Enhancing the Performance of Genetic Algorithms in Combinatorial Optimization of Large and Difficult Problems

Using Variable Adaptive Mutation Rates Controlled by 'Inbreeding'

Daniel Smullen*
UOIT
2000 Simcoe St. N
Oshawa, ON, Canada
daniel.smullen@uoit.net

Jonathan Gillett
UOIT
2000 Simcoe St. N
Oshawa, ON, Canada
jonathan.gillett@uoit.net

Joseph Heron
UOIT
2000 Simcoe St. N
Oshawa, ON, Canada
joseph.heron@uoit.net

Shahryar Rahnamayan
UOIT
2000 Simcoe St. N
Oshawa, ON, Canada
shahryar.rahnamayan@uoit.ca

ABSTRACT

Blah blah blah, here is our abstract.

Keywords

Genetic Algorithms, Combinatorial Optimization, N Queens Problem, Variable Mutation

1. INTRODUCTION

Genetic algorithms serve an important role in various applications, but particularly useful areas are those where they can perform stochastic generation of solutions for combinatorial problems. One notable example of this is the N-Queens problem. One approach to solving the N-Queens problem using a genetic algorithm is to implement chromosomes which model the position of each queen on the chessboard. The permuted values stored in the chromosome represent the sequential row positions of each queen, encoded into binary values. Each column value is the index, since there can never be more than one queen per column. Thus, the column positions increase by one for each queen and the intersections (referred to as collisions) are calculated per row or on the diagonal to determine whether a solution has been found. As with any genetic algorithm, evaluating the fitness of the chromosomes per generation is required to determine if a usable solution has been generated.

By tuning the parameters of the genetic algorithm, performance can increase or decrease based on the solution landscape of the problem encountered. We have found that by replacing a static mutation value with an adaptive value controlled by chromosome similarity, better performance can be achieved than using static values, particularly when there is no *a priori* knowledge about the ideal static mutation rate. This is compounded when the solution landscape is large, such as in higher-order N-Queens problems.

*Corresponding author.

GECCO2014 2014 Victoria, BC, Canada

1.1 Background and Problem Domain

Like many other complex optimization problems, the N-Queens problem becomes orders of magnitude more complex as the number of queens and the size of the chessboard increases. Finding all solutions is a simple but non-trivial problem. When two or more queens share a row or diagonal, a collision occurs for each pair of queens, meaning that the board state is not a solution.

The Eight-Queens problem is the classical version of the puzzle, and even with this configuration the problem is computationally expensive. There are $\binom{64}{8}$ arrangements of the queens on a standard 8×8 board. Exhaustive deterministic evaluation has shown that there are only 92 distinct solutions. Even including optimizations such as imposing a constraint to place only one queen to a single row, there are still 8^8 possible distinct arrangements. In higher-order versions of the problem, combinatorial explosion occurs. The Nine-Queens problem has 352 distinct solutions. The Ten-Queens problem has 724. Table **??** on page **??** shows the growth of the number of solutions.

While the number of solutions is known for problems involving up to 26 queens, there is currently no known formula to determine the exact number of distinct solutions. That is to say, higher order N-Queens problems are currently intractable with existing methods. Deterministic methods for approaching these problems are largely useless, and the problem remains unsolved.

- (ADD CITATIONS!
- Cite me![**?**, **?**, **?**, **?**, **?**, **?**, **?**]

1.1.1 The N-Queens Puzzle

The natural question raised when approaching simple problems is why a deterministic or brute force strategy is not

used. Unfortunately, these approaches cannot work within human time-frames except in smaller problem landscapes. The issue is that a direct linear traversal of the landscape does not yield fruitful results easily in the case of the N-Queens problem. This stems from the fact that the fitness of each sequential solution doesn't matter. The problem cannot be solved by producing a solution that has queens which will attack each other. Therefore, only solutions with the absolute maximum, 100% fitness, are useful solutions. What is more, as the problem landscape increases in size, the number of distinct solutions does not increase proportionally. A full brute force traversal of the landscape would be required to find all the distinct solutions, which is increasingly expensive in higher order problems.

Other stochastic solutions are impractical with this type of problem. This is because finding solutions among the landscape of possible board states yields a tremendous amount of duplicate solutions. Some method is required to direct the random traversal of the landscape toward distinct solutions and away from those which have already been found. The problem with this statement, is that the nature of randomness cannot be constrained in this way - it will no longer be random. Therefore methods such as genetic algorithms become an attractive solution approach because they represent a 'smarter' traversal of the problem landscape than what is offered by a linear brute force traversal, or a uniformly random traversal of the landscape. Less time is spent on areas of the landscape which are not likely to be solutions. This leads us toward the core motivation of our solution's behaviour.

- (ADD CITATIONS!)

1.1.2 Motivation

Modelling the randomness of the solution landscape as seen in the N-Queens problem is impossible with known mathematical methods. To produce a geometric solution landscape would require that the dependent axis of the coordinate plane maps each sequential value to a uniformly random value in the actual solution landscape. Conceptually this is very difficult to imagine, but the reason why this conceptual strategy is useful will become clearer. First, we will discuss why a new approach is necessary in comparison to traditional GA approaches.

There are several 'short-cuts' which are available when approaching the N-Queens problem because of the nature of the chess board. Since it is square, one solution in fact can be used to yield many other unique solutions using symmetry operations (reflection and rotation of the board). This means that up to 8 solutions can be generated by finding one distinct solution. However, symmetry operations cannot be used to find all distinct solutions because many symmetrical reconfigurations of the board are equivalent. In the example of the Eight-Queens problem, there are 11 board configurations which yield 8 solutions, and 1 which yields only 4. The result is still 92 unique solutions, with the majority of them calculable through symmetry operations alone.

Using symmetry operations, our initial exploration into solving the N-Queens problem yielded solutions quickly, but not quickly enough to beat deterministic solution generation methods. This was especially apparent in lower order problems where the solution landscape is small. We began to explore biological definitions of genetics, and sought inspiration from the fact that nature has an apparent system for maintaining genetic variation within organisms. For example pure-bred dogs often tend to have significant health problems which directly result from inbreeding. Royal families often use inbreeding to keep their line 'pure', which also results in genetic abnormalities. (CITE THIS)

These facts led us toward a theoretical approach to apply the negative effects of inbreeding towards genetic algorithms. Our approach would make the mutation rate variable based on chromosome similarity, suggesting the natural mutant products of inbreeding as a result of a too-similar biological chromosome. Our initial solution worked exceptionally well, providing all 92 solutions nearly instantaneously in comparison to our previous static mutation rate GA which required a few seconds. This motivated us to pursue the implications of our strategy further.

1.2 Related Work

- This is where you can cite all of the references Joseph found regarding other types of variable mutation. - Joseph can help you out with this section

2. OUR APPROACH

- Describe the variable mutation rate algorithm and the two parameters that it has in comparison to fixed mutation's one param, (step size for changing the mutation rate and the inbreeding threshold used to adjust the mutation rate).

2.1 Applying Variable Mutation Rates

- For explaining the variable mutation rate there is no need for a formal algorithm essentially it boils down to the following:

- For each generation evaluate the population chromose similarity (see below for the population chromosome similiarity algorithm)

- If the population chromosome similarity is less than the threshold increase the mutation rate by the step size. If the similarity is greater than the threshold decrease the mutation rate by the step size. This results in the mutation rate adjusting up or down such that the population chromosome similarity approaches an equilibrium near the inbreeding threshold.

2.1.1 Chromosome Similarity Algorithm

- For explaining the chromosome similarity algorithm (which is a little more complicated to avoid the naiive O(n!) solution, instead it is now linear complexity). Even though in terms of theoretical big-O complexity it is constant (since it's just based on the population size which is constant) it is worthwhile mentioning that from our empirical evidence using profiling of the code using the naiive O(n!) solution has a significant impact for larger population sizes (so much so that it caused a significant degrade in performance motivating us to profile the code and improve the original algorithm and create the following linear complexity algorithm)[**?**]

2.2 Implementation

- An overview of our GA implementation so that someone wishing to replicate our results knows how we implemented our solution.

- Could potentially provide a URL to the source code implementation on github, I think this would be a very good idea.

```
Function Similarity(chromosomes)
    input  : An array of chromosomes
    output: Fraction of chromosomes that are similar
    similar ← 0
    matched ← false
    length ← length of chromosomes
    // Sort using an arbitrary sorting algorithm
    sorted ← Sort(chromosomes)
    for i ← 0 to length − 1 do
        if sorted[i] == sorted[i + 1] then
            similar ← similar + 1
            matched ← true
        else if matched then
            similar ← similar + 1
            matched ← false
        end
        // Case where the last item is a match
        if matched and (i + 1 == length − 1) then
            similar ← similar + 1
        end
    end
    return similar/ length
end
```

Algorithm 1: Chromosome similarity function

2.2.1 Chromosome Design

- Describe the chromosome design, the ordering of each gene of the chromosome represents the horizontal position of each queen on the board and the corresponding vertical position of the queen is an integer stored in the gene (array index) of the chromosome.

–> I think it would be wortwhile to have a very small diagram highlighting this that shows the correlation between a chromosome to the queens on a small (4x4 or 5x5) chessboard

2.2.2 Fitness Function

- Fitness function, each pair of queens that have a vertical or diagonal collision is recorded as a value of 2 (one collision from the perspective of each queen). This number is then represented as a fraction over 1 such that a chromosome with a larger number of collisions will have a much lower fitness value.

- If no collisions occur the default value is 1 resulting in a fitness of $(1/1 = 1)$ which is a solution to the problem.

- For queens with multiple collisions each pair of queens will be recorded as 2 collisions, in the event that there are three pairs of queens with collisions, one pair with a vertical collision, one with a diagonal, and one with a separate diagonal collision the fitness value would be (1 / 6).

- In the event where there are multiple types of collisions (say two pairs of queens each with vertical collisions, and one with a diagonal collision) the fitness value would be (1 / 8), for a higher number of collisions this often makes the fitness value even lower than just a multiple of 2 x pairs of collisions, the fitness value decreases more than linearly for a higher number of collisions.

–> Might be helpful to give a diagram

–> Due to the ordering of the queens in the chromosome there can only be vertical or diagonal collisions (since two

```
Function Fitness(chromosome)
    input  : A single chromosome
    output: A fitness value for the chromosome
    collisions ← 0
    length ← length of the chromosome
    for i ← 0 to length − 1 do
        // Check each gene against the current
        j ← (i + 1)  mod length
        while j != i do
            y_i ← chromosome[i]
            y_j ← chromosome[j]
            // Check for vertical collision
            if y_i == y_j then
                collisions ← collisions + 1
            end
            // Check for diagonal collision
            if abs((i − j) / (y_i − y_j)) == 1 then
                collisions ← collisions + 1
            end
            j ← j + 1
            j ← j  mod length
        end
    end
    if collisions == 0 then
        return 1
    else
        return 1 / collisions
    end
end
```

Algorithm 2: Fitness function

queens can never be on the same row).

2.2.3 Selection Method

- Roulette wheel selection method, by generating a random floating point number and selecting a chromosome value where the random number lies within the bounds. The range of random values that can be chosen changes based on the maximum upper and lower bounds of the sum of the fitness of the chromosomes in the population.

- Each of the ranges of values that a particular chromosome can have is weighted based on the fitness of the chromosome. For example in a population with 4 chromosomes two of which are solutions (fitness 1) and two with a single pair of collisions (fitness $1/2 = 0.5$) the ranges of values for each chromosome would be as follows:

[0, 1), [1, 2), [2, 2.5), [2.5, 3), and the bounds of the random floating point value generated for the roulette wheel selection would be [0, 3).

2.2.4 Chromosome Operations

- Crossover operation (70% chance), cloning (30% chance), (cite paper justifying why we used these values, they are often recommended/used values)

- Background mutation operation, applied to chromosomes, this is variable rather than a traditional fixed value, e.g. a 5% value results in a 5% chance of mutation being applied to chromosomes.

- Mutation operator changes ONE of the genes of the chromosome randomly, which results in changing the the y coordinate of a random queen in the chromosome to a random value within the range of possible y values.

- Mutation operator DOES NOT result in an invalid chromosome, it is limited to the range of possible (valid) y values.

2.2.5 Chromosome Evaluation

- If a chromsome of the current population has a fitness value of 1 it is compared to the list of previous solutions to see if it is unique. If the solution is distinct the rotation (rotating the solution an additional three times) and reflection (performing reflection on the solution) followed by three additional rotations operations are applied to find a total of 8 solutions. Each solution found after rotation and reflection is compared to the list of previous solutions to verify that it is a distinct solution.

–> We do not keep duplicate solutions, this is important since someone may think that out of the 1000's of solutions we found many of them are just duplicates, when in fact they are all DISTINCT solutions.

- The current population is then replaced with the new population that was created by applying the crossover, cloning, and mutation operations.

2.3 Methodology

- Implementation in Java

- Experiments were conducted on the HPC facilities provided by SHARCNET

- describe the study and control (diff. fixed mutation rates, list them all, vs. variable mutation rate with a fixed inbreeding threshold of 15%), this had 1 fixed param (inbreeding threshold)

–> This should probably be put in a table. I'll create one for you

- N queens problem sizes used: For 8 - 16 queens used each n queens size, for 16 - 26 used each even sized N queens problem, lastly a test using 32 queens.

- Number of generations for each N queens problem (10 million generations for all except 32 queens), 50 million generations were used for 32 queens given the increased complexity of the problem.

- sample sizes (30, 15, 10)

–> I will give you a table with each n queens problem, the sample sizes used for each and the number of generations, this would reduce a lot of text needed to explain the items.

–> justify why the sample sizes were reduced for larger problems given the computational limitations of SHARCNET (max 256 jobs, 7 days CPU time, regardless and your job would be killed). We would like larger sample sizes (100+ runs) but given the CPU time limitations of SHARCNET could not.

- describe the data collected, explain how each of the attributes such as population fitness, similarity are calculated based on the mean of the population for 1000 generations at a time (mean of means), rather than the mean of each population for each generation (TOO MUCH DATA!)

2.4 Why Not Fixed Mutation?

- This should be part of the end of results basically summarizing the pros and cons of fixed mutation and variable with reference to how our results show that in our case (for certain larger problems) variable mutation was better.

3. RESULTS

- cite specific results and why we think they are important, what is the significance of them and how they support our results/hypothesis that in the case of N Queens var. mutation is better than fixed.

- cite the result showing the best fixed mutation vs. the variable mutation for each N-queens problem, try and use both the figure and the table, the table has some additional interesting information which cannot be conveyed in the image alone.

–> I will create the table for you, essentially showing each N queens problem, the best results for fixed mutation vs. the best results for variable mutation

- cite interesting results of the fat boxplots in the range of chromosome similarity for the optimal fixed mutation rates, use the "person stepping" analogy and how that certain fixed mutation rates had the widest range in chromosome similiarty allowing it to hone in on solutions faster.

- Try and incorporate one of the scatter plots as well that show very interesting results and see if you can use it to compare/contrast the results of the following plots

- variable mutation rate scatter plot - variable mutation rate similarity scatter plot - best fixed mutation rate similarity scatter plot

4. CONCLUSIONS

- Wait until the rest of the paper & we have more feedback before writing the conclusions.

- Further research into adjusting the inbreeding threshold, for the purpose of the research a constant inbreeding threshold of 15% was used, however further research could be done in testing different thresholds.

- Comparing the results of variable mutation with fixed mutation using other types of combinatorial/optimization

Table 1: Number of unique solutions to the N-Queens Problem

N	8	9	10	11	12	13	14	15	16	17	18	19	
Distinct Solutions	92	352	724	2680	14200	73712	365596	2279184	14772512	95815104	666090624	4968057848	3

problems such as TSP, constraint satisfaction problem (CSP), etc.

- Variable population size based on the amount of inbreeding (if you have a lot of inbreeding in nature the organisms will have higher mutatation rate and deformities, the population will shrink)

- Having the mutation operator affect more than one gene if it goes $> 100\%$?

5. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you.

- Shahryar
- SHARCNET

APPENDIX

A. RESULTS

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Tables

- Additional tables can go here

A.2 Figures

- Additional figures can go here

A.3 Source Code

- Could link to github and possibly the github wiki, or explanation of the R analysis source code.

A.4 Raw Data

- Link to the dropbox URL containing the actual data from SHARCNET used for the research