CPSC 433: Artificial Intelligence - Assignment Search Problem - Input and Output Definition

(Version 1)

Your system should be able to take as input a text file and positive integer inputs. The simplest way to do this is through the command line, but a GUI is an option as well if your group so chooses. The text file will contain a description of the specific problem being solved, and the positive integer inputs will be used for soft constraint penalty values and as additional multiplier weightings to soft constraint categories.

Example command line execution

java CPSC433F25Main filename $w_{minfilled}$ w_{pref} w_{pair} $w_{secdiff}$ $pen_{lecturemin}$ $pen_{tutorialmin}$ $pen_{notpaired}$ $pen_{section}$

java CPSC433F25Main input.txt 1 0 1 0 10 10 10 10

This minimal format is mandatory, additional parameters can be included (like control weights) with arguments that follow these.

The definition and usage of these specific weights and penalties will be addressed later in this document. The four penalties should already be familiar from the prior document discussing the project problem description. The four weights were not in that document.

You will generate a search instance (your starting state) based on the text file, which contains a few key words and then describes the information as tables.

Every key word header (Ex. **Lecture slots**:) will be in every input file, although sometimes there will be no data under a header. For example, if there are no paired lectures/tutorials, or no pre-assigned lectures/tutorials. The quantity of empty lines between each table is not guaranteed. Each line in the input file (that is not a key word header line or empty line) is a row in one of the key word tables. You should use keyword headers as table breaks. The fields within a row are separated by commas! Please note, that extra spaces, or missing spaces around commas may occur in valid input files.

The general scheme of an input text file is as follows:

Name:
Example-name

Lecture slots:
Day, Start time, lecturemax, lecturemin, allecturemax

Tutorial slots:
Day, Start time, tutorialmax, tutorialmin, altutorialmax

Lectures:
Lecture Identifier, alrequired

Tutorials:
Tutorial Identifier, alrequired

Not compatible:
Lecture Identifier, Lecture Identifier
Lecture Identifier, Tutorial Identifier
Tutorial Identifier, Tutorial Identifier

Unwanted:
Lecture Identifier, Slot day, Slot time
Tutorial Identifier, Slot day, Slot time

Preferences:
Slot day, Slot time, Lecture Identifier, Preference value
Slot day, Slot time, Tutorial Identifier, Preference value

Pair:
Lecture Identifier, Lecture Identifier
Lecture Identifier, Tutorial Identifier
Tutorial Identifier, Tutorial Identifier

Partial assignments:
Lecture Identifier, Slot day, Slot time
Tutorial Identifier, Slot day, Slot time

Naturally, this requires some additional explanations:

- Possible days for all references to lecture slots are **MO** and **TU** only (due to the additional hard constraint of the UofC that lectures on monday to be taught on wednesday and friday at the same time, resp. lectures taught on tuesday to be taught on thursday at the same time).
- Possible days for all references to tutorial slots are **MO**, **TU**, and **FR**.
- The possible start times for all available slots are stated in the problem description given prior.
- Every slot that is available must have an entry in the input file. If one of the possible slots does not occur in the input file, we assume that $\boldsymbol{lecturemax}$ and $\boldsymbol{lecturemin}$ (resp. $\boldsymbol{tutorialmax}$ and $\boldsymbol{tutorialmin}$) is 0, which means that no lectures (tutorials) can be scheduled into this slot!
- Active learning rooms are a subset of rooms available at a slot time so $\boldsymbol{allecturemax} \leq \boldsymbol{lecturemax}$ and $\boldsymbol{alltutorialmax} \leq \boldsymbol{tutorialmax}$. A lecture assigned to a lecture slot would count against $\boldsymbol{lecturemax}$ regardless of is $\boldsymbol{alrequired}$ property being true or not. If $\boldsymbol{alrequired}$**=true** the that assignment counts against $\boldsymbol{allecturemax}$ as well.
- We schedule only lectures and tutorials that occur in the listings:
  *Exception: if we have under **Lectures**: lectures starting with CPSC 351 and/or CPSC 413, we have to schedule CPSC 851 and/or CPSC 913 in the required slots (note, they are scheduled into tutorial slots) and have to satisfy all the regular constraints that are given in the problem description.*
- A Lecture Identifier has the form described in the problem description (see also the example below).
- A Tutorial Identifier has the form described in the problem description (see also the example below).
- The entries after "**Not compatible**" identify all lectures and tutorials that are not compatible with each other. But note that there have to be also some additional built-in incompatibilities, as described in the problem description! Note also that incompatibilities are symmetric!
  Finally, there is no particular order of the 3 types of statements, they can be mixed in this section!
- The entries under "**Unwanted**" provide a list of lectures/tutorials and slots in which we do not want the lectures/tutorials to be scheduled. Lecture statements and tutorial statements can occur in any possible order.
- "**Preferences**" can be expressed for lectures and tutorials, and we do not require a particular order here. i.e. it is not necessary to list all lectures first! If a preference entry does not refer to a valid time slot, you can ignore this

entry (but you might want to print out a warning)! Note that we can have preferences for lectures/tutorials and slots that also appear in the **"Unwanted**" list.

- The entries under "**Pair**" identify lectures that we want to be scheduled at the same time, if possible. (don't count these twice, only once for the pair altogether when it isn't fulfilled)
  Again, there is no particular order of the 3 types of statements, they can be mixed in this section!
- The order of the entries under "**Partial assignments**" is not fixed, again. If an entry is not valid (lecture with a slot that is not a lecture slot or tutorial with a slot that is not a tutorial slot) terminate with an error message.

Some of the questions regarding input-files from previous years were:

1. is the format static?, i.e. do the key words occur in the given sequence - > Yes
2. what to do with additional blanks in inputs? -> your parser should filter them
3. what kind of GUI do I expect? -> CMD line is sufficient (GUI very optional), regarding your output, see below.

The positive integer inputs will mostly have to be concerned with the various parameters that deal with the soft constraints and essentially for the function $\boldsymbol{Eval}$. I would like you to define $\boldsymbol{Eval}$ using four subfunctions $\boldsymbol{Eval_{minfilled}}$, $\boldsymbol{Eval_{pref}}$, $\boldsymbol{Eval_{pair}}$ and $\boldsymbol{Eval_{secdiff}}$ that are producing a value for each of the 4 types of soft constraints.

Then we have

$$\boldsymbol{Eval}(assign) = \boldsymbol{Eval_{minfilled}}(assign) * w_{minfilled} +$$
$$\boldsymbol{Eval_{pref}}(assign) * w_{pref} +$$
$$\boldsymbol{Eval_{pair}}(assign) * w_{pair} +$$
$$\boldsymbol{Eval_{secdiff}}(assign) * w_{secdiff}$$

with $w_{minfilled}$, w_{pref}, w_{pair} and $w_{secdiff}$ being weights that tell the system how important the different soft constraints are. Note that I will have examples that require as values for these weights a 0! Reminder $pen_{\text{lecturemin}}$, $pen_{tutorialmin}$, $pen_{notpaired}$ and $pen_{section}$ are being used by the individual **Eval** sub-functions. $\boldsymbol{Eval_{minfilled}}$ uses $pen_{\text{lecturemin}}$, $pen_{tutorialmin}$. $\boldsymbol{Eval_{pair}}$ uses $pen_{notpaired}$.

$\boldsymbol{Eval}_{\boldsymbol{secdiff}}$ uses $pen_{section}$. $\boldsymbol{Eval}_{\boldsymbol{pref}}$ uses the specific individual values in input file under "**Preferences**:".

The following is a short example input file that your parser should be able to parse without error (note there is some inconsistent spacing in this input):

Name:
ShortExample

Lecture slots:
MO, 8:00, 3, 2,0
MO, 9:00,3,2,1
TU, 9:30, 2, 1,2

Tutorial slots:
MO, 8:00, 4, 2,4
TU, 10:00,2,1,2
FR, 10:00, 2, 1,0

Lectures:
CPSC 231 LEC 01,true
CPSC 231 LEC 02,true
DATA 201 LEC 01,false
SENG 300 LEC 01,false

Tutorials:
CPSC 231 LEC 01 TUT 01,true
CPSC 231 LEC 02 TUT 02,true
DATA 201 LEC 01 LAB 01,false
SENG 300 TUT 01,false

Not compatible:
CPSC 231 LEC 01 TUT 01, CPSC 231 LEC 02 TUT 02
SENG 300 LEC 01, CPSC 231 LEC 01
SENG 300 LEC 01, CPSC 231 LEC 02
SENG 300 TUT 01, CPSC 231 LEC 02
CPSC 231 LEC 01, SENG 300 TUT 01

Unwanted:
CPSC 231 LEC 01, MO, 8:00

Preferences:
TU, 9:30, CPSC 231 LEC 01, 10
MO, 8:00, CPSC 231 LEC 01 TUT 01, 3
TU, 9:30, CPSC 231 LEC 02, 10
TU, 10:00, CPSC 231 LEC 01 LAB 02, 5
MO, 8:00, CPSC 231 LEC 02 LAB 02, 1
MO, 10:00, CPSC 231 LEC 02 LAB 02, 7

Pair:
DATA 201 LEC 01, SENG 300 LEC 01

Partial assignments:
DATA 201 LEC 01, MO, 8:00
DATA 201 LEC 01 LAB 01, FR, 10:00

As output from your system, I expect an assignment presented in the following form and the **Eval**-value that your system assigns to the assignment. There are several ways how we could output an assignment, but I prefer it ordered by lectures. This means that we order the lectures alphabetically. After each lecture, the tutorials for the lecture (also alphabetically). At the end of each lecture or tutorial, output the assigned slot using the assigned days and times that identified it in the input file.

Here is an example (that is not the solution to the above input example, please create your own test examples for the system functionality):

Eval-value: 30
CPSC 231 LEC 01 : MO, 10:00
CPSC 231 LEC 01 TUT 01 : TU, 10:00
CPSC 231 LEC 02 : MO, 14:00
CPSC 231 LEC 02 TUT 02 : MO, 8:00
SENG 300 LEC 01 : TU, 9:30
SENG 300 TUT 01 : MO, 8:00
DATA 201 LEC 01 : MO, 8:00
DATA 201 LEC 01 LAB 01 : FR, 10:00