Genetic Algorithms Applied to the Knapsack Problem

Christopher Queen
Department of Mathematics
Saint Mary's College of California
Moraga, CA

Essay Committee:
Professor Sauerberg
Professor Jones

May 16, 2016

Contents

1 Abstract

A genetic algorithm is a relatively new search algorithm that attempts to solve a problem using a process that mimics biological evolution. Genetic algorithms have been applied to many different problems including aircraft design, financial forecasting, and cryptography. To understand the process of a genetic algorithm, we will apply a genetic algorithm to the knapsack problem, walk through the steps a genetic algorithm might take, and analyze the results of each process. We will first examine how to find solutions to a knapsack problem without a genetic algorithm, and then we will define a genetic algorithm and apply it to a knapsack problem.

2 Knapsack Problem

2.1 Overview

Imagine you have a knapsack that can only hold a specific amount of weight and you have some weights laying around that you can choose from. The knapsack problem deals with finding combinations of those weights to reach the target weight for the knapsack. As a simple example, if we have weights weighing $1lb, 2lb$, and $5lb$, can we find some combination of these weights to reach a target weight of $6lb$? The solution to this problem is clearly the combination of $1lb$ and $5lb$. Stated mathematically, given some number of weights and a target $t \in Z^+$, is it possible to express t as a sum of distinct weights? To explore the knapsack problem in terms of sets and subsets, let S be a set such that $S \subset Z^+$. Is it possible to find a subset S' of S whose elements sum to a target t?

2.2 Application

Using a knapsack cipher of size 8, we can encrypt plaintext into ciphertext by converting solutions to a knapsack problem into the binary representation of an ASCII character. Suppose we want to encrypt the characters in the string "HELLO". First we need to construct a set $S \subset \mathbb{Z}^+$ of size 8. To make this knapsack problem easily solvable, we will construct S with elements $x_1, x_2, ..., x_8$ such that $x_{i+1} \geq 2x_i$. The subset S will be defined as

$$S = \{126, 262, 524, 1117, 2234, 4491, 8995, 17990\}$$

Converting the characters of the string to their binary representations of ASCII characters and summing the elements of S that are located in the same position as every 1 in the binary representations, we can construct the following table.

Plain Text	ASCII	Target Value
H	01001000	2496
E	01000101	22743
L	01001100	6987
L	01001100	6987
O	01001111	33972

Now suppose that we want to decrypt this message and we only have the previous set S and the target value for each character. There are many different algorithmic approaches to this problem that vary in speed and efficiency. In this case we will define a greedy algorithm to solve the problem by consecutively selecting the largest weights to see if they can be used in the subset solution.
To find a target t with a subset of size n,

- Set an integer $j = n$
- While $j > 1$
  - If $t \geq x_j$ where $x_j \in S$
    1. Add x_j to the solution subset S'
    2. Set $t = t - x_j$
  - Set $j = j - 1$

If the elements in S' do not sum to t when $j = 1$ (when the loop terminates), then the solution subset does not exist.

Using the algorithm above, the maximum number of elements the algorithm need to check is n, therefore this is an $O(n)$ algorithm. This is a very efficient algorithm. If we were to use an algorithm that simply tested all possible combinations of elements in S, it would be an $O(2^n)$ algorithm, which is far less efficient. Knapsack problems are NP-hard problems.

3 Genetic Algorithms

3.1 Overview

As stated in the introduction, a genetic algorithm is a type of search algorithm that mimics Charles Darwin's theory of evolution. Darwin stated that through natural selection, the most fit individuals in an environment survive and pass their traits on to future generations while the least fit individuals die away along with their traits. It's important to note that each individual's fitness level is determined by it's environment. Polar bears are more "fit" in the arctic for example because their white fur allows them to blend in with the snow and catch their prey more easily, while brown bears are spotted easily and are unable to catch any food. In terms of genetic algorithms, an "environment" is any kind of problem that the algorithm needs to solve. So if a genetic algorithm is being applied to finding a solution to $x + 5 = 10$, then $x + 5 = 10$ is the algorithm's "environment". Individuals in a genetic algorithm are possible solutions to the problem at hand. In the case of finding solutions to $x + 5 = 10$, all possible x values are possible solutions to the problem, although most options are not true solutions. These concepts will be explored in more depth in later sections of this paper.

3.2 Applications

This paper focuses on how to apply a genetic algorithm to solve a knapsack problem with knapsack sequence of size 16. The following example with a set S and target t will be used throughout the remainder of this paper.

$$S = \{2642, 1914, 8847, 5726, 1640, 7266, 4127, 9869, 3320, 1870, 3293, 1161, 4727, 9248, 4145, 1664\}$$
$$t = 3534$$

The solution subset to this knapsack problem is the set

$$S' = \{1870, 1664\}$$

since $1870 + 1664 = 3534 = t$.

In this problem, our variables S and t make up our environment. Taking a look at the set S, we can create a binary representation of S' by setting a 1 in the position of elements taken from S and a 0 in the position of elements not taken from S.

$$S' = \{1870, 1664\} = 0000000001000001$$

Taking a look at a unicode table, we can see that this binary number is the binary representation for the letter "A ". The following sections describe the process a genetic algorithm carries out to find the solution to this knapsack problem.

3.3 Process

To begin, the genetic algorithm randomly generates some number of potential solutions to the problem we are trying to solve. In this paper we call these solutions "individuals". A genetic algorithm loops through until a termination condition is met, hopefully resulting with one of the solutions being the solution to the problem. As the algorithm loops through, it compares the fitness function's results with other individuals. Individuals that are seen as more favorable go through a reproduction process to create a new generation of individuals. This is done through recombination where a new potential solution is generated by selecting a combination of elements from both parent solutions or through mutating a single individual. The different techniques used for the selection process are crossover, mutation, and reproduction. These processes will be discussed more thoroughly later on in this paper.

To provide an understanding of each process that a genetic algorithm undertakes, we will use a simple example of a genetic algorithm applied to a knapsack problem following the steps detailed in the image below.

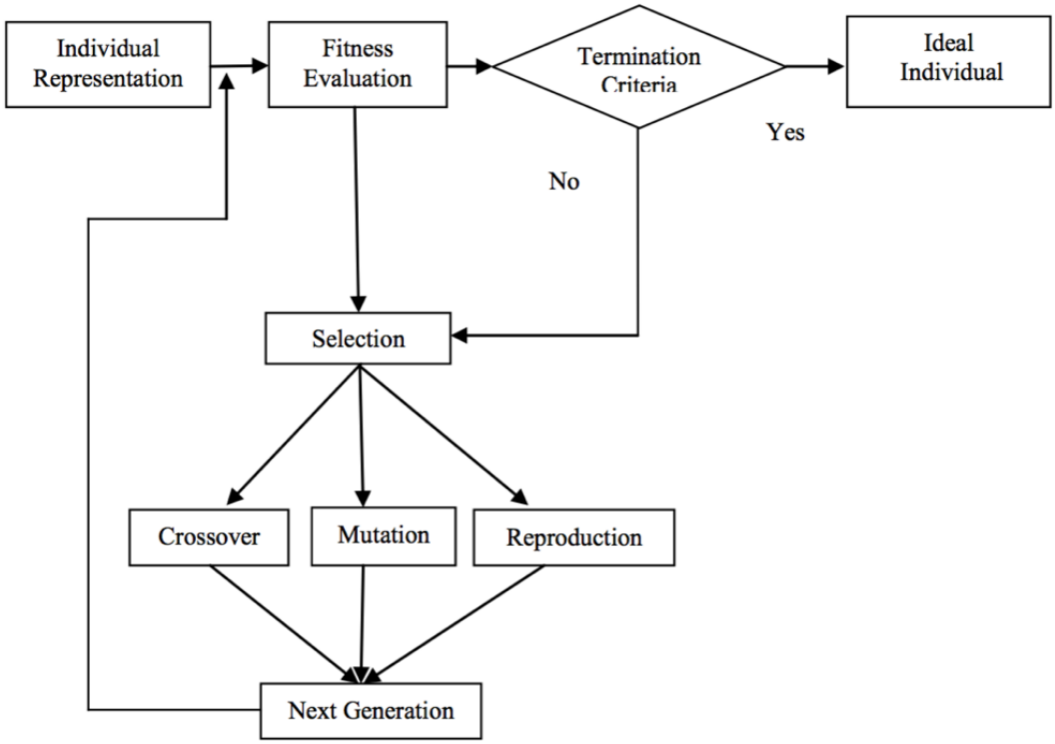


Fig 1: Genetic Algorithm Process

3.4 Individual Representation

As was stated previously, an individual in a genetic algorithm is the possible solution to a given problem. With a knapsack problem, a possible solution to the problem is a subset $S' \subset S$ where the elements of S' are intended to sum to a target t. In section 2.2, we found the binary representation of the subset S'. We will think of individuals in this environment as binary representations of subset solutions of the knapsack problem. Since our knapsack sequence is of size 16, any 16 digit binary number can be an individual this particular genetic algorithm. Since a subset solution can potentially be any subset of S, the number of possible individuals in our search space is $2^{16} = 65536$.

When a search space is so large, finding a solution to the problem is like finding a needle in a haystack. In most cases, the author of a genetic algorithm will not know where to start and so the genetic algorithm will need to create some number of individuals to make up it's initial population at random. The number of individuals to populate the first generation is a parameter that needs to be set by the author. The number of individuals in the initial population will affect the efficiency of the algorithm, as we will see in section 4.2. Let's assume the genetic algorithm randomly generated the following individuals to make up the initial population. For simplicity, this algorithm will be set to generate only three individuals to create the initial population.

$$1111100101110101$$
$$1101100000010000$$
$$0010000000000001$$

3.5 Fitness Evaluation

Converting our individuals in our initial population to their subset representations and summing the elements of each subset, we have the following data.

$$\begin{aligned}
1111100101110101 &= \{2642, 1914, 8847, 5726, 1640, 9869, 1870, 3293, 1161, 2248, 1664\} \\
&= 40874 \\
1101100000010000 &= \{2642, 1914, 5726, 1640, 1661\} \\
&= 13083 \\
0010000000000001 &= \{8847, 1664\} \\
&= 10511
\end{aligned}$$

As we can see, none of the subset sums above equal our target value of $t = 3534$. Some of these subset sums are closer than others however. We can see that the individual 0010000000000001 whose subset representation sums to 10511 is the closest to the target $t = 3534$.

In order for the algorithm to determine which individuals are closest to the target, or which individuals are most fit, the genetic algorithm uses a

fitness function to evaluate each individuals fitness value. Fitness functions vary depending on the context of the algorithm. If implemented poorly or incorrectly, a fitness function can result in misleading data or inefficiencies. Observe the following fitness function [1] for example.

$$f(S') = 1 - \left(\frac{t - sum(S')}{t}\right)^{1/2}.$$

In this case, we want this fitness function to generate a number between $0 < x \leq 1$ where the closer the value is to 1, the more "fit" the individual is. If the fitness value is equal to 1 then the individual associated with the fitness value is a solution to the problem. This function works as long as the $sum(S') \leq t$. If the sum of the subset is greater than the target value however, our fitness function will result in misleading data. Therefore we need to add to our fitness function to handle cases where $sum(S') > t$.

$$f(S') = 1 - \left(\frac{t - sum(S')}{t}\right)^{1/2} \qquad (\text{if } sum(S') \leq t)$$

$$f(S') = 1 - \left(\frac{(sum(S') - t)}{max(t, sum(S) - t)}\right)^{1/6} \qquad (\text{if } sum(S') > t)$$

This function will now generate a value between $0 < x \leq 1$ for any individual passed through the function. Now that we have defined our fitness function, we will pass each individual in our population through the fitness function and achieve the following results.

$$f(1111100101110101) = 0.095$$
$$f(1101100000010000) = 0.279$$
$$f(0010000000000001) = 0.316$$

We can see that the individuals can be ranked from most fit to least fit in the following way.

$$0010000000000001 \quad (1)$$
$$1101100000010000 \quad (2)$$
$$1111100101110101 \quad (3)$$

3.6 Selection

Now that the genetic algorithm can determine which possible solutions are more fit than the others, the algorithm will use some selection criteria to perform one of the following operations on specific individuals: reproduction, mutation, or crossover. The selection criteria is another parameter of the algorithm determined by the author. The selection criteria will also influence the efficiency of the algorithm so it is important that the criteria is selected thoughtfully. The most effective way to determine what selection criteria is most efficient is to run the algorithm several times and analyze the results with different criteria. If the algorithm were dealing with thousands of individuals to select from, it would select individuals using "tournament selection". In tournament selection, individuals are grouped into "tournaments" to be selected for recombination based on their fitness values compared to the other individuals in their tournament. In this example, since we are dealing with a small population pool, we will define the selection process in the following way: crossover the two most fit individuals, mutate the least fit individual, and reproduce the most fit individual.

3.7 Crossover

As stated in the previous section, our algorithm will crossover the two most fit individuals. So we will crossover 1101100000010000 and 0010000000000001. The different crossover techniques we can use are single-point crossover, two-point crossover, uniform crossover, and arithmetic crossover [3]. In this example we will use a single point crossover. This essentially means that we will be combining half of the elements in the first set with half of the elements in the second set.

In this single point crossover, we will combine the first 8 binary digits of the individual with the largest fitness evaluation with the last 8 binary digits

of the other individual. Visually, we will be creating a new individual by combining the boldfaced binary digits illustrated below

0010000000000001
11011000**00010000**
0010000000010000 (New Solution)

Converting our new solution back to a subset of S, we have $\{8847, 1661\}$. Passing this value through our fitness function we have

$$f(0010000000010000) = 0.324$$

3.8 Mutation

Our least fit individual 1111100101110101 will go through a mutation process. Mutation on binary digits is the process of inverting certain bits in the binary number [3]. Bit inversion simply means that a 1 is changed to a 0 and vice versa. Since 1111100101110101 the farthest individual from the solution, it will undergo a mutation process that will drastically change it's subset representation. In general, the number of bits and position of bits to be inverted are chosen at random. In this mutation process, we will invert the first 5 bits of the individual.

1111100101110101 (Before mutation)
0000000101110101 (After mutation)

This subset representation of the new individual 0000000101110101 is

$$S' = \{9869, 1870, 3293, 1161, 9248, 1664\}$$

whose elements sum to 27105. This sum is certainly closer to t than the individuals previous sum of 40874. Passing the new individual through the fitness function, we have

$$f(0000000101110101) = 0.162.$$

The fitness value for this new individual is closer to 1 than it's previous fitness value of 0.095 so the mutation process happened to produce an individual closer the solution to the knapsack problem.

3.9 Reproduction

Reproduction is the process of simply passing an individual from a past generation onto the next generation. Since the individual 0010000000000001 from the first generation produced the highest fitness value of 0.316 of all other individuals in the first generation, it may be useful to keep it around. It may be helpful to crossover this individual with less fit individuals to find the solution, or it can be mutated in the case that it is the least fit individual in some generation.

After performing the crossover, mutation, and reproduction operations, the second generation is 0010000000000001, 0010000000010000, and 0000000101110101.

Observing the new fitness values, we can see that overall our new generation is closer to the solution than our previous generation.

First Generation

$f(1111100101110101) = 0.095$
$f(1101100000010000) = 0.279$
$f(0010000000000001) = 0.316$
Average fitness: 0.230

Second Generation

$f(0010000000000001) = 0.316$
$f(0010000000010000) = 0.324$
$f(0000000101110101) = 0.162$
Average fitness: 0.267

As we can see, the average fitness value has increased by 0.037 from the first generation to the second generation. It is ideal that our average fitness value increased, but without running the algorithm through several more generations, we do not have any insight into how efficient our algorithm is yet. Once we know how many generations the algorithm needed to create in order to find a solution, we can change parameters around to optimize the algorithm.

3.10 Termination Condition

As was stated in section 3.5, in this particular genetic algorithm, the fitness function generates a number $0 < x \leq 1$. In this case, a fitness value closer to 0 means the individual is less fit and a value closer to 1 means the individual is more fit. If an individual's fitness value is equal to 1, then that individual is a solution to the problem. Therefore a fitness value of 1 is the termination condition in this algorithm. If none of the individuals from a particular generation generate a fitness value of 1, then the algorithm passes each the individuals through the recombination process to create further generations. In the case that the algorithm cannot generate a fitness value of 1 after creating some number of generations, the algorithm could change it's parameters and try again. An example of this would be to mutate different bits or crossover different individuals when creating new individuals.

4 Genetic Algorithm Efficiency

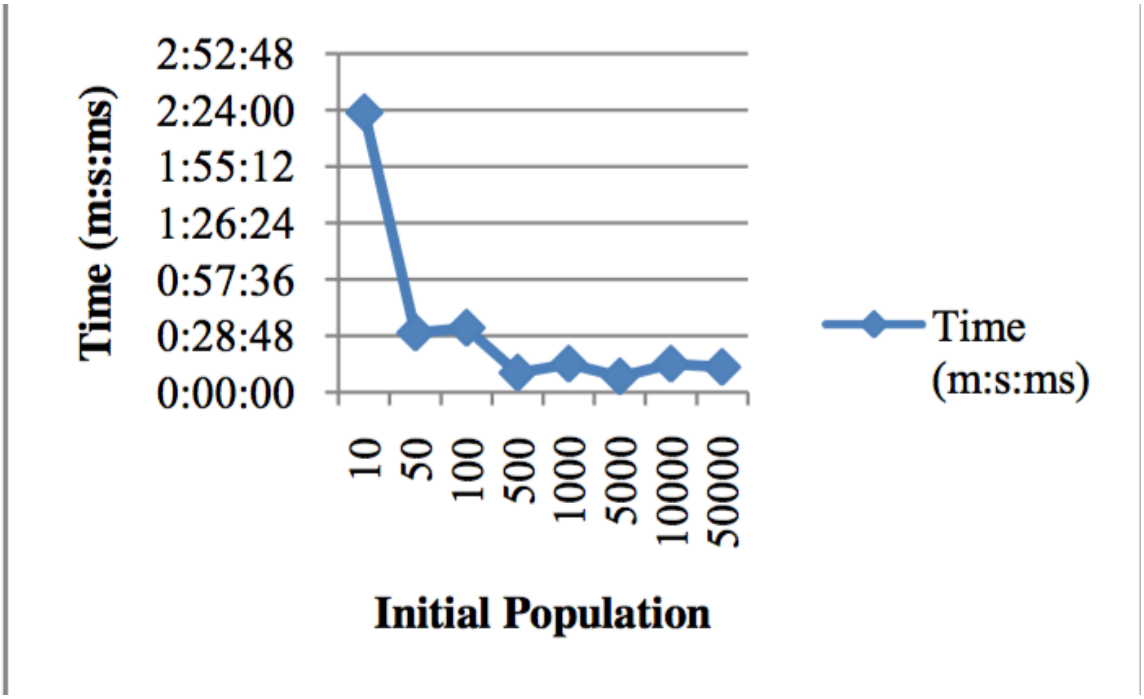
4.1 Introduction

As we have seen, there are many factors that can increase and decrease the efficiency of the algorithm. This section will reference the efficiency of a genetic algorithm applied to a knapsack cipher of size 16 created by Dr. R. Geetha Ramani and Lakshmi Balasubramanian. Their algorithm was applied to decrypting the sentence "The quick brown fox jumps on lazy dog.". The following data represents the maximum time taken and number of generations required to reach the a character in the string of text.

4.2 Efficiency of Population Changes

In the example from section 2, we chose a population size of three individuals for the sake of simplicity. As we can see from the graph below, it is usually the case that the more individuals that exist, the faster the algorithm can find the solution. The graph does show however that the algorithm takes slightly longer when the population size is greater than 10000 than when the population size is 5000. This means that the time taken for the algorithm is not directly proportional to the number of individuals in the population,

but is generally faster with more individuals than with fewer individuals.



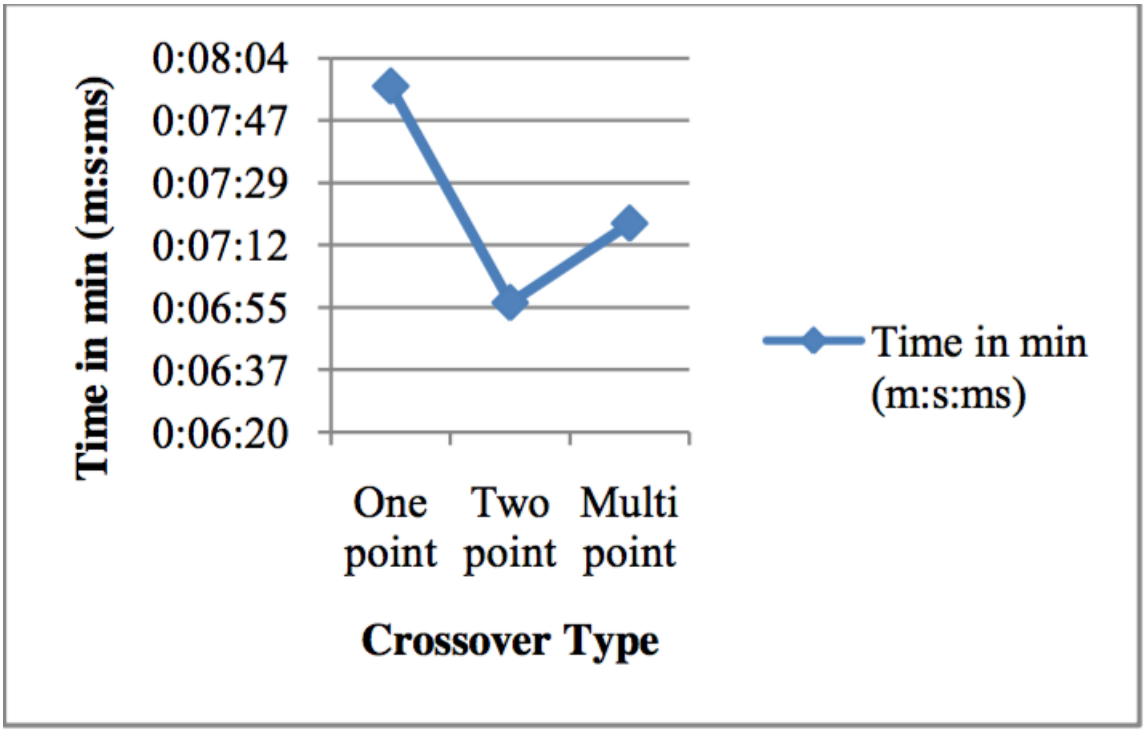
4.3 Effect of Crossover Type

In the example crossover operation from section 2.7, we used a single point crossover, meaning the new individual was creating by combining only one part of the parent individuals. In a two-point crossover, two parts of the parent individuals are used to create the new individual. For example, if we crossover 10011011 and 00111010 by selecting the first two and last two digits from 10011011 and the middle four digits from 00111010, we have

10011**11**
00**111**10
10111011 (New Solution)

A multi-point crossover is done by crossing over multiple parts of the parent binary digits. The following chart shows the difference in the time taken

to find the solution using one point, two point, and multi-point crossover. We can see here that using two point conversion resulted in the fastest time for the algorithm to find a solution.



4.4 Effect of Recombination Type

The following table represents the number of generations and time taken to reach the solution when given certain probabilities for recombination techniques. In case 1 for example, the probability that the algorithm will crossover elements to create the next generation is 90%, where the probability for mutation is 10% and reproduction is 0%. The results from this chart tell us that the algorithm in this case is most efficient when using crossover

the most, never using reproduction, and hardly using mutation.

	CP	MP	RP	Generations	Time in min (m:s:ms)
CASE 1	0.9	0.1	0.0	20	0:07:56
CASE 2	0.8	0.1	0.1	33	0:08:06
CASE 3	0.1	0.8	0.1	48	0:11:11
CASE 4	0.5	0.5	0.0	40	0:12:04
CASE 5	0.1	0.9	0.0	25	0:10:30

5 References

1. Dr. Ramani, L. Balasubramanian, Genetic Algorithm solution for Cryptanalysis of Knapsack Cipher with Knapsack Sequence of Size 16, International Journal of Computer Applications, 2011

2. B. Delman, Genetic Algorithms in Cryptography, Rochester Institute of Technology, 2004

3. Obitco. (n.d.). III. Search Space. Retrieved April 28, 2016, from http://www.obitko.com/tutorials/genetic-algorithms/search-space.php