Breaking Down the “N-Knights” Problem in Chess

Vasupradha Ramji

Research for Introduction to Discrete Structures

COT3100H

April 26th, 2024

Introduction

The game of Chess' existence has been dated back to as early as the 6th century AD, along with numerous variations and alterations of the game's pieces and rules. Within chess itself lies a vast range of mathematical and algorithmic problems and dilemmas that have proven useful even outside of merely a strategic game. It has consequently garnered revolutionary breakthroughs in mathematics and theory, including a plethora of solutions proposed by mathematicians regardings its logistics and complexity. For instance, the game of Chess has been dissected into calculations using set theory, game theory, probability, and graph theory among many other concerts prevalent in the realm of mathematics and algebra.

In addition to the nature of the game itself alongside the gameplay, chess has also served as the setting for a range of abstract questions and theories that expand it further than an eight by eight board with two players. Rather, these discrete questions pose hypothetical scenarios outside of the game and incorporate algorithms and logic external to the classic game. One such problem is the Eight Queens Problem.

The central premise underlying the Eight Queens Problem is that given a chessboard of dimensions 8 squares by 8 squares and, hypothetically, 8 queen pieces, a solution must be determined in which each of the eight queens can be arranged on the board without risking any two Queens from being attacked by another. On a standard chessboard, there are 4,426,165,368 possible arrangements of 8 different queens on a chessboard. The number of possible solutions that abide by the above constrictions is 92. However, the actual number of distinct solutions, which are neither symmetrical or rotated from any other solution, dwindles down to 12, as

reiterated by Kesri and Pattnaik from the School of Computer Engineering in KIIT University, India (Kesri and Pattnaik, 2012, pg. 1).[1] Having published their paper in the International Journal of Computer Applications, they had worked with Vaibhav Kesri from NIT Kurukshetra, India in examining the nth Queens problem further and pinpointing the derivation of a unique solution as they had defined it. Clearly, the problem harbors complexity and requires an algorithmic and mathematical approach to encapsulate the complete scope of it.

Real World Applications

In addition to its applications in mathematics, it also harbors real-world applications and has served as a basis for the logic behind concepts in other fields as well. For instance, Rok Sosic and Jun Gu had published a case study on a local search algorithm that they had developed as an alternative to the backtracking search algorithm, which various other researchers had utilized. In their own words, "Some direct applications of this result include very large scale integration (VLSI) testing, air traffic control, modem communication systems, data/message routing in a multiprocessor computer, load balancing in a multiprocessor computer, computer task scheduling, computer resource management, optical parallel processing, and data compression." (Sosic and Gu, 1994, pg. 661).[2]

[1] Kesri, Vishal. Kesri, Vaibhav. Pattnaik, Prasant Ku. An Unique Solution for N queen problem. 2012. Pg. 1-2

[2] Sosic, Rok. Gu, Jun. Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem. 1994.

Background

Admittedly, there is a method to the madness involved in this problem, some of which I have uncovered through experimenting and working through the logic of the problem in my own free time. As the Queen piece in chess can either move diagonally and horizontally any number of spaces on the board as long as its destination is vacant, essentially, it encompasses the moves of a Pawn, King, Rook, and Bishop. The only piece that it cannot move like is the Knight, which shifts up or down two blocks and either left or right one block in an L-shape movement, or similarly, one block up or down and two blocks left or right. For this reason, if the Queens were placed in such a way that they were each a Knight's move away from each other, they cannot kill each other.

Of course, there are additional restrictions such as only one Queen being in each row and column, as well as no two Queens being diagonal to one another. Employing this logic, I have been able to find two distinct solutions to the problem by placing the Queens abiding by these restrictions. Moreover, the same research paper published by Kesri and Pattnaik has mentioned logic that aligns with my findings as well, which is evident in Figures 2 and 3, which illustrate the concept of the Queen not being able to move as Knight would and a solution that reflects this respectively (Kesri and Pattnaik, 2012, pg. 2). They have even taken it a step further in attempting to determine unique solutions through incorporating values in the empty squares as well and how many Queens can attack those particular spaces and uncovered an algorithm that runs in linear time.

Mathematicians and analysts of the Eight Queens problem have also developed a variation known as the n-Queens Problem, in which the scope is expanded to accommodate an n number of Queens followed by a board of n * n squares. Evidently, many such variations exist in a similar manner. To demonstrate, there have been a variety of studies conducted on the problem if it involved different pieces, such as a Rook or a Pawn as demonstrated by a paper by R. Douglas Chatham, in which he writes "The proof also involves taking known solutions to the M queens problem for some M < N, and adding extra rows, columns, queens, and pawns in systematic ways to obtain patterns that can be verified as being N + k queens problem solutions." (Chatham, 2017, pg. 205).[3] In his approach, he formulated the composition of each solution into matrix representations as shown in Figures 3 and 4, which each demonstrate a possible solution to two different N + k problems (pg. 206), and has provided equations and formulas that he had constructed in explanation to this, while even considering how the problem would vary if the Queens were replaced by Rooks in order to further prove the alternating sign matrix algorithm.

It is evident that there exist a plethora of methods and algorithms that have been used to find solutions and expand this problem further, such as the backtracking method as detailed by Jay P. Fillmore and S. G. Williamson's paper on backtracking, in which they consider how the backtracking algorithm, which is "a search algorithm to determine all elements (x_1, …, x_n) of a Cartesian product X_1 * … * X_n, which satisfy a given true-false valued "criterion" function: $\varphi(x_1, ..., x_n)$ = true." (Fillmore and Williamson, 1974, pg. 41), can be combined with recursion in order to generate solutions for the Eight Queens Problem (pg. 45).[4] This can also be extended to the

[3] Chatham, R. Douglas. Reflections on the N + k Queens Problem. 2017.
[4] Fillmore, Jay P. Williamson, S. G. On Backtracking: A Combinatorial Description of the Algorithm. 1974. Pg. 45-49

n-Queens problem and can be represented by a matrix, as the aforementioned research paper by Chatham has demonstrated.

Clearly, within the Eight Queens problem itself and the chess board is a vast scope of questions to be answered and mathematical theories that have been developed. Through exploring previously done research into this topic and familiarizing myself with the academic discussions that have been published thus far, I have gained a good general understanding of how the problem and theories work. Therefore, I intend to extend my research further than the standard Eight Queens Problem and incorporate an n number of Knights instead. By experimenting with the Knight piece and how this would alter the problem and solutions, I will be able to gain a deeper understanding about the mathematics in the background.

Research Inquiry

In order to approach an “n-Knight’s” Problem, I began familiarizing myself with any other research that had been done towards this approach, as well as identifying any gaps in which I can objectively formulate assumptions and solutions as well as create a basis for my research. In particular, I aim to examine a standard scale 8 x 8 chess board and determine the maximum number of knights pieces that can be placed such that no two knights are at risk of attacking one another. A simple solution has already been uncovered in which each Knight can be placed in either the black or white squares, due to a Knight in a black square always moving to a white square and vice versa, which I would like to delve deeper into. By doing so, I would like to generate solutions to the following questions:

- In a 8 x 8 standard chess board, what is the largest number of Knights that can be placed such that none of the pieces are in danger?
- If the 8 x 8 chess board was to be expanded to accommodate n x n number of squares, how will this alter the circumstances of the problem and solution?
- Is there any correlation between the number of Knights and the value of n

As I had traversed through previously conducted research on Knights specifically in relation to the Eight Queens Problem, I had pinpointed three main parts of this problem and conclusions to consider. In turn, these will aid in my simulation of this problem and further analysis.

To begin with, the Knight piece is evidently less restricted than the Queen, Bishop, and Rook. Whereas each of the latter pieces are able to move any number of spaces on the board, the Knight piece can only move three squares away. In the context of objectively placing Knights on the board such that no piece will be a Knight's move away from another piece, this will create a solution in which more pieces can be placed on the board.

On a similar note, a research paper by James A. Storer that had been published in the Journal of Computer and System Sciences also goes as far as to generalize that "as the board size gets large, standard knights seem to become worth less since they can only travel a distance of three squares on a given move…" (Storer, 1980, pg. 77).[5] This has also allowed for me to derive that in an n-Knights problem, the number of Knights that can be placed in a particular arrangement increases as n increases, and decreases as n decreases.

Finally, the unique L-shaped movement of the Knight piece in chess is distinct from any other piece. For this reason, similar to how one solution to the n-Queens problem was to place Queens a Knight's move away from one another, it is logical to assume that the Knight pieces can be placed a Rook's, Bishop's, Pawn's, and King's move away from another. In fact, in their paper titled "The Mathematical Knight", Noam D. Elkies and Richard P. Stanley have generated a solution in which 32 mutually non-attacking Knights can be placed in each white square, which would ensure that they can only attack the black squares (Elkies and Stanley, 2003, pg. 25).[6] Theoretically, in an arrangement that follows this pattern, the Knights are each diagonal to one another, as a Bishop's move is.

[5] Storer, James A. On the Complexity of Chess. 1980. Pg. 77
[6] Elkies, Noam D. Stanley, Richard P. The Mathematical Knight. 2003. Pg. 25

Research Methods

Due to the assumption that this pattern and algorithm would be the most likely solution to this problem, I had begun brainstorming how to simulate it through a computer program in the language C, while documenting my thought-process, any limitations, and additional considerations that arise. I had chosen to implement this problem using the backtracking strategy, which is characterized by iterations through the board while recursively placing pieces down into safe spaces. However, if a piece cannot be placed in a certain spot, the program will "backtrack" back to the previous piece and move it to the next column or safe space before checking whether it leads to any conflict. If this recursive call in turn returns false, or 0 in the case of my code - meaning it cannot be placed there, it will backtrack once more until a piece is correctly moved such that it is not at risk of being attacked. Therefore, the program will end successfully only if the function iterates until it's over the bounds of the last row, or equal to n when n is the number of rows. On the other hand, if the function cannot determine a possible conclusion and backtracks all the way back to the first piece, then it will return that no solution is possible.

For reference, I had studied and tested an n-Queen problem code that my Introduction to Discrete Structures professor, Arup Guha, and Computer Science 1 professor, Tanvir Ahmed, had posted, which had incorporated backtracking as well. Nonetheless, my problem would have to be implemented differently and possess additional constraints due to the pieces being Knights rather than Queens.

The process

To begin with, the basis behind the n-Queen problem is that an n number of Queens must be placed on the chess board such that no two Queens attack one another, as mentioned before. Naturally, this indicates that only one Queen can be placed in each row and column, therefore leading to the possibility of iterating through each row until n Queens are placed correctly. Furthermore, in the case of backtracking, a programmer will be able to ensure that the program backtracks to the right piece by unmarking the previously placed Knight. This is due to the Queen being able to move any number of spaces on the board as long as the criteria of the moves are met. However, a Knight can only move at most two squares in any direction, followed by one in another direction. Therefore, my program must be able to iterate through each column as well as row recursively. In other words, multiple Knights can be placed in a single row, which appears to complicate this strategy. As such, I decided to construct an alternate function to verify whether a square is safe to place a Knight in, drawing inspiration from Professor Ahmed's isSafe function. Evidently, I had to come up with an alternative logic to accommodate for the Knight's move, but I had found that the method he used to check whether any of the rows or columns checked were out of bounds, meaning that they should not be checked lest a segmentation fault occur, simple to implement. Thus, the real challenge was to determine how to account for the Knight's moves. Initially, I had begun with mathematical reasoning in order to brainstorm.

In order to visualize a Knight's move, I created a table with 5 * 5 squares as a Knight's move cannot go past these dimensions. I then placed the Knight in the middle square, and marked any possible squares that can be attacked as well as a pattern in the squares not marked, as shown below.

row/col	*-2*	*-1*	*0*	*1*	*2*
-2	*diagonal*	**X**	*same column*	**X**	*diagonal*
-1	**X**	*diagonal*	*same column*	*diagonal*	**X**
0	*same row*	*same row*	**Knight**	*same row*	*same row*
1	**X**	*diagonal*	*same column*	*diagonal*	**X**
2	*diagonal*	**X**	*same column*	**X**	*diagonal*

Initially, I had attempted to generate an algorithm that would incorporate the idea that the product of two odd numbers will always be an odd number, ensuring that I can pinpoint which cells to check, as well as that the squares on the same column, row, and diagonal are safe. However, it proved tedious and inefficient, and I had examined the table that I had constructed before settling on a negative, positive number system as shown. The main premise is that the Knight can move 2 squares at most in each direction, therefore allowing me to label the rows and columns with -2, -1, 0, 1, 2. I had then identified the squares that the central Knight could move to, which are (-2, -1), (-1, -2), (1, -2), (2, -1), (-2, 1), (-1, 2), (1, 2), and (2, 1). In this way, I had constructed my isSafe function by creating arrays to store the x and y of each of these pairs, before adding the numbers for each pair to the central knight's row and column. This had then allowed me to check those particular squares for any Knights as long as they were within the bounds.

Breaking down the recursive logic

With the isSafe logic figured out and tested, I had then moved onto working through the logic of the solveKnights function which would recursively place Knights on the board until either a solution is generated, or no solution is returned. Due to the nature of the Knight's move being held in juxtaposition to the Queen's move, I could not recursively increase the row alone as was done with the n-Queens problem. Instead, I had to update the columns as well accordingly. For this reason, I had decided to recursively move through each square individually. Furthermore, since I was moving across the board square by square, I also had to account for the fact that the loop would have to jump from the last column of one two to the first column of the next row and vice versa in the edge cases. As a result, I had used the modulus operator in order to calculate the new column in relation to the size of each row with the line, col = col % n;

Therefore, if a square was deemed safe, the program would increment the column by 1 until the end of a row is reached, in which case it would move to the next row. Similarly, if it has reached the end of the last row without placing down the maximum number of knights, it would return 0 and backtrack. If a recursive call then returns 0, I have also implemented code after the recursive call to remove the knight at the previous square and move back a column as well as the opposite logic for moving to the last column of the previous row if the column goes out of bounds.

Finally, if a square is not safe and the isSafe function returns 0, then the program would shift to the next column and employ the same earlier edge case logic. In this way, I had ensured that each square could be recursively moved to when the function is looped until a value is

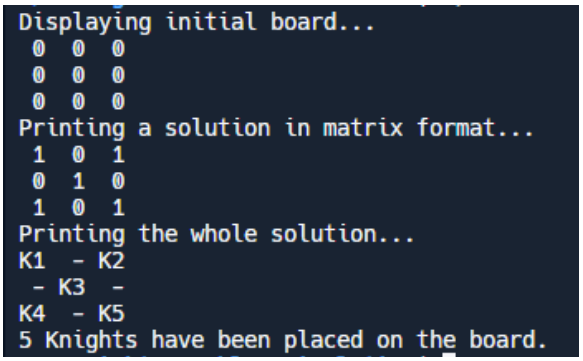
returned. With the main solver function having been written, I had then weaved together the program by calling each function appropriately and displaying the board in matrix form, as well as with characters. One other aspect of the code had been the inclusion of an integer variable with the maximum number of knights that could be placed according to a formula.

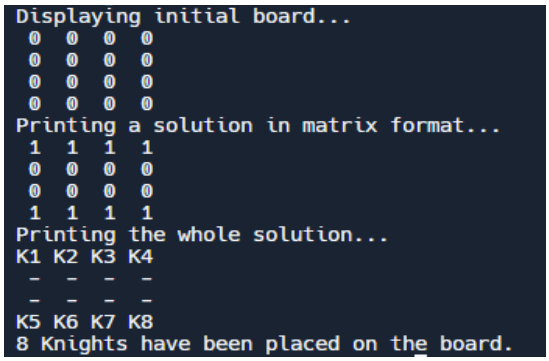
Results

While my algorithm can generate solutions in different patterns, I had mainly worked with an alternating squares pattern in order to work out the logic. Therefore, I had been able to optimize the runtime of the code and specify the base case for recursion by determining a formula for the maximum number of Knights.

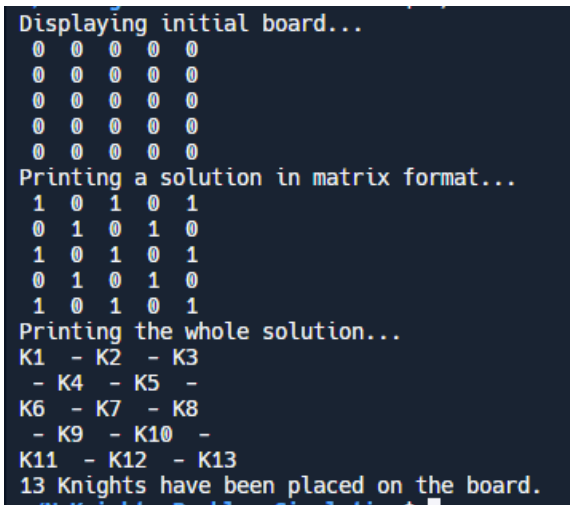
The number of knights that can be placed in an alternating pattern on a 8 * 8 boards would be 32, due to there being 32 white and 32 black squares. However, no number could evenly divide into a board with an n that is odd, due to the total number of squares always being odd. This can be proven in number theory with the definition of an odd integer, such that $(2d_1 + 1)(2d_2 + 1)$ will always result in an odd product. Thus, there will be an additional square in whatever color the first square's color falls into. Since the first Knight will always be placed in row 0, column 0, there will be one more Knight than there will be squares of the opposite color. In other words, the number of Knights can be calculated with the formula $\frac{n^2}{2}$, in which the decimal is rounded up. This in turn served as the basis for my calculations and predictions of the results, which had then aligned perfectly with what was generated when the code was run.

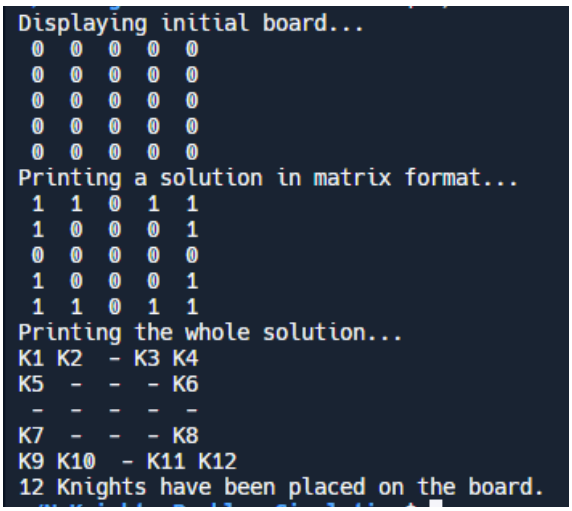
First, I had worked with a 3 * 3 board and a 4 * 4 board to generate the following:





This had then validated the formula due to $\frac{3^2}{2}$ being 4.5, which is 5 rounded up. Similarly, $\frac{4^2}{2} = 8$. This had allowed me to incorporate this formula with code that would add 1 to the total number of squares if it were odd, before dividing by 2 to get the maximum number of knights. Interestingly, by changing the maximum number of knights and the formula's code, I have been able to generate different patterns such as the ones below.





These two boards have the same 5 * 5 dimensions, but feature different arrangements of the Knights due to the formula rounding down instead in the case of an odd number. This is a

result of the code reaching the maximum number of knights that had been calculated as 12 before fully backtracking as it had with 13 Knights. In this way, I had been able to derive that with the exception of a 1 * 1 board, the maximum number of knights that can be placed will coincide with this formula, and at least half of the board can be occupied by Knights with all values of n or one more than half depending on if n is odd. This had then been incorporated with the following formula:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2} = \frac{n^2}{2} \rightarrow \frac{(n*n)}{2} \text{ if n} \in \text{the set of odd } Z^+$$

$$\frac{1}{2} + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2} = \frac{(n*n)+1}{2} \text{ if n} \in \text{the set of odd } Z^+$$

Above is the derivation of the formula that I had incorporated, which has clearly been simplified from a summation problem that reflects the loops through which I had created the board in my code. Abiding by the logic that half of the squares would be occupied when even, and the result would be rounded up, or added to 0.5 in other words, I have come up with these summations and the appropriate closed forms.

Analysis

Overall, the code was successful in simulating the n-Knights problem and displaying the first possible arrangement of the pieces. While the code does generate the solution for values of n that I have been able to test, the runtime increases exponentially and can prove inefficient for simulating larger boards with n values over 8, as the number of squares would drastically jump from 64 to 81 between n = 8 and n = 9. Furthermore, my method of iterating through the board had been to loop through each individual square, which had resulted in the higher runtimes.

Nevertheless, this code has proven that it is possible to generate solutions for an N-knights problem encompassing backtracking similar in nature to the n-Queen problem, as well as revealed the math that operates behind it with proof that it works. All things considered, with a little over 200 lines of code in my program, backtracking proved to save lines of code, but there must exist a faster algorithm that could assist in solving this problem, such as permutations and loops.

In order to delve deeper into the results of my code and compare it to any previously done programs involving knights, I had found an insight into code that could simulate the Knight's Tour problem in Chess, which is another prominent dilemma popular in the programming community. John R Gerlach had published a paper on this very topic titled “The Knight’s Tour in Chess - Implementing a Heuristic Solution” in which he details his approach to the Knight’s Tour problem, part of which had been the inclusion of a series of steps for the rows and columns of -2, -2, 1, and 2.[7] Evidently, this method has proven reliable in mapping out a knight’s move and outlining the squares to move to as the Knight only has 8 possible moves. In

[7] Gerlach, John R. The Knight’s Tour in Chess - Implementing a Heuristic Solution. 2015. Pg. 2-3.

researching similar problems to the n-Queens problem, I had gained insight into the underlying logic that drives many of the pieces and rules of chess, and the endless possibilities are compelling and enthused me about the game even more.

Conclusion

Throughout the process of conducting this research, I had been able to identify any limitations and contingencies that I must pay particular attention to while coding the n-Knight's problem, and it has successfully run and displayed results that validated my findings and predictions. In researching the applications and specifics of the backtracking algorithm, I had overall become more adept in the uses of recursive functions as well as problem solving. Moreover, I had also been able to derive mathematical equations and formulas to explain the phenomena that had occurred with the maximum number of knights allowing for the Knights to be placed in alternating squares, which admittedly is intuitive but can be solidified by a computer algorithm.

Further Research

While I have demonstrated the n-Knight's variation of the famed n-Queen problem, I have not yet been able to run the code for large values of n. To do this, it would be necessary to alter the algorithm and uncover a more efficient method of iterating through the board, perhaps with permutations, but it does not seem probable as the Knight is a unique piece that can be placed in the same row, column, and diagonal unlike the other pieces. However, it might be possible to achieve faster speeds and more optimal runtimes with a higher-level programming language, such as Java or Python.

Furthermore, aside from the Knight and Queen, there is also the Bishop and Rook to examine in a similar way and further research could be conducted on whether arrangements such as these would be possible for these pieces as well. For this reason, there are a multitude of ways in which to expand on what has already been discovered and uncover the inner workings of chess, possibly with algorithms other than backtracking. Nevertheless, conducting this research has allowed me to build upon my understanding and knowledge of logical problems involving mathematics in the background, as well as sharpened my understanding of backtracking as a whole.

References

Sosic, Rok. Gu, Jun. *Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem.* 1994.

Fillmore, Jay P. Williamson, S. G. *On Backtracking: A Combinatorial Description of the Algorithm.* 1974. Pg. 45-49

Chatham, R. Douglas. *Reflections on the N + k Queens Problem.* 2017.

Storer, James A. *On the Complexity of Chess.* 1980. Pg. 77

Elkies, Noam D. Stanley, Richard P. *The Mathematical Knight.* 2003. Pg. 25

Kesri, Vishal. Kesri, Vaibhav. Pattnaik, Prasant Ku. *An Unique Solution for N queen problem.* 2012. Pg. 1-2

Gerlach, John R. *The Knight's Tour in Chess - Implementing a Heuristic Solution.* 2015. Pg. 2-3.