PROJECT 3

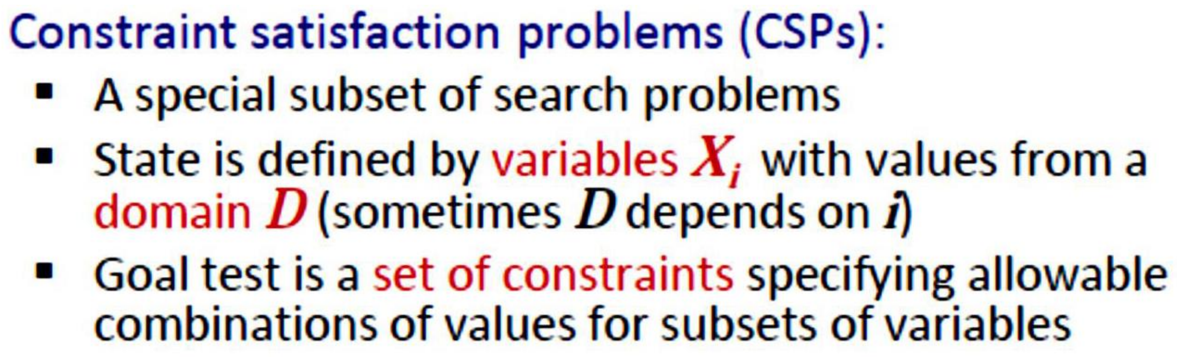
TA ALLOCATION FOR COURSES

CSE 537: ARTIFICAL INTELLIGENCE

(ABHINAV MISHRA & GOWTHAM SRINIVASAN)

In this project, we have provided an implementation for allocation of TAs in a department using Constraint Satisfaction Problems approach. We have implemented three variants of this solution approach which are,

1. Simple Backtracking Search
2. Backtracking Search with Forward Checking
3. Backtracking Search with Forward Checking coupled with Constraint Propagation



Constraint satisfaction problems (CSPs):

- A special subset of search problems
- State is defined by variables X_i with values from a domain D (sometimes D depends on i)
- Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Problem Definition:

In this problem, we are given following data,

1. **Course Schedules**: This data provides the information about lecture schedules of all the courses.
2. **Course Recitation**: This data provides information about recitation schedules of all courses.
3. **Course Details:** This data provides information about number of students enrolled in the course and if the TA is required to attend the lecture for the course.
4. **Course Skill:** This data provides the information about the skill that a TA should possess to get assigned to that course.
5. **TA Skill:** This data provides information about the skills possessed by TAs.
6. **TA Responsibility:** This data provides information about times when TA will not be available for assistantship.

Following are the constraints defined for the problem:

1. TA should have free time during recitation.
2. TA should have free time during the lecture if TA has to attend the lecture if data 3 mandates TA's presence during the lecture.
3. No. of TAs determined for each course based on following constraints,
   a) 25 <= Students < 40 => 0.5 TA
   b) 40 <= Students < 60 => 1.5 TA
   c) 60 <= Students => 2.0 TA
4. TA must possess at least one skill required by the course.
5. All course recitation are 90 minutes long.
6. All course lectures are 80 minutes long.

Implementation Details:

1. Simple Backtracking Search:

Implementation for this approach is provided in ***backtracking_plain.py***.

This code contains following method,

1. **main():** This method is the driver method for the implementation which reads the data using load_data utility provided I load_data.py module and initiates backtracking search by calling backtracking_search(ta_capacity, course_capacity_req) method.

2. **backtracking_search(ta_capacity, course_capacity_req):** This method creates a solution dictionary and triggers recursive backtracking call.

3. **recursive_backtracking_search(ta_capacity, course_capacity_req, solution):** This method is the core method of implementation. In this method an unassigned course is picked and TA from the TA list is assigned to this course. Before assigning the TA to the course check constraints methods are called to assess the viability of TA assignment. Once a TA is assigned a recursive call is made again to assign TAs to remaining courses. If the results of recursive call is failure which means if some course remained unassigned then different TA is assigned and recursive call is made again. In this way a solution list is generated which is later used to find an optimal solution based on maximum TA assignment. If recursive call result is not failure then the current assignment and result of remaining course assignment is combined and returned as an optimal solution.

4. **check_constraint(ta, course, num_skill_req, solution):** This method is central point of constraint checking which triggers different constraint check methods, combines

the results and return the result to recursive backtracking method for further processing.

5. **check_timing_constraint(ta, course):** This method checks if TA responsibility time is conflicting with course recitation time and course lecture time if attendance is required.

6. **check_skill_constraint(ta, course, num_skill_req):** This method checks if TA is a single skill match between TA and course.

7. **check_course_timing_clash_constraint(ta, course, solution):** A TA can be assigned to two course. It could lead to a use case where both course have times clashing between either lecture or recitation or lecture and recitation. This method detects such clashes.

8. **get_available_ta_capacity(ta, solution):** During the assignment of TA, it has to be checked if the TA which is being considered for assignment has already been assigned previously so that only remaining capacity is considered during this current assignment. This method provides the available TA for capacity for a TA based on assignment done so far.

2. Backtracking Search with Forward Checking:

Implementation for this approach has been provided in ***backtracking_forward_checking.py***. Implementation of this approach is on the lines of simple backtracking search. Contrary to previous approach, in forward checking after the assignment is done, domain values for all remaining courses are reduced based on the current assignment. This logic has been implemented by enhancing

recursive_backtracking_search(ta_capacity, course_capacity_req, solution):

Since this logic is already present in recursive backtracking, we don't have to check for valid available capacity during the backtracking assignment as a result,

get_available_ta_capacity(ta, solution)

method has been removed.

3. Backtracking Search with Forward Checking and Constraint Propagation:

Implementation of this approach can be found in ***constraint_propogation.py***. In constraint propagation, first TA assignment is done followed be forward checking. After forward checking constraint propagation is called recursively to eliminate conflicting assignment.

Apart from the methods defined in previous implementation, few additional important methods have been introduced in this implementation.

1. **constraint_propagation(course_domain_dict_copy,solution_copy**):
   Constraint propagation is done in addition to the forward checking, after every assignment. Constraint propagation is a recursive method which is used to reduce the domain of each course. In this method, we pick the courses which have only one possible assignment and assign the TAs to those courses. Then, we do a forward checking and reduce the domain for other courses and recursively call constraint propagation again. The method exits, when all courses have more than one possible assignment because in this case, the domain cannot be reduced anymore.

2. **update_course_ta_in_dict(course_domain_dict_copy, unassigned_course, t_a):**
   This method is a part of the forward checking process, and is used to reduce the TA domain for the other courses based on the current assignment.

Execution:

In order to execute any of the files mentioned above, please update the path used for loading the data.

1. ***backtracking_plain.py***: Update path in main function at line 13.
2. ***backtracking_forward_checking.py:*** Update path in main function at line 13.
3. ***constraint_propogation.py:*** Update path in main function at line 14.