LINGI2261: Artificial Intelligence
Assignment 4: Local Search and Propositional Logic

François Aubry, Cyrille Dejemeppe, Yves Deville
November 2015

Guidelines

- This assignment is due on **Wednesday 9 December, 6:00 pm**.
- *No delay* will be tolerated.
- Not making a *running implementation* in *Python 3* able to solve (some instances of) the problem is equivalent to fail. Writing some lines of code is easy but writing a correct program is much more difficult.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Indicate clearly in your report if you have *bugs* or problems in your program. The online submission system will discover them anyway.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated. The consequences of *plagiarism* is *0/20 for all assignments*.
- Answers to all the questions must be delivered at the INGI *secretary* (paper version). Put your names *and your group number* on it. Remember, the more concise the answers, the better.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No report or program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.

Deliverables

- The answers to all the questions in paper format **at the secretary**. **Do not forget to put your group number on the front page**.
- The following files are to be submitted on *INGInious* inside the *Assignment 4* task(s):
  - `maxvalue`: *maxvalue* local search for wedding problem, to submit on INGInious in the *Assignment4: Wedding Problem: maxvalue* task.
  - `randomized_maxvalue`:*randomized maxvalue* local search for wedding problem, to submit on INGInious in the *Assignment4: Wedding Problem: randomized maxvalue* task.
  - `get_clauses` and `get_nb_vars`: methods to solve the RPG equipment problem (section 2.2), to submit on INGInious in the *Assignment4: RPG Equipment Problem* task.

1 The Wedding Problem (13 pts)

A big wedding with n guests is being organized and they are having trouble choosing the tables that should be assigned to each guest. There is a total of t tables (t divides n) and we know the affinities bewtween the guests. The goal is to find a layout that maximizes the total affinity.

More formally, the affinity between each pair of guests is given in a n by n matrix A. The affinities are integer values. A negative value in cell $A[i][j]$ means that person i dislikes person j. A zero value means that person i is indifferent to person j and a positive affinity means that person i likes person j. Note that we do not assume affinities to be symmetric, that is, we can have $A[i][j] \neq A[j][i]$. The values in the diagonal have no meaning and are set to 0.

If T_i denotes the persons at table i, your goal as an event planner is to maximize:

$$\sum_{i=1}^{t} \sum_{p_1 \neq p_2 \in T_i} A[p_1][p_2]$$

The format for describing the different instances on which you will have to test your programs is the following:

Input format

n
t
$A[0][0]$ $A[0][1]$ ... $A[0][n-1]$
$A[1][0]$ $A[1][1]$ ... $A[1][n-1]$
...
$A[n-1][0]$ $A[n-1][1]$ ... $A[n-1][n-1]$

In the instance files, n is the number of guests and t is the number of tables. You can assume that t divides n. Then follows the matrix A giving the affinities between the guests. This is represented by n lines each with n integers separated by singles spaces. The j-th integer on the i-th line represents $A[i][j]$, the affinty between guest i and guest j.

The goal is to assign n/t guests to each table so that the total affinity is maximized. You may assume that $n/t \geq 3$.

For this assignment, you will use *Local Search* to find good solutions to the wedding problem. The test instances can also be found on Moodle. A template for your code is also provided. The output format **must** be the following:

Output format

V_{init}
$T_1[0][0]$ $T_1[0][1]$ ... $T_1[0][s-1]$
$T_1[1][0]$ $T_1[1][1]$ ... $T_1[1][s-1]$
...
$T_1[t-1][0]$ $T_1[t-1][1]$ ... $T_1[t-1][s-1]$
V_{final}
$T_2[0][0]$ $T_2[0][1]$ ... $T_2[0][s-1]$
$T_2[1][0]$ $T_2[1][1]$ ... $T_2[1][s-1]$
...
$T_2[t-1][0]$ $T_2[t-1][1]$ ... $T_2[t-1][s-1]$

Where V_{init} is the value of the initial solution (described later) and T_1 represents the table assignment in that solution. Each line represents the guests on that table, **sorted** by index. Guests are numbered from 0 to $n-1$. The numbers are separated by single spaces.

The second part is the same but contains the final solution. V_{final} is the value of that solution and T_2 is the table assignment in the final solution.

Diversification versus Intensification

The two key principles of Local Search are intensification and diversification. Intensification is targeted at enhancing the current solution and diversification tries to avoid the search from falling into local optima. Good local search algorithms have a tradeoff between these two principles. For this part of the assignment, you will have to experiment this tradeoff.

Questions

1. Formulate the wedding problem as a Local Search problem. You are given a template on Moodle: *wedding.py*. Implement your own extension of the *Problem* class from `aima-python3`. For this assignment, the moves considered in the algorithms are: swap the tables assigned to two guests that are on different tables.
2. **(8 pts)** Implement the `maxvalue` and `randomized maxvalue` strategies. To do so, you can get inspiration from the *randomwalk* function in *search.py*.
   (a) `maxvalue` chooses the best node (i.e., the node with maximum value) in the neighborhood, even if it degrades the current solution. The `maxvalue` strategy should be defined in a function called *maxvalue* with the following signature: *maxvalue(problem,limit=100,callback=None)*.
   (b) `randomized maxvalue` chooses the next node randomly among the 5 best neighbors (again, even if it degrades the current solution). The `randomized maxvalue` strategy should be defined in a function called *randomized_maxvalue* with the following signature: *randomized_maxvalue(problem,limit=100,callback=None)*.

   To construct the initial solution, denote by $s = n/t$ the number of guests in each table. Use the following greedy algorithm: while there is a person p that is not assigned to a table, assign p and the $s - 1$ unassigned persons with which p has more affinity to a new table. In case of ties assign the person with smallest index.
   For example, if there are 2 tables and 6 guests with affinities:

	0	1	2	3	4	5
0	0	-1	5	-3	0	0
1	1	0	-1	-4	-2	1
2	0	1	0	0	-1	-1
3	-5	0	0	0	-2	1
4	-1	5	5	-2	0	0
5	3	-1	-2	-3	2	0

   In the first table we put persons $0, 2$ and 4 because the first unassigned person is 0 and the two persons that maximize affinity are 2 and 4. Person 4 ties with person 5 so we choose person 4 because $4 < 5$. Thus the other table has guests $1, 3$ and 5.
3. **(2 pts)** Compare the 2 strategies implemented in the previous question between each other and with `randomwalk`, defined in *search.py* on the given wedding instances. Report, in a table, the computation time, the value of the best solution and the number of steps when the best result was reached (`Node.step` may be useful). For the second and third strategies, each instance should be tested 10 times to eliminate the effects of the randomness on the result. When multiple runs of the same instance are executed, report the means of the quantities.
4. **(3 pts)** Answer the following questions:
   (a) What is the best strategy?
   (b) Why do you think the best strategy beats the other ones?
   (c) What are the limitations of each strategy in terms of diversification and intensification?
   (d) What is the behavior of the different techniques when they fall in a local optimum?

2 Propositional Logic (7 pts)

2.1 Models and Logical Connectives (1 pts)

Consider the vocabulary with four propositions A, B, C and D and the following sentences:

- $(A \wedge \neg B) \vee (B \wedge \neg C)$
- $(A \Rightarrow \neg B) \wedge \neg(C \vee \neg D)$

Questions

1. For each sentence, give its number of valid interpretations, i.e. the number of times the sentence is true (considering for each sentence **all the proposition variables** A, B, C and D).

2.2 RPG Equipment Problem (6 pts)

Let us assume you are playing a Role Playing Game (RPG) in which you have to reach a treasure at the top of a dungeon. To climb to the last floor of the dungeon, you have to go through a succession of levels. Each level contains a succession of enemies you have to defeat to reach the next level. Each enemy can only be defeated by your character if he has a given set of abilities. At the beginning of each level, a merchant sells pieces of equipment to your character. To each piece of equipment corresponds a set of abilities (some abilities can be provided by several pieces of equipment). Each piece of equipment presents exactly one conflict with another equipment piece. A conflict (equ_1, equ_2) is such that buying one of these equipment pieces is allowed (buying equ_1 or equ_2 is allowed) but you cannot buy both (buying (equ_1, equ_2) is not allowed).

Let us assume you have played the game for a long time before and your character has a huge amount of coins in his possession such that you are not limited by the amount of money when considering the purchase of equipment pieces. Let us also assume at the beginning of each level that the equipment purchased on the previous level is broken and cannot be used any more (i.e. at the beginning of each level, the player begins with no equipment). The relations between the equipment pieces, abilities and enemies are the following ones:

Provides when piece of equipment E provides ability A, if the player buys piece of equipment E he possesses the ability A.

IsProvided when ability A is required to beat the level, the player has to buy at least one equipment E such that ability A is provided by equipment E (an ability can be provided by several pieces of equipment).

Conflicts when pieces of equipment E_1 and E_2 are in conflict, the player cannot buy both pieces together. However, the player is allowed to buy one or zero of these equipment pieces.

Requires when enemy M requires the ability A, the player has to possess ability A to defeat enemy M.

Given a level (i.e. a collection of enemies) and a merchant (i.e. a collection of pieces of equipment), the problem consists in finding a set of pieces of equipment the player has to buy to the merchant to be able to defeat all the enemies from the level. For this assignment, we will only search for a satisfying set without bothering about its size. The player can of course possess some unneeded abilities.

Below is an example of merchant file containing the equipment pieces available:

Mini Merchant File Example

```
Equipment:  Lightning Pharis's Hat
Abilities:  Illusionary Dexterity, Eternal Wings
Conflicts:  Black Knight Shield

...

Equipment:  Air Elite Cleric Leggings
Abilities:  Impressive Necromancy, Fast Kick
Conflicts:  Fire Darksword
```

Below is an example of level file containing the enemies to defeat:

Mini Level File Example

```
Enemy:  Zombie
Requirements:  Eternal Wings, Fast Kick
```

A solution for the mini examples we showed above would be to buy the `Lightning Pharis's Hat` and the `Air Elite Cleric Leggings`.

Questions

1. Explain how you can express this problem with propositional logic. What are the variables and how do you translate the relations and the query?
2. Translate your model into Conjunctive Normal Form (CNF) and write it in your report.

On the iCampus site, you will find several files to download:

- `Merchant.gz` is a representation of the merchant at the beginning of each level. For this assignment, we consider the merchant is the same at the beginning of each level.
- `Level_0005.gz`, `Level_0010.gz` and `Level_0025.gz` are representations of levels containing respectively 5, 10 and 25 enemies.
- `rpg.py` is a Python module to load and manipulate the merchant and level files described above.
- `play.py` is a python file used to play the RPG equipment problem.
- `studentSol.py` is the skeleton of a Python module to simulate the RPG game on a given level with a given merchant.
- `minisat.py` is a simple Python module to interact with MiniSat.

- `minisat-ingilabs.tar.gz` is a pre-compiled MiniSat binary that should run on the machines in the computer labs.

MiniSat is a small and efficient SAT-solver we will use. Either use the pre-compiled binary from iCampus or download the sources from `http://minisat.se`. A quick user guide can be found at `http://www.dwheeler.com/essays/minisat-user-guide.html`, but that should not be needed if you use the `minisat.py` module. Extract the executable `minisat` from `minisat-ingilabs.tar.gz` in the same directory that `minisat.py` to be able to use the latter script.

To play the RPG equipment problem, enter the following command in a terminal:

```
python3 play.py MERCHANT_FILE LEVEL_FILE
```

where `MERCHANT_FILE` is the merchant (e.g., `Merchant.gz`) file and `LEVEL_FILE` is a level file (e.g., `Level_0005.gz`).

Questions

3. Modify the functions `get_clauses(merchant, level)` and `get_nb_vars(merchant, level)` in `studentSol.py` such that they solve the RPG equipment problem. Submit your functions on INGInious inside the *Assignment4: RPG Equipment Problem* task.