# Solving Subset-Sum Problem by using Genetic Algorithm Approach

Jahanzeb Maqbool Hashmi
201224444
jahanzeb@ajou.ac.kr
Dept. of Computer Engineering,
Ajou University

November 27, 2012

## 1 Introduction

Genetic Algorithms are evolutionary algorithms used to solve non-deterministic polynomial time problems generally known as *np*-problems. The search space of the problems are too large to be solve by conventional approaches. The basic idea of Genetic approach is analogous to biological evolution where only the *fittest* candidates among the species survive in the next generation. The basic evolutionary operators (*recombination* and *mutation*) create necessary diversity and thereby facilitate novelty while *selection* ensures the quality.

The general flow of evolutionary algorithms [1] are given in Algorithm 1

**Algorithm 1:** Program Flow of an evolutionary Genetic Algorithm

```
Initialize the population with initial random seed;
while exact or closest to exact solution not reached so far do
    SELECT parents;
    RECOMBINE pairs of parents;
    MUTATE the resulting offspring;
    EVALUATE new candidates;
    SELECT individuals for the next generation;
end
```

In this term-project, an attempt is made to solve the well-known '*Subset-Sum*' problem using Genetic approach. The rest of the report is organized as follows: Section 2 describes the *Subset-Sum* problem in general. Section 3 describes the genetic algorithm, flow chart and implementation details of the problem. Section 4 demonstrates the implementation with the help of a simple and easy to understand example. In the end, section 5 concludes the work.

## 2 Subset-Sum Problem

Subset-Sum problem is an important algorithm which has its applications in wide range of domains like complexity theory, cryptography and cooperative voting games [2]. Several researches have been done and the researchers showed that it belongs to the *np*-complete class of problems [3].

In the *Subset-Sum* problem, a set W of n integers and a large integer $\tau$ (target) are given. We are interested in finding a subset S such that the sum of elements of S should be equal to the target $\tau$. The mathematical description of the problem is stated as below. Let;

$$W = \{w_1, w_2, w_3, \ldots, w_n\} \tag{1}$$

be a set of integers and $\tau$ be a large positive integer, *then* choose a set S, such that;

$$S \subset W \quad \wedge \quad \sum_{i=1}^{n} S_i \leq \tau \tag{2}$$

Above is the basic *Subset-Sum* problem. The Genetic version of the problem is discussed in section 3.

## 3 Subset-Sum Genetic Approach

### 3.1 Mathematical Model and Flow

Since we have the understanding of the problem itself, now we will try to solve this problem using genetic approach. We will modify the equation (2) a bit and formulate the problem once again. We introduce a vector $\vec{x}$ as a feasible solution. Let;

$$FeasibleSolution : \vec{x} = \big(x_1, x_2, x_3, \ldots, x_n\big), where \quad x_i \in \{0, 1\} \tag{3}$$

$$\sum_{i=1}^{n} w_i x_i \leq \tau \quad for i = 1, 2, \ldots, n. \tag{4}$$

$$ObjectiveFunction : P(\vec{x}) = \sum_{i=1}^{n} w_i x_i \leq \tau, where \quad \vec{x} = \big(x_1, x_2, x_3, \ldots, x_n\big) \tag{5}$$

In order to apply GA to *Subset-Sum* problem, the binary string $\vec{x}$ is to be chosen as *genotype*. The fitness function to evaluate the individuals is implemented as described in [2], and is given in equation (6)

$$FitnessFunction : f(\vec{x}) = s.\big(\tau - P(\vec{x})\big) + \big(1 - s\big).P(\vec{x}) \tag{6}$$

where,

$$s = \begin{cases} 1 & \text{if } \big(\tau - P(\vec{x})\big) \geq 0, \quad means \quad \vec{x} \text{ is feasible} \\ 0 & \text{otherwise} \end{cases}$$

## 3.2 Algorithm and Analysis

Figure 1 shows the flow chart of the Subset-Sum problem using genetic approach. The algorithm is shown in Algorithm7 and39. An important point to be noted is that these two algorithms do not cover very basic details like utility functions etc. The reader is advised to go through the source code of the implementation for the sake of clarity.

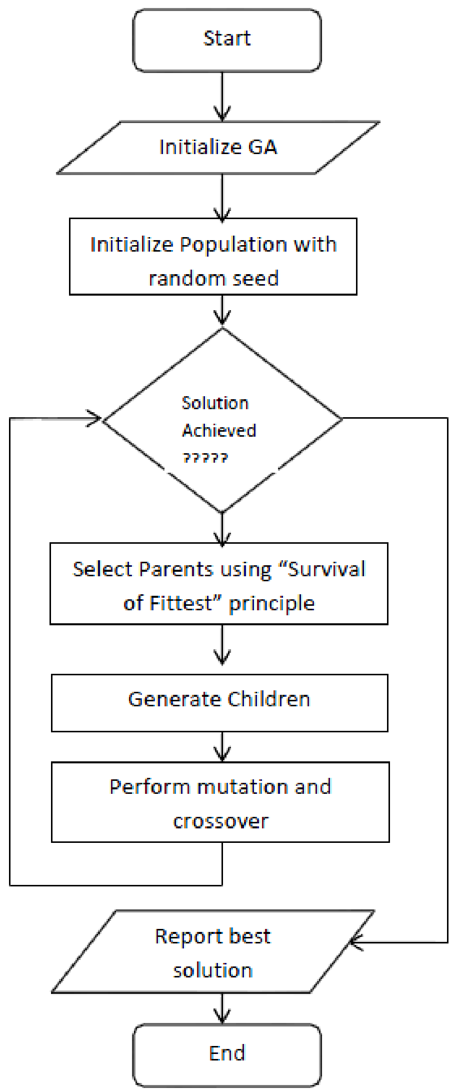


Figure 1: Subset-Sum Genetic Algorithm Flow chart

**Algorithm 2:** Subset-Sum Genetic Algorithm

```
Input: A finite set A = {a_1, a_2, ..., a_n} of integers and a target T.
Let P be the initial randomly seeded population.
numGenerations ← 50000
count ← 0
while count < numGenerations do
    ProduceNextGeneration(P, A, T)
end
return
```

**Algorithm 3:** Produce Next Generation Algorithm

**Input**: Initial population $P$, $A$ and target $T$

$P_n \leftarrow \phi$<br>
**while** $P_n.size < P.size$ **do**<br>
    Let $i$, $j$, $k$ and $l$ be 4 distinct random integers.<br>
    Choose 4 chromosomes $ch1$, $ch2$, $ch3$, $ch4$ at these random indices from $P$.<br>
    Check the fitness between $ch1$ and $ch2$, and between $ch3$ and $ch4$ and let the winners be two parents.<br>
    $w1 \leftarrow winner_{12}$<br>
    $w2 \leftarrow winner_{34}$<br>
    Perform *uniform crossover* on $w1$ and $w2$ with *probability* 0.5 and generate 2 new children *child1* and *child2*.<br>
    $Prob_{mutate} \leftarrow 0.01$<br>
    $r \leftarrow random()$<br>
    **if** $r < prob_{mutate}$ **then**<br>
        $k \leftarrow random(child1.size)$<br>
        **if** $child1(k) = 1$ **then**<br>
            $child1(k) \leftarrow 0$<br>
        **else**<br>
            $child1(k) \leftarrow 1$<br>
        **end**<br>
        $k \leftarrow random(child2.size)$<br>
        **if** $child2(k) = 1$ **then**<br>
            $child2(k) \leftarrow 0$<br>
        **else**<br>
            $child2(k) \leftarrow 1$<br>
        **end**<br>
    **end**<br>
    $isChild1Good \leftarrow child1_{fitness}$ *is better than* $w1_{fitness}$<br>
    $isChild2Good \leftarrow child2_{fitness}$ *is better than* $w2_{fitness}$<br>
    **if** $isChild1Good$ **then**<br>
        $P_n.add(child1)$<br>
    **else**<br>
        $P_n.add(w1)$<br>
    **end**<br>
    **if** $isChild2Good$ **then**<br>
        $P_n.add(child2)$<br>
    **else**<br>
        $P_n.add(w2)$<br>
    **end**<br>
**end**<br>
$P \leftarrow P_n$<br>
**return**

Now, lets have a quick look at the analysis of this algorithm. Generally, the time complexity of the genetic algorithms depends upon the fitness function. There were two ways to implement this problem, either call $produceNextGeneration$ untill the solution is found or fix the number of $Generations$ big enough so that the solution can converge before reaching that limit. For the sake of term project, the latter one is used and the limit is set to 50000 because of time complexity of the algorithm. Let $N_p$ and $N_a$ be the size of the population and the size of the problem set respectively. In Algorithm 7 the outer while loop runs untill 50000 number of generations and for each iteration it calls $produceNextGenerations$. In Algorithm 39 the while loop at $line2$ runs untill the $N_p$. In this while loop, there are other loops that iterates over each over each gene $g_j$ of each chromosome $P_i$ in operations like $crossover$ and $fitness$ (as mentioned before that they are not listed in above algorithms for the sake of clarity, you can view the code for complete picture). Thus these $for$ loops iterate untill the size of each chromosome $N_c$ which is as equal to $N_a$. So the complexity of the $produceNextGeneration$ is;

$$T(n) = 50000 * N_p * N_c \tag{7}$$

$$T(n) = O(N_p * N_c) \tag{8}$$

by ignoring constant value and all the lower order terms. Please note that this time complexity is subject to fixing the number of generation to some constant $k$.

### 3.3 Implementation

The implementation of the *Subset-Sum* problem is done using Java programming language. Pure object-oriented approach is used to design the problem. The '*SubsetSumGA*' is the main class of the program and '*Chromosome*' class represents a candidate of the population. To hold the total population we use the collection i.e. *LinkList* of *Chromosomes*. The program runs until some fixed number of generations say $n$. The reason of using fixed number of generations is that the time complexity should be minimized for the sake of completion of term project. The size of $n$ is kept too large so that the solution should be able to converge.

## 4 Example

Lets us have one simple example of the *Subset-Sum* problem using genetic approach and observe how the solution converges. Consider a set of positive integers $A$ and $N_a$, $N_p$ and $N_c$ are the sizes of $A$, *population* and *chromosome* respectively where $N_g$ is the total number of generation.

$A = \{4, 2, 2, 3, 2\}$ , $N_a = 5$ , $N_p = 5$ , $N_c = 5$ , $N_g = 5$, Target $\tau = 6$

$$G_0 = \begin{cases} (1,0,0,0,0) & \text{sum} = 4 \\ (0,0,1,0,1) & \text{sum} = 4 \\ (1,1,0,1,0) & \text{sum} = 9 \\ (1,1,1,0,0) & \text{sum} = 8 \\ (1,1,0,1,1) & \text{sum} = 11 \end{cases}$$

$$G_1 = \begin{cases} (1,0,0,0,0) & \text{sum} = 4 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,0) & \text{sum} = 4 \\ (0,0,1,0,1) & \text{sum} = 4 \\ (1,0,1,0,0) & \text{sum} = 6 \end{cases}$$

$$G_2 = \begin{cases} (0,1,1,0,1) & \text{sum} = 6 \\ (0,0,1,0,1) & \text{sum} = 4 \\ (1,0,1,0,0) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \end{cases}$$

$$G_3 = \begin{cases} (1,0,0,0,1) & \text{sum} = 6 \\ (0,1,1,0,1) & \text{sum} = 6 \\ (1,0,1,0,0) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \end{cases}$$

$$G_4 = \begin{cases} (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \end{cases}$$

$$G_5 = \begin{cases} (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \\ (1,0,0,0,1) & \text{sum} = 6 \end{cases}$$

As we can see from this simple example that the initial population varies above and below the required solution $\tau = 6$ but after few generations due to cross-over and mutation of the genes, the solution converges.

# 5 Conclusion

As we have seen that the genetic algorithms are very useful for solving np-complete type of problems. The time complexity of such algorithms varies depending upon the fitness function. This subset-sum problem which we solved here corresponds to these type of problems. We have seen the mathematical model, the flow of the program as well as the algorithm. We also observed with the help of an example that how genetic approach works. Finally, we agree that the genetic approach works well where the problem size is too big to be solved by conventional approaches. Note that genetic approaches may not perform well for polynomial time algorithms.

## 6 Disclaimer

This work is done for the sake of completion of Advanced Algorithms course requirements taught by Dr. Satchidananda Dehuri. You are allowed to use this document and the source code it comes with for personal learning purpose ONLY. Any other kind of use which also includes sharing on a public forum is strictly prohibited and subject to the written permission of the author.

## References


[1] A. Eiben and J. Smith, *Introduction to evolutionary computing.* springer, 2008.

[2] R. Wang, "A genetic algorithm for subset sum problem," *Neurocomputing*, vol. 57, pp. 463–468, 2004.

[3] B. Dietrich, L. Escudero, and F. Chance, "Efficient reformulation for 0–1 programsmethods and computational results," *Discrete Applied Mathematics*, vol. 42, no. 2, pp. 147–175, 1993.