CS 312: Artificial Intelligence Laboratory

Lab 3 report

— B Siddharth Prabhu (200010003)

— Sourabh Bhosale (200010004)

- **Introduction:**

The objective of this task is to implement Heuristic Search Algorithms**.** Uniform Random-4-SAT is a family of SAT problems distributions obtained by randomly generating 4-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 4 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 *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.*

- **Description:**

1. State space:

The state space is all possible permutations of truth values of all literals. Here, let us take an example having 5 clauses which need to be satisfied in order to reach to goal state. Each clause consists of 4 literals randomly selected from 4 variables and their negations. Each literal can only have value either 1 or 0 which may or may not satisfy all clauses. So, if a state is not goal state, then we may perturb 1 or even more number of literals in order to get neighbour states which could be possibly goal state.

Eg: [0,0,0,0] → perturb 1 literal / bit at a time to get

neighbour states→ ([1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]).

It is possible that one of these combinations of bits / literals satisfies one or more clauses of the boolean expression.

2. Start node and goal node:

Start node is [0,0,0,0] for this 5 clause 4-SAT problem. It is possible that this satisfies all the 5 clauses and this is a valid goal state. If not, perturb one or more bits at a time to get its neighbours for further proceedings.

Goal node is a node with a literal list [A, B, C, D] consisting of boolean values A, B, C, D, which when put into a given 4-SAT expression returns 1 , i.e., it satisfies the 4-SAT CNF expression.

- **Pseudocode:**

a) GoalTest :

The global optimum of the heuristic occurs when all of the clauses are satisfied. So, if the current state's heuristic is equal to the number of clauses, then we have reached the goal.

```
function GoalTest (current_state) :
        if all clauses are satisfied :
            return True
        else :
            return False
```

b) MoveGen :

This function populates the neighbour list of a given current node, according to the following pseudocode:

```
function MoveGen (current_state, perturb_num, tt = None):
    initialise empty list for neighbours
    if tt:
        generate all_combos using perturb_num
        for each combo in all_combos:
            make a copy of current state
            for bitindex in combo:
                if tt of bitindex is 0:
                    flip bit at bitindex
                    update tt of bitindex
                    update tt of other indices
                append copy to neighbours
        return neighbours
    else :
        generate all_combos using perturb_num
        for each combo in all_combos:
            make a copy of current state
            for bitindex in combo
                flip bit at bitindex
            append copy to neighbours
        return neighbours
```

- **Heuristic Function:**

  Heuristic Function returns an integer equal to the total number of clauses satisfied in the formula. We have used a dictionary to record the occurrence of literals and their values with respect to the current state, as key-value pairs.

  Example:

  If current state is [0011] then its corresponding dictionary will be

  { ‘A’ : 0, ‘B' : 0, ‘C’ : 1, ‘D’ : 1,

  ‘~A’ : 1, ‘~B’ : 1, ‘~C’ : 0, ‘~D’ : 0 }

- **Pseudocode**:

```
for clause in formula :
        if clause is satisfied:
                heuristic += 1
return heuristic
```

- **Beam search analysis for different beam lengths:**

Input CNF	Beam Width	No. of states in path	No. of states explored
['D', '~A', '~B', 'C'], ['~C', 'A', 'B', 'D'], ['~C', 'D', '~A', '~B'], ['C', '~A', 'D', 'B'], ['~B', '~C', 'A', '~D']	1	1	1
	2	1	1
	3	1	1
	4	1	1
['B', '~A', '~D', 'C'], ['C', 'D', '~B', 'A'], ['~B', '~D', '~A', 'C'], ['~D', 'A', 'C', 'B'], ['A', 'C', 'B', 'D']	1	2	2
	2	2	2
	3	2	2
	4	2	2
['~D', '~C', 'B', '~A'], ['B', '~D', '~C', 'A'], ['~D', '~A', '~C', '~B'], ['D', 'B', '~A', '~C'], ['D', 'C', 'A', 'B']	1	2	2
	2	2	2
	3	2	2
	4	2	2
['A', 'B', 'C', 'D'], ['~A', '~B', '~C', '~D'], ['~A', 'B', 'C', 'D'], ['A', '~B', 'C', 'D'], ['A', 'B', 'C', '~D']	1	2	2
	2	2	2
	3	2	2
	4	2	2

Beam Search for different Beam Widths

- **Tabu search for different values of tabu tenure:**

Input CNF	Tabu Tenure	No. of states in path	No. of states explored
['D', '~A', '~B', 'C'], ['~C', 'A', 'B', 'D'], ['~C', 'D', '~A', '~B'], ['C', '~A', 'D', 'B'], ['~B', '~C', 'A', '~D']	1	1	1
	2	1	1
	3	1	1
	4	1	1
['B', '~A', '~D', 'C'], ['C', 'D', '~B', 'A'], ['~B', '~D', '~A', 'C'], ['~D', 'A', 'C', 'B'], ['A', 'C', 'B', 'D']	1	2	2
	2	2	2
	3	2	2
	4	2	2
['~D', '~C', 'B', '~A'], ['B', '~D', '~C', 'A'], ['~D', '~A', '~C', '~B'], ['D', 'B', '~A', '~C'], ['D', 'C', 'A', 'B']	1	2	2
	2	2	2
	3	2	2
	4	2	2
['A', 'B', 'C', 'D'], ['~A', '~B', '~C', '~D'], ['~A', 'B', 'C', 'D'], ['A', '~B', 'C', 'D'], ['A', 'B', 'C', '~D']	1	2	2
	2	2	2
	3	2	2
	4	2	2

Tabu Search for different Tabu Tenures

- **Comparison of Variable Neighbourhood Descent (VND), Beam Search, Tabu Search:**

Input CNF	Algorithm	No. of states in path	No. of states explored
['B', '~A', '~D', 'C'], ['~A', 'B', 'D', '~C'], ['D', 'C', 'A', 'B'], ['~B', '~C', '~A', 'D'], ['D', 'A', '~B', '~C']	Beam Search	2	2
	Tabu Search	2	2
	VND	2	2
['A', 'D', '~B', 'C'], ['C', 'B', '~A', 'D'], ['C', '~A', 'B', 'D'], ['B', 'A', 'C', '~D'], ['D', '~C', 'B', 'A']	Beam Search	1	1
	Tabu Search	1	1
	VND	1	1
['C', 'D', 'A', 'B'], ['~B', '~D', 'C', 'A'], ['~B', '~C', 'D', '~A'], ['B', '~A', '~D', 'C'], ['C', '~D', 'A', '~B']	Beam Search	2	2
	Tabu Search	2	2
	VND	2	2
['B', '~C', '~A', 'D'], ['~D', '~B', '~A', '~C'], ['A', '~B', '~C', 'D'], ['B', '~C', 'D', '~A'], ['B', '~C', 'A', '~D']	Beam Search	1	1
	Tabu Search	1	1
	VND	1	1

Comparison of Variable neighbourhood descent, Beam Search, Tabu Search.

- - Comparison of Variable neighbourhood descent, Beam Search, Tabu Search for other types of inputs:

Input CNF	Algorithm	No. of states in path	No. of states explored
['D', 'A', 'B'], ['B', 'C', 'A'], ['D', 'A', 'C'], ['~C', '~B', 'A'], ['~A', 'D', 'C']	Beam Search	3	5
	Tabu Search	3	3
	VND	3	3
['C', '~A', 'B'], ['A', 'C', 'B'], ['B', 'A', '~C'], ['~B', 'D', 'C'], ['~D', '~A', '~B']	Beam Search	1(Stuck in local Maxima)	1(Stuck in local Maxima)
	Tabu Search	3	3
	VND	2	2
['B', '~A', 'D'], ['A', '~B', 'C'], ['~C', '~B', 'D'], ['A', 'B', 'D'], ['A', 'B', '~D']	Beam Search	1(Stuck in local Maxima)	1(Stuck in local Maxima)
	Tabu Search	3	3
	VND	2	2
['A', '~D'], ['B', '~A'], ['A', 'B'], ['~B', 'C']	Beam Search	1(Stuck in local Maxima)	1(Stuck in local Maxima)
	Tabu Search	4	4
	VND	2	2

Comparison of Variable neighbourhood descent, Beam Search, Tabu Search: Other clause lengths.

- **CONCLUSION:**

We can conclude that such k-SAT problems 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 Beam Search, Tabu Search & VND.

In our observations, we can see that most of the cases have around 1 or 2 explored states. This is due to the nature of the problem itself; the initial state is [0, 0, 0, 0], so for satisfiability, each clause must have at least one negated term. The chances of every clause containing at least one negated term is pretty high, i.e., if [A, B, C, D] is one of the clauses, only then will the initial state not be a goal state.