The Knight's Tour - Evolutionary vs. Depth-First Search

V. Scott Gordon
California State University, Sacramento
6000 J st., Sacramento, CA 95819
Email: gordonvs@ecs.csus.edu

Terrill J. Slocum
Quarterwave Corporation
5780 Labath Ave., Rohnert Park, CA 94928
Email: terry@quarterwave.com

Abstract- **A genetic algorithm is used to find solutions to the standard 8x8 knight's tour problem, and its performance is compared against standard depth-first search with backtracking. The binary encoding is described, along with a simple repair technique which can be used to extend tours that have reached impasse. The repair method is powerful enough on its own to find complete tours, given randomly generated bitstrings. But when used in conjunction with a genetic algorithm, considerably more solutions are found. Depth-first search is shown to find more solutions under certain conditions, but the genetic algorithm finds solutions more consistently for arbitrary initial conditions.**

I. The Knight's Tour

The Knight's Tour is as follows: given a chessboard of specified size and dimension, find a sequence of legal knight moves such that the knight touches every square once and only once. Mordecki [1] shows that on an 8x8 board, an upper bound on the number of tours is approximately 1.305×10^{35}. Lobbing and Wegener [2] computed the exact number of *re-entrant*, or cyclic tours (those that end on the start square) as being 13,267,364,410,532. But the number of blind alleys is significantly greater.

The problem has captured the imagination of chess players and mathematicians for centuries. Taylor, Euler and Lagrange, for example, all worked on the knight's tour, Taylor often being credited for proposing it as a mathematical problem in the early-1700s, and Euler for giving it serious analysis in the mid-1700s after several of his contemporaries had already found solutions [3]. The Belgian chess master Koltanowski made a career of performing knight's tour exhibitions. Audience members would fill a chessboard with names or dates, and after a minute of study, Koltanowski would recite a knight's tour calling out the names on the squares, while blindfolded.

Much like the well-known "traveling salesman" problem, it can be solved using depth-first search with backtracking, simply by trying every possible sequence, backing up when reaching an impasse and trying a different route. Other algorithms have also been used. Warnsdorff (1823) devised a method [4] which does not require backtracking and finds one solution immediately, but fails for n>75. Parberry [5] used both divide-and conquer, and a neural network, for building larger tours out of smaller ones.

II. Genetic Algorithms

Genetic algorithms are search algorithms based loosely on the principles of natural evolution, particularly genetic evolution. They have been useful for optimization problems, such as finding the shortest path through a set of cities. By applying simplified notions of selection, crossover, mutation, and survival of the fittest to an artificial population of candidate solutions, a genetic algorithm can, with relatively little tailoring of the solution method to the problem domain, evolve solutions. The concept of the genetic algorithm was described by Holland in 1975 [6], but the technique did not become popular until the mid-1980s.

The genetic algorithm used here is the Simple Genetic Algorithm (SGA) described by Holland [6] and later by Goldberg [7]. In an SGA, the genetic operators are applied successively to random members of the population until a new population of size equal to the old population is generated. Gradual improvement in the quality of the solutions is achieved by selecting strings for recombination that tend to be more fit; i.e., that themselves represent better solutions than the other strings in the population.

We chose to use an *elitist* version of the SGA. In our version of elitism, after a new population is generated, the most fit individual from the previous generation is always copied over the least fit individual in the new generation. Elitism guarantees that the fitness of the most fit individual found so far, plotted over time, never decreases.

III. Encoding

The binary encoding described here was also developed independently by Miler Lee [8] at Stanford University. Both Lee and present author Slocum introduced their encoding methods in class projects. Lee used a different repair method than Slocum (described here).

Assuming a fixed starting square, a complete knights tour is a sequence of 63 moves. A cyclical knights tour is a sequence of 64 moves. Although we did not consider whether or not a tour was cyclical, we utilized an encoding that represented 64 moves, to allow for the possibility of considering cyclical tours in the future.

Depending on the square on which the knight resides, there can be from 2 to 8 possible moves. Thus, a move can be represented in three bits using binary encoded values such as shown below in Figure 1.

0-7803-8515-2/04/$20.00 ©2004 IEEE

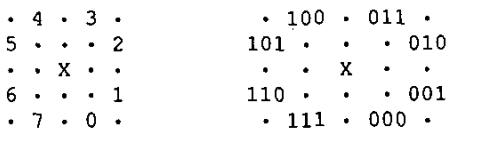

Figure 1. numeric representations for legal knight moves from square X

A series of 64 knight moves can thus be represented with a binary string of length 64x3=192 bits. A starting square is chosen, and moves then proceed relative to the current square. Thus, for start square E4 (the center of the board) a string which starts:

```
110110001010101....
```

is decoded into the series of digits:

```
110-110-001-010-101... = 66125...
```

which, in algebraic chess notation, represents the tour fragment:

```
(E4) - C3 - A2 - C1 - E2 - C3...
```

Note that in this example, square C3 is visited twice, already violating the definition of a legal tour. This gives rise to a natural method for evaluating the fitness of strings: by simply counting the number of legal moves it represents. That is, it is decoded and its path traversed until the knight makes an illegal move (jumps off the board) or the knight revisits a square already visited. The number of legal moves made up to that point, thus an integer from 1 to 63, is the fitness of the string. In the previous example, the value of the string is 4, since four legal moves:

```
- C3 - A2 - C1 - E2
```

were made. The bits which follow an illegal move are ignored. A fully legal tour is thus known ahead of time to have a fitness value of 63. We always ignore the last three bits, since we are not at this time concerned with whether the tour is cyclic.

An interesting property of this encoding is that it gives a heavy bias to bits that appear early in the string. A change in a string alters the starting point for all subsequent moves. Thus a change in the first few moves of a low fitness string could result in a substantial increase in fitness, and vice-versa, whereas a change after the first illegal move may have no effect at all. If a string has near optimum fitness, a change in the beginning of the string could decrease its fitness drastically. This is one of the reasons that we chose to use an elitist genetic algorithm.

IV. Extending Partial Tours Using Repair

Our first experiments using the implementation described above did not yield any legal tours even when several million strings were evaluated over the course of many thousands of generations. Lee [8] also made this observation, and therefore limited his runs to smaller boards. We instead devised a simple method of *repairing*, or extending the tour using the rightmost, as yet unused portion of the partial solutions, as follows.

For each string in the population, the point at which it was no longer evaluated (i.e., where the knight jumped off the board or back onto a previous square) is where repair occurs. At that point, the move indicated (the 3-bit chunk) is checked to see if substituting another 3-bit pattern will allow the tour to proceed. Since there are only 7 possible replacement values, this is not overly expensive. If a substitution cannot be made which extends the tour, evaluation of that string stops. If a replacement can be made which extends the tour, the evaluation then can proceed to the right, as before. No backtracking is performed.

For instance, in the previous example, evaluation of the string reached an impasse when square C3 was encountered for a second time. Thus, the string:

```
110110001010101...
```

having reached impasse at the rightmost substring 101, would consider using 000 instead. From E2, this would represent a move off of the board, so substring 001 would be considered. From E2, this represents a move to G1, which has not yet been visited and is therefore a legal move. The string therefore continues to be evaluated, having been changed to:

```
110110001010001...
```

Each string is therefore still evaluated left-to-right, repairing when an impasse is reached. Since the repair actually modifies the string, the population is altered due to the process of fitness evaluation. The implications of altering a population during fitness evaluation has been explored by Whitley et. al (1994) [9].

Starting at square E4, our first run found a tour at generation 60, and is shown in Figure 2 (the bitstring has been translated to octal for readability, along with a chessboard showing the sequence of moves)

Although we were not concerned with cyclic tours, we did note that the third solution found (at generation 99 of the first run), was in fact cyclic (shown in Figure 3).

```
string = 43650360167247123336707235246 6
  3671602124642256571064102143461072

      05 48 07 32 03 46 29 20
      08 33 04 47 30 21 60 45
      49 06 31 02 61 44 19 28
      34 09 50 43 22 27 62 59
      53 14 35 10 01 18 41 26
      36 11 54 51 42 23 58 63
      15 52 13 38 17 56 25 40
      12 37 16 55 24 39 64 57
```

Figure 2. first knights tour found, at generation 60.

```
string = 436503601672471233361472565714
  2761602124570254301603452070565417

      05 28 07 32 03 26 23 20
      08 31 04 27 48 21 56 25
      29 06 33 02 55 24 19 22
      34 09 30 47 42 49 54 57
      63 14 35 10 01 18 41 50
      36 11 64 43 46 51 58 53
      15 62 13 38 17 60 45 40
      12 37 16 61 44 39 52 59
```

Figure 3. first cyclic knights tour found, at generation 99.

V. Test Methodology

A. Test Methodology -- GA + Repair

The elitist SGA with repair was used, with string length 192 encoded as previously described. The population size was 50, with one-point crossover at a rate of 80%, and mutation rate of 1% (that is, each bit is flipped with probability 0.01). We chose these GA parameters because they were commonly used in other research literature.

We ran the algorithm for 20,000 generations, which results in 1 million individuals being evaluated. Naturally, it is possible (in fact, quite likely) for there to be duplicates among the 1 million strings, since the process of selection results in copying individuals from one generation to the next. Therefore, the GA probably evaluated considerably less than 1 million different strings.

This process took 50 seconds on a Dell Pentium 3 laptop running Windows 2000 and Visual C++. We ran the experiment 5 times per start square, for each of the 64 possible start squares, recording the number of distinct tours found in each of the 64x5=320 runs.

B. Test Methodology -- Repair only

As stated earlier, without repair the GA was unable to find any tours. Since adding repair enabled solutions to be found, it was necessary to determine whether the GA was doing any useful work, or if repair alone would suffice.

Since the GA was generating 1 million strings over the course of its 20,000 generations, we generated 1 million strings randomly and applied repair to each of them, counting the number of complete tours that were produced. We also ran this experiment 5 times per start square for each of the 64 possible start squares, recording the number of distinct tours found in each of the 64x5=320 runs. We also noted that the time to complete these runs was comparable to that of the GA, namely, about 50 seconds per run.

C. Test Methodology -- Backtracking

Solving a knight's tour using backtracking is a simple exercise commonly given as an undergraduate programming exercise. Searching the web provides numerous examples of illustrative algorithms and class assignments. We felt it was important to compare the performance of the GA against this standard algorithm under similar conditions.

A start square is selected, and a move is chosen from the sequence shown in Figure 1. If the move is legal, the process repeats from the new square. This continues until the move either jumps off the board, or lands on a square already visited. Then, the last move is retracted and the next move in sequence is tried. This was run on the same laptop, also for 50 seconds. Each of the 64 possible start squares was considered, and the number of tours found in 50 seconds, for each start square, was recorded.

On the Internet, one finds a variety of descriptions regarding the observed performance of this backtracking algorithm on the knight's tour. Many such sources describe the 8x8 problem as being too large for simple backtracking to find solutions in such a short period of time; in fact, one such academic site indicated that it would take over a million years to even find one solution! Incorporating the Warnsdorff technique mentioned earlier is a frequently-mentioned way of speeding the discovery of solutions. While true, we felt that it utilized too much chess-specific information, as we wanted to use only generic search techniques. Besides, to our surprise, we discovered that the supposition of simple backtracking being infeasible for finding 8x8 tours was incorrect -- given the right start conditions, we were able to quickly find thousands.

VI. Results

A. Results -- GA + Repair

The GA was run for 50 seconds, time enough for 20,000 generations and thus 1,000,000 string evaluations (including the repair algorithm). The number of *distinct* complete tours were tallied for each run (duplicate tours, which presumably could happen frequently due to genetic selection, were not counted). The chessboard in Figure 4 shows, for each square, the average number of complete tours found starting at that square, averaged over 5 runs each. Figure 5 gives relevant statistics:

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

Figure 4. GA+Repair: Avg. # of tours found in 20,000 gens, starting at each square (min and max values are shown in boldface)

Total number of runs (64x5)	320
Average # of tours per run	89
Most tours found in one run	642
Least tours found in one run	0
% of runs with no tours found	6%

Figure 5. GA + Repair: Performance statistics.

B. Results -- Repair only

One million strings were generated randomly for each start square, and the repair algorithm was applied to each string, taking about 50 seconds for each run. The number of distinct complete tours were tallied for each run. The chessboard in Figure 6 shows, for each square, the average number of complete tours found starting at that square, averaged over 5 runs each. Figure 7 gives relevant statistics:

24.0	2.2	2.4	3.0	8.3	3.2	2.8	5.6
8.0	2.6	1.0	0.8	2.6	1.4	4.4	2.2
6.8	1.0	**0.0**	2.0	1.8	0.4	1.0	2.2
5.4	2.0	1.8	2.8	3.4	3.4	0.2	3.0
2.4	1.8	2.2	3.4	2.2	2.4	1.6	3.6
1.4	0.4	0.8	1.0	1.6	2.0	0.2	1.2
3.6	2.0	0.6	1.2	0.2	0.6	2.0	1.0
7.0	1.4	0.6	2.2	1.4	0.8	1.2	5.2

Figure 6. Repair only: Avg. # of tours in 1,000,000 strings, starting at each square (min and max values shown in boldface)

Total number of runs (64x5)	320
Average # of tours per run	3
Most tours found in one run	29
Least tours found in one run	0
% of runs with no tours found	20%

Figure 7. Repair only: Performance (values rounded to the nearest integer)

C. Results -- Backtracking

Depth-first search with backtracking was run for 50 seconds, for each start square. The number of complete tours were tallied for each square. The chessboard in Figure 8 shows, for each square, the number of complete tours found starting at that square. Figure 9 shows relevant statistics:

0	0	0	0	0	0	0	31
0	0	0	0	0	9	0	0
0	0	0	235	0	0	0	0
1155	0	0	0	0	0	0	0
1204	0	0	0	0	0	0	212
0	0	0	36	513	0	0	0
0	0	0	0	0	0	0	0
4256	0	0	2677	0	0	0	7

Figure 8. Backtracking: Number of tours found starting at each square.

Total number of runs	64
Average # of tours per run	161
Most tours found in one run	4256
Least tours found in one run	0
% of runs with no tours found	83%

Figure 9. Backtracking: Performance statistics.

We also repeated the backtracking runs for a different sequence of moves. Each time a square is visited, the 8 possible knight moves must be tested in turn. For the above runs, the order of moves tried is given in Figure 1, in which the moves are tested in circular order. We also tried the following relative order: *0-4-2-5-1-7-3-6.*

The chessboard in Figure 10 shows the number of complete tours found starting at each square, for the new search sequence. Figure 11 shows relevant statistics.

0	0	202	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	203	0	0	0	0	0	0
0	0	0	0	0	0	51	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 10. Backtracking: Number of tours found starting at each square. Sequence used is *1-5-3-6-2-8-4-7* relative to Figure 1.

Total number of runs	64
Average # of tours per run	7
Most tours found in one run	203
Least tours found in one run	0
% of runs with no tours found	95%

Figure 11. Backtracking: Performance statistics for revised sequence.

VII. CONCLUSIONS

Consider the summary of data as shown in Figure 12. The experiments yielded a number of surprises. We were pleasantly surprised by the performance of the GA in finding complete 8x8 tours so reliably, regardless of the initial conditions. But we were equally intrigued by the ability of simple backtracking to find large numbers of tours, if one is willing to try a variety of starting conditions.

We presented a repair algorithm which was shown to find individual tours reliably. In fact, given arbitrary initial conditions, the repair algorithm alone applied to random strings appears to be more likely to find a solution than sample backtracking. It found tours 80% of the time, whereas backtracking found tours only 5 to 17% of the time for the initial conditions we considered.

Incorporating a genetic algorithm with repair was shown to find roughly 30 times as many tours as using repair alone. It found tours in 94% of the runs, also considerably better than backtracking. Also, the GA appears to be much less sensitive to the starting conditions than is backtracking; the number of tours found ranged from 19 to 420 for the GA, while for backtracking it ranged from 0 to 4256 (with the vast majority of start squares yielding no solutions with backtracking). Genetic search seems to be considerably more reliable, if less sporadically spectacular, for this application.

Simple backtracking often performed extremely well, finding in one case over 4000 8x8 tours in less than one minute. In fact, it found more tours than any other method.

This is in stark contrast to statements commonly made on websites and undergraduate homework assignments. But its performance is very sensitive to the start square, and the order in which squares are tried. This explains why so many computer programmers believe that it is incapable of finding solutions to the 8x8 problem, since there is an 85-95% chance that a particular run, for an initial set of conditions, will find no tours even for lengthy runs. In some ways, backtracking performed the best, but in other ways, it performed the worst.

This discovery suggests a simple modification for using backtracking to solve the knight's tour quickly. Since it was observed that certain start squares yield many tours quickly, simply have the algorithm try each start square for a few seconds, and if no tours are found, try the next start square. Once a start square is found which yields some solutions, allow that run to continue and many solutions will be found.

The genetic algorithm was shown to be a particularly reliable method of finding 8x8 knight's tours. With a population size of only 50, finding on average 89 tours among the <1,000,000 candidates is indicative of a well-directed search, especially for a space of this nature (10^{35} tours in a search space of 2^{192} strings). Repair assists the genetic algorithm dramatically. Yet, the hill-climbing method used borrows no chess-specific knowledge, and does no backtracking or look-ahead. Nor does repair alone produce very good results. The two methods compliment each other nicely, as neither does well alone.

VIII. FUTURE WORK

The work presented here utilized what seemed to us to be the simplest and most obvious genetic encoding and recombination operators. It is likely that other operators may realize significant performance improvements. In particular, we plan next to try Whitley's edge recombination operator [10] which seems well-suited to this application.

	Backtracking	**Backtracking**	**GA+repair**	**Repair alone**
	(sequence A)	***(sequence B)***		
run-time	50 seconds	50 seconds	50 seconds	50 seconds
# evaluations	N/A	N/A	1 million	1 million
number of runs	64 x 2 = 128	64 x 2 = 128	64 x 5 = 320	64 x 5 = 320
avg # tours per run	161	7	89	3
max # tours found	4256	203	642	29
min # tours found	0	0	0	0
% runs with tours	17%	5%	94%	80%

Figure 12. Summary of results

REFERENCES

[1] Mordecki, E., "On the Number of Knight's Tours," Prepublicaciones de Matemática de la Universidad de la República, 2001/57, 2001.

[2] Löbbing, M. and Wegener, I., "Branching Programs and Binary Decision Diagrams. Theory and Applications," in SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, PA, 2000.

[3] Ball, W.W.R. and Coxeter, H.S.M., Mathematical Recreations and Essays, 13th ed., Dover, New York, 1987.

[4] Warnsdorff, H.C., "Des Rösselsprungs einfachste und allgemeinste Lösung," Schmalkalden, 1823

[5] Parberry, I., "An Efficient Algorithm for the Knight's Tour Problem," Discrete Applied Mathematics, Vol. 73, pp. 251-260, 1997

[6] Holland, J., Adaptation in Natural and Artificial Systems, 1st ed., Univ. of Michigan, 1975. 2nd ed. 1992 by MIT Press.

[7] Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

[8] 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.

[9] Whitley, D., Gordon, V.S., and Mathias, K., "Lamarckian Evolution, the Baldwin Effect and Function Optimization," Parallel Prob. Solving from Nature 3, Israel, pp 6-15, 1994

[10] Whitley, D., Starkweather, T., and Shaner, D., "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination," in Handbook of Genetic Algorithms, Davis, editor. Van Nostrand Reinhold, pp 320-372, 1991