Enumerating Knight's Tours using an Ant Colony Algorithm

Philip Hingston
Edith Cowan University
2 Bradford St, Mt Lawley, 6050
Australia
p.hingston@ecu.edu.au

Graham Kendall
The University of Nottingham
Nottingham
UK
gxk@cs.nott.ac.uk

Abstract- In this paper we show how an ant colony optimisation algorithm may be used to enumerate knight's tours for variously sized chessboards. We have used the algorithm to enumerate all tours on 5x5 and 6x6 boards, and, while the number of tours on an 8x8 board is too large for a full enumeration, our experiments suggest that the algorithm is able to uniformly sample tours at a constant, fast rate for as long as is desired.

1 Introduction

A *Knight's Tour* is a Hamiltonian path in 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 such that it visits each square of an 8x8 chessboard exactly once. Murray (Murray 1913) traces the earliest solutions to this problem back to an Arabic text in 840 ad. The text describes two tours, one by Ali C. Mani (Figure 1) and the other by al-Adli ar-Rumi (Figure 2). The second is called a *closed* tour, as the knight could complete a circuit with one more move, while the first one is merely an *open* tour. The problem has been much studied since that time. Leonhard Euler carried out the first mathematical analysis of the problem, presenting his work to the Academy of Sciences in Berlin in 1759 (Euler 1766). Other well-known mathematicians to work on the problem include Taylor, de Moivre and Lagrange.

McKay calculated the number of closed tours on a standard 8x8 chessboard to be 13,267,364,410,532 (McKay 1997). An upper bound of the number of open tours was found to be approximately 1.305×10^{35} (Mordecki 2001). The search space is even larger. For example, if we were to define a tour using a pair of integers between 1 and 8 for the position of the start square, and a sequence of 63 such integers to choose which of the possible 8 knight's move to take for each move, we would be searching a space of size 8^{65}, or approximately 5×10^{58}.

It is not surprising, given its long history, that there are many approaches for producing knight's tours. A depth-first search, with backtracking, is perhaps the most obvious, though rather slow. A heuristic approach due to Warnsdorff, although dating back to 1823, is perhaps the most widely known approach (Warnsdorff 1823). 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 towards the end of the tour the knight will visit squares that have more moves available. Using the heuristic greatly increases the likelihood of finding a complete tour, but obviously tours that do not satisfy the heuristic cannot be discovered.

34	47	22	11	36	49	24	1
21	10	35	48	23	12	37	50
46	33	64	55	38	25	2	13
9	20	61	58	63	54	51	26
32	45	56	53	60	39	14	3
19	8	59	62	57	52	27	40
44	31	6	17	42	29	4	15
7	18	43	30	5	16	41	28

Figure 1 - Open tour due to Ali C. Mani

60	11	56	7	54	3	42	1
57	8	59	62	31	64	53	4
12	61	10	55	6	41	2	43
9	58	13	32	63	30	5	52
34	17	36	23	40	27	44	29
37	14	33	20	47	22	51	26
18	35	16	39	24	49	28	45
15	38	19	48	21	46	25	50

Figure 2 - Closed tour due to al-Adli ar-Rumi

0-7803-9363-5/05/$20.00 ©2005 IEEE.

When only one or a few Knight's tours are wanted, a number of efficient methods exist, for example, using divide and conquer methods (Parberry 1997), or neural networks (Takefuji and Lee 1994).

A recent approach to finding many knight's tours utilised a genetic algorithm (Gordon and Slocum 2004). The authors used a simple genetic algorithm (Goldberg 1989), encoding an attempted knight's tour as a sequence of 64x3 bits. Each triple represents a single move by the knight, with the fitness being defined as the number of legal moves (maximum = 63) before the knight jumps off the board or revisits a square. The last 3 bits were ignored as the authors were not concerned with finding closed tours. If a candidate tour led to an illegal move, a repair operator was used to check the other seven possible knight's moves and replace the illegal move with a legal move if there is one, and then attempt to continue the tour, doing more repairs if needed. Without this repair operator, the genetic algorithm found no complete tours. Adding repair functionality allowed tours to be discovered. The maximum number of tours they reported in a single run, which consisted of 1,000,000 evaluations, was 642, a rate of 0.000642 tours per attempt. By contrast, a naïve depth first search yields approximately 0.000003 tours per attempt.

In (Hingston and Kendall 2004), the current authors introduced an ant colony optimisation algorithm for generating knight's tours, which produced about 0.076 tours per attempt. In this paper, we improve on the algorithm, and also investigate its performance on smaller boards, where a complete enumeration is possible, making analysis easier.

2 The Ant Colony Optimisation Algorithm

In this section, we describe the ant colony optimisation algorithm that we designed to enumerate knight's tours. We first review the basics of ant colony optimisation (ACO), and present the ACO algorithm that we introduced in (Hingston and Kendall 2004). It is similar to the well-known Ant Systems algorithm introduced by Dorigo *et al.* (Dorigo, Maniezzo et al. 1996). We then describe a new modification utilising multiple restarts of the earlier algorithm.

2.1 Ant Colony Algorithms

Ant colony optimisation algorithms are based on the natural phenomenon 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. Ant colony optimisation (ACO) algorithms were introduced by Marco Dorigo in his PhD thesis (Dorigo 1992) and later in the seminal paper in this area (Dorigo, Maniezzo et al. 1996). In (Dorigo, Maniezzo et al. 1996) 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. Ants can increase their chance of finding the shorter route by preferentially choosing the one with more pheromone. There is positive feedback, in that the more successful a behaviour proves to be, the more desirable it becomes. This form of behaviour is known as *stigmergy* or *autocatalytic behaviour*.

This idea has been transformed into various search algorithms, by augmenting the probabilistic nature of ant movements following pheromone trails, with a problem specific heuristic. See (Cordon, Herrera et al. 2002) for a review of algorithms based on this idea. In the most famous example, ants can be used to search for solutions for the traveling salesman problem. Each ant in the colony traverses the city graph, depositing pheromone on edges between cities. High levels of pheromone on an edge indicate that it is part of relatively shorter tours found by previous ants. When deciding when to move from one vertex (city) to another, ants take into account the level of pheromone on the candidate edges along with a heuristic value (distance to the next city for the TSP). The combination of pheromone and heuristic probabilistically determines which city an ant moves to next.

2.2 One-shot Ant Colony Algorithm for Enumerating Knight's Tours

In this section, we describe the ant colony optimisation algorithm introduced in [**Error! Bookmark not defined.**] to discover knight's tours. We will call this algorithm the ant colony enumeration (ACE) algorithm. As for the TSP, ants traverse a graph, depositing pheromones as they do so. 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 successfully visits all the squares on the board will have discovered a knight's tour. Here is a pseudo-code of the algorithm:

```
Initialise the chessboard
For each cycle
  Evaporate pheromones
  For each starting square
    Start an ant
    While not finished
      Choose next move
      Move to a new square
    If tour is complete, save it
    Lay pheromone
  Update pheromones
```

Note that we found it advantageous to search for solutions from all starting squares simultaneously, rather

than running the algorithm multiple times, once for each starting square. We hypothesise that there is information sharing between ants starting on different squares. That is, an ant starting on one square can utilise the knowledge gained by ants starting on more remote squares – knowledge that is more difficult 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*. Note that 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 the corresponding edges. We use $dest_{row,col,k}$ to denote the square we would reach if we followed edge k from square (row, col).

Initialising the chessboard

Initially, a small amount of 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. This prevents the level of pheromone becoming unbounded, and allows the ant colony to "forget" old information. In our algorithm, we implemented this by reducing the amount of pheromone on each edge once per cycle, using the update formula:

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

where $0 < \rho < 1$ is called the *evaporation rate*. After some experimentation, we set this to 0.25.

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, $tabu = \{(startRow, startCol)\}$. Each ant also remembers its start square, and its sequence of moves. Initially, $moves = <>$, an empty list.

Choosing the next move

To choose its next move, an ant computes, for each edge leading to a square not in its tabu list, the following quantity:

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

where $\alpha \geq 0$, the *strength*, is a constant that determines how closely the ant follows the pheromone. It then chooses edge k with probability:

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

Thus the ant is more likely to choose moves whose edges have more pheromone.

Moving to the new square

In some ACO's, ants deposit pheromone as they traverse each edge. Another alternative, which we use in our algorithm, is to deposit pheromone only after the ant has completed an attempted tour. Hence, having chosen edge k, the ant 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 itself on a square where all potential moves lead to a square in her tabu list. If it has visited all the squares on the chessboard, it has succeeded in finding a valid knight's tour. If not, it has completed a partial tour.

Lay pheromone

When she has finished her attempted tour, the ant retraces its steps and adds pheromone to the edges that it 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:

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

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. Actually, it is never 0 for an edge that is used. For example, the shortest possible failed attempt has three moves. In these experiments, we set Q to 1.0. Once all the ants have completed their attempted tours, we then update the pheromone trails using the formula:

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

2.3 Ant colony algorithm with multiple restarts

In our previous work (Hingston and Kendall 2004), we ran the ACE for only 100,000 cycles at a time, i.e. 6,400,000 attempted tours. We noted that the rate of production of new tours initially rises as the colony learns a good arrangement of pheromone levels, then starts to drop away when learning stops and the ants start to repeat tours found earlier. We speculated that this problem might be overcome by periodically resetting the pheromone levels, so that the ants would explore different areas of the search space. We call this multiple-restart version of the algorithm MACE. Pseudo-code for MACE is just

```
For each repeat
  Initialise the chessboard
  Execute the one-shot algorithm
```

We have since found that performing multiple restarts is essential when working on smaller board sizes, because

the one-shot algorithm quickly converges and covers only a limited portion of the search space.

An idea similar to mutiple restarts has been used before in the context of optimisation. The MACO algorithm (multiple ant colony optimization) proposed by Sim *et al.* (Sim and Sun 2003) uses multiple ant colonies, each with their own type pheromone. Ants are repulsed by the pheromone of ants from different colonies. This is applied in networking applications, where several routes through a network are sought and the desire is to minimise interference. It is doubtful if this method would scale to our situation, where thousands of restarts are used. Multi-colony ant algorithms have also been studied as a way to parallelize ACOs (see e.g. (Middendorf, Reischle et al. 2000)).

3 Choosing alpha and the number of cycles

To implement the multiple restart algorithm, we must decide how often to restart – i.e. how many cycles to use within each repeat. In this section we describe the method we used to make this choice. The method also allows us to choose a suitable value for α, the strength parameter.

For a given value of α, we plot the number of attempted tours against the average production rate up to that number of attempts. This average increases as the ants learn a good pattern of pheromones, and then starts to drop once the pheromone levels have converged and ants begin to repeat earlier tours. We choose the number of attempts that gives the peak value of average production rate. Finally, we select the value of α for which this peak average production rate is greatest.

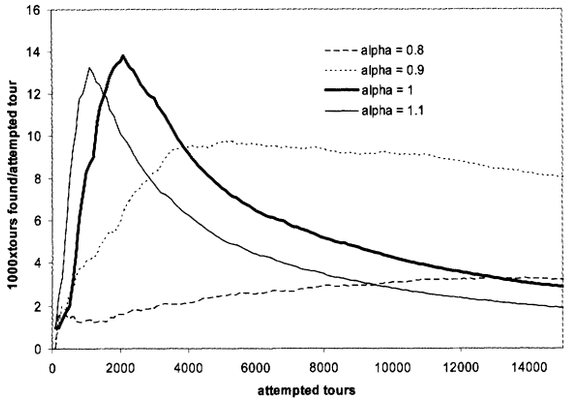

Figure 3 - Mean production rates for 5x5 knight's tours

Figure 3 illustrates the method for a 5x5 chessboard, where the best rate was approximately 0.0138 tours per attempted tour, which occurs with an α value of 1.0, after 2100 attempts (about 84 cycles). Note that each data point on the graph is actually an average value over 10 runs of the one-shot algorithm. The obvious extension of this method could be used to select values for other algorithm parameters – the evaporation rate and update rate.

4 Experiments

In this section, we report on experiments comparing the efficiency and effectiveness of MACE with that of alternative methods. While we are most interested in the traditional 8x8 chessboard, we begin with 5x5 and 6x6 boards, as these are small enough to make complete enumeration practical, affording an opportunity to analyse the performance of the algorithm more completely. (Note that there are no knight's tours for 3x3 or 4x4 boards.) We then extrapolate to the 8x8 case. We have not considered the 7x7 case, as it is too large for complete enumeration, and not of much interest of itself.

4.1 The 5x5 board

Table 1 shows the results from running a full enumeration by depth-first search on a 5x5 board. In total 1,728 tours were found (none of which is a closed tour) from 14,897,808 attempts. This is a production rate of about 0.000116. Table 2 shows the number of "Warnsdorff" tours on a 5x5 board (320 in total). Using the heuristic, only 856 attempts are needed to find these 320 tours – a much higher production rate (0.374) - but less than 20% of the existing tours can be found this way. For larger board sizes, this proportion is much smaller.

As expected, the tables have several symmetries, so we can determine all the table entries from those in a triangle in the top left of the table. For larger boards, we will only show that triangle.

Using α = 1.0 and 84 cycles per restart, we ran the multiple restart ant colony algorithm until all 1,728 tours were found. We repeated this procedure for 20 runs, recording the number of attempted tours needed for each run. The mean number of attempted tours required was 1,734,370, with a minimum of 1,231,451 and a maximum of 2,521,101. Thus, the average production rate was 0.001 tours per attempt, nearly 10 times the rate achieved by depth-first search. This is less than the rate for one repeat (0.0138), because the sets of tours found in each repeat are not disjoint sets, so some of the tours found later in the search will have already been found in earlier repeats. Nevertheless, the ant algorithm requires far less attempts than depth–first search does. However, the computational time is longer for the ant algorithm as each tour carries more computational overhead: each ant must calculate the probabilities needed to choose each move; the pheromone levels need to be updated after a tour has been completed by 25 ants (i.e. one in each square) and so on.

Figure 4 shows the number of tours found versus the number of attempts for a single run of various algorithms. The ant algorithm is initially very fast, but tails off later. The first 1,700 solutions are found in about the same time it takes to find the last 28. We picture each repeat as exploring a randomly chosen patch of the search space. Patches explored by later repeats overlap with earlier ones. We can use this picture to develop an empirical model of the performance of the algorithm: let us assume that each patch is randomly placed, that within each patch, each tour is equally likely to occur. Then at each

attempt, the algorithm is equally likely to discover any tour (in this case, the probability of discovering a particular tour on a particular attempt would be p = 0.0138/1728). The probability that a particular tour has not been disovered by the n^{th} attempt would therefore be $(1-p)^n$. Hence, the expected number of tours found by n attempts would be $N \times \left(1-(1-p)^n\right)$, where N is the total number of tours. This is plotted in Figure 4 as the dotted line labelled "formula". While the fit is quite good, we shall see in the 6x6 case that it is not quite correct – the assumption of equal probabilities for all tours must be wrong.

Table 1 Exhaustive depth-first search on a 5x5 board

Total tours found = 1,728. None of the tours is closed. The table entries are tours tried/**tours found**

625,308 **304**	727,156 **0**	595,892 **56**	727,156 **0**	625,308 **304**
727,156 **0**	601,036 **56**	384,804 **0**	601,036 **56**	727,156 **0**
595,892 **56**	384,804 **0**	252,400 **64**	384,804 **0**	595,892 **56**
727,156 **0**	601,036 **56**	384,804 **0**	601,036 **56**	727,156 **0**
625,308 **304**	727,156 **0**	595,892 **56**	727,156 **0**	625,308 **304**

Table 2 Warnsdorff tours on a 5x5 board

The number of Warnsdorff tours is 320. The table entries are tours tried/**tours found**

32 **32**	72 **0**	20 **16**	72 **0**	32 **32**
72 **0**	20 **16**	2 **0**	20 **16**	72 **0**
20 **16**	2 **0**	64 **64**	2 **0**	20 **16**
72 **0**	20 **16**	2 **0**	20 **16**	72 **0**
32 **32**	72 **0**	20 **16**	72 **0**	32 **32**

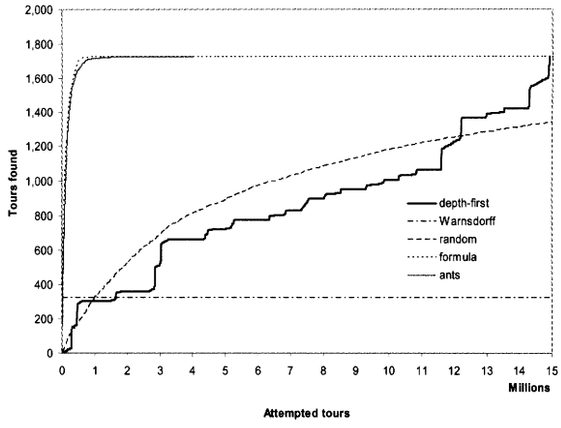

Figure 4 - Tours found on a 5x5 chessboard using various algorithms

We have also included another line on the plot, labelled "random". This was obtained by running the ant colony algorithm with learning (i.e. pheromone update) disabled. This corresponds roughly to making random choices at each move, and it does surprisingly well on 5x5 boards, at least compared to a full depth-first search. The effect of learning by the ant colony is clearly shown in the comparison between ants with learning and ants without. For larger board sizes, this random variant is not able to find tours at a reasonable rate.

4.2 The 6x6 board

We now turn to the case of a 6x6 chessboard. This is more challenging than the 5x5 one, but it is still possible to completely enumerate all the tours.

Table 3 shows the number of tours found in an exhaustive enumeration on a 6x6 board. Just the top-left corner of the table is shown – the rest can by filled in using symmetries. Depth first search required 210,036,568,392 attempts, giving a production rate of 0.0000316. We have not shown the table for Warnsdorff tours, but there are 1,984, of which 360 are closed, requiring 2,914 attempts to find them. This is such a tiny proportion of the total number of tours that we will not consider the Warnsdorff heuristic further.

Using the technique described in Section 3, we determined to use α = 1.0 (with a mean production rate of 0.0467), and 260 cycles per repeat for the ant algorithm.

Figure 5 shows the performance of the various algorithms. Note that we have switched to a log-scale on the x-axis so that the performance of all the algorithms can be seen together. The plot clearly shows the superiority of the ant colony algorithm over depth-first search. The ant colony requires roughly 1% of the number of attempts that depth-first search does. The Warnsdorff and random algorithms would barely climb above the x-axis were we to include them in this plot. Also shown is the predicted performance on the ant colony algorithm based on the formula developed above. The formula gives a fairly good fit up until around 30,000,000 attempts, or 850,000 tours found, after which the actual production

rate drops below the prediction. We hypothesise that this is because some tours are actually harder to find than others, so the production rates drops after most of the "easy" tours have been found.

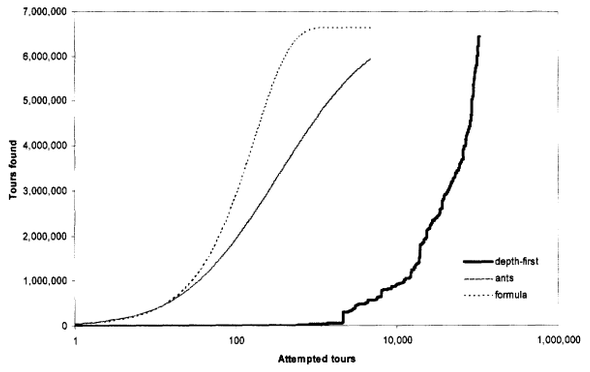

Figure 5 - Tours found on a 6x6 chessboard using various algorithms

We should also mention at this point an implementation problem that we encountered. For boards of 6x6 and larger, actually storing the tours as they are found, so that we can check whether the tour has already been found, becomes an issue. Note that uniqueness is guaranteed for the depth-first search, so there is no need to store the tours, but we need to check for uniqueness with the ant algorithm. We found that there were too many tours to be kept in memory, even with some compression, and storing, retrieving and checking them from file is a considerable performance penalty. We therefore switched to a hashing scheme to check uniqueness.

Table 3 Exhaustive depth-first search on 6x6 board
Total tours found = 6,637,920. 19,724 are closed tours. The table entries are tours tried/**tours found.** Note that the 6x6 table can be completed using symmetry.

7,083,683,400 **524,486**		
7,294,926,164 **289,050**	6,543,995,877 **173,402**	
6,284,771,475 **115,837**	4,394,081,182 **49,578**	2,933,905,179 **52,662**

The scheme used is as follows: we created H distinct hash functions, over an address space of size 12x1024x1024. We created H bitmaps of this size. When a tour was found, we set the corresponding bits in the H bitmaps as determined by the hash functions. When a candidate new tour was found, we checked to see whether the H bits for the candidate tour were already set. If any bit was not set, then the candidate is guaranteed to be a new tour. If all bits were already set, then it is likely that the tour has already been found. Assuming the hash functions are independent and generate uniformly distributed values, the chance of a collision even when nearly all the tours have been found is less than roughly 2^{-H}. We set H to 25, sufficient to make the chance of a collision around 1 in 33,000,000. In practice, this means that we should expect very few, if any, collisions. Some testing confirmed that this was the case. Note that, even if there were a few collisions, this would make our performance measurement for the ant colony algorithm a conservative one.

4.3 The 8x8 board

Finally, we return to the original problem, enumerating knight's tours on an 8x8 chessboard. As mentioned at the start, the number of tours here is enormous, and we cannot even consider attempting to enumerate them all. Hence we can do no better than to run the algorithms for a certain number of attempted tours or for a certain amount of time, and examine the performance up to that point.

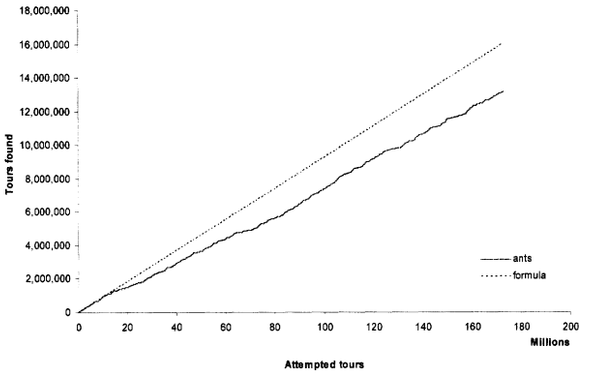

Figure 6 - Tours found on an 8x8 chessboard

Figure 6 shows the number of tours found by MACE on an 8x8 chessboard on one such run. We used the method of Section 3 to determine settings of $\alpha = 1.0$, and 27,000 cycles per repeat (mean production rate of 0.0926). At the end of this run, MACE had found 13,124,464 tours from 172,800,000 attempts, a rate of 0.076. The formula predicts 16,000,000 solutions from this number of attempts. We believe that the reason for the discrepancy in this case is that the method of Section 3 yields a biased estimate of the production rate (since we are choosing the maximum value from a set of samples). The effect due to non-uniform probabilities that we saw in the 5x5 case and 6x6 case would not be apparent here as we have only enumerated a tiny fraction of the existing tours. The rate of 0.076 agrees with the earlier value found in (Hingston and Kendall 2004).

By contrast, after the same number of attempts, depth-first search had found 6 tours. This is why we haven't

included depth-first search in the plot. After 6,129,510,000 attempts, the total number of tours found was 5,728. Recall that the genetic algorithm approach had a production rate of only 0.000642, supposing that this could be maintanied over longer runs. Warnsdorff's heuristic has already been discounted because of its lack of coverage.

For interest, Table 4 shows the percentage of tours found for each starting point. It can be seen that the corners and the centre are the easiest places to start. The corners might be expected to be relatively easy, because there are only two ways to move into or out of these squares, so getting them out of the way early should help (an application of Warnsdorff's heuristic). We have no explanation as to why the centre squares are relatively easy. The hardest are the squares that are a knight's move from the corners. This may be because a tour starting from one of these squares must either jump straight to the corner or end in the corner, so the other 5 choices for the first move make completing the tour difficult.

Table 4 - Percentage of tours found for each starting square on an 8x8 board

1.93	1.66	1.49	1.80	1.79	1.55	1.72	1.93
1.69	1.52	0.93	1.45	1.48	1.03	1.51	1.64
1.51	1.06	1.47	1.65	1.70	1.44	1.00	1.47
1.76	1.42	1.63	1.93	1.95	1.69	1.41	1.81
1.79	1.50	1.63	1.90	1.97	1.65	1.46	1.79
1.49	1.09	1.43	1.63	1.68	1.46	1.05	1.48
1.77	1.51	0.93	1.44	1.45	0.93	1.53	1.70
1.92	1.66	1.41	1.82	1.77	1.50	1.67	1.94

5 Conclusions

We have shown how an ant colony optimisation algorithm can be employed to enumerate knight's tours on variously sized chessboards. The method is able to generate nearly uniform samples of the solution space using several orders of magnitude fewer attempts than other methods.

This is an unusual application for ant colony optimisation algorithms, which are usually used to search for a single (optimum) solution. With the knight's tour, we are interested in finding all the (equally) optimal solutions. There is always a tension in stochastic search methods like ACO, evolutionary algorithms, simulated annealing etc between *exploration* and *exploitation.* In this kind of application, while the exploitation component is present (compare MACE with random search as in Figure 4) the exploration component of the search is emphasised. It would be interesting to know what other problems exist where all solutions are required.

There are various ways in which MACE might be further enhanced. For example, depth-first search can be sped up by recognising partial tours as hopeless earlier, and we could do the same with MACE. We could add a local search component, or try some of the other enhancements and variations developed for other ant colony optimisation algorithms. We experimented with adding a heuristic component to the move selection rule (based on the Warnsdorff heuristic), as is normally done in ant colony optimsation. Though this increased the production rate, we abandoned it because it biased the distribution of tours found. We could have spent more effort to optimise learning rate and evaporation rate.

In future work, we intend to explore some of these options in order to tackle several related problems: large boards and magic tours. Large boards are interesting because Warnsdorff's heuristic becomes very slow and fails on some board sizes. There are other heuristics, but the problem is far from solved. The problem of finding magic tours is also difficult. A tour is *magic* if the table of numbers formed as in Figure 1 and Figure 2 forms a magic square (i.e. the row, column and diagonal sums are all equal). A tour is *semi-magic* if just the row and column sums are equal. For example, there are no magic tours on an 8x8 board, and there are only 2,240 semi-magic ones. Magic tours are rare and difficult to find. Perhaps MACE can be used as the basis for an efficient search for magic square

References

Cordon, O., F. Herrera, et al. (2002). "A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends." Mathware and Soft Computing **9**(2-3): 141-175.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms. Milan, Politecnico di Milano.

Dorigo, M., V. Maniezzo, et al. (1996). "The Ant System: Optimization by a Colony of Cooperating Agents." IEEE Transactions on Systems, Man, and Cybernetics-Part B **26**(1): 29-41.

Euler, L. (1766). "Solution d'une question curieuse qui ne paroit soumise a aucune analyse." Mémoires de l'Académie Royale des Sciences et Belles Lettres de Berlin **15**: 310-337.

Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley.

Gordon, V. S. and T. J. Slocum (2004). The Knight's Tour - Evolutionary vs. Depth-First Search. IEEE Congress on Evolutionary Computation (CEC'04), Portland, Oregon.

Hingston, P. and G. Kendall (2004). Ant Colonies Discover Knight's Tours. AI 2004: Advances in Artificial Intelligence: The 18th Australian Joint Conference on Artificial Intelligence, Cairns, Springer.

McKay, B. D. (1997). Knight's tours of an 8x8 chessboard, Department of Computer Science, Australian National University.

Middendorf, M., F. Reischle, et al. (2000). Information Exchange in Multi Colony Ant Algorithms. IPDPS 2000 Workshops.

Mordecki, E. (2001). "On the number of Knight's tours." Pre-publicaciones de Matematica de la Universidad de la Republica, Uraguay **2001/57**.

Murray, H. J. R. (1913). History of Chess, Clarendon Press.

Parberry, I. (1997). "An efficient algorithm for the Knight's tour problem." Discrete and Applied Mathematics **73**: 251-260.

Sim, K. M. and W. H. Sun (2003). " Ant Colony Optimization for Routing and Load Balancing: Survey and New Directions." IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans **33**(5): 560-572.

Takefuji, Y. and K.-C. Lee (1994). "Finding knight's tours on an M×N chessboard with O(MN) hysteresis McCulloch-Pitts neurons." IEEE Transactions on Systems, Man and Cybernetics **24**(2): 300-306.

Warnsdorff, H. C. v. (1823). "Des Rösselsprungs einfachste und allgemeinste Lösung." Schmalkalden.