Ant Colonies Discover Knight's Tours

Philip Hingston[1] and Graham Kendall[2]

[1] Edith Cowan University, Australia
p.hingston@ecu.edu.au
[2] The University of Nottingham, UK
gkx@cs.nott.ac.uk

Abstract. In this paper we introduce an Ant Colony Optimisation (ACO) algorithm to find solutions for the well-known Knight's Tour problem. The algorithm utilizes the implicit parallelism of ACO's to simultaneously search for tours starting from all positions on the chessboard. We compare the new algorithm to a recently reported genetic algorithm, and to a depth-first backtracking search using Warnsdorff's heuristic. The new algorithm is superior in terms of search bias and also in terms of the rate of finding solutions.

1 Introduction

A *Knight's Tour* is a Hamiltonian path of a graph defined by the legal moves for a knight on a chessboard. That is, a knight must make a sequence of 63 legal moves visiting each square of an 8x8 chessboard exactly once. Murray [1] traces the earliest solutions to this problem back to an Arabic text in 840 ad. Leonhard Euler carried out the first mathematical analysis of the problem in 1759 [2]. Other well-known mathematicians to work on the problem include Taylor, de Moivre and Lagrange.

There is interest in finding both *open* and *closed* tours. A closed tour has the extra property that the 63rd move ends on a square that is a knight's move away from the start square, so that the knight could complete a Hamiltonian circuit with a 64th move. Closed tours are more difficult to find. An upper bound of the number of open tours was found to be approximately 1.305×10^{35} [3]. Löbbing and Ingo [4], calculated the number of closed tours, later corrected by McKay to be 13,267,364,410,532 tours [5]. Though there are many tours, the search space is even larger, at around 5.02×10^{58}.

A depth-first search, with backtracking, is perhaps the most obvious search method, though rather slow. A heuristic approach due to Warnsdorff in 1843, is the most widely known approach [6]. Using Warnsdorff's heuristic, at each move, the knight moves to a square that has the lowest number of next moves available. The idea is that the end of the tour will visit squares that have more move choices available.

A recent approach to finding knight's tours used a genetic algorithm [7]. This used a simple genetic algorithm [8], encoding a knight's tour as a sequence of 63x3 bits. Each triple represents a single move by the knight, with the fitness being defined by the number of legal moves (maximum = 63) before the knight jumps off the board or revisits a square. If a candidate tour leads to an illegal move, a repair operator checks the other seven possible knight's moves, replaces the illegal move with a legal move if

G.I. Webb and Xinghuo Yu (Eds.): AI 2004, LNAI 3339, pp. 1213–1218, 2004.

there is one, and then attempts to continue the tour, repairing as needed. Without repair, the genetic algorithm found no complete tours. With repair, the maximum number of tours reported in a single run of 1,000,000 evaluations was 642.

2 The Ant Colony Algorithm

Ant colony optimization (ACO) algorithms are based on the observation that ants, despite being almost blind and having very simple brains, are able to find their way to a food source and back to their nest, using the shortest route. ACO's were introduced by Marco Dorigo [9], [10]. In [10] the algorithm is introduced by considering what happens when an ant comes across an obstacle and has to decide the best route to take around the obstacle. Initially, there is equal probability as to which way the ant will turn in order to negotiate the obstacle. Now consider a colony of ants making many trips around the obstacle and back to the nest. As they move, ants deposit a chemical (a *pheromone*) along their trail. If we assume that one route around the obstacle is shorter than the alternative route, then in a given period of time, a greater proportion of trips can be made over the shorter route. Thus, over time, there will be more pheromone deposited on the shorter route. Now the ants can increase their chance of finding the shorter route by preferentially choosing the one with more pheromone. This sets up a positive feedback cycle, known *stygmergy* or *autocatalytic behaviour.*

This idea has been adapted to derive various search algorithms, by augmenting pheromone trails with a problem specific heuristic. In the most famous example, ants can be used to search for solutions of the traveling salesman problem (TSP). Each ant traverses the city graph, depositing pheromone on edges between cities. High levels of pheromone indicate an edge that is in shorter tours found by previous ants. When choosing edges, ants consider the level of pheromone and a heuristic value, distance to the next city. The combination determines which city an ant moves to next.

We now present the new ACO algorithm that we use to discover knight's tours. As for the TSP, ants traverse a graph, depositing pheromones as they do so. In this case of the Knight's Tour Problem, the vertices of the graph correspond to the squares on a chessboard, and edges correspond to legal knight's moves between the squares. Each ant starts on some square and moves from square to square by choosing an edge to follow, always making sure that the destination square has not been visited before. An ant that visits all the squares on the board will have discovered a knight's tour.

We found it best to search for solutions from all starting squares simultaneously. We hypothesise that an ant starting on one square can utilize the knowledge gained by ants starting on more remote squares – knowledge that is harder to obtain from other ants starting on the same square.

We need some notation to describe the algorithm in detail. First, we define $T_{row,col,k}$ to be the amount of pheromone on the k^{th} edge from the square in row *row* and column *col*. For squares near the edge of the chessboard, some moves would take the knight off the board and are illegal. We set $T_{row,col,k} = 0$ for those edges. We use $dest_{row,col,k}$ to denote the square reached by following edge *k* from square (row,col).

Initialising the Chessboard. Initially, some pheromone is laid on each edge. In our simulations we used $T_{row,col,k} = 10^{-6}$ for all edges corresponding to legal moves.

Evaporating Pheromones. Pheromones evaporate over time, preventing levels becoming unbounded, and allowing the ant colony to "forget" old information. We implemented this by reducing the amount of pheromone on each edge once per cycle, using:

$$T_{row,col,k} \rightarrow (1-\rho) \times T_{row,col,k}$$

where $0 < \rho < 1$ is the called the *evaporation rate*.

Starting an Ant. Each ant has a current square (row, col) and a tabu list, which is the set of squares that the ant has visited so far. Initially, $(row, col) = (startRow, startCol)$, and $tabu = \{(startRow, startCol)\}$. Each ant also remembers her start square, and her sequence of moves. Initially, $moves = <>$, an empty list.

Choosing the Next Move. To choose her next move, an ant computes, for each edge leading to a square not in her tabu list, the following quantity:

$$p_k = (T_{row,col,k})^{\alpha}$$

where $\alpha > 0$, the *strength* parameter, is a constant that determines how strongly to favour edges with more pheromone. She then chooses edge *k* with probability:

$$prob_k = \frac{p_k}{\sum_{j:dest_{row,col,j} \notin tabu} p_j}$$

Moving to the New Square. In some ACO algorithms, ants deposit pheromone as they traverse each edge. Another alternative, which we use in our algorithm, is for no pheromone to be deposited until the ant has finished her attempted tour. Hence, having chosen edge *k*, she simply moves to $dest_{row,col,k}$, and sets:

$$tabu \rightarrow tabu \cup \{dest_{row,col,k}\}, \text{ and}$$

$$moves \rightarrow moves + < k >$$

Keep Going Until Finished. Eventually, the ant will find herself on a square where all potential moves lead to a square in her tabu list. If she has visited all the squares on the chessboard, she has found a valid knight's tour, otherwise a partial tour.

Lay Pheromone. When she has finished her attempted tour, the ant retraces her steps and adds pheromone to the edges that she traverses. In order to reinforce more successful attempts, more pheromone is added for longer tours. We have found that we obtain slightly better results by reinforcing moves at the start of the tour more than those towards the end of it. Specifically, we define, for each ant *a*, for each row and column, and each edge *k*:

$\Delta T_{a,row,col,k} = Q \times \frac{(|moves| - i)}{(63 - i)}$, if ant *a*'s i^{th} move used edge *k* from *row*, *col*, and

$$\Delta T_{a,row,col,k} = 0 \text{ , otherwise}$$

where the parameter Q is the *update rate*, and the value 63 here represents the length of a complete open tour. Thus, each ant contributes an amount of pheromone between 0 and Q. Once all ants complete their attempted tours, we update the pheromone using the formula:

$$T_{row,col,k} \rightarrow T_{row,col,k} + \sum_{a} \Delta T_{a,row,col,k} \cdot$$

3 Experiments and Results

In this section we describe the experiments that we conducted and present the results we obtained. While the Knight's Tour is a puzzle or mathematical curiosity, it is a special case of an important NP-complete graph theoretic problem - that of finding a Hamiltonian path in a graph. In many applications, one is interested in finding Hamiltonian paths that optimize some figure of merit (such as tour length in TSP), so algorithms that generate Hamiltonian paths for evaluation are required. (Though in the case of TSP, finding paths is not the hard part, as the graph is usually well connected.) With this in mind, the aim of these experiments is to gather evidence on how well the ant colony algorithm does at generating as many knight's tours as possible. In addition, it is desirable that the algorithm achieves coverage of the complete set of knight's tours, and not be biased towards generating particular kinds of tours.

Firstly, we ran searches using a standard depth-first search with a fixed ordering of moves, and similar searches using Warnsdorff's heuristic to determine candidate moves. These experiments provide a baseline, indicating how difficult it is to locate complete tours. We then ran searches using our ant colony algorithm.

A naïve depth-first search was implemented, using the fixed move ordering given in [7]. For each possible starting square, we ran the search until 100,000,000 tours had been tried. The algorithm found an average of 308.6 complete tours for each square, all open.

We also implemented a variant of depth-first search using Warnsdorff's heuristic, in which a move is only considered valid if it obeys the heuristic. All these searches ran to completion, so we effectively enumerated all tours that obey the heuristic. The total number of "Warnsdorff tours" was found to be 7,894,584 - a tiny fraction of the total number of tours. About 15% (1,188,384) of these are closed tours. This variant is clearly very efficient in generating knight's tours, but it is also highly biased - indeed it is unable to reach most of the search space. The high proportion of closed tours found suggests that the portion of the search space that is reached is highly atypical.

For the ant colony algorithm, we first did some experimentation to discover a good set of parameters, settling on the following: evaporation rate $\rho = 0.25$; update rate $Q = 1.0$; strength $\alpha = 1.0$; in each cycle, start one ant from each start square; and greater pheromone update for moves near the end of a tour. If the evaporation rate is too low, there is not enough exploration, whilst if it is too high, there is not enough exploitation. Strength also affects this balance. Starting ants from all start squares in each cycle produces an order of magnitude more solutions compared to running the

search once for each starting square. For the update rule, we also tried adding a constant amount of pheromone to each edge of a partial tour, or adding an amount proportional to the length of the partial tour. Both were inferior to the chosen formula.

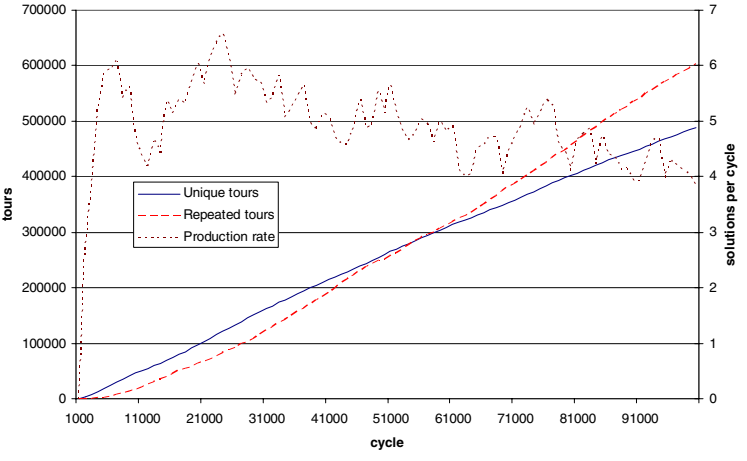

Fig. 1. Mean performance of the ant colony algorithm

Fig. 2. Pheromone patterns at the completion of two runs of the ant colony algorithm

With these choices made, we ran the ant colony algorithm 20 times for 100,000 cycles each time. The mean number of unique complete tours found in each run was 488,245.4 (with, on average, 9,192.0 closed tours), three orders of magnitude better than the genetic algorithm. A better competitor is the heuristic depth-first search, which is more efficient than the ant colony, but only finds Warnsdorff tours.

Fig. 1 shows the mean number of unique tours discovered and the number of repeated tours for progressively more cycles. It also shows the "production rate" - the

number of new tours found per cycle. Production rate increases for about the first 20,000-25,000 cycles, while the ant colony is learning a good pheromone pattern. After this, repeated tours are found, and the production rate slowly falls. A remedy might be to restart the algorithm after a few thousand cycles. We tried this idea, running the algorithm multiple times for 5,000 cycles each time. In no case were any tours discovered in more than one run. Fig. 2 shows pheromone patterns from two typical runs when the patterns have more or less converged. Each pattern has eight 8x8 grey scale rectangles. Each rectangle shows pheromone levels for one of the eight knight's moves at each square on the chessboard, darker grey indicating more pheromone. Patterns for different runs are quite different from each other.

4 Conclusion

We have introduced a new ant colony algorithm for discovering knight's tours on an 8x8 chessboard. The new algorithm is able to discover tours efficiently, without the bias of existing heuristic approaches. Just as graph theory itself was developed in the 18th century to study problems such as the Konigsberg Bridge Problem and the Knight's Tour Problem, this algorithm should be adapted easily to solve other problems involving Hamiltonian paths or cycles in other graphs.

References

[1] Murray H.J.R. (1913) History of Chess
[2] Euler L. (1766) Mémoires de l'Academie Royale des Sciences et Belles Lettres, Année 1759, vol.15, pp. 310–337, Berlin.
[3] Mordecki E. (2001) On the Number of Knight's Tours. Pre-publicaciones de Matematica de la Universidad de la Republica, Uruguay, 2001/57 (http://premat.fing.edu.uy/)
[4] 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.
[5] McKay B.D. (1997) Knight's tours of an 8x8 chessboard, Tech. Rpt. TR-CS-97-03, Dept. Computer Science, Australian National University.
[6] Warnsdorff H.C. (1823) Des Rösselsprungs einfachste und allgemeinste Lösung. Schmalkalden
[7] 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
[8] Goldberg D. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley
[9] Dorigo M. (1992). Optimization, Learning and Natural Algorithms. Ph.D.Thesis, Politecnico di Milano, Italy, in Italian
[10] Dorigo M., V. Maniezzo & A. Colorni (1996). The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41