Journal of Computer Sciences and Applications, 2015, Vol. 3, No. 2, 33-39
Available online at http://pubs.sciepub.com/jcsa/3/2/3
© Science and Education Publishing
DOI:10.12691/jcsa-3-2-3

Genetic Algorithm Based Solution to SAT-3 Problem

Umme Aiman[*], Nausheen Asrar

Department of Computer Science, Jamia Hamdard, New Delhi, India
*Corresponding author: aimanraza16@gmail.com

Received March 18, 2015; Revised April 03, 2015; Accepted April 16, 2015

Abstract SAT-3 is an NP-complete problem for determining whether there exists a solution satisfying a given Boolean formula in the Conjunctive Normal Form, wherein each clause has at most three literals. Existing approaches of this problem take exponential time and are also memory inefficient. The work uses Genetic Algorithms for finding an optimal solution to this problem. The central idea is the intelligent exploitation of a random search used to solve optimization problems. The work explores previous works to direct the search into regions of better performance within the search space, thus reducing the time and space complexity. It thus establishes the ability of Genetic Algorithms for finding optimal solutions from a huge set of solutions. The work has been implemented and analyzed with satisfactory results.

Keywords: *NP complete problem, genetic algorithm, SAT-3 problem, intraceability, optimal solution*

Cite This Article: Umme Aiman, and Nausheen Asrar, "Genetic Algorithm Based Solution to SAT-3 Problem." *Journal of Computer Sciences and Applications*, vol. 3, no. 2 (2015): 33-39. doi: 10.12691/jcsa-3-2-3.

1. Introduction

A boolean expression is made up of clauses. Each clause has literals. If a clause has at most three literals connected by disjunction, and the clauses in turn are connected through conjunctions, the expression is that of SAT3. The problem is a satisfiability problem. The problem calls for finding the values of the literals that make the given expression true. Consider an expression where in there are n literals. Since a literal can have two values, there can be 2n possible set of values of the given set of literals, at each set of values; the expression needs to be evaluated. The solution, by the above method (brute force), therefore becomes computationally very expensive.

There is another way of handling the problem. The use of heuristic techniques like Genetic Algorithms would make the search of the solutions easy and hence tackle the given problem efficiently. GAs are heuristic processes based on the concept of survival of the fittest [1]. The operators like crossover, mutation etc help one to achieve the task. Many problems have been successfully solved using GAs. The present work uses GAs to solve the SAT3 Problem.

The paper has been organized as follows. Section 2 presents the literature review, section 3 presents Genetic Algorithm, section 4 presents NP Complete problems. In section 5 we discuss existing approaches to SAT-3 problem; section 5.1 describes, section 5.2 states, section 5.3 presents, section 5.4 presents, section 5.5 describes. In section 6 we present the proposed solution, section 7 presents result, section 8 presents the conclusion and section 9 presents references.

2. Literature Review

A systematic review was undertaken to summarize the existing approaches and their restrictions, find gaps in current research and propose areas for further investigation. The review has been carried out in accordance with the guidelines proposed by Kitchenham [2]. Table 1 presents the results of the review.

3. Genetic Algorithm

GAs works on the principle of survival of the fittest. [1] Each generation is represented by a series of character strings similar to the chromosome of DNA. [15] Each individual is a point in the search space. [16] They are then made to go through a series of evolutionary processes. The individuals compete for resources. Genes from successful individuals propagate throughout the population, making successive generations more suited to their environment.

After an initial population is randomly generated, the algorithm evolves through three operators:

Selection- equates to the survival of the fittest;
Crossover- represents mating between individuals;
Mutation- introduces random modifications.
The main steps of GA are explained below.

Randomly generate an initial population. Each chromosome has n cells. The population is generated by the following algorithm:

```
for each chromosome
for each cell
If (random () % 2 == 0)
Cell=1
```

```
else
Cell=0
end if
end
end
```

The chromosomes generated are a combination of 0s and 1s. A chromosome is a representation of the values assigned to the variables.

Find out the truth value of each chromosome. The truth value is whether the chromosome results in a "TRUE" or a "FALSE" value for the given formula.

CROSSOVER: Randomly select two chromosomes and break them from a randomly selected crossover point. A third child chromosome is formed by joining the first part of the first chromosome and the second part of the second chromosome. Similarly a fourth child chromosome is formed by joining the second part of the first chromosome and the first part of the second chromosome. Now the truth value for these two newly formed chromosomes are found out. Crossover rate is between 2 and 5.

MUTATION: A random chromosome is selected and one of its random bits is complemented and the truth value is calculated. The number of mutations is given by the formula.

No. of mutation = (No. of cells in a chromosome * No. of chromosomes * mutation rate) / 100

The mutation rate is less than 1 and was taken as 0.5 in our implementation. The number of TRUE value is counted. Figure 1 shows the basic steps in GA.

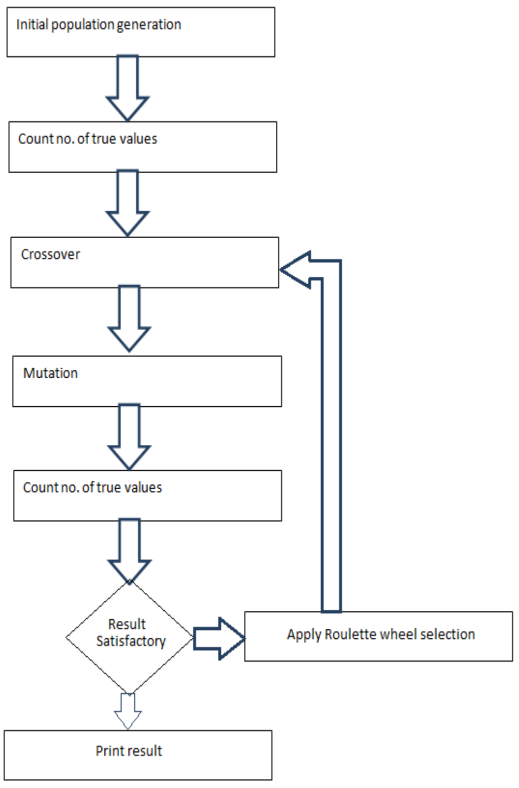

Figure 1. Basic steps in GA

4. NP Complete Problem

There are basically two types of problems

- Polynomial-time problems.
- NP problems.

Polynomial-Time problems-Problems that have algorithms with worst case running time of O(nk), where k is a constant, are called tractable others are called intractable or super-polynomial [17].

NP problems- Another class of algorithms that are "verifiable" in polynomial time is called NP problems. Any problem in P is also in NP [18].

We say that a language M, defining some decision problem, is NP-hard if every other language L in NP is polynomial-time reducible to M. If a language M is NP –hard and it is also in the class NP itself, then M is NP-complete [19].

Thus, an NP –complete problem is, in a very formal sense, one of the hardest problems in NP, as far as polynomial-time computability is concerned. If anyone finds a deterministic polynomial-time algorithm for even one NP-complete problem, then P=NP. [19] Figure 2 shows the relation of complexity classes.

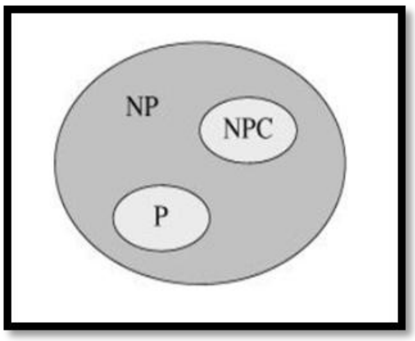

Figure 2. Relation of Complexity Classes

Definition: A Boolean formula is in 3SAT if it is in 3CNF form and is also satisfiable. A Boolean formula is in 3CNF if it is of the form

$C1 \wedge C2 \wedge \cdots \wedge C3$

Where, each Ci is an "AND" of three or less literals. [20].

5. Existing Approaches To SAT3 Problem

5.1. Brute Force Algorithm

The simplest but also the most time consuming idea. The main approach is to try every possible valuation until a satisfying one is found or there are no more valuations left. It is guaranteed to find a solution but has a slow run-time performance [21].

Implementation of this algorithm involves taking a number of variables and the Boolean function as input from the user. Then a 2D array is used as the data structure to store all the possible 2n combinations of values of all the variables. Then each row of the 2D array is used to insert values of variables in the given Boolean function to check its satisfiability, thus in a worst case scenario, all the possible combinations are tried until the solution is obtained.

Complexity-This approach works well for small inputs i.e. Small no of variables and simple functions. But since it is exponential in nature, its run-time complexity is O (2n) in the worst case. Thus, when the number of variables becomes large the algorithm fails to find optimal solution within practical time limit.

Table 1. Literature Review

S. No	Problem	Proposed solution	Conclusion
1.	N-puzzle problem – It involves having a set dimension of puzzle space (usually 3x3 for an 8-puzzle) and denoting the dimensions as N being the columns and M being the rows. In the puzzle space, there is a random arrangement of cells/blocks, with one empty space that enables adjacent cells/block to slide into them. Since the pieces are square blocks, only the top, bottom, left and right blocks adjacent to the empty space may slide into its place. The cells can either be numbered or printed with a fragment of the whole picture that the rearranged puzzle should show. [3]	GA is used. Index determination and threshold determination are done during the process.	• GA based approach results in better time complexity than the existing ones. • The complexity of the GA based algorithm is-O (log22m-1) ~O (m-1) [4].
2.	Vertex cover problem- Vertex cover of a graph G is a set of vertices such that each edge of G is incident to at least one vertex in the set. The resultant set is said to cover the edges of G. A minimum vertex cover is a vertex cover of smallest promising size. The vertex cover number is the size of a minimum vertex cover [5].	Standard GAs was used. A chromosome of genetic population can serve as a path from the initial state to goal state.	• GA based technique generates a very near solution to the problem. • It gives an optimal solution in lesser time and lesser moves as compared to approximation algorithms. • GA's are known to be better than randomized algorithm as they are more robust and has complexity very less than the existing approaches. [6]
3.	Maximum clique problem- Maximum clique problem finds the fully connected sub graphs from a given graph G= (V, E) in an arbitrary connected graph where V= {1, 2,….n} is set of vertex and E represents edges(x, y) set of G. The maximum clique problem is to find the maximum clique in a graph[7].	GA is used.	•When the no. of vertices was small, the solutions obtained were accurate. •For a large number of vertices, optimized solutions were obtained by applying the process of moderation. •Overall the resultant algorithm has very less complexity than other existing solutions. [8]
4.	Subset sum problem- In the SSP, a set W of n integers and a large integer C are given. We are interested in finding a subset S (of W) whose elements sum is closest to, without exceeding, C.[9]	The GA is applied. The steps involved are:- • Sorting • Calculate limit point • Generate Population • Mapping • Calculating the sum • Reducing the population • Cross over •Mutation • Moderation [10]	• For smaller values of sum, the GA based solution produces accurate results of approx. 60% of total sample space. •In other cases, process of moderation and Roulette Wheel selection were employed to enhance obtained solutions. [10]
5.	Post correspondence problem- It is defined as a finite set of pairs of strings (gi, hi) (1<=i<=s) over alphabet ∑. A solution to this instance is a sequence of selection i1i2……in (N>=1) such that the strings gi1 gi2…..gin and hi1, hi2…….hin formed by concatenation are identical. The no. of pairs in a PCP instance, s in the above is called its size[11].	Standard GA is used.	•The Application of Genetics to randomized algorithm approach to this problem has produced optimal solutions for small inputs. •Only, in a small number of cases, the algorithm failed to produce satisfactory results. [12]
6.	Travelling salesman problem- given a, collection of cities, the problem is to determine the shortest route which visits each city precisely once and then returns to its starting point.[13]	Improved version of GAs is applied	The algorithm • Successfully solved problems up to 229 cities. • Produced optimal results for problem up to 442 in an acceptable time limit. • Problems with higher no. of cities [maximum 666 cities] could be approximately solved by constructing a tour with length 0.04% over the optimum.[14]

5.2. Davis Putnum Algorithm

The algorithm works on a propositional formula ф. ф should be in CNF form. The algorithm treats ф as a set of clauses that can be manipulated. It defines 3 basic principles for obtaining the answer.

5.2.1 Unit Propagation Rule

Rule I deals with simplifying the formula for determining satisfiability by analyzing one-literal clauses. There are three cases to be distinguished.

Case 1: an atomic formula p occurs as a one-literal clause in both positive and negated form. In this case the resulting formula ф is unsatisfiable, because the empty clause can be deduced by means of resolution.

Case 2: Case 1 does not apply and an atomic formula p occurs as a one literal clause. Then all clauses that contain p can be deleted and -p can be removed from the remaining clauses.

Case 3: Case 1 does not apply and an atomic formula -p occurs as a one literal clause. Then all clauses that contain -p can be deleted and p can be removed from the remaining clauses.

5.2.2. Pure literal Rule

It deals with redundant clauses. It is generally used as a preprocessing step for efficiency reasons. If only p or only -p occurs in the formula, then all clauses which contain p or -p may be deleted.

5.2.3. Resolution Rule

Resolution can be applied to two clauses that have complementary literals L and L' such that L ≡ -L'. With the help of the resolution step these two clauses can be combined in a single clause called the resolvent that contains all the literals without L and L' [21].

Complexity-The fact that this is an exponential algorithm follows directly from the fact that regular resolution is exponential. This means that this algorithm will generate an exponential number of clauses and thus is very memory inefficient. The lower bound complexity of this algorithm has been proven to be 2cn for some constant c > 0 [21]. Figure 3 depicts the basic steps involved in Davis Putnum Algorithm.

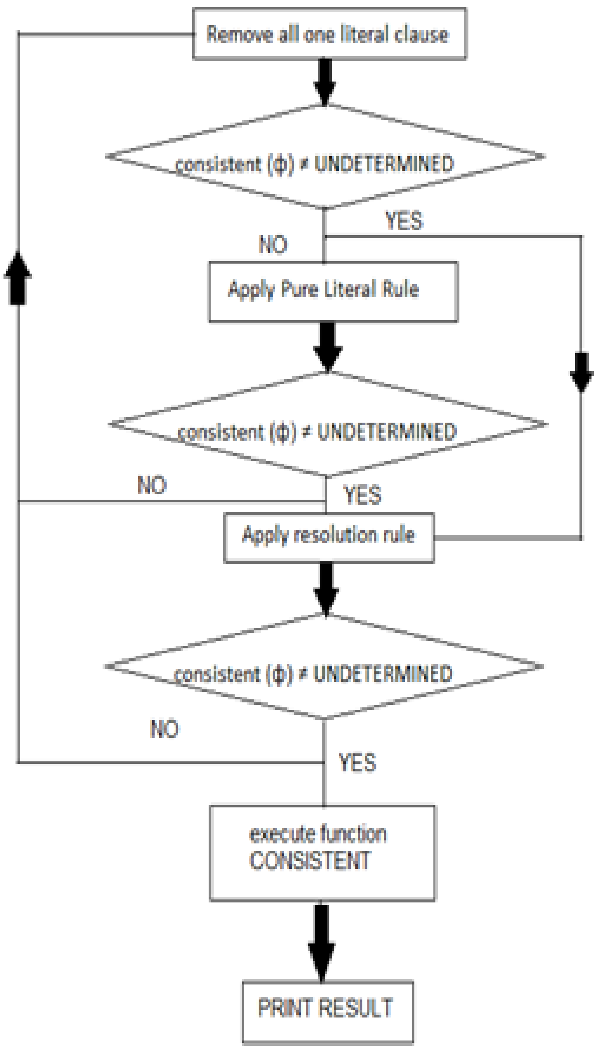
Remove all one literal clause
consistent (ϕ) ≠ UNDETERMINED
YES
NO
Apply Pure Literal Rule
consistent (ϕ) ≠ UNDETERMINED
NO
YES
Apply resolution rule
consistent (ϕ) ≠ UNDETERMINED
NO
YES
execute function CONSISTENT
PRINT RESULT

Figure 3. Basic steps in Davis-Putnum algorithm

5.3. Davis Logemann and Loveland Algorithm

In this algorithm rule III of Davis Putnum Algorithm was replaced by splitting rule in order to limit the amount of memory used during runtime.

- The simplifying of the formula is now usually done as a preprocessing step for efficiency reasons.
- Heuristics for the branching variable are used
- Backtracking has been replaced by back jumping or non-chronological backtracking in iterative versions of the DLL, so that similar parts of the same tree are not searched over and over again. [21]
- New clauses can be ascertained during the search process to increase the amount of branches that can be cut away from the search tree later.
- Improvements in Boolean Constraint Propagation algorithms are introduced.
- Random restarts of the algorithm that make use of previously learned information increase the chances of coming to a solution faster
- For these improvements to be efficient and also to allow non-chronological backtracking the DLL algorithm is usually implemented iteratively [21].

Complexity- In general, DPLL requires exponential time (splitting rule!)! Heuristics are needed to determine which variable should be instantiated next and which value should be used.

DPLL is polynomial on Horn clauses, i.e., Clauses with at most one positive literal -A1,V…V-An V B [22].

5.4. Binary Decision Diagrams (BDD)

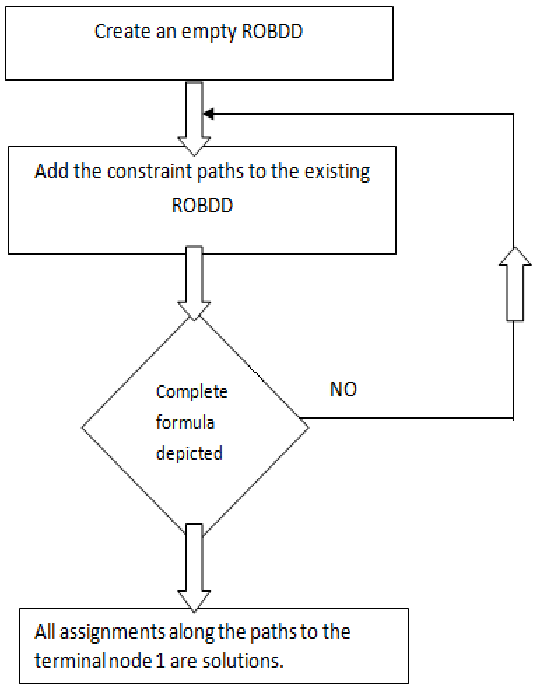
Create an empty ROBDD
Add the constraint paths to the existing ROBDD
Complete formula depicted
NO
All assignments along the paths to the terminal node 1 are solutions.

Figure 4. Basic steps in Binary Decision Diagrams

A BDD represents a formula ϕ (seen as a Boolean function) as a rooted directed acyclic graph G = (V, T, E). Each non-terminal node or vertex v(i) ϵ V is labeled by a variable x and has edges directed towards two successor nodes, called the then-child and the else-child. In a graphical representation, the then-child represents x = 1 via an uninterrupted line and the else-child represents x = 0 via a dotted line. The lines are depicted as undirected for simplicity, but must always be considered as directed to the next node. Each terminal node t(i) ϵ T is labeled with 0 or 1 and does not have any children. For a given assignment of the variables, the value of the function is found by tracing a path from the root to a terminal vertex following the branches indicated by the values assigned to the variables [21].

Using reduced order Binary Decision Diagram a set of satisfying assignments is created by looking at the constraints.

The process is to continuously add clauses to an empty BDD until the perfect pattern is described. By adding all the constraint paths to the BDD we get the complete representation of the function for which the satisfiability is being checked. This method solves the problem by building up the solution set in order to verify satisfiability.

Complexity- ROBDDs perform worse, both in terms of space and time complexity in under constrained and critically constrained set of problems. It only works best in over constrained sets of problems [21]. Figure 4 shows the steps involved in Binary Decision Diagram approach.

5.5. Set Theory

Set theory based approach decides the un-satisfiability of a given formula. It describes the problem in set theory and investigates what is needed to decide the un-satisfiability of a given sat3 formula.

For a SAT3 function ϕ of n variables the set V_φ of possible valuations (variable assignments) v is determined, for which v(ϕ) evaluates to 1. Here v is a mapping {x0::: xn-1} → {0,1} that instantiates every variable to 0 or 1. So each function vi can be depicted as a unique number in binary representation, which is in the set 2n that denotes n-tuples over a binary alphabet {0, 1} n. If any clause that is part of ϕ is unsatisfiable, then ϕ is also unsatisfiable. That is, v(ci) = 0 => v(ϕ) = 0. Using this information, any 3-SAT formula can be proven (un) satisfiable using sets [21].

Complexity-This solution is not computationally possible with a high amount of clauses and takes more time than applying brute-force approach to all 2n possible valuations [21].

6. Proposed Solution

This work proposes Gas based solution to SAT-3 problem. The algorithm has been presented as follows. The algorithm uses the blend of the theory of natural selection and conventional optimization.

```
ALGORITHM: SAT-3 problem
1. cell->number of variables
2. expression->given Boolean formula
3. chromosome-> representation of values assigned to
   the variables of given Boolean formula
4. expression_evaluate->truth value of expression
5. //initial population generation
6. for each chromosome //no. of iterations/initial
   population ??
7. for each cell
8. If (random () % 100> 50)
9. cell=1;
10. else
11. cell=0;
12. end if
13. end for
14. end for
15. //count number of true values
16. for each chromosome
17. If (expression_evaluate == TRUE)
18. count++;
19. end if
20. end for
21. //crossover
22. for each chromosome-> till crossover_rate
23. chromo some 1=random () % population;
24. chromo some 2=random () % population;
25. crossover_point=random () % cell;
26. for each cell-> beginning to crossover_point
27. chromo some 3[i] = chromo some 1[i];
28. chromo some 4[i] = chromo some 2[i];
29. end for
30. for each cell->crossover_point to end
31. chromo some 3[i] = chromo some 2[i];
32. chromo some 4[i] = chromo some 1[i];
33. end for
34. end for
35. for each chromosome
36. If (expression_evaluate==TRUE)
37. count++;
38. end if
39. end for
40. //mutation
41. for each chromosome->till mutation_rate
42. mutation_point=random () % cell;
43. If (cell[mutation_point]==1)
44. cell[mutation_point]=0;
45. else
46. cell[mutation_point]=1;
47. end if
48. end for
49. for each chromosome
50. If (expression_evaluate==TRUE)
51. count++;
52. End if
53. End for
```

7. Results

The following tables (Table 2, Table 3, Table 4, Table 5) show the obtained results. Due to time constraints we have considered the case of 4 variables and 7, 8, 9 and 10 clauses. Further analysis with more number of variables and clauses will be considered for future work.

In the following tables the first column shows the no. of true values generated randomly when initial population is formed.

The second column shows the Cross-over Rate. The third column shows the number of true values generated w.r.t the mentioned crossover rate. The fourth column shows the Mutation Rate. Finally, The last column shows the no. of true values generated w.r.t the mentioned mutation rate.

Table 2 shows the results for 4 variables and 7 clauses, Table 3 shows the results for 4 variables and 8 clauses, Table 3 shows the results for 4 vriables and 9 clauses, Table 3 shows the results for 4 variables and 10 clauses.

Table 2. 4 Variables, 7 Clauses (#{A}||#{B}||#{C})&&(#{D}||!#{A}||#{C})&&(!#{B})&&(!#{C}||!#{A}) &&(#{A}||!#{B})&&(!#{A}||!#{B}||!#{C})&&(!#{D}||#{C}||#{A})

True Values (Random)	Crossover rate	True Values (crossover)	Mutation rate	True values (mutation)
0	2	0	0.1	1
1	2	1	0.3	1
2	2	2	0.6	0
1	2	1	0.8	0
1	3	1	0.1	1
2	3	2	0.3	2
1	3	1	0.6	1
2	3	2	0.8	1
1	4	1	0.1	0
1	4	1	0.3	1
1	4	1	0.6	1
2	4	2	0.8	2

Table 2. 4 Variables, 8 Clauses (#{A}||#{B}||#{C})&&(#{B}||#{C}||#{D})&&(#{C}||#{D}||#{B})&&(#{ A}||#{B}||#{D})&&(#{B}||#{D}||#{A})&&(#{A}||#{B}||!#{C})&&(!#{A} ||#{C}||#{D})&&(#{C}||!#{D})

True values (random)	Crossover rate	True values (crossover)	Mutation rate	True values (mutation)
2	2	2	0.1	2
2	2	2	0.2	3
4	2	4	0.3	5
3	2	2	0.4	4
2	3	2	0.1	2
2	3	2	0.2	2
3	3	3	0.3	3
1	3	1	0.4	3
3	4	3	0.1	4
3	4	3	0.2	3
2	4	2	0.3	2
2	4	2	0.4	4

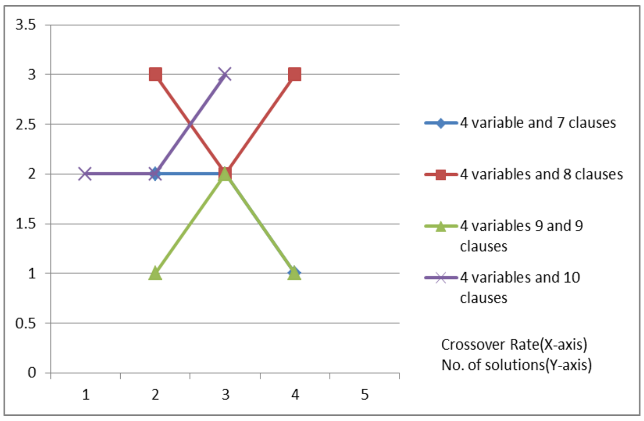

Figure 5. Graph showing variation in no. of solution with different Crossover Rates

Table 3. 4 Variables, 9 Clauses (#{A}||#{B}||#{C})&&(#{D}||!#{A}||#{C})&&(!#{B})&&(!#{C}||!#{A}) &&(#{A}||!#{B})&&(!#{A}||!#{B}||!#{C})&&(!#{D}||#{C}||#{A})&&(! #{A}||#{B}||#{C})&&(!#{A}||#{B}||#{D})&&(!#{A}||!#{B}||!#{D})

True values (random)	Crossover rate	True values (crossover)	Mutation rate	True values (mutation)
1	2	1	0.1	2
2	2	2	0.3	3
2	2	2	0.6	2
2	2	2	0.8	0
1	3	1	0.1	1
3	3	3	0.3	3
1	3	1	0.6	1
2	3	2	0.8	2
2	4	2	0.1	1
1	4	1	0.3	1
0	4	0	0.6	1
0	4	0	0.8	1

Table 4. 4 Variables, 10 Clauses (#{A}||#{B}||#{C})&&(#{D}||!#{A}||#{C})&&(!#{B})&&(!#{C}||!#{A}) &&(#{A}||!#{B})&&(!#{A}||!#{B}||!#{C})&&(!#{D}||#{C}\||#{A})&&(!#{A}||#{B}||#{C})&&(!#{A}||#{B}||#{D})&&(!#{A}||!#{B}||!#{D})

True Values (Random)	Crossover rate	True Values (crossover)	Mutation rate	True values (mutation)
3	2	3	0.1	3
1	2	1	0.2	1
1	2	1	0.3	3
4	2	3	0.4	4
3	3	2	0.1	3
2	3	2	0.2	2
2	3	2	0.3	3
2	3	2	0.4	2
2	4	2	0.1	2
2	4	2	0.2	4
3	4	3	0.3	2
3	4	3	0.4	4

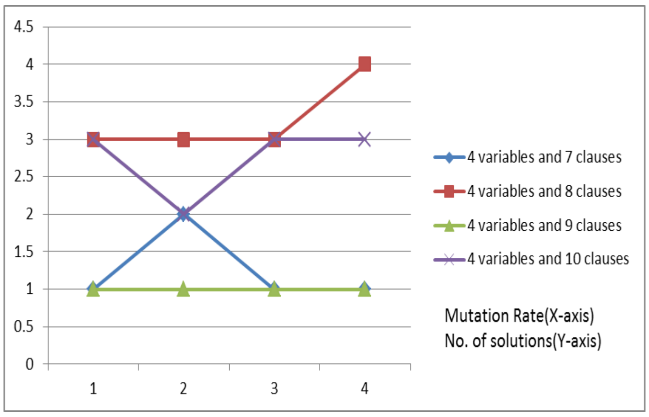

Figure 6. Graph showing variation in no. of solution with different Mutation Rates

Figure 5 depicts the graph showing variation in no. of solution with different Crossover Rates. Figure 6 depicts the graph showing variation in no. of solution with different Mutation Rates.

8. Conclusion

The paper successfully implements Gas based algorithm for SAT-3 problem. The algorithm has been analyzed for various crossover rates and mutation rates as explained in the previous section. It can be seen from the graph presented in the previous section. It can be seen from the graph presented in the previous section that the results are optimal for COR=3 and MR=0.3.

The extension to this work would analyze the algorithm for a larger domain to establish the veracity of the algorithm. It is also intended to use other artificial intelligence based techniques and compare the results with the above techniques.

References

[1] David E. Goldberg, “Genetic Algorithm”, Pearson Education.

[2] Barbara Kitchenham, (July 2003), “Procedures for Performing Systematic Reviews”, Keele University Technical Report TR/SE-0401 ISSN: 1353-7776 and NICTA Technical Report 0400011T.1

[3] Drogoul, A., and Dubreuil, C. "A distributed approach to n-puzzle solving." Proceedings of the Distributed Artificial Intelligence Workshop. 1993.

[4] Harsh Bhasin and NehaSingla, (August 2012),” Genetic based Algorithm for N-Puzzle Problem”, International Journal of Computer Applications (0975-8887), Volume 51-No. 22.

[5] Eric Angel, Romain Campigotto and Christian Laforest, Algorithms for the Vertex Cover Problem on Large Graphs.

[6] Harsh Bhasin and Gitanjali, (2012), "Harnessing Genetic Algorithm for Vertex Cover Problem", International Journal on Computer Science and Engineering (IJCSE), Vol. 1, Issue 2, pp. 218-223.

[7] Naresh Kumar, Deepkiran Munjal, Modified Genetic Algorithm for Maximum Clique Problem, International Journal of Computer & Organization Trends-Volume 3 Issue 4-May 2013.

[8] Harsh Bhasinand Rohan Mahajan, (2013)," Genetic Algorithms Based Solution To Maximum Clique Problem"‖, International Journal of Computer Science and Engineering, Vol. 4.

[9] Rong Long Wang, A genetic algorithm for subset sum problem, New Aspects in Neurocomputing: 10th European Symposium on Artificial Neural Networks 2002.

[10] Harsh Bhasin and NehaSingla, (2012)," Modified Genetic Algorithms Based Solution to Subset Sum Problem", (IJARAI) International Journal of Advanced Research in Artificial Intelligence, Vol. 1, No. 1.

[11] Ling Zhao, Notes in Computer Science Volume 2883, 2003, pp 326-344, Tackling Post's Correspondence Problem.

[12] Harsh Bhasin and Nishant Gupta, (2012), "Randomized algorithm approach for solving PCP", International Journal on Computer Science and Engineering (IJCSE),Vol. 4, Issue 1, pp. 106-113.

[13] Genetic algorithms for the travelling salesman problem: A review of representations and operators-P Larrañaga, CMH Kuijpers, RH Murga, I Inza…-Artificial Intelligence …, 1999-Springer.

[14] H Braun, On solving travelling salesman problems by genetic algorithms,-Parallel Problem Solving from Nature, 1991-Springer.

[15] Lintao Zhang, Searching For Truth: Techniques For Satisfiability of Boolean Formulas, PhD Thesis, Princeton University, 2003.

[16] Yogesh S. Mahajan, Zhaochui Fu and Sharad Malik, Zchaff 2004: An Efficient SAT Solver, Lecture Notes in Computer Science, 2005, Volume 3542/2005, 898.

[17] www.personal.kent.edu/.../Algorithms/MyAlgorithms/.../npComplete.html

[18] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, "Introduction to Algorithms", the MIT Press, Cambridge, Massachusetts, USA, 2009.

[19] ww3.algorithmdesign.net/sample/ch13-np.pdf

[20] www.cs.umd.edu/~gasarch/TOPICS/sat/SATtalk.pdf

[21] "Solving 3-SAT", (2012), Peter Maandag, Bachelor Thesis, Radboud University Nijmegen.

[22] "Foundations of Artificial Intelligence, Satisfiability and Model Construction", Wolfram Burgard, Bernhard Nebel and Martin Riedmiller.

[23] "Implementing the Davis-Putnam Method", (1999), Hantao Zhang (Dept. of Computer Science, The University of Iowa), Mark E. Stickel (Artificial Intelligence Center, U.S.A).

[24] www.cs.ucc.ie/~dgb/courses/ai1/03-notes.pdf

[25] Algorithms for Random 3-SAT, Abraham D. Flaxman, Microsoft Research

[26] A. Mishchenko, An Introduction to Zero-Suppressed Binary Decision Diagrams. http://www.ee.pdx.edu/~alanmi/research/, 2001.

[27] J. M. Howe and A. King, A Pearl on SAT Solving in Prolog, In Functional and Logic Programming, volume 6009 of Lecture Notes in Computer Science, pages 165-174, Springer, 2010

[28] Norbert Manthey, Improving SAT Solvers Using State-of-the-Art Techniques, Thesis, Technische Universitt Dresden (2010).

[29] Wikipedia-online encyclopedia.

[30] http://www.stumptown.com/diss/chapter2.html