CS 312: Artificial Intelligence Laboratory
Lab 2 Report

S V Praveen - 170010025 Deepak H R - 170010026

January 27, 2020

1 Introduction

The objective of this task is to simulate Best First Search, Hill Climbing, Variable Neighbourhood Descent, Beam Search and Tabu Search for the Job Allocation problem. Given, Cost matrix $C(NxN)$, where each $C(i,j)$ represents the cost of Person i for Job j. The problem asks us to find an optimal allocation of N Tasks to N unique people. A solution space approach is explored and nearly optimal values of allocation can be achieved by adding constraints. The solution-space consists of an 1 x n vector [The i'th position indicates the job Index assigned to i'th person]. The start state is $\{0,1,2,..N-1\}$. The goal state is an optimal allocation that satisfies the required total cost constraint.

2 Brief Description

2.1 State Space

A space S is a (1 x N) vector, where $S[i]$ represents the Job Index assigned to Person i. Since, a person must be uniquely assigned to a task, the solution space consists of $N!$ possible solutions.

2.2 Start State

For the sake of simplicity, the start state is assigned as

$$\{0,1,2,..N-1\}$$

2.3 Goal State

The goal state G is a permutation of $\{0,1,2,..N-1\}$ if it satisfies a total cost constraint c(a positive integer), ie,

Let $h(x) \leftarrow$ total cost of assignment of state x
If $h(G) < c$ and G is a valid state,
Then G is goal state.

3 Pseudo Code

3.1 MoveGen(state)

The function takes a state as input and returns a set of states that are reachable from the input state in one step. In this case, since a solution space approach is used, neighbours are obtained by shuffling the tasks assigned for d people, where $d \leftarrow$ density.

Algorithm 1 moveGen(state, d=2)

```
procedure MOVEGEN(state, d=2)
    nextStates ← ()                                   ▷ initialize nextStates to empty set
    for every combination C of d indices in {0,1,2,...,N−1} do
        for every permutation P of C do
            new = shuffleAssignment(state , P)         ▷ Shuffles state assignment for indices in P
            nextStates.append(new)
    return nextStates                                 ▷ nextStates are required moves generated
```

3.2 GoalTest(state, constraint)

Returns true if the input state is goal and false otherwise. The procedure first tests whether N jobs are uniquely assigned to N people. It also checks if the total cost of the state, ie, h(state) satisfies the constraint.

Algorithm 2 goalTest(state)

```
procedure GOALTEST(state)
    uniqueJobs = ()                                   ▷ Empty Set
    for Job j in state do
        uniqueJobs.insert(j)
    if size(uniqueJobs) == N and h(state) < constraint then
        return true
    else
        return false
```

4 Heuristic

The Job Allocation problem is an optimisation problem which has only one meaningful heuristic when considering the solution space approach. The heuristic used is

$$h(x) := \sum_{i=0}^{N-1} C(i, x[i]) \tag{1}$$

An alternative heuristic $g(x)$ makes use of the number of ancestor states and $h(x)$ Let $p(x)$ be the number of ancestors of state x

$$g(x) := h(x) + k * p(x) \tag{2}$$

where k is a fixed positive constant.

5 Analysis and Observations

5.1 Best First Search

Best First Search finds an optimal allocation as it explores every possible state. The two heuristics differ in the time taken and number of states explored. The below table summarizes the results obtained.

Using heuristic $h(x)$

Input	No. states explored	Time taken (s)
input1.txt	302	0.018719
input2.txt	30	0.002456
input3.txt	132	0.008143
input4.txt	231	0.011227

5.2 Best First Search VS Hill Climbing

The below table summarizes the results obtained between Best First Search and Hill Climbing algorithm. The density used for the movegen function is 3. As expected, the Hill Climbing algorithm runs faster than Best First Search due to its greedy nature and lesser number of states explored. A drawback seen from the Hill Climbing algorithm is the fact that it does NOT always find an optimal solution.

Algorithm	Parameter			
	Input	No. states explored	Time taken(s)	Optimal Solution
BFS	input1.txt	302	0.018719	Yes
HC	input1.txt	8	0.014438	Yes
BFS	input2.txt	30	0.002456	Yes
HC	input2.txt	3	0.001466	Yes
BFS	input3.txt	132	0.008143	Yes
HC	input3.txt	6	0.006709	Yes
BFS	input4.txt	231	0.011227	Yes
HC	input4.txt	5	0.006227	No

5.3 Beam Search Analysis

The table summarizes the observations obtained for different beam widths. The problem does not entirely take the advantage of beam search due to the high unlikelihood of two states having the same heuristic value. Finding a heuristic, that would allow this to happen and at the same time drive the algorithm to an optimal state seemed impossible. The optimal beam width is highly dependent on the input. A beam width of 2 turned out to be a good candidate while performing Beam Search.

Input	Parameter			
	Beam width	No. states explored	Time taken(s)	Optimal Solution
input1.txt	2	7	0.012238	Yes
	4	7	0.012844	Yes
	8	7	0.014765	Yes
input2.txt	2	2	0.001615	Yes
	4	2	0.001177	Yes
	8	2	0.001237	Yes
input3.txt	2	5	0.005541	Yes
	4	5	0.005953	Yes
	8	5	0.004201	yes
input4.txt	2	4	0.006135	No
	4	4	0.006937	No
	8	4	0.005376	No

5.4 Tabu Search Analysis

The table summarizes the observations obtained for different tabu tenure values. 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.

Input	Parameter			
	Tabu Tenure	No. states explored	Time taken(s)	Optimal Solution
input1.txt	2	7	0.008690	Yes
	4	3	0.176319	Yes
	6	2	4.305657	Yes
input2.txt	2	2	0.000868	Yes
	4	1	0.000720	Yes
	6	1	0.000720	Yes
input3.txt	2	5	0.003395	Yes
	4	2	0.051434	Yes
	6	1	0.339309	Yes
input4.txt	2	4	0.003016	No
	4	3	0.057453	Yes
	6	2	0.503820	Yes

5.5 Comparison of Variable neighborhood descent, Beam Search, Tabu Search

The algorithms are similar in the number of states they explore. Tabu Search is nearly always faster than other algorithms, however, it does not guarantee an optimal solution. VND outperforms the other algorithms in finding an optimal solution.

Input	Algorithm	Parameter		
		No. states explored	**Time taken(s)**	**Optimal Solution**
input1.txt	VND	7	0.117301	Yes
	Beam	7	0.012238	Yes
	Tabu	7	0.008690	Yes
input2.txt	VND	7	0.004418	Yes
	Beam	2	0.001615	Yes
	Tabu	2	0.000868	Yes
input3.txt	VND	5	0.044563	Yes
	Beam	5	0.005541	Yes
	Tabu	5	0.003395	Yes
input4.txt	VND	5	0.049682	Yes
	Beam	4	0.006135	No
	Tabu	4	0.003016	No

6 Conclusion

From the results, it is seen that Best First Search always finds an optimal solution at the cost of time, as it explores all possible N! states in the solution space. On the other hand, the Hill Climbing Algorithm, has lesser execution time due to its greedy nature but it cannot guarantee an optimal solution. Beam Search is expected to do better than Hill Climbing as it searches a bigger state space in a reasonably similar amount of time. Variable Neighbourhood Descent is seen to outperform Hill Climbing in finding an optimal solution, due to the denser movegen function used, at the cost of execution time. Finally, Tabu search performs similar to VND but may or may not find an optimal solution.