Improved Knight Coverings

Frank Rubin
59 DeGarmo Hills Road
Wappingers Falls, NY 12590
http://www.ContestCen.com

Abstract: A computer program for finding knight coverings of a chessboard is described, and some improved coverings for boards of sizes 19x19 through 25x25 are shown.

Introduction: I recently added a puzzle concerning chess knights to my website *www.ContestCen.com*. In order to provide the answer to the puzzle, I wrote a simple computer program to solve it. I soon realized that the program could easily be modified to solve the knight covering problem. To my surprise, the resulting coverings matched all of the current best coverings up through the 18x18 board, and beat the current best for sizes 19x19 through 23x23 and 25x25 recently reported by Fisher [1], who also describes the history of the problem.

The knight covering problem is a special case of the graph domination problem, where we seek a minimum subset of the nodes of an undirected graph such that every node in the graph is either in the subset or adjacent to a node in the subset. For the knight covering problem the nodes represent the squares of the chessboard, and two nodes are adjacent if there is a knight move between the corresponding squares.

The program

The program is a simple recursion. Here is a brief outline of the algorithm.

Start with an empty board.

Repeat

Choose the next square that is neither attacked nor occupied.

Try a knight in every position that either attacks or occupies the chosen square.

Evaluate each such position.

If the number of covered squares is now adequate, continue to add more knights, otherwise remove the newest knight.

Until the board is covered.

These steps will be described in detail in the following sections.

Search order

The program searches the board for the next uncovered square in diagonal order, like

1	2	4	7	11
3	5	8	12	16
6	9	13	17	20
10	14	18	21	23
15	19	22	24	25

Thus square (1,1), the upper left square, is considered first, then (1,2), (2,1), (1,3), and so forth. This keeps the covered area compact, rather than strung out, which would occur if the search had been done straight across the rows.

Duplicate detection

It is desirable to detect and remove duplicate board positions early in the process to avoid considering the same placements many times. Duplicate detection is largely a question of keeping enough information about previous trials, and requires a great deal of storage. In this case the program was running on an old (pre-Pentium) PC, and had very limited storage accessible. The approach was to use two hash tables. A 31-bit pseudorandom number was assigned to each square on the board, and the hash value of a position was the xor (exclusive-or) of these values for each occupied square.

Two tables each containing 1 bit for each possible hash value were kept. The checks were made at two preset points in the generation process, when roughly 20% and 30% of the knights were placed, respectively. When a board position of the selected size was generated, the bit was tested. If it were 0, the program set the bit and proceeded with the recursion; otherwise it abandoned the position as a duplicate. By choosing suitable depths in the recursion for which the number of positions generated was considerably smaller than the table sizes, the number of false positives was kept low, so the chance of abandoning a favorable position was likewise kept low.

Redundant knights

Another strategy for avoiding wasted effort is to remove redundant knights. As additional knights are placed on the board, sometimes all of the squares covered by one knight become covered or occupied by later knights. A board position containing redundant knights can never lead to a minimum solution, so such positions should be abandoned as early as possible.

The check for redundant knights is relatively time-consuming. After some experimentation, the best trade-off was to perform the check after placing every fourth knight.

Evaluating board positions

The key to keeping the number of board positions manageable during the recursion is to evaluate each trial placement, and pursue only the most promising. The evaluation is very simple, the program counts the number of squares that are covered. A position with 20 knights covering 132 squares is clearly more promising than one covering only 131 squares.

A table is kept giving a cutoff for the number of squares that must be covered by a given number of knights. Any position with fewer squares covered is abandoned.

This is the tricky part. There is no *a priori* way of establishing these cutoffs. It is initially done by hand. A rough guess is made, the program is run, and the number of board positions considered for each number of knights is printed at the end. If it looks too low the cutoff is reduced; if it looks too high the cutoff is increased.

As each board size is tried, the table used for the last board gives the starting point for the next larger board. After gaining some experience with smaller boards, the cutoffs for the next-larger board can be estimated with decent accuracy.

Duplicate printing

Detecting duplicate solutions is done separately from detecting duplicate board positions. The same hash values are used, but this time the hash value for each solution is kept, along with a second hash value derived by xoring the hash values of the squares immediately to the right of each knight. This effectively gives us a 62-bit hash value for each solution, making the likelihood of a false match negligible. The hash values for the last 25 solutions are kept in a list. Each new solution is added to the top of the list, and each time a solution is regenerated it is moved to the top of the list. This reduces the search time for detecting duplicate solutions. Typically 75% to 90% of solutions are duplicates.

Eliminating humans

The human element can be largely eliminated by using an additional table. In this table we keep the number of board positions that we would like to consider for any given number of knights. These numbers are based on our experience with the running times for the program, and how much computer time we are willing to devote the problem. When the number of board positions exceeds this limit the program increases the cutoff number.

At the end of the run, these new cutoffs can be written directly into the program source code. In this way, the computer can refine its own program.

Results

The program has matched the previous best coverings [1] for board sizes from 10x10 through 19x19 and 24x24. For sizes 20x20 through 23x23 and 25x25 the program has improved on the best coverings, as follows:

Board size	20	21	22	23	24	25
Prior best	63	71	76	84	88	97
New best	62	68	75	83	88	96

The program also beat the simulated annealing solutions of Jackson and Pargas [2] for all boards from 16x16 through 20x20 by 2 to 3 knights.

What's next?

The question naturally arises, can these coverings be improved? Which boards are most likely to have smaller coverings? The following table can help choose the best place to search. It shows the density, or mean number of squares covered by each knight for board sizes 3x3 through 26x26. A table shows the small increases and decreases in density better than a graph.

3	3	3	11	21	5.762	19	57	6.333
4	4	4	12	24	6	20	62	6.452
5	5	5	13	28	6.036	21	68	6.485
6	8	4.5	14	32	6.125	22	75	6.452
7	10	4.9	15	36	6.25	23	83	6.373
8	12	5.333	16	40	6.4	24	88	6.545
9	14	5.786	17	46	6.283	25	96	6.510
10	16	6.25	18	52	6.231	26	102	6.627

We would expect the table to be irregular for small boards because of parity, edge effects, and granularity. As boards grew larger, we would expect a fairly steady improvement as the center of the board started to dominate. There should also be some sharp increases where highly patterned very dense coverings occur. Such increases are seen at 10x10, 12x12, 16x16 and 24x24.

In an earlier version of this paper, the best covering found for 19x19 was 58, giving a density of 6.224, indicating that 19x19 was a prime choice for improvements. After producing the table, and seeing this, I retried 19x19 and was able to reduce the covering to 57 knights. The best remaining places to seek improvement appear to be 18x18 and 23x23.

Non-attacking knights

A variant of the knight covering problem is the non-attacking knight covering problem, where we add the additional constraint that the knights cannot attack each other. The anonymous referee has pointed out that the 21x21 covering can be changed to a non-attacking cover by moving the knight at (17,2) to (15,1). It would be straightforward to modify the program to produce non-attacking coverings.

Sample Coverings

For large boards it is not yet feasible to list and categorize all of the minimal coverings. One sample for each size is illustrated. These were chosen for their elegance, using as few clusters of knights as possible, and prefering symmetric clusters and cluster placements where possible.

References

[1] David C. Fisher, "On the NxN Knight Cover Problem" submitted to Ars Combinatoria.

[2] Anderson H. Jackson, Roy P. Pargas, "Solutions to the NxN Knights Covering Problem," J. Recr. Math. 23(1991), pp 255-267.