Common Search Strategies and Heuristics With Respect to the N-Queens Problem

CS504 Term Project

Sheldon Dealy

December 10, 2004

Abstract

The N-Queens problem is examined and programmatically implemented for Depth First Search, Depth First Search with improvements, Branch and Bound, and Beam Search. Several heuristics are presented and implemented with each of the searches. Results were analyzed for number of nodes generated, number of nodes traversed, and relative execution time. While heuristics were found which gave Branch and Bound and Beam Search a significant edge over DFS, there exist polynomial time algorithms using complete board assignment and heuristic repair methods which are purported to do better.

Contents

List of Figures

Chapter 1

Introduction

The attraction of the N-Queens problem is that it is simple to state, yet complicated to solve efficiently. The size of the solution space grows exponentially as N^N. The N-queens problem can stated thus; "Given an NxN chessboard, find a way to place N queens upon the board such that no queen threatens another." Solutions exist for N not equal to $2, 3$.

In this paper we propose to contrast and compare how an implementation of DFS search without heuristics compares to implementations of Beam Search, and Branch and Bound Search, and a heuristically enhanced version DFS. Each of the searches is used in combination with each of four search heuristics to be enumerated later.

The structure of this paper is as follows. Chapter 2 describes some variations in the problem and approaches to solving the N-Queen problem. In chapter 3, a brief history of the N-Queens problem is presented. Chapter 4 discusses the searches and heuristics employed in the experiments. In chapter 5, I address how the experiments were implemented. Chapter 6 presents the results with an analysis for each heuristic. In chapter 7, conclusions from the experiments are discussed. Finally in chapter 8, the experiments are summarized for what has been shown.

Chapter 2

Variations

For the archetype version of N-Queens, there are three basic kinds of algorithm strategies commonly used to solve the N-Queens problem. First, there are algorithms used to find all solutions for a given N. Second, there are algorithms which find the fundamental solutions or solutions which are not isomorphic to another solution using rotation or symmetric reflection [11, 4]. Third, there are algorithms which generate one or more, but maybe not all solutions. For purposes of comparison, the implementation given here is based on finding the first solution for a given N.

There exist some interesting variations of the N-Queens problem statement. These problem variations include:

- Finding the minimum number of queens required to cover all squares on the board such that no queen threatens another.
- N-Queens on a wrapped board (toroidal), introduced by G. Polya [14].
- 3-D Queens, using an $NxNxN$ board (generalizable to multiple dimensions) [15].

2.1 Novel Approaches

Two polynomial time implementations are presented because of their interesting and highly successful approaches to the N-Queens problem.

Gradient Based N-Queens

The gradient based N-Queens search begins with a random assignment of queens to the board. The assignment requires that each queen is given a row and column position such that no two queens occupy the same row or column. After the queens have been assigned their locations, to obtain a solution it remains to resolve conflicts on the diagonals with the other queens. To remedy diagonal conflicts, the algorithm performs a series of swaps to the queen's column positions with queens in other rows in an attempt to find a non-conflicting board state. If a swap will reduce the number of conflicts, it is performed, otherwise it is not performed. The search proceeds in the gradient or direction containing the fewest conflicts. If no solution can be found, then a new permutation of the board is generated. The gradient-based N-Queens implementation was solved for $N = 500,000$ in 1990 [10].

Heuristic Repair Based N-Queens

The Heuristic Repair approach begins, similar to the gradient-based approach, by generating a complete assignment of N-Queens on an NxN board. Subsequent to the full board initialization, the algorithm incrementally repairs conflicting queens by taking each queen with a conflicting board assignment and evaluate positions having lesser conflicts. Unlike gradient-based search, conflicting queens are pushed onto a stack so that different permutations of positions with the conflicting queens can be backtracked over if an immediate solution is not found. The Heuristic Repair approach was solved for $N = 1,000,000$ in 1993 [12].

Chapter 3

Historical Highlights of N-Queens

The first published reference to the Queens problem was an 1848 article by a German chess enthusiast Max Bezzel. The article was about the 8-queens variant and published in the German chess newspaper *"Schachzeitung"*.

Karl Gauss took a passing interest in the problem after reading an 1850 article written by Franz Nauck, a mathematician who eventually was the first to discover all 92 solutions to the 8-Queens problem.

The N-Queens problem has since captured the interests of many mathematicians, chess players, and more recently computer scientists.

An interesting side note to the N-Queens problem is historical error with regards to whom to credit for introducing the original problem. In 1874, J.W.L. Glaisher published a mathematical article about the Queens problem. In this article he gave credit of the original Queens problem to Franz Nauck. This mistake has since been a study in the propagation of historical error [5]. It has been speculated that while Glaisher had access to the correct facts at his disposal, perhaps his understanding of the original German was poor. This mistake continues to this day (e.g. see [8]) and it should be a lesson to those who value accuracy not to blindly trust the statements made by downstream research.

Chapter 4

Search Strategies and Heuristics

4.1 Searches

There were four searches implemented in the experiments; one search as a control, and three searches which were capable of using heuristics to improve their search efficiency.

- Depth First Search (control). Depth First Search (DFS) without improvements was implemented as a control to provide a baseline against which to judge the other searches with the heuristics.

- Depth First Search with Improvements (DFS-I). My implementation of DFS-I evaluates the child nodes of each parent node an orders them in reverse order of their respective values before pushing them onto the DFS stack.

- Branch and Bound (BnB). BnB was implemented on top of a Best First Search[1]. The lower bound is an estimate of the shortest distance to the solution and was calculated from the heuristic plus the distance from the root. The upper bound is a measure of the worst acceptable distance to a solution (pessimistic bound); calculated by adding

[1]I tried BnB with a plain queue and with priority queue. It did not seem to make much difference which of the two data structures was used for BnB.

a constant to the lower bound. The constant was determined empirically. Pruning was done at the time of node creation, i.e. nodes were not re-evaluated after being placed in the queue.

- Beam Search. Beam Search was implemented on top of a Best First Search. The width of the beam is calculated as a function of N and was found to work well at around $4 * N$ with the heuristics given. Using smaller constants with N, solutions to a given Beam Search run were not always assured.

4.2 Heuristics

There were four heuristics tested with each of the three heuristics-capable searches.

- H0. No heuristic; a baseline to compare searches with added heuristics.

- H1. Distance is the number of squares from previously placed queen (local density). The distance heuristic is simple to compute and is not very informed.

- H2. Mean distance between previously placed queens (an aggregation of H1). The mean distance heuristic is still pretty quick to calculate, but is still not very informed. It was my hope that cheap searches like H1 and H2 would yield worthwhile results, but as I will show in the results, these heuristics did not perform very well.

- H3. The number of open squares on board. For each queen placed on the board, the row, column, and diagonals in which the queen is placed is covered by the queen. The number of open, or uncovered squares is inefficient to calculate or estimate because the squares covered by the diagonals may be calculated multiple times. The more open squares, the more opportunities there are to place a queen. H3 was a more informed heuristic.

- H4. Mean hamming distance between all queens on the board. The mean hamming distance takes the hamming distance between Q_1 and all the other queens placed, then the hamming distance between Q_2 and all the other queens is calculated, etc.. Even after removing duplicate hamming calculations, this heuristic is expensive to compute. The idea is that knowing the mean hamming distance between queens might make it easier to predict which placements would lead to a solution.

Chapter 5

Implementation Methods

The search space was implemented as a graph. Each node in the graph represents a partial solution to the problem. The goal of implementation was to compare the relative efficiency of different search strategies when used with each of the different heuristics. In order to achieve this goal, all searches were implemented using the same base code. The data structures used to direct the searches were different to allow for the different graph traversals; the searches were implemented using either a queue or a stack.

It was presumed that a good heuristic-search combination would perform relatively better when compared to other heuristics of a similar, but less fortunate implementation. A solution is considered acceptable when all n queens are placed without conflicts. Each node was implemented as a vector to save space considering the number of possible permutations. Manipulations of legal moves were implemented through indices into the vector. The indices into the vector represent rows, while the values at the row indices of the vector represent the column indices [13].

The beginning state is a queen placed at random on a space in the top row. A legal move to the next state is one that does not *immediately* threaten another queen. Child nodes were generated, evaluated, and placed on a stack for DFS search or a priority queue with

Beam Search and BnB. Alternate paths are attempted when it is not possible to legally place another queen on the board and the number of queens is less than N.

Searches were evaluated by pairing the searches with each heuristic. Statistics were measured for number of nodes generated, number of nodes traversed, and the relative execution time to find a solution. Each configuration was run 150 times with the initial start position being chosen at random. The mean of the results was calculated and the algorithms graphed against each other for each of the heuristic possibilities.

Chapter 6

Results

6.1 Analysis: No heuristic (H0)

The searches were run without using any heuristics to provide a baseline. In all criteria (see figures 6.1-6.3), DFS was clearly the most efficient. Intuitively, without heuristics, it was expected that Branch and Bound would revert to a breadth first search and Beam Search would give mixed results. Actually Beam Search performed comparably to DFS in the number of nodes traversed while generating an excessive search space. In reality, Branch and Bound was the search which performed erratically.

6.2 Analysis: Simple distance measure heuristic (H1)

Performance using the simple distance measure was less than hoped for (see figures 6.4-6.6). DFS and DFS-I generated the smallest number of nodes per solution. Beam search, DFS, and DFS-I all traversed about the same number of nodes to obtain a solution while Branch and Bound performed poorly. Branch and Bound found a solution in much less time than the other searches with H1, however BnB did not always find a solution.

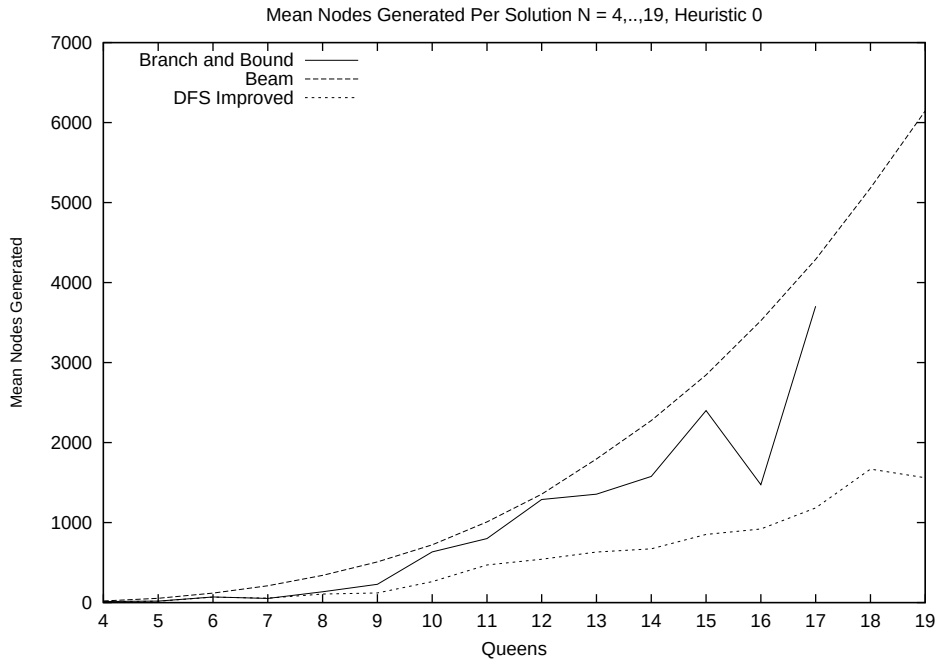

Figure 6.1: Nodes Generated Per Solution, No Heuristic

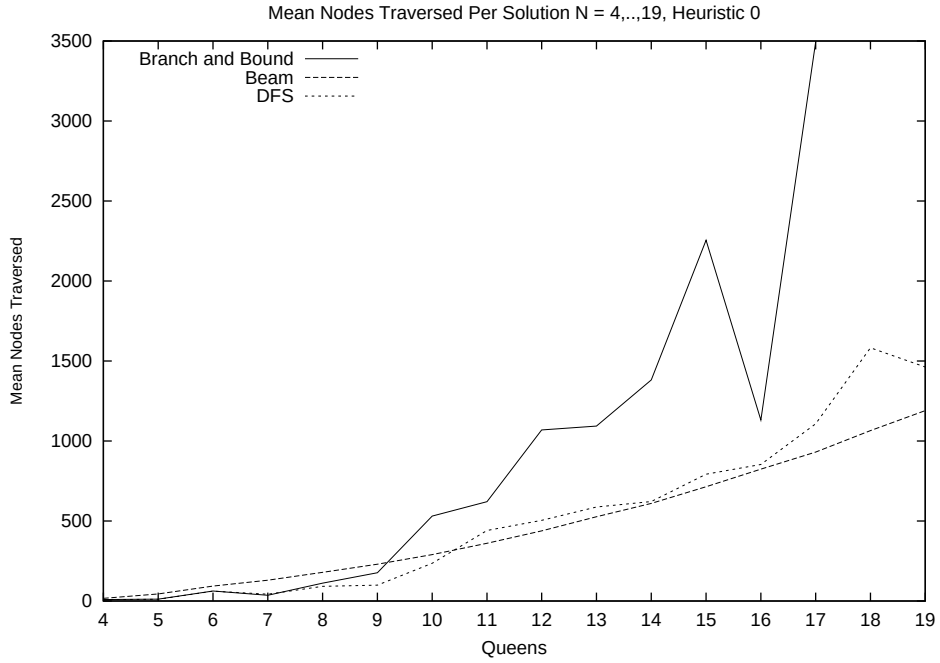

Figure 6.2: Nodes Traversed Per Solution, No Heuristic

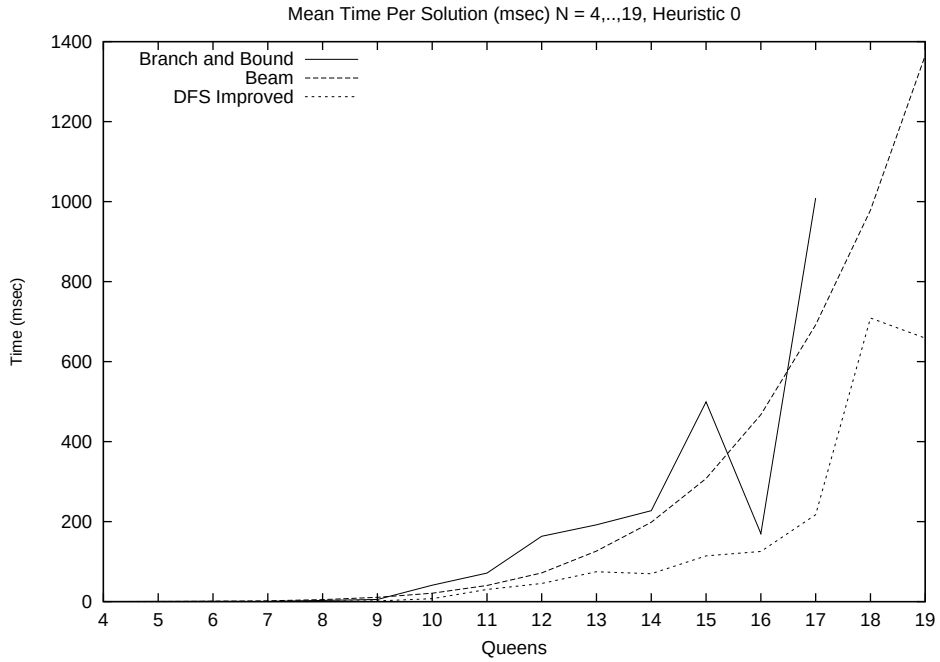

Figure 6.3: Time to a Solution, No Heuristic

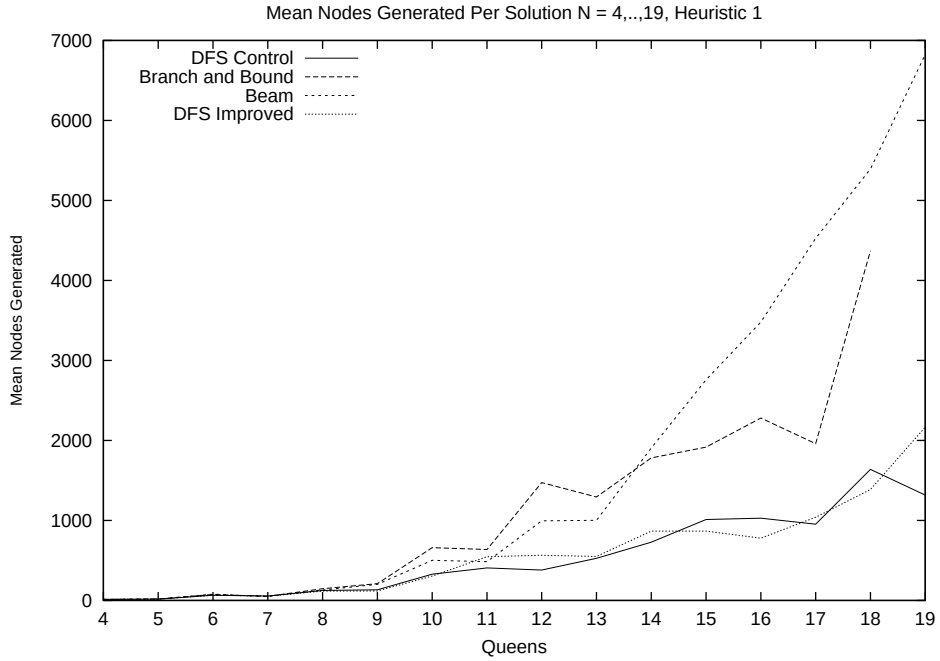

Figure 6.4: Nodes Generated Per Solution, Distance From Previous Queen (H1)

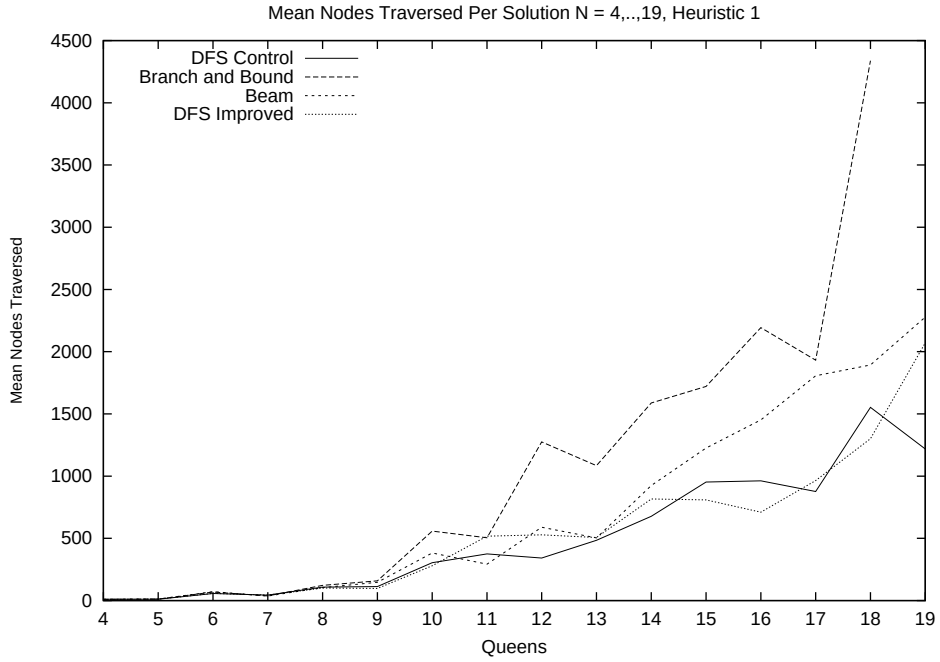

Figure 6.5: Nodes Traversed Per Solution, Distance From Previous Queen

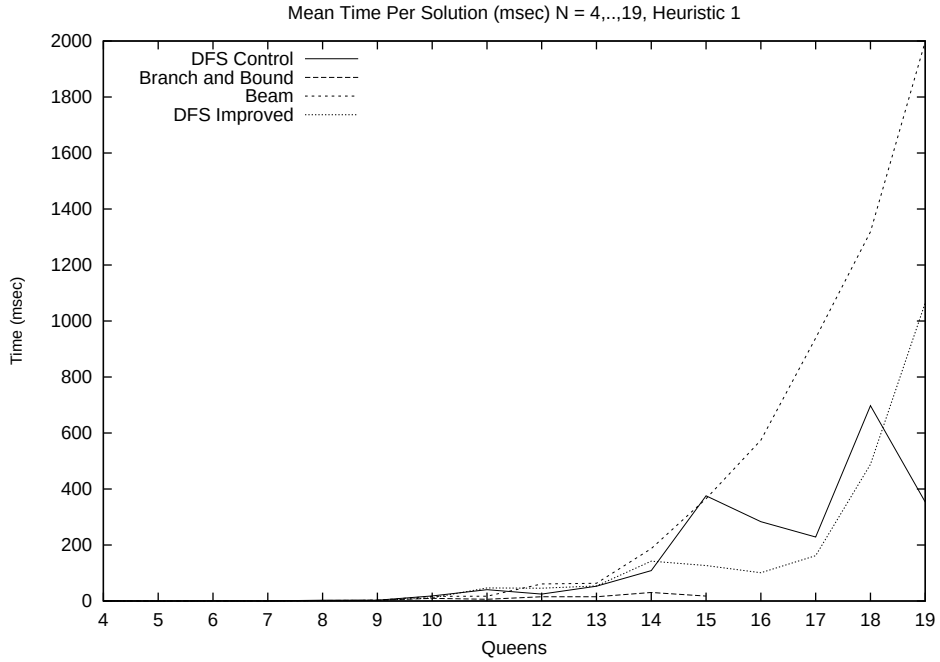

Figure 6.6: Time to a Solution, Distance From Previous Queen

6.3 Analysis: Aggregation of the simple distance measure heuristic (H2)

DFS and DFS-I generated a fraction of the nodes that either Beam or BnB searches generated (see figure 6.7). Beam Search traversed a number of nodes comparable to DFS and DFS-I to achieve a solution, (see figure 6.8). The time for Beam Search was only slightly higher per solution than DFS and DFS-I (see figure 6.6). Branch and Bound performed poorly in all performance areas measured with H2. DFS and DFS-I performed about the same.

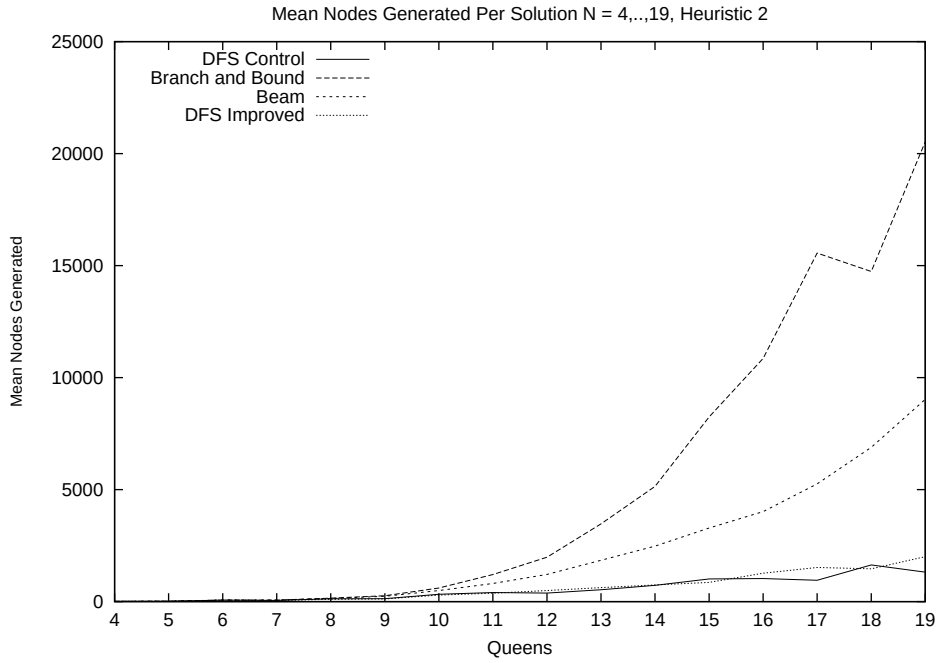

Figure 6.7: Nodes Generated Per Solution, Mean Distance From Previous Queens (H2)

6.4 Analysis: Number of open squares heuristic (H3)

BnB, DFS, and DFS-I all generated a similar size search space and yielded a similar number node traversals to find a solution with similar search times, (see figures 6.10-6.12). Beam Search fared better, generating about a third fewer nodes and traversing only half as many

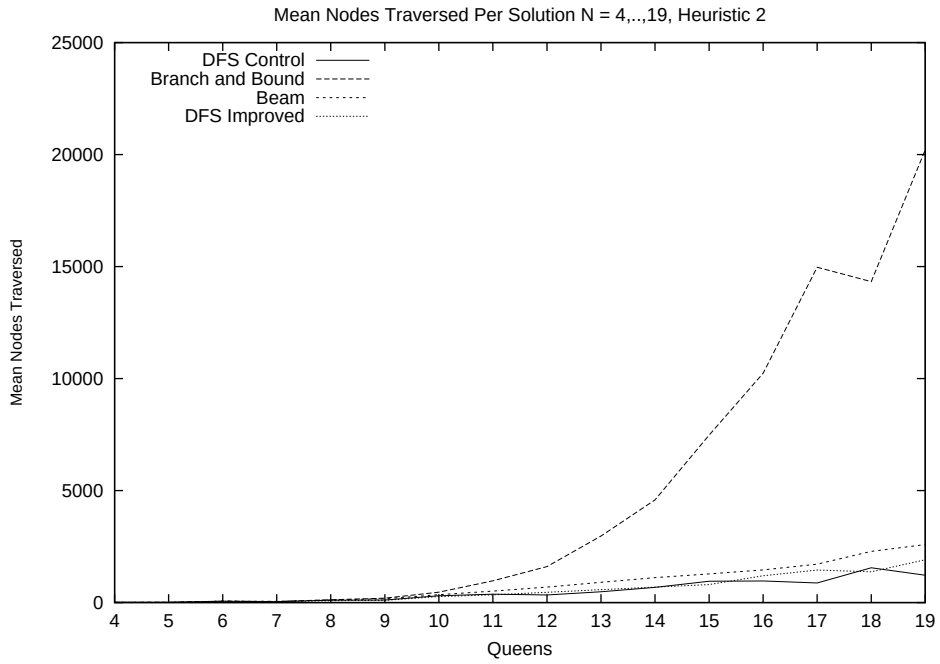

Figure 6.8: Nodes Traversed Per Solution, Mean Distance From Previous Queens

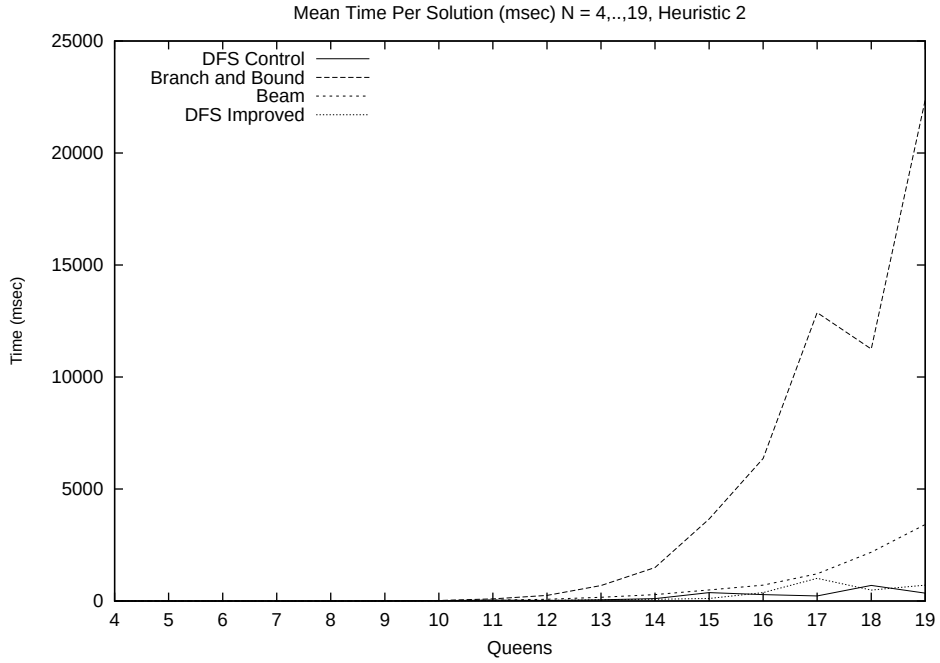

Figure 6.9: Time to a Solution, Mean Distance From Previous Queens

nodes on average to find a solution compared to the other searches. Surprisingly, Beam Search executed in linear time.

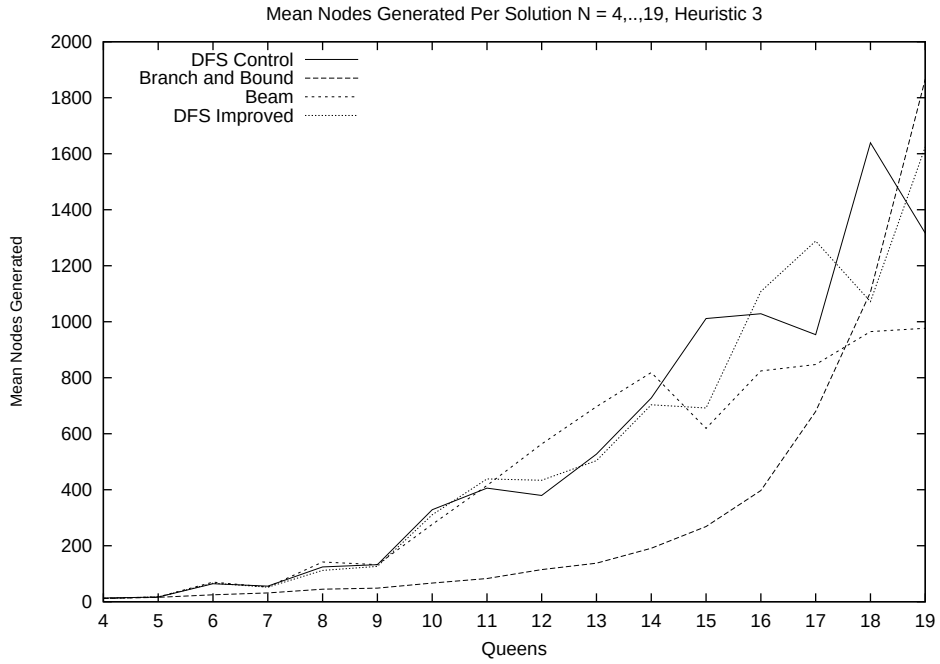

Figure 6.10: Nodes Generated Per Solution, Free Squares (H3)

6.5 Analysis: Mean hamming distance heuristic (H4)

Beam Search, DFS, and DFS-I all generated similar size search spaces and node traversals, (see figure 6.13). Beam Search and DFS times performed about the same. DFS-I actually performed slightly worse than the other searches for time to a solution, (see figure 6.6). It is speculated that the poor search time for DFS-I can be attributed to computational overhead. On the other hand, BnB generated a compact solution space, traversed few nodes, and was incredibly fast – when it actually generated a solution.

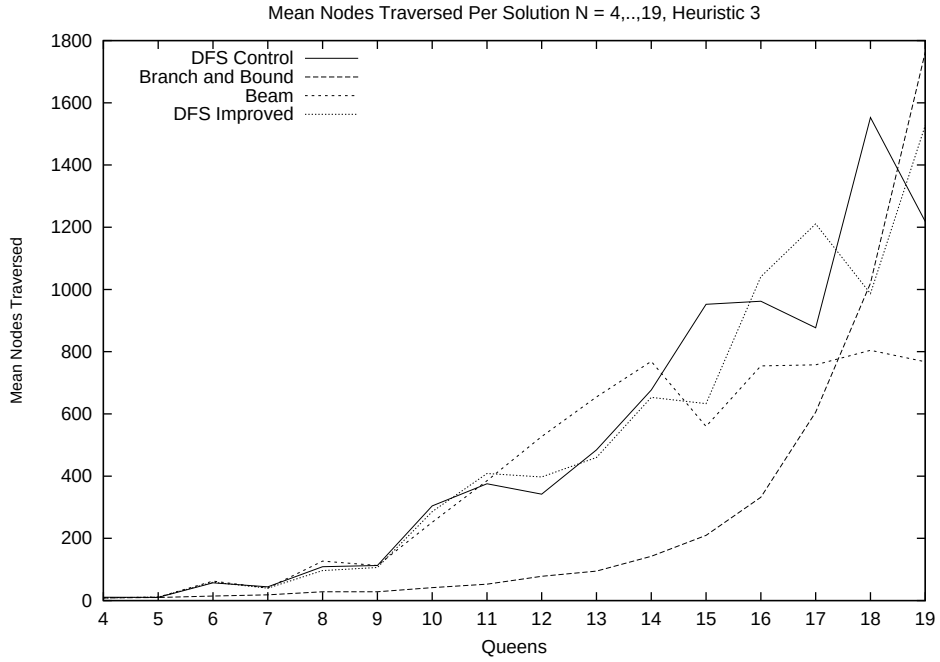

Figure 6.11: Nodes Traversed Per Solution, Free Squares

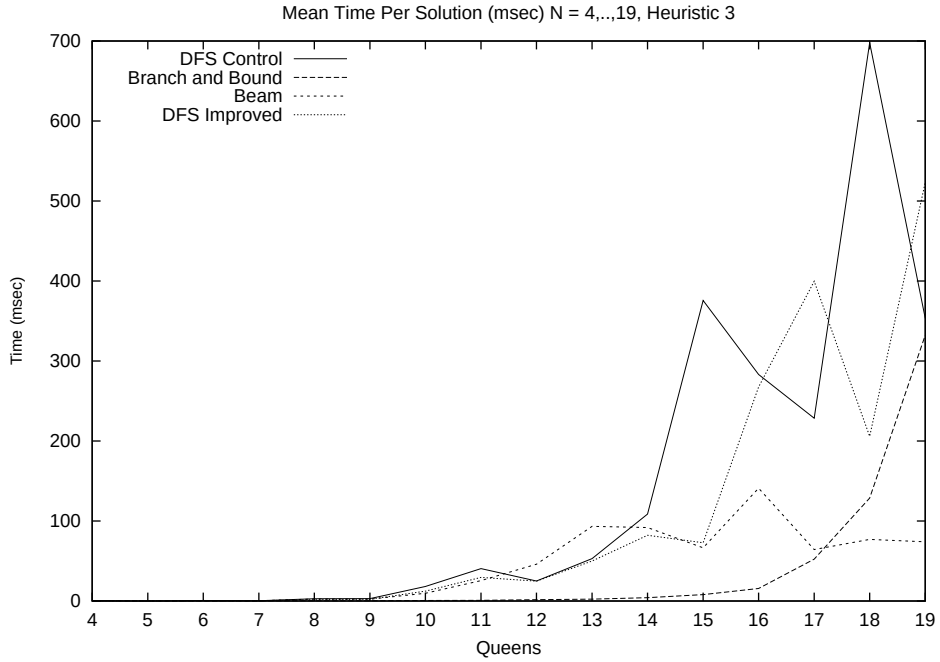

Figure 6.12: Time to a Solution, Free Squares

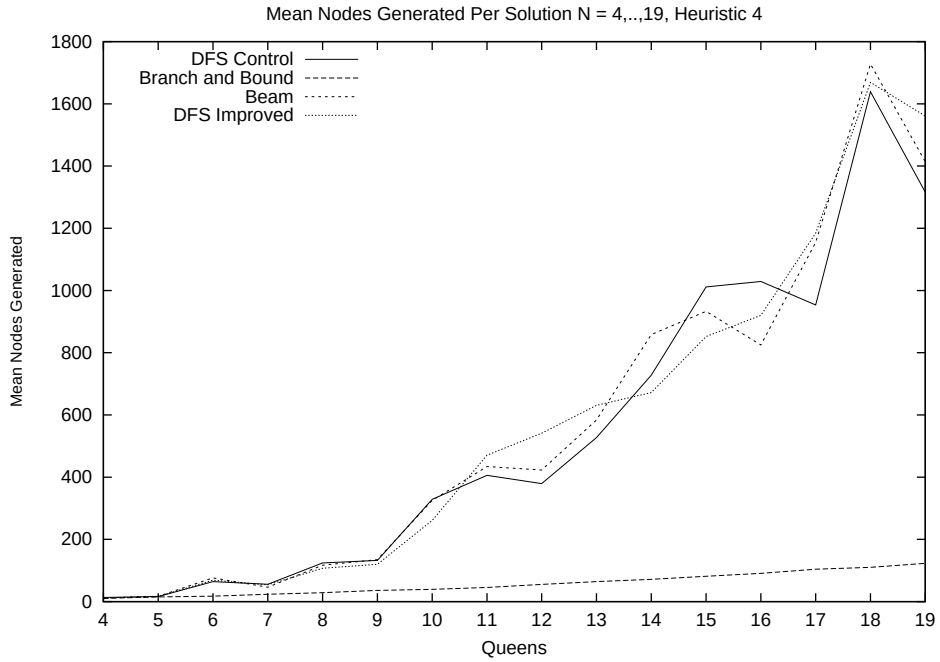

Figure 6.13: Nodes Generated Per Solution, Mean Hamming (H4)

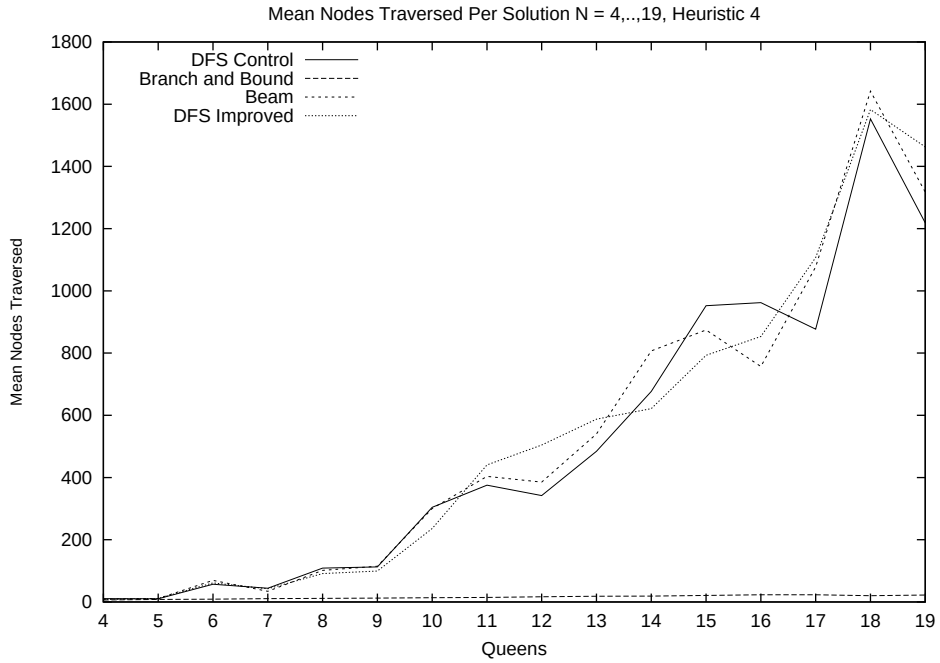

Figure 6.14: Nodes Traversed Per Solution, Mean Hamming

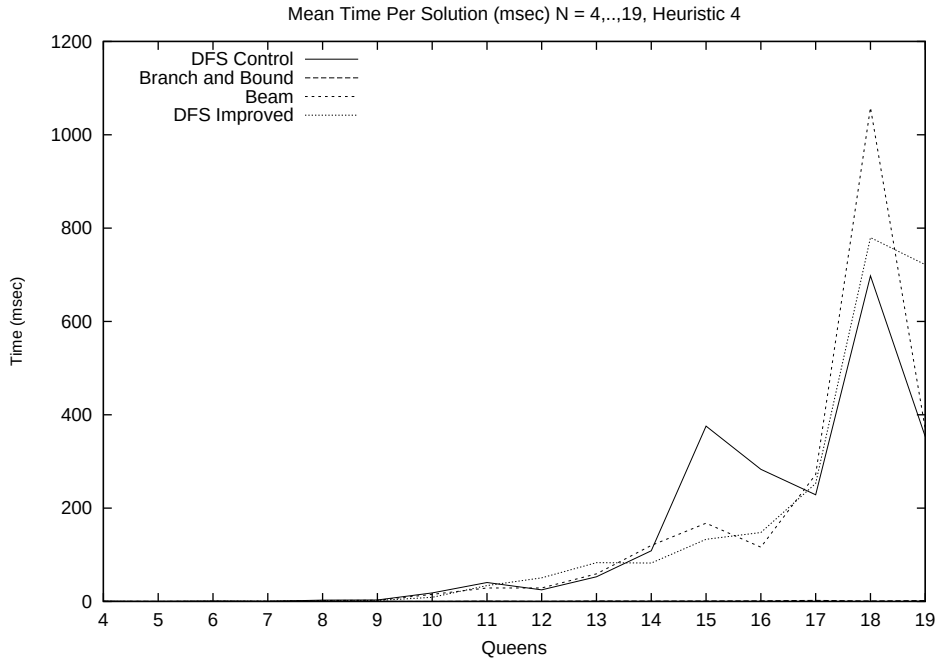

Figure 6.15: Time to a Solution, Mean Hamming

Chapter 7

Discussion

It was probably naive to think that less informed heuristics like the simple distance measures of heuristics H1 and H2 would yield solutions better than DFS, but curiosity got the better of me and they were implemented anyway. A slight but discernible improvement was seen between H1 and H2 for Beam Search, but not enough to choose Beam Search over DFS.

Additional computation was shown to yield a more informed search, but as was seen in the results from H4, the Mean Hamming Distance, performance of DFS-I was degraded from the computational overhead. Clearly, the added information for improved DFS was not enough to pay for the additional overhead.

Better heuristics may have other interesting trade-offs, e.g. BnB used linear space, but did not always find a solution. However, the speed of BnB was so fast with H4, that time to a single solution even with failure, was far less than either Beam Search, DFS, or DFS-I.

7.1 Challenges

What the graphs and results do not show is that for small N, some initial configurations permit no solution. For larger values of N, some initial configurations require considerable

computation to obtain a solution, depending upon the search. With that in mind, for purposes of gathering statistics, the starting point in each run was chosen at random. Random starting positions ensured that searches had neither the best nor the worst starting position each time, but that the results were drawn over all possible positions. Child nodes were chosen in a similar way.

Parameters were carefully tuned to match heuristics with searches. It is not claimed that the parameters used matching each search with a given heuristic were the optimal combination. However, the parameters used seemed to work best at the time.

Results were difficult to evaluate. Even after much reflection, it is still possible to draw startling conclusions. For instance, the BnB results using the Mean Hamming Distance heuristic appeared to be a serious bug until it was realized that the time to failure was also quite fast. The narrow search gave BnB a chance to either find a solution or fail quickly an order of magnitude faster than any of the other searches on average [1].

[1]Time to failure was not measured computationally, but was observed directly.

Chapter 8

Summary

N-Queens is an interesting and useful exercise in applied algorithm heuristics. It is my opinion, gained from the literature [12, 10], that there may be more fruitful lines of experimentation that exploit the repairs of a randomly generated board rather than to build a solution incrementally.

Some heuristics worked better with a given search than others. Beam Search matched well with the Open Squares heuristic while Branch and Bound performed well using the Mean Hamming Distance heuristic.

Improved DFS didn't enhance the functionality such that it would be preferable to plain DFS. This lack of functionality could be explained in part to implementation idiosyncrasies. On the other hand, following in the footsteps of Karl Popper, the hypothesis that DFS with minimal heuristics is better than DFS alone is just false.

Bibliography

[1] Unsolved Problems in Number Theory 2nd edition, Richard K. Guy ed., 1994, Springer Verlag, New York.

[2] Gauss's Arithmetization of the Problem 8 Queens, J. Ginsburg, January 1938, Scripta Mathmatica, Vol. 5, No. 1, Yeshiva College, New York.

[3] The N-Queens Problem, I. Rivin, I. Vardi, P. Zimmermanm, Aug.-Sept. 1994, The American Mathematical Monthly, Vol. 101 No. 7.

[4] Mathematical Recreations, Maurie Kraitchik, 1942, W.W. Norton and Co., New York.

[5] Gauss and the Eight Queens Problem: A Study in Miniature of the Propagation of Historical Error, Paul J. Campbell, Nov. 1977, Historia Mathematica, Vol. 4 No. 4.

[6] http://mathworld.wolfram.com/QueensProblem.html, Summary of n-queens problem with bibliography.

[7] http://www.liacs.nl/home/kosters/nqueens.html, Large bibliography for the n-queens problem maintained by Walter Kosters.

[8] The N-Queens Problem, C. Letavec and J. Ruggiero, May 2002, Informs Transactions on Education, Vol. 2, No. 3.

[9] Different Perspectives of the N-Queens Problem, C. Erbas, S. Sarkeshik, and M. Tanik, 1992, ACM.

[10] A Polynomial Time Algorithm for the N-Queens Problem, R. Sosic and J. Gu, 1990, SIGART, Vol. 1, No. 3.

[11] Isomorphism adn the N-Queens Problem, P. Cull and R. Pandey, Sept. 1994, SIGCSE Bulletin, Vol. 26, No. 3.

[12] Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems, S. Minton, A. Philips, M. Johnston, P. Laird, 1993, Journal of Artificial Intelligence Research, Vol. 1.

[13] Computer Algorithms, Horowitz, Sahni and Rajasekaran, 1997, Computer Science Press, New York.

[14] Construction Through Decomposition: A Divide-and-Conquer Algorithm for the N-Queens Problem, B. Abramson and M. Yung, 1986, IEEE

[15] http://www.csse.monash.edu.au/∼lloyd/tildeAlgDS/Recn/Queens3D/, L.Allison, C.N.Yee, and M.McGaughey, 1988, Monash University, Australia