CSE 537- Artificial Intelligence
Project-3 Report

Team Members:
1. Rajendra Kumar R (109785402)
2. Sri Sruti Mandadapu (109749215)

Files Submitted:
1. csp.py:
   The code for Constraint Propagation which implements the three algorithms.
2. Read_Input.py:
   Reads from the input file and creates data structures to store the data from the file.
3. Test_Input_Files(Folder) :
   This folder contains all the test files with which the code is executed and verified.

Assumptions:
1. If a TA does not have any skill, the he/she will not be considered for assignment.
2. If a course does not have any specific skills requirement, then any TA can be assigned to it.
3. An assignment is considered to be complete if all the TAs are assigned to their full availability or all the courses have their assignment full.
4. A TA can be partially assigned too, i.e. TA having a single course assigned to 0.5 is valid.

Algorithms Implemented:

1. Plain Backtracking Search:

The ***backtracking_search*** function implements this algorithm. At each step, a value form the domain (Courses) is assigned to a variable (TA) if the given constraints are satisfied. If the assignment fails, then the algorithm backtracks, that is the previous assigned values are removed and new assignments are made to the variables.

Sample output:
When this algorithm is run with the "testInput2" file, following is the obtained output:

```
=======Constraint Propagation=======
Variables :  {'TA4': 1, 'TA5': 1, 'TA6': 1, 'TA7': 1, 'TA1': 1, 'TA2': 1, 'TA3': 1, 'TA8': 1, 'TA10': 1, 'TA9': 1}

Domains:  {'CSE114': 1.5, 'CSE306': 0.5, 'CSE307': 0, 'CSE110': 2, 'CSE101': 1.5}

Total number of nodes expanded for Backtracking Search: 284

Total time taken by the backtracking search algorithm: 0.00226712226868 seconds

Assignment:  {'TA4':  [['CSE306',  0.5]],  'TA5':  [['CSE110',  1.0]],  'TA6':  [['CSE110',  1.0]],  'TA7':
```

```
[['CSE101', 1.0]], 'TA1': [['CSE114', 1.0]], 'TA10': [['CSE114', 0.5]], 'TA8': [['CSE101', 0.5]]}
```

2. Backtracking Search with Forward Checking:

The ***forward_checking*** function implements this algorithm. As in backtracking, a class is assigned to a TA at each step. When the value of a course becomes zero, it is removed from the list. This decreases the number of domain values to be checked for each TA whether they satisfy the constraints or no.

Sample output:
When this algorithm is run with the "testInput2" file, following is the obtained output:

```
=======Constraint Propagation=======
Variables :  {'TA4': 1, 'TA5': 1, 'TA6': 1, 'TA7': 1, 'TA1': 1, 'TA2': 1, 'TA3': 1, 'TA8': 1, 'TA10': 1, 'TA9': 1}

Domains:  {'CSE114': 1.5, 'CSE306': 0.5, 'CSE307': 0, 'CSE110': 2, 'CSE101': 1.5}

Total number of nodes expanded for Backtracking Search with Forward Checking: 161

Total time taken by the Forward checking algorithm: 0.00187110900879  seconds

Assignment: {'TA4': [['CSE306', 0.5]], 'TA5': [['CSE110', 1.0]], 'TA6': [['CSE110', 1.0]], 'TA7': [['CSE101', 1.0]], 'TA1': [['CSE114', 1.0]], 'TA10': [['CSE114', 0.5]], 'TA8': [['CSE101', 0.5]]}
```

Maximum skills matched (checked how ?)

3. Backtracking Search with Forward Checking and Constraint Propagation:

The ***arcConsistency*** method implements this algorithm. Each variable (TA) is checked if it is consistent with all the other variables. In other terms, TA1 is said to be consistent with TA2 if only if for every course(domain value) for TA1 there is a valid course(domain value) for TA2. This consistency check is performed before assigning courses to the TAs .

Sample Output:
When this algorithm is run with the "testInput2" file, following is the obtained output:

```
=======Constraint Propagation=======
Variables :  {'TA4': 1, 'TA5': 1, 'TA6': 1, 'TA7': 1, 'TA1': 1, 'TA2': 1, 'TA3': 1, 'TA8': 1, 'TA10': 1, 'TA9': 1}

Domains:  {'CSE114': 1.5, 'CSE306': 0.5, 'CSE307': 0, 'CSE110': 2, 'CSE101': 1.5}

Total number of nodes expanded for Backtracking Search with Forward Checking and Arc Consistency: 28

Total time taken by the Forward checking algorithm: 0.000289916992188  seconds
```

Assignment: {'TA1': [['CSE101', 0.5]], 'TA2': [['CSE101', 1]], 'TA3': [['CSE110', 1]], 'TA8': [['CSE110', 1]], 'TA9': [['CSE114', 1]], 'TA10': [['CSE114', 0.5], ['CSE306', 0.5]]}

Observations:
The number of nodes explored and the time taken by each algorithm are estimated. The results are shown below:

Input file: testInput2

Number of Nodes :

S.No.	Algorithm	Nodes
1.	Pure Backtracking Search	284
2.	Backtracking+ Forward checking	161
3.	Backtracking+ Forward checking+ Constraint Propagation	28

Time Taken:

S.No.	Algorithm	Time (in seconds)
1.	Pure Backtracking Search	0.00226
2.	Backtracking+ Forward checking	0.00187
3.	Backtracking+ Forward checking+ Constraint Propagation	0.00028

Testing:

- Unzip the file, AI-Project03-rraghupatrun_smandadapu.zip
- Open PyCharm.
- Open the file "csp.py" from the un-zipped directory.
- Run the program using "Shift + F10" or "ALT + u" - Click on "Run csp"
- Select the algorithm to run for the given input file.
- Hit Enter.

Optimizations:
1. In Forward Checking algorithm, if a course is fully assigned, it is removed from the list which is reducing the number of values that need to be checked for consistency, unlike Backtracking search where all the values are considered for consistency check. This reduces the number of nodes that are being explored at each level.
2. In the Constraint Propagation algorithm, only the courses that are valid, i.e. consistent with the corresponding TA are added to the list, which reduces the search time.
3. If a course requires a TA or no (i.e. the number of students are less than 25) is checked before the consistency check. This reduces the running time of the algorithms.

Challenges:
1. The given input file dataset_AI_CSP was very inconsistent. We made a few modifications to it. The new file is testInput1. Also, we implemented the below logic to handle this dataset:

While checking if the TA skills match a particular course's required skills in the function

skillTest we have checked if more than 80% of the skills required are not being satisfied by the TA, then the assignment is not considered.

Conclusion:

1. Backtracking is a brute force constraint checking algorithm, and it explored 284 nodes for the given input as shown on the Observations section.
2. Forward checking is an efficient backtracking as it prunes the domain of each variable as we traverse to the leaves of the Graph/Tree. Due to this we achieved the assignment by visiting just 161 nodes.
3. Constraint Propagation (Arc Consistency) has made the failure prediction even before the assignment, due to this the assignment is very simple and easy. However the problem is a very dense graph in which every node is connected to every other node so the arc check is costly, however there is a tradeoff at the assigning the value to the variables.