CS 314, Lab 3 - Report

Shriram Ghadge (180010015), Rishit Saiya (180010027)

February 6, 2021

1 Abstract

In this task, Heuristic Search Algorithms had to be implemented. Uniform Random-4-SAT is a family of SAT problems distributions obtained by randomly generating 3-CNF formulae in the following way: For an instance with `n = 4` variables and `k = 5` clauses, each of the `k` clauses is constructed from 3 literals which are randomly drawn from the `2n` possible literals (the n variables and their negations) such that each possible literal is selected with the same probability of $\frac{1}{2n}$. Clauses are not accepted for the construction of the problem instance if they contain multiple copies of the same literal or if they are tautological (i.e., they contain a variable and its negation as a literal). Each choice of n and k thus induces a distribution of Random-4-SAT instances. Uniform Random-4-SAT is the union of these distributions over all n and k.

2 Literature

2.1 State Space

Every state is defined using class having literal values. Each variable is either `0` or `1`. In case of Tabu Search, the state representation is modified a bit and one more list to store Tabu Tenure values is added to state as below.

2.1.1 VND & Beam Search

In the above mentioned algorithms, the state used are as follows:

$$State = ([a, b, c, d])$$

2.1.2 Tabu Search

For Tabu Search, the state used are as follows:

$$State = ([a, b, c, d], [M1, M2, M3, M4])$$

For Example: Assuming that all literal values are 0 and it is allowed to change is represented as ([0,0,0,0]) or maybe ([1,1,1,1]) for VND & Beam Search Algorithms. Whilst in Tabu Search such a state maybe ([0,0,0,0], [0,0,0,0]).

2.2 Start Node & Goal Node

For the aforementioned algorithms, the start and Goal Nodes will be calculated in the same logic as before. We are starting with all literal values 0 and it is allowed to change any literal in the next move.

2.2.1 Start Nodes for VND & Beam Search

In the above mentioned algorithms, the start node used are as follows:

$$State = ([0, 0, 0, 0])$$

2.2.2 Start Nodes for Tabu Search

For Tabu Search, the start node used are as follows:

$$State = ([0, 0, 0, 0], [0, 0, 0, 0])$$

2.2.3 Goal Node

Goal node is a node with a literal list [a,b,c,d] such that boolean values of a,b,c,d when put into a given 4-SAT expression returns 1 i.e it satisfies the 4-SAT CNF expression.

3 Pseudo Codes

In the following subsections, pseudo codes for important functions in the code are explained.

3.1 movGen(state)

The function takes a state as input and returns a set of states that are reachable from the input state in one step. [Algorithm 1]

3.2 GoalTest(state)

This function returns True if the input state is goal and False otherwise. [Algorithm 2]

3.3 variableNeighborhoodDescent(x, k_{max})

The variable neighborhood descent (VND) method is obtained if a change of neighborhoods is performed in a deterministic way. It is presented in Algorithm 3, where neighborhoods are denoted as N_k, $k = 1, \ldots, k_{max}$.

Algorithm 1 movGen(state)

```
1: procedure MOVEGEN(state)
2:     nextStates ← ()                          ▷ initialize nextStates to empty set
3:     for neighbour n of state in order(HeuristicValues) do
4:         bit = negation(bit)                                          ▷ a := ã
5:         neighbours ← new.node()
6:     return nieghbours
```

Algorithm 2 goalTest(state)

```
1: procedure GOALTEST(state)
2:     if [a, bc, d] satisfies CNF then
3:         return true
4:     return false                                          ▷ state is not goal
```

3.4 Beam Search

Beam search is a heuristic search method. It is given in as Algorithm 4. *containsGoal*() is a function that determines whether the goal state has been reached. *score*() function uses a heuristic function to score states. *prune*() function selects the best states to keep.

3.5 Tabu Search

We used Tabu Search Algorithm as shown in Figure 1.

4 Heuristic Function

Heuristic Function returns an integer equal to the total number of clauses satisfied in the formula. It follows the below logic:

```
if(clauseSatisfied()){
    value++;
}
else
    continue;
return value;
```

5 Beam Search Analysis for different Beam Lengths

In the following cases, where the number of states explored are compared across algorithms, the values are observed as in Table 1. The Beam Width was varied from 1 to 4 for 3 different clauses and corresponding initial states.

Algorithm 3 VND(x, k_{max})

```
1: procedure VND(x, k_max)
2:     k ← 1
3:     while k = k_max do
4:         x' ← arg min_{y∈N_k(x)} f(y)        ▷ Find the best neighbor in N_k(x)
5:         x, k ← NeighborhoodChange(x, x', k)   ▷ Change neighborhood
6:     return x
```

Algorithm 4 beamSearch(state)

```
1: procedure BEAMSEARCH(state)
2:     initialStates ← currentStates
3:     while not containsGoal(currentStates) do
4:         next(currentStates) =: candidatesStates
5:         score(candidatesStates)
6:         currentStates := prune(candidatesStates)
7:     return state
```

6 Tabu Search for different values of Tabu Tenure

In the following cases, where the number of states explored are compared across algorithms, the values are observed as in Table 2. The Tabu Tenure was varied from 1 to 4 for 3 different clauses and corresponding initial states.

7 Comparison of Variable Neighborhood Descent, Beam Search, Tabu Search: Nodes explored by each

In the following cases, where the number of states explored are compared across algorithms, the values are observed as in Table 3. The following assumptions were made:

```
Beam Width = 2
Tabu Tenure = 2
```

8 Conclusion

With reference of above mentioned data, we can say that, for such k-SAT problem need to be compared across different parameters like Tabu Tenure and Beam Width depending across algorithms. Also, number of states explored to reach Goal Node is compared across different algorithms namely Beam Search, Tabu Search & VND and clear optimal Tabu Tenure is 1 and Optimal Beam Width is 1 for 4-SAT type problems.

```
1.  Tabu Search {
2.  max_frequency ←25n, frequency← 0
3.  current_solution = generate randomized solution;
4.  best_solution = current_solution;
5.  while (frequency <= max_frequency n) {
6.  accept_transaction = 0;
7.  while (accept_transaction = 0) {
8.  generate transaction;
9.  if (move  ∉ tabu_list) {
10. accept_ transaction = 1;
11. update tabu_list;
12. update current_solution;
13. else
14. accept_ transaction = 0;
15. }
16. }
17. if (current_solution > best_solution){
18. best_solution ←current_solution;
19. }
20. frequency← frequency+1;
21. }
22. report best_solution;
```

Figure 1: Tabu Search Pseudo Code

Clause **Initial state**	**Beam Width**	**States Explored**
'A', 'B', 'd'], ['B', 'C', 'D'], ['B', 'a', 'c'], ['D', 'b', 'c'], ['a', 'b', 'c']] Initial state : {a': 1, 'b': 0, 'c': 1, 'd': 1, 'A': 0, 'B': 1, 'C': 0, 'D': 0}	1	1
	2	1
	3	1
	4	1
'A', 'B', 'C'], ['B', 'C', 'c'], ['B', 'a', 'c'], ['B', 'c', 'd'], ['b', 'c', 'd']] Initial state : {a': 1, 'b': 1, 'c': 1, 'd': 1, 'A': 0, 'B': 0, 'C': 0, 'D': 0}	1	5
	2	5
	3	5
	4	5
'A', 'B', 'c'], ['A', 'c', 'd'], ['B', 'C', 'd'], ['B', 'a', 'c'], ['a', 'c', 'd']] Initial state : {a': 1, 'b': 1, 'c': 0, 'd': 0, 'A': 0, 'B': 0, 'C': 1, 'D': 1}	1	9
	2	9
	3	9
	4	13

Table 1: Beam Search Analysis for different Beam Lengths

Clause **Initial state**	**Tabu Tenure**	**States Explored**
[['A', 'B', 'b'], ['A', 'B', 'c'], ['B', 'c', 'd'], ['C', 'D', 'a'], ['C', 'a', 'b']] Initial state : {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'A': 1, 'B': 1, 'C': 1, 'D': 1}	1	1
	2	1
	3	1
	4	1
[['A', 'D', 'b'], ['B', 'D', 'a'], ['B', 'c', 'd'], ['D', 'a', 'c'], ['a', 'b', 'd']] Initial state : {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'A': 1, 'B': 1, 'C': 1, 'D': 1}	1	5
	2	7
	3	3
	4	5
[['A', 'B', 'c'], ['A', 'c', 'd'], ['B', 'C', 'd'], ['B', 'a', 'c'], ['a', 'c', 'd']] Initial state : {'a': 0, 'b': 1, 'c': 0, 'd': 0, 'A': 1, 'B': 0, 'C': 1, 'D': 1}	1	8
	2	7
	3	5
	4	5

Table 2: Tabu Search for different values of Tabu Tenure

Clause **Initial state**	**Algorithms**	**States Explored**
[['A', 'B', 'D'], ['A', 'B', 'b'], ['A', 'c', 'd'], ['C', 'D', 'a'], ['D', 'a', 'b']] Initial state : {'a': 1, 'b': 0, 'c': 1, 'd': 1, 'A': 0, 'B': 1, 'C': 0, 'D': 0}	Beam Search	1
	Tabu Search	1
	VND	1
[['A', 'D', 'b'], ['B', 'C', 'd'], ['D', 'a', 'c'], ['D', 'b', 'c'], ['a', 'b', 'c']] Initial state : {'a': 0, 'b': 1, 'c': 0, 'd': 1, 'A': 1, 'B': 0, 'C': 1, 'D': 0}	Beam Search	5
	Tabu Search	5
	VND	5
[['A', 'B', 'c'], ['A', 'c', 'd'], ['B', 'C', 'd'], ['B', 'a', 'c'], ['a', 'c', 'd']] Initial state : {'a': 1, 'b': 1, 'c': 0, 'd': 0, 'A': 0, 'B': 0, 'C': 1, 'D': 1}	Beam Search	9
	Tabu Search	8
	VND	8

Table 3: Comparison of Variable Neighborhood Descent, Beam Search, Tabu Search: Nodes explored by each