Artificial Intelligence-Fall 2022

Project 03-Constraints Satisfaction Problems

Ehsan Hosseini-Mohammad Afshari

Mohammad Afshari

Overview

As we have reached the end of the academic semester, we have to divide the halls of the faculty between the groups, we have n halls and m groups, and each group is trying to choose at least one of the n halls to hold their exams. But there are a few points:

1. Each hall is only for one group
2. No group can choose two halls that are next to each other.
3. Each group likes certain halls and chooses only among them.

Now, the education director of the faculty asked us to model and solve this problem as a constraint satisfaction problem.

Mohammad Afshari

Phase one

In this phase we have to model problem as csp
Given:

- N: the number of halls (h_1, h_2, ..., h_n)
- M: the number of groups (g_1, g_2, ..., g_m)
- M lines, each indicating the halls that the corresponding group likes to choose
- E: the number of constraints
- E lines, each indicating two adjacent halls that cannot be assigned to the same group

Variables:

- X_i, where 1 <= i <= N, is an integer variable that represents the g hall i.
- Domain of X_i is a subset of {1,.., M}, based on the halls that each group likes.

Constraints:

1. Each hall can only be assigned to one group: ∀ i, 1 <= i <= N, X_i is assigned to exactly one group.
2. No two groups can be assigned to adjacent halls: ∀ i,j, where (i, j) is element of [1,N], X_i ≠ X_j.
3. Each group can only choose among the halls it likes: ∀ i, where 1 <= i <= N, X_i is assigned to a group j only if the hall i is in the list of halls that group j likes to choose.

Objective: The objective is to find an assignment of halls to groups that satisfies all constraints, and to return a list of n integers, where each integer represents the group assigned to the corresponding hall.

Phase one includes below modules:

- **Input**

  in which we have the generate_problem() function that is used for generating a problem based on inputs given such as constraints.

- **CSP**

  In the CSP module we have the CSP class in which we store the current state of the process meaning we store all the halls with all their attributes, constraints and assigned_halls which tells us if a certain hall is assigned or not.

- **Hall**

  in the Hall module we have the Hall class with the following attributes:

    - Preferences:an array of numbers used for storing the preferences of each Hall (domain).each number represents a certain group like “ce” or “me”.

    - Neighbors:an array of Halls used for storing the halls that are connected to the very Hall with the attribute.(which is given as constraints in input).

- **Main**

  The flow of the program starts from here and all modules will be used here

- **Utils**

  This module contains some auxiliary functions

    - has_conflict : checks for conflicts
    - goal_test: checks if assignment is complete
    - zero_prefrences: checks if preference list is empty or not
    - set_uncommon_prefrences: remove common preferences from first hall

Mohammad Afshari
ehosseini8001@gmail.com

Phase Two

Phase two includes the below modules:

- **Algorithms**
  this module contains the following methods:

    - **backTracking:**backtracking method is used to implement the backtracking algorithm.In backtracking algorithm we start with

an initial state and we use a triple,which consists of our CSP state , a certain hall that we are trying to assign a group to.every time first we check for the terminality of the state by calling the method “zero_preferences()”function which takes an state as argument and check if there is any hall with zero preferences which our problem is over and cannot be continued.

- **forward_checking()**:forward checking gets a CSP state as a problem ,hall_index and group _index.
  In forward checking we are only concerned with the neighbors preferences of a certain hall being assigned a certain group,specifically we remove the certain group given as parameter(group_index) from the neighbors of the specified hall with the given hall_index.

- **Heuristics**
  - **MRV**
    This module contains following methods:
    - find_Least_preference: Given a CSP problem, it returns the smallest number of preferences among all unassigned halls.
    - find_highest_neighbor: Given a CSP problem and the smallest number of preferences, it returns the index of the hall with the highest number of neighbors among all halls with that number of preferences.
    - MRV: The main function that returns the hall with the smallest number of preferences and the highest number of neighbors.

  - **LCV**
    Implementation of the Least Constraining Value (LCV) heuristic

The input to the code is an instance of the CSP class and an integer hall, representing the index of the hall for which the LCV heuristic should be applied.

ehosseini8001@gmail.com

Phase Three

Phase three includes the AC3 function

- **AC3**: this method takes a CSP state as a problem and a conflict list.AC3 is used for reducing conflicts by all the adjacent halls and the group assigned to them.specifically in AC3 we have a set on conflicts that is our edges or constraints array (array of tuples for showing edges or contect halls) then we go through while loop comparing the preferences of nodes two by two and removing the common preferences in the “set_uncommon_preferences”method we do have to be careful not to cause any terminality condition the one that concerns us is not having any preference at all which is caused by reducing the preferences of halls.but if we don't catch any terminality condition we move on and enqueue all the neighbors of the first hall (which preferences got reduced)so as to adjusting the preferences of all the neighbors with respect to the new preferences value of the current hall number one (hall_one) AC3 function is used to enhance the performance by reducing the number of preferences and looking for any terminality condition which is created by not having any preference possible at all.having the AC3 called on every state of the backtracking algorithm is somehow a termination test and tries to cut off any unnecessary expansion.