Assignment-3 Lab Report

R.Chidaksh (200010046), R.Sai Krupakar (200010047)

January 10, 2022

1 Problem Statement:

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 Description about the domain:

2.1 State space

State space is the set of all possible combinations of truth values. Here every state is represented using literals. Each literal can take either 0 or 1.

2.2 Start state and Goal state

2.2.1 Start state:

Start state is the initial state from which we move towards the goal state so as to make the all clauses true(1). There exists five clauses and each clause is the combination of the literals that are generated randomly.

2.2.2 Goal state:

Goal state is the state in which each and every clause equals to 1(True). Her the SAT equation holds True. Goal state is the values of all literals at which the SAT equation is true.

2.3 MOVEGen and GOALTEST:

2.3.1 MOVEGEN

MOVEGEN function takes the current node as the input. After taking the current node the neighbours of the node are generated by toggling one of the bits each time. If we are struck at any state in Variable Decent Search then we can toggle two bits at a time to generate the neighbours .

Pseudocode of MOVEGEN:

```
1: procedure moveGen(state)
2:     open_list ← ( )                                  .
3:     for neighbours n of state in order(Heuristic Values) do
4:         bit = negation(bit)
5:         neighbour ← new.node()
6:     return nieghbour
```

2.3.1 GOALTEST

GOALTEST function helps us to find whether the current state is goal state or not. The function returns True when the heuristic value of the current node and the goal node are same or it returns False. If the function returns True then the process terminates or else it continues.

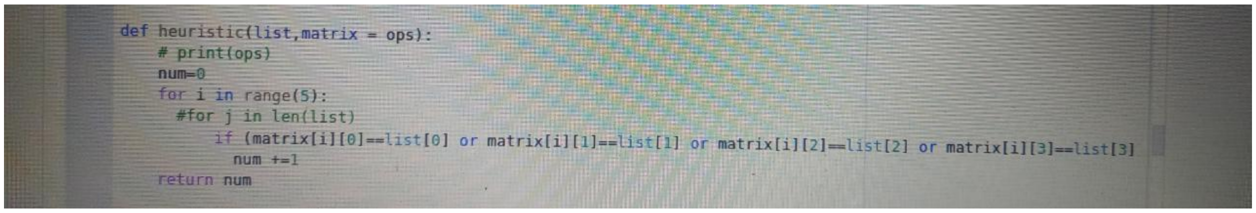
Pseudocode of GOALTEST:

Algorithm 2 goaltest(state)

```
1: procedure goalTest(state)
2:     if [a,bc,d] satisfies CNF then
3:         return true
4:     return false
```

3.Heuristic Function:

The heuristic function we used here returns an integer equal to the number of clauses that are true(1) for the current node. The heuristic value of the goal node is equal to the number of clauses.



```
def heuristic(list,matrix = ops):
    # print(ops)
    num=0
    for i in range(5):
      #for j in len(list)
          if (matrix[i][0]==list[0] or matrix[i][1]==list[1] or matrix[i][2]==list[2] or matrix[i][3]==list[3]
            num +=1
    return num
```

4. Beam Search Analysis for different Beam Lengths:

Beam search uses BFS .At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. Beam search only stores fixed number of best states at each level (called the beam width). Only those states are expanded next. The greater the beam width, the fewer states are pruned.

Clause Initial state	Beam Width	States Explored
[A,B,C,D]=[1,0,1,1] Goal State=[1,0,0,1]	1	2
	2	2
	3	2
	4	2
[A,B,C,D]=[1,0,0,1] Goal State=[1,0,0,1]	1	1
	2	1
	3	1
	4	1
[A,B,C,D]=[0,0,0,0] Goal State=[0,0,0,0]	1	1
	2	1
	3	1
	4	1

Table 1: Beam Search Analysis for different Beam Lengths

5. Tabu search for different values of tabu tenure:

In Tabu search the length is fixed initially. Once the heuristic keeps revolving around the local optimal value, our search algorithms fails. For this we can use an algorithm Tabu search where we let the program check states, though initially in other search algorithms it doesn't allow, Tabu search allows to go against the required heuristic to a certain limit. This maximizes the possibility of finding the goal state.

Clause Initial state	**Tabu Tenure**	**States Explored**
[A,B,C,D]=[1,0,1,1] Goal State=[1,0,0,1]	1	2
	2	2
	3	2
	4	2
[A,B,C,D]=[1,0,0,1] Goal State=[1,0,0,1]	1	1
	2	1
	3	1
	4	1
[A,B,C,D]=[0,0,0,0] Goal State=[0,0,0,0]	1	1
	2	1
	3	1
	4	1

Table 2: Tabu Search for different values of Tabu Tenure

6. Comparison of Variable neighbourhood descent, Beam Search, Tabu Search: Nodes explored by each:

Clause Initial state	Algorithms	States Explored
[A,B,C,D]=[1,0,1,1] Goal State=[1,0,0,1]	Beam Search	2
	Tabu Search	2
	VND	2
[A,B,C,D]=[1,0,0,1] Goal State=[1,0,0,1]	Beam Search	1
	Tabu Search	1
	VND	1
[A,B,C,D]=[0,0,0,0] Goal State=[0,0,0,0]	Beam Search	1
	Tabu Search	1
	VND	1

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

7. Results and Inferences

In VND search algorithm, we can see that, it performs better than the other algorithms in finding an optimal solution with less number of explored states though it takes more time than others. we can say that VND takes lesser number of states, but more time since it has to calculate for all the states, on repeatedly increasing the neighbour-density.

In Tabu search we can see that, It is seen that as the tabu tenure increases lesser states are visited and there is more chance of finding an optimal solution. Empirically, the optimal tabu tenure found is 4. For problems of higher dimensions, a greater tabu tenure might have to be used.

In Beam search we can see that, the problem does not entirely take the advantage of beam search because it is less likely that the two states have the same heuristic value. The optimal beam width is highly dependent on the input. A beam width of 2 found to be suited for most of the inputs comparing the optimality and number of explored states.

8.Conclusion

Here since the number of states explored is not large all the three algorithms are exploring same number of states in many cases. Comparing the results from the third table, the algorithms are similar in the number of states they explore. Tabu Search is faster than other algorithms when it comes to bigger inputs, however it does not guarantee an optimal solution. Beam search doesn’t always guarantee a solution. VND gives us a chance to check many other neighbours other than what we limit the algorithm to, by toggling more than just one bit.