Bangladesh University of Engineering & Technology
Department of Computer Science & Engineering
CSE 402: Artificial Intelligence Sessional
Session July 2016

# Implementation of $k$-SAT with Genetic Algorithm

---

## 1 Introduction

In this assignment you have to implement Genetic Algorithm to solve $k$-SAT problem. An instance of $k$-SAT problem is a propositional formula, where each clause has $k$ number of literals. Consider an example of an instance of 3-SAT problem.

$$(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (p_1 \vee \neg p_3 \vee p_4)$$

The objective is to check whether there is an assignment of values to these variables such that the given formula becomes true. The assignment $p_1 = false, p_2 = true, p_3 = true, p_4 = true$ makes the given formula true. This problem is called satisfiability problem (SAT). We consider formulas only in conjunctive normal form.

In our problem, there will be $k$ number of literals in each clause of a formula. There can be any number of clauses and any number of variables in the formula. Our objective is to find an assignment of the variables by Genetic Algorithm so that the given formula becomes true.

Implement another version of $k$-sat problem called Exactly-$m$ $k$-sat problem. In this case, the assignment has to be done in such a way, so that exactly $m$ literals in a clause are true. For example, in an exactly-1 3-sat problem, only one literal is true and other two must be false in each clause.

# 2 Implementation

Genetic Algorithm is given in Figure 1.

**Algorithm** *The Genetic Algorithm (GA)*

```
popsize ← desired population size
P ← {}
for popsize times do
    P ← P ∪ {new random individual}
Best ← □
repeat
    for each individual P_i ∈ P do
        AssessFitness(P_i)
        if Best = □ or Fitness(P_i) > Fitness(Best) then
            Best ← P_i
    Q ← {}
    for popsize/2 times do
        Parent P_a ← SelectWithReplacement(P)
        Parent P_b ← SelectWithReplacement(P)
        Children C_a, C_b ← Crossover(Copy(P_a), Copy(P_b))
        Q ← Q ∪ {Mutate(C_a), Mutate(C_b)}
    P ← Q
until Best is the ideal solution or we have run out of time
return Best
```

Figure 1: Genetic Algorithm

## 2.1 Representation

Each individual is a vector of length $l$ for $l$ number of literals. Each element of the vector is either $true$ or $false$.

## 2.2 Initial Population

Let, the population size be $N$. Randomly generate $N$ number of unique individuals of length $l$. For maintaining uniqueness of each individual, keep a hash value for each individual to check. Checking manually with whole population for uniqueness will take $O(n^2)$, and is thus infeasible.

## 2.3 Fitness Function

Generally better fitness means a better individual that is closer to optimal solution. Fitness $f_i$ for an individual $i$ is calculated in the following manner.

$$f_i = \sum_{c \in C} evaluate(c)$$

Here, $C$ denotes the set of clauses of the $k$-SAT instance. $evaluate(c)$ is 1, if the clause is true, otherwise 0 for the assignment of the literals for the individual $i$.

## 2.4 Selection

We will use Tournament Selection where you will choose $t$ number of individuals from the population randomly, and select the best individual among the $t$ individuals according to fitness function. Use $t = 5$ for the experiment.

**Algorithm** *Tournament Selection*

$P \leftarrow$ population
$t \leftarrow$ tournament size, $t \geq 1$

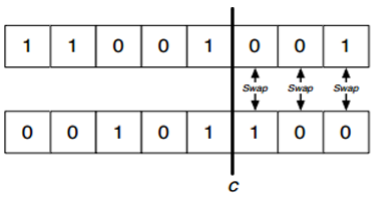
*Best* $\leftarrow$ individual picked at random from $P$ with replacement
**for** $i$ from 2 to $t$ **do**
    *Next* $\leftarrow$ individual picked at random from $P$ with replacement
    **if** Fitness(*Next*) $>$ Fitness(*Best*) **then**
        *Best* $\leftarrow$ *Next*
**return** *Best*

Figure 2: Tournament Selection

## 2.5 Crossover

Two individuals are selected from the population for crossover, and are called parents. Let, $P_a, P_b$ are two parents. Two types of crossovers are taken into consideration with 50% probability for each.

- **One Point Crossover:** It randomly picks an integer $c$ in the range $[1, l]$, inclusive, and swaps all the indexes greater than $c$.



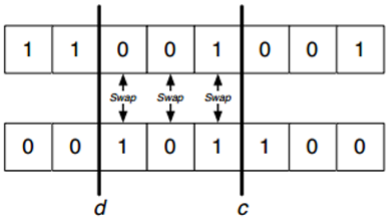

**Algorithm** *One-Point Crossover*

$\vec{v} \leftarrow$ first vector $\langle v_1, v_2, ...v_l \rangle$ to be crossed over
$\vec{w} \leftarrow$ second vector $\langle w_1, w_2, ...w_l \rangle$ to be crossed over

$c \leftarrow$ random integer chosen uniformly from 1 to $l$ inclusive
**for** $i$ from c to $l$ **do**
    Swap the values of $v_i$ and $w_i$
**return** $\vec{v}$ and $\vec{w}$

Figure 3: One Point Crossover

- **Two Point Crossover:** It randomly picks two numbers $c$ and $d$ in the range $[1, l]$, inclusive, and swap the indexes between them.




**Algorithm** *Two-Point Crossover*

$\vec{v} \leftarrow$ first vector $\langle v_1, v_2, ...v_l \rangle$ to be crossed over
$\vec{w} \leftarrow$ second vector $\langle w_1, w_2, ...w_l \rangle$ to be crossed over

$c \leftarrow$ random integer chosen uniformly from 1 to $l$ inclusive
$d \leftarrow$ random integer chosen uniformly from 1 to $l$ inclusive
**if** $c > d$ **then**
    Swap $c$ and $d$
**for** $i$ from $c$ to $d - 1$ **do**
    Swap the values of $v_i$ and $w_i$
**return** $\vec{v}$ and $\vec{w}$

Figure 4: Two Point Crossover

### 2.6 Mutation

We will use bit-flip mutation. For each element of the vector, flip a coin of a certain probability $p$ (often $1/l$, where $l$ is the length of the vector). Each time the coin comes up heads, flip the bit.

**Algorithm** *Bit-Flip Mutation*

1: $p \leftarrow$ probability of flipping a bit
2: $\vec{v} \leftarrow$ boolean vector $\langle v_1, v_2, ...v_l \rangle$ to be mutated

3: **for** $i$ from 1 to $l$ **do**
4: **if** $p \geq$ random number chosen uniformly from 0.0 to 1.0 inclusive **then**
5: $v_i \leftarrow \neg(v_i)$
6: **return** $\vec{v}$

Figure 5: Bit Flip Mutation

# 3 Input

See sample input file. You can go to this address, https://toughsat.appspot.com, and generate as much $k$-SAT problems as you want.

# 4 Experiment and Results

Generate a table as shown for each of the input given for different $N$ for both $k$-sat and exactly-$m$ $k$-sat.

Table 1: Experiment

| $k$ | $v$ | $c$ | $N$ | No. of generations | | Best-Fitness | |
|---|---|---|---|---|---|---|---|
| | | | | $k$-sat | exactly-$m$ $k$-sat | $k$-sat | exactly-$m$ $k$-sat |
| | | | | | | | |

# 5 Conclusion

For any query or confusion please feel free to email me at ahmaadsabbir@gmail.com.

Prepared by-
Sabbir Ahmad
Lecturer
Department of CSE, BUET
ahmadsabbir@cse.buet.ac.bd
ahmaadsabbir@gmail.com
http://teacher.buet.ac.bd/ahmadsabbir