Exploration of Genetic Algorithms Through the Domains of Knapsack and N Queens.

Fraser Mince
Abilene Christian University
801 Vista Ln
Abilene Tx 79601
972-834-9632
bfm09a@acu.edu

ABSTRACT

In this paper I apply genetic algorithms to the domains of both Knapsack and N Queens. I find that knapsack works exceptionally well while N Queens sometimes returns answers with a couple of queens in the wrong place.

Categories and Subject Descriptors

I.2.2 [ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY]: Nonnumerical Algorithms and Problems – *Pattern Matching.*

General Terms

Algorithms, Performance, Experimentation, Theory.

Keywords

Fitness, evaluation, genetic algorithm, multithread, island theory, crossover, two point crossover, genome, bitset, N Queens, Knapsack.

1. INTRODUCTION

In this project I experimented with different domains and methods of using genetic algorithms, in order to make a program that can efficiently find the answer to the respective domains. This involved determining encodings, writing evaluation functions, dealing with mutation, dealing with collision. I decided to experiment on two different domains: N Queens, and Knapsack.

2. N QUEENS

2.1 Basics and Encoding

The basic idea of N Queens is finding how you can put N Queens on a N by N chess board. I first tried to implement the N Queens encoding using a N * N length bitset with 1s for the queens and 0s for everything else. Unfortunately I was having to go way out of my way to make sure each genome was a valid solution with no longer than N queens. These genomes also took much more space lots of that space filled with the unnecessary information (the zeroes).

2.2 Improved Encoding

To simplify the process and boost efficiency I changed to an integer encoding. The index of the numbers represent the columns and the numbers represent the row. This means that each encoding will always be a collection of N queens. This also means that unnecessary information is not stored, cutting down on time to evaluate the genomes and calculate the answers.

2.3 Evaluations

I evaluated N Queens by checking if each queen had a conflict. I only allowed each queen to have one conflict giving each genome to have between 0 and N conflicts. I used this number and found the average of every genome in the set then calculating avg/evaluation. I reversed these from the standard evaluation/avg because I wanted the genomes with smaller evaluations to reproduce more. I used these numbers in order to determine the probability of genome reproduction.

2.4 Fitness

For my fitness function I used remainder stochastic sampling to create my intermediate population. If the number is 1.0 or greater I include the genome in the intermediate population floor(fitness) times. Regardless of what the original number is I then subtract fitness - floor(fitness) and use that as a probability that the genome will be included again.

2.5 Breeding

In order to produce the next generation of offspring I used a two point crossover. I picked a random start and end for the two point crossover and treated the string as if it were a circular object. I then took the pieces between start and end from the first parent and everything else from the second. I also included a 5% chance for mutation on a random bit within the child. This child is only added to the population if it is better than the worst member of the population. In this case the worst member is removed and replaced with the new child.

2.6 Convergence

For convergence I originally assumed that I would wait until each genome was the same. I soon realized that this would pose a problem. In N Queens there are many cases where there will be several different Genomes with the same evaluation. This means they will not be replaced and will not converge. Instead I define convergence as the state where every genome has the same evaluation score.

2.7 Cataclysmic Mutation

Upon convergence I cataclysmically mutate, skipping over the first genome in my population in order to preserve the original and then giving every number in every genome after that a 50% chance to mutate randomly to a different number.

2.8 The Island Model

I implemented a multi-threaded island model. I took one thousand different populations that run at the same time and after ten cycles I get the best genome from each population and give it to the other populations. This allows for more diversity of populations and allows for interbreeding between populations.

2.9 Effectiveness

Though I experimented mostly on N Queens I discovered that it is a hard problem to solve using genetic algorithms. Because there is an absolute answer we are trying to reach there are many times (especially on bigger sets) in which my algorithm will still have conflicts within the best answer I get back. While it will give back an answer back much sooner than brute force it is especially obvious when the answer is not optimal.

2.10 Experimentation

I experimented with several different variables to see what would improve my algorithms performance. I changed the number of times I cataclysmically converge(finally deciding on 1000). This allows me to get zero much more often at the cost of taking longer. I also experimented with the number of thread before discovering that the only way the number of thread effects runtime is that the more there are the more likely one is to take a long amount of time. This is not a large bottleneck so I decided to change from five threads to one thousand.

3. KNAPSACK

3.1 Basics and Encoding

For Knapsack I used a binary encoding with each bit representing if a item in a sack kept or ignored. I use these numbers to produce a close to optimal answer of weight and value pairs. The evaluation is simply the value. If the Item combination makes the bag overweight then the evaluation is zero. The fitness is calculated in the exact same way as in N Queens except it is eval/avg.

3.2 Breeding

The main significant difference between the breeding of N Queens and Knapsack is that Knapsack uses a random crossover. This just means that each bit of the child is randomly chosen between the two parents. Since there is so significance to the actual patterns in knapsack we instead opt to ignore these patterns and choose inheritance on an individual bit by bit level.

3.3 Effectiveness

This program seems to work very well. It is able to solve the problem with a near optimal answer and in a much faster amount of time than the brute force method. As the chart below states close to optimal answers can be found in a fraction of the amount of time it takes to do brute force.

4. SUMMARY

Genetic algorithms are a powerful tool that allows a NP problem to be taken and solved with a near maximum answer in a much faster amount of time. These algorithms are powerful tools that make use of a novel strategy to solve problems in a unique way.

5. FUTURE WORK

Continue to expand my multithreaded island method. This is the first time I have ever multithreaded anything so I do not know if I did it right though it seems to work. Look at other strategies for optimizing the breeding and selection process.

Knapsack Evaluation Number(Recursive Calls Vs Evaluations)

	5 Items	10 Items	15 Items	20 Items	25 Items	30 Items	35 Items	40 Items
Brute Force	32	1,024	32,768	1,048,5	33,554,	1E+09	3E+10	1E+12
Genetic	2,653	636	6,810	8,725	10,388	12,507	17,453	18,349

Knapsack Evaluation Values								
	5 Items	**10 Items**	**15 Items**	**20 Items**	**25 Items**	**30 Items**	**35 Items**	**40 Items**
Brute Force	16	43	61	90	115	134	143	175
Genetic	16	43	61	90	115	129	143	175

Percentage of Times That There is Zero Collisions (Out of Ten)						
Number of Queens	**5.0**	**8.0**	**14.0**	**20.0**	**30.0**	**40.0**
Genetic	1.0	1.0	1.0	0.5	0.4	0.3