CS312 - Artificial Intelligence Lab
Assignment 3

Group 10

Rudraksh (190030038), Sumedhsingh (190030042)

Likhilesh (190030024)

Introduction

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 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.

1 Description of Domain

1.1 State Space: A state space is the set of all possible configurations of a system. In this case we have a state representation by string of size 4 representing 4 literals. And each

literal takes two values 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

1.2 Start State & Goal State

We are starting with all literal values 0 and it is allowed to change any literal in the next move. And Goal state implies if we get all the clauses value to 1.

1.3 Pseudo Codes

The section below has pseudo codes of a few important functions.

1.3.1 moveGen(state)

This function generates and returns the states that can be reached from a given state in one step. Note that at most 6 new neighbours are possible for a given state.

Algorithm 1 moveGen(state)

```
moveGen(state,k)
neighboursList
  if(k==1):
      neighboursList <=get all states by changing one one bit
  else if(k==2):
      neighboursList <=get all states by changing one two bits
 else if(k==3):
      neighboursList <=get all states by changing one three bits
 else if(k==4):
      neighboursList <=get all states by changing one four bits

return neighboursList
```

1.3.2 goalTest(state, goalState)

This function returns true if the given state is goal state otherwise returns false.

Algorithm 2 goalTest(state, goalState)

```
goalTest(state)

 if [a, b, c, d] satisfies CNF then
      return true
 return false
```

1.3.3 Beam Search

Beam search is extension of Best first Search based on Heuristic Algorithm. The only difference is now we take multiple best nodes based on beam width instead on one best node and carry on the same way.

Algorithm 3 Beam Search

```
Data: Graph (G), start node (s), goal node (g), beam width (B)
Result: Path with lowest cost
Function beamSearch(G,s,g,B)
open Lists <- s
closed List <- empty list
path <- empty list
while open list is not empty do
b <- best node from open List
openList.remove(b)
closedList.add(b)
if b is g then
path.add(b)
return path
```

end
N <- neighbors(b)
for n in N do
if n is in neither closedList nor openList then
open List.add(n)
else if n is in open List then
if path with current parent <= path with old parent
then
Replace parents of n
end
else if n is not in closed List then
openList.add(n)
end
end
if number of nodes in open List > B then
openList <- best B nodes in open List
end
end
return path
end

1.3.4 Variable Neighbourhood Descent

The variable neighbourhood descent (VND) method is obtained if a change of neighbourhoods is performed in a deterministic way. Where neighbourhoods are denoted as Nk, k= 1 ... kmax.

Algorithm 4 Variable Neighbourhood Descent

---

input : starting solution,s0
input : neighborhood operators ,{Nk},k=1,..,kmax
input : evaluation function ,f

```
k <= 1
while k<=kmax do
  s <= the best neighbor in Nk (current)
  if f(s) < f(currrent) then
     current <=s
     k<=1
  else
     k<= k+1
   end if
end while
```

1.3.5 Tabu Search

We have used tabu search as shown below.

Algorithm 5 Tabu Search

```
Sbest = S0 /Construct Initial Solution/
bestCandidate = Sbest
TabuList <- [] /Initialize TabuList/
while (NOT StoppingCondition ()) do
Generate candidate solutions in the neighborhood of Sbest
set Scandidate as the first candidate in the Sbest Neighborhood
for (Scandidate in Sbest Neighborhood) do
if (Scandidate NOT in TabuList AND fitness(SCandidate) >
fitness(bestCandidate)) then
best Candidate <- Scandidate
end if
```

end for
if (fitness(bestCandidate) > fitness(Sbest)) then
Sbest <- bestCandidate
end if
Update TabuList (push the best Candidate)
if (tabutenure is over for candidate)
Remove the candidate from the Tabu List
end if
end while
return Sbest

2 Heuristic Function

Our Heuristic value ranges from 0 to 5, based on logic mentioned below.

Algorithm 3 Heuristic function

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

3 Beam Search Analysis

Initial State	Clauses + Initial State ({'a': 0, 'b': 0, 'c': 0, 'd': 0})	Beam width	States Explored	Solution found
({'a': 1, 'b': 0, 'c': 0, 'd': 1})	cbAD AbDC ABDC cbaD dCBa	1	2	"1000"
		2	2	"1000"
		3	2	"1000"
		4	2	"1000"
({'a': 0, 'b': 0, 'c': 0, 'd': 0})	bAcD BdAC dcAb cAdb cBad AcBd cadB bCAD BCda BcdA ABdc ACbD DbAC cbda BdCA bAdC adCB acdb aBdc cAdB AcBd caDB CbAD bCAd dbaC cbda DABC abCd abDc CbdA	1	3	"0011"
		2	4	"0011"
		3	5	"0011"
		4	6	"0011"
({'a': 1, 'b': 0, 'c': 0, 'd': 0})	DCAb cDBA abDc CBdA acdB cdBA aDcB cABd aDCB cBAD baDC bcaD CDaB dCAb bdcA BAcD CBDA abcD acBd BADc ABcd BDcA ADbC bCaD BdCa dcba dbAC cDAB aBcd BAcD ACdb dcbA BDaC Dacb ADbC bADc bCDa bdcA caDb cDBa	1	7	"0010"
		2	4	"0010"
		3	5	"0010"
		4	6	"0010"

4 Tabu Search Analysis

Initial State	Clauses + Initial State ({'a': 0, 'b': 0, 'c': 0, 'd': 0})	Tenure	States Explored	Solution found
({'a': 1, 'b': 0, 'c': 0, 'd': 1})	cbAD AbDC ABDC cbaD dCBa	1	2	"1000"
		2	2	"1000"
		3	2	"1000"
		4	2	"1000"
({'a': 0, 'b': 0, 'c': 0, 'd': 0})	bAcD BdAC dcAb cAdb cBad AcBd cadB bCAD BCda BcdA ABdc ACbD DbAC cbda BdCA bAdC adCB acdb aBdc cAdB AcBd caDB CbAD bCAd dbaC cbda DABC abCd abDc CbdA	1	3	"0011"
		2	3	"0011"
		3	3	"0011"
		4	3	"0011"
({'a': 1, 'b': 0, 'c': 0, 'd': 0})	DCAb cDBA abDc CBdA acdB cdBA aDcB cABd aDCB cBAD baDC bcaD CDaB dCAb bdcA BAcD CBDA abcD acBd BADc ABcd BDcA ADbC bCaD BdCa dcba dbAC cDAB aBcd BAcD ACdb dcbA BDaC Dacb ADbC bADc bCDa bdcA caDb cDBa	1	7	"0010"
		2	7	"0010"
		3	8	"0010"
		4	5	"0010"

5 Comparison of above searches

Initial State	Clauses	Algorithms	States Explored	Solution found
({'a': 1, 'b': 0, 'c': 0, 'd': 1})	cbAD AbDC ABDC cbaD dCBa	Beam (2)	2	"1000"
		VND	2	"1000"
		Tabu (2)	2	"1000"
({'a': 0, 'b': 0, 'c': 0, 'd': 0})	bAcD BdAC dcAb cAdb cBad AcBd cadB bCAD BCda BcdA ABdc ACbD DbAC cbda BdCA bAdC adCB acdb aBdc cAdB AcBd caDB CbAD bCAd dbaC cbda DABC abCd abDc CbdA	Beam (2)	4	"0011"
		VND	3	"0011"
		Tabu (2)	3	"0011"
({'a': 1, 'b': 0, 'c': 0, 'd': 0})	DCAb cDBA abDc CBdA acdB cdBA aDcB cABd aDCB cBAD baDC bcaD CDaB dCAb bdcA BAcD CBDA abcD acBd BADc ABcd BDcA ADbC bCaD BdCa dcba dbAC cDAB aBcd BAcD ACdb dcbA BDaC Dacb ADbC bADc bCDa bdcA caDb cDBa	Beam (2)	4	"0010"
		VND	6	"0010"
		Tabu (2)	7	"0010"

6 Overall Observations and Conclusions

1) For above comparison we have considered beam width = 2 and tabu tenure = 2.
2) Beam Search is same as hill climbing but better, as chances of getting to the solution is more.
3) VND is better than tabu in term of state explored because tabu barred some state which might be possible solution.