<Spring 2015: Algorithms>

Constraint Satisfaction Problems: Implementing Scheduling Algorithm
(DEADLINE: 6/7 23:59)

Disclaimer: This assignment is adapted from a project developed by Juan Mancilla and Oscar Sanchez at the University of Illinois at Urbana-Champaign

- **Goals:**

1. To dust off your Java programming skills;
2. To learn how you improve the algorithm;
3. To understand constraint satisfaction problems utilizing backtracking search.

Scheduling is a common CSP. You are to find a feasible meeting schedule that satisfies the constraints of each of the employees in a company. The description of our scheduling problem is as follows:

- There is a set of M employees.
- There is a set of N meetings, each of which has a duration of one time unit.
- The schedule has a maximum number T of time slots.
- Each employee has a set of meetings that he or she must attend.
- The scheduled time slots for meetings must enable the participating employees to travel between the respective meeting locations.

First, give a formal specification of this problem as a CSP (i.e., what are the variables, what are the domains of the variables, and what are the constraints). Then, implement backtracking search to find an assignment that satisfies each of these three problems in attached **hw.zip** (see below for the explanation of the problem files and format of the solution). In your report, give your solution (or say that none exists), as well as the number of attempted variable assignments for each instance. Experiment with different variable and value selection heuristics, as well as (optionally) with more advanced inference methods, to try to reduce the amount of backtracking as much as possible. Discuss your choices in the report.)

- **Format of the problem files**

Each problem file describes an instance of the scheduling problem, specifying the total number of meetings to schedule, the number of employees, which meetings each employee must attend, the times required to travel between meetings, and the total number of available time slots in the schedule. Notice that two meetings may be assigned to the same time slot as long there is no employee that must be in both of them. Consider the following sample problem:
Number of meetings: 5
Number of employees: 6
Number of time slots: 5

Meetings each employee must attend:
1: 2 5
2: 2 3
3: 1 5
4: 1 4 5
5: 2 4
6: 1 3
// This specifies the time required to travel from one meeting location to the other.
// For example, getting from meeting 2 to meeting 3 requires one time slot,
// and getting from meeting 1 to meeting 5 requires two time slots.

Travel time between meetings:

	1	2	3	4	5
1:	0	1	2	1	2
2:	1	0	1	1	1
3:	2	1	0	1	1
4:	1	1	1	0	1
5:	2	1	1	1	0

Here is the solution for the sample problem (you should report yours in a similar format):
Meeting 1 is scheduled at time 1
Meeting 2 is scheduled at time 1
Meeting 3 is scheduled at time 5
Meeting 4 is scheduled at time 3
Meeting 5 is scheduled at time 5

- **Algorithm Analysis**

Implementation is not difficult but we are going to more focus on the efficiency of your implemented algorithm. You need to show how efficiently your algorithm can solve provided problems. The problem3, which is the largest input, might take huge amount of time (it might take a day depending on your implementation). **Let's try to reduce the amount of backtracking as much as possible. Show your chosen strategies to improve your algorithm as well as run your algorithm at the presentation on 6/12** (Grading will be done during the presentation). Good Luck !!

Submission Instructions

One designated person from the group will need to submit on e-class by the deadline. Submission must consist of the following two attachments:

1. A **report** in **PDF format**. The report should briefly describe your implemented solution and fully answer all the questions posed above. **Remember: you will not get credit for any solutions you have obtained, but not included in the report.**

   As before, all group reports need to include a brief **statement of individual contribution**, i.e., which group member was responsible for which parts of the solution and submitted material.

   The name of the report file should be **teamname_assignment.pdf**. Don't forget to include the names of all group members and the number of credit units at the top of the report.

2. Your **source code** compressed to a **single ZIP file**. The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If we find it necessary to run your code in order to evaluate your solution, we will get in touch with you.

   The name of the code archive should be **teamname_assignment.zip**.