17 : 28
5 : 42	9 : 49	11 : 23	7 : 34	12 : 32	12 : 9	5 : 20	7 : 34
1 : 91	2 : 15	9 : 16	3 : 38	12 : 43	12 : 62	11 : 22	9 : 54

Figure4. Heuristic Only, left value: first generation that produce the first tour (average), right value: # of tours found in 5 runs for the square. (16000 Generations)

3.3 GA with repair

Gordon and Slocum in their approach; GA with repair, they applied repair to partial tours before calculating the chromosome fitness. They used the following parameter for the genetic algorithm. Population size was 50, with one-point crossover at a rate of 80% and mutation rate of 1%. They ran the algorithm for 20,000 generations, which results in 1 million individuals being evaluated [8].

Figure5 shows the results of their approach, each square in the figure shows the average number of complete tours found starting at that square (20000 generations were evaluated) [8].

402	108	61	86	112	97	99	185
252	64	33	81	44	121	142	47
147	70	56	85	172	148	66	48
245	130	112	78	67	89	55	86
112	30	44	80	95	78	62	47
74	67	44	68	20	72	**19**	80
51	79	58	120	95	51	54	39
116	25	36	85	70	76	66	92

Figure5. GA with Repair Average # of tours found in 20,000 generations, starting at each square

3.4 Method Comparison

We refer to Gordon and Slocum [8] for the results of GA with repair, repair alone and backtracking. We developed a program to test GA with repair and repair alone and most of the time we produced the same results that they have shown [8]. Table6 gives a summary of the performance of the different approaches used to solve the knight tour problem.

Applying heuristic with GAs added no considerable time overhead compared to the repair method. In fact, both approaches took the same period of time to evaluate the same number of candidate chromosomes. As we can see in the table, GA with heuristic turns 1.51% of the total evaluated chromosomes to complete tours whereas GA with repair performs much less with only 0.57% of the chromosomes resulted in complete tours. Heuristic and repair are worse with less percentages. Backtracking performs very well in terms of number of tours found but not better than GA with heuristic and also backtracking is highly dependent on the starting square. 83% of runs, backtracking fails to find any complete tour. Repair and heuristic fail 20% of runs to find tours compared to backtracking. GA with repair fails 6% and GA with heuristic again performs much better than all of the other approaches with only 1% of runs failing to find complete tours.

Comparing other GAs factors between GA with heuristic and GA with repair we can see that with heuristic performs much better. For example, in 22 complete tours were found in only one generation out of the 50 chromosomes in the generation compared to only 16 tours in the case of GA with repair. Five generations were needed on average to produce the first complete tour using GA with heuristic compared to 7 generations in the case of GA with repair. Finally, 25% of runs using GA with heuristic produce a complete tour in the first generation versus 20% in the case of GA with repair.

4 Conclusions

Pure genetic algorithm was proven to fail to find solutions for the knight tour problem because it is highly random. Backtracking, although it finds large number of solutions, it is highly dependent on the initial conditions of the search problem. In addition, it fails in most of the runs to find any solution. Repair alone finds too few solutions and in many cases no solutions are found Heuristic alone perform better than repair alone because choosing the next move is selective in the case of heuristic but it is random in the case of repair. This gives greater chances for heuristic to guide the knight to a complete tour. Even so,

Table1. Results of applying different approaches to solve Knight tour problem.
Note: (-) means that the data are not available or not applicable for that field.

	GA with heuristic	GA with repair	Heuristic only	Repair Only	Backtracking
# tours found	12,084	5,696	1,979	192	10,304
#evaluations	800,000	1000,000	800,000	1000,000	-
Hit Ratio (tours found/evaluated)	1.51%	0.57%	0.25%	0.02%	-
average # tours found per square	189	89	31	3	161
Failed runs – no tours were found	1%	6%	20%	20%	83%
Maximum # tours in one generation	22	16	-	-	-
Average # generations to produce a tour	4.9	6.5	-	-	-
Runs produced tours in the first generation	25%	20%	-	-	-

heuristic alone finds few solutions because later in the tour-search the algorithm stops when the knight has no more legal moves to make.. The problem is the same for repair alone. A better approach is genetic algorithms where they apply GA operators on the chromosomes and then select the best chromosomes to be incorporated into the next generation. Our contribution is to combine heuristic with GA to provide a better method for selecting candidates and turning them into complete tours (full solutions). This process is highly effective in finding solutions. It was superior to all of the other approaches discussed earlier in the paper. The most important factor is that GA with heuristic fails only 1% of the time to find a solution compared to other approaches such as GA with repair which fails 6% of the time and also heuristic alone and repair alone both of which fail 20% of the time and backtracking which fails 83%. For many real world applications such as game playing (chess for example) or path determination, a solution (sequence of moves sometimes) must be found within a fixed a mount of time. So spending the time doing calculation to discover at the end that no solution could be found and the calculation must be restated is a poor approach. However, in the case of GA with heuristic it is almost guaranteed (99%) to find a solution in the first run. Another time factor is that 25% of the runs GA with heuristic find solutions in the first generation and only 5 generations are needed on average to construct the first solution compared to 20% and 7 generations in the case of GA with repair. So GA with heuristic not only provides better guarantee of finding a solution, but it also finds the solution in less time, better than all of the other approaches. Another factor is the large number of solutions that could be found using GA with heuristic. GA with heuristic has a high hit-ratio (# complete tours found/ # of tours evaluated) of , 1.51%, which is 2.65 times better than the 0.57% ratio for GA with repair and much better than 0.25% and 0.02% for heuristic alone and repair alone approaches respectively.

In summary, GA with heuristic provides a greater guarantee of finding solutions to the Knight's tour problem than other approaches. In addition, it finds a greater number of solutions and with less time on average the better than all of the other approaches.

5 Future work

The performance of genetic algorithms depends on the problem in hand and also on the different GA operators. It is also the case when applying heuristic to GAs. A deeper analysis is needed about how different values for GA operators might affect the overall performance of the algorithm. Using other heuristic functions might be an option but care is needed in calculating these heuristics as to not add a large overhead to the algorithm.

6 References:

[1] Murray H.J.R. (1913) History of Chess
[2] Jelliss, G. P. Introducing Knight's Tours, survey 2004 http://homepages.stayfree.co.uk/gpj/ktn.htm(june-2006)
[3] Jelliss, G. P. Knight's Tour notes. 2004 http://www.ktn.freeuk.com/1a.htm (june-2006)
[4] Gunno Törnberg. Knight's Tour http://web.telia.com/~u85905224/knight/eknight.htm. (june-2006)
[5] Holland, J., Adaptation in Natural and Artificial Systems, 1st ed., Univ. of Michigan, 1975. 2nd ed. 1992 by MIT Press.
[6] Friedel. F. The Knight's Tour. http://www.chessbase.com/columns/column.asp?pid=163 (june-2006)
[7] Löbbing M. and Wegener I. (1996) The Number of Knight's Tours equals 33,439,123,484,294 – Counting with Binary Decision Diagrams. Electronic Journal of Combinatorics. 3(1), R5
[8] Gordon V.S. and Slocum T.J. (2004) The Knight's Tour – Evolutionary vs. Depth-First Search. In proceedings of the Congress of Evolutionary Computation 2004 (CEC'04), Portland, Oregon, pp. 1435-1440
[9] Lee, M., "Finding Solutions to the Knight's Tour Problem Using Genetic algorithms," Genetic Algorithms and Genetic Programming at Stanford 2000. Published by Koza, J. at Stanford University, 2000.
[10] Whitley, D., Gordon, V.S., and Mathias, K., "Lamarckian Evolution, the Baldwin Effect and Function Optimization," Parallel Prob. Solving from Nature, Israel, pp 6-15, 1994