LINFO1361: Artificial Intelligence
Assignment 4: Local Search and Propositional Logic

Auguste Burlats, Yves Deville
April 2023

Guidelines

- This assignment is due on **Wednesday 17 May, 6:00 pm**.
- *No delay* will be tolerated.
- *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.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No 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*.
- The answer to questions must be given by filling in the latex template provided. The final file must be submitted on *gradescope*. No report sent by email will be accepted.
- Nothing must be modified in the template except your answer that you insert in the *answer* environments as well as your names and your group number. The names are provided through the command *students* while the command *group* is used for the group number. The dimensions of *answer* fields *must not* be modified either. For the tables, put your answer between the "&" symbols. Any other changes to the file will *invalidate* your submission.
- To submit on gradescope, go to `https://gradescope.com` and click on the "log in" button. Then choose the "school credentials" option and search for *UCLouvain Username*. Log in with your global username and password. Find the course LINFO1361 and the Assignment 4, then submit your report. Only one member should submit the report and add the second as group member.
- For those who have not been automatically added to the course, at the right bottom of gradescope homepage, ckick on "Enroll on course" button and type the code *86KJVB*.
- Check this link if you have any trouble with group submission `https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members`

Deliverables

- The answers to all the questions in a report **on INGInious. Do not forget to put your group number on the front page as well as the INGInious id of both**

group members.

- The following files are to be submitted:
  - `atomplacement_max.py`: your local search for the Atom Placement problem, to submit on INGInious in the *Assignment4: Atom Placement max value* task.
  - `atomplacement_random_max.py`: your *randomized max value* local search for the Atom Placement problem, to submit on INGInious in the *Assignment4: Atom Placement randomized max value* task.
  - `tapestry_solver.py`: which contains `get_expression` method to solve the Tapestry problem, to submit on INGInious in the *Assignment4: Tapestry Problem* task.

1 The Atom placement problem (13 pts)

This exercise deals with a simplification of a physics problem aiming at finding the ground state of a crystal. Your goal is to place a given set of atoms among provided sites. Atoms needs to be placed in such a way to minimize the energy associated to the interaction between them.

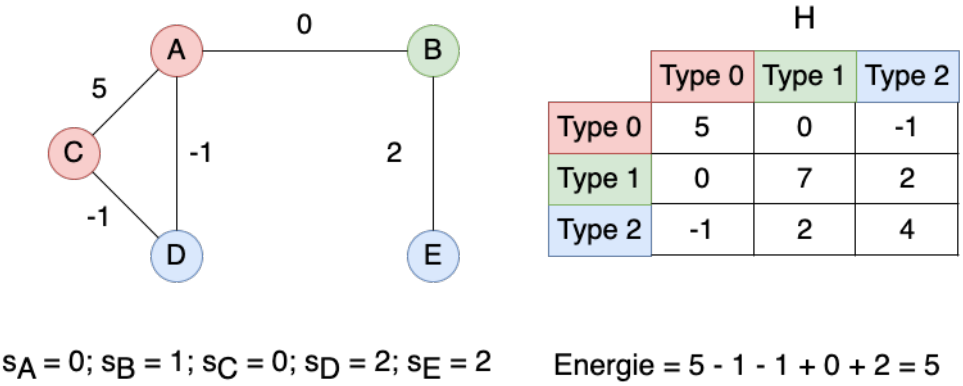

Figure 1: Example of an atom placement problem

You are provided with graph $G(V, F)$ where V corresponds to a set of sites where atoms will be placed and F is a set of edges. The graph is simple, unweighted and non-directed. If there is an edge $(i, j) \in F$, then the sites i and j are said to be neighbors. You are required to place a set of atoms N on the sites (we assume that $|N| = |V|$). There are k types of atoms. Let s_v be the type of the atom assigned on the site v. Two atoms placed in neighbor sites will interact. The energy associated to this interaction depends on the type of each node. The possible energies are given in an $k \times k$ matrix H. If two neighbors contain atoms of types i and j, the energy associated to their interaction is $H_{i,j} = H_{j,i}$ (H is symmetric). The elements of H can be positive, negative, or zero. The problem is to assign the atoms of N to the sites V in such a way as to minimize the total energy. The total energy is given by: $E = \sum_{(i,j)\in F} H_{s_i,s_j}$. E is therefore the sum of the interaction energies. Figure 1 shows an example with $k = 3$ and $N = \{0, 0, 1, 2, 2\}$.

All the atoms in N must be placed. A site can only contain one atom. A number of atoms

of each type is given. Thus, if the instance says that there are 5 atoms of type 1, then the solution must contain 5 atoms of type 1, not 4 or 6.

1.1 Objective function

As explained before, the objective function is the sum of the energies of each interaction :

$$E = \sum_{(i,j)\in F} H_{s_i,s_j} \tag{1}$$

where the set of variables to assign is $s = \{s_0, s_1, s_2, .., s_{|V|-1}\}$.

The solution must follow the constraints :

$$\forall i \in V : s_i \in \{0, \cdots, k-1\} \tag{2}$$

and

$$\forall a \in \{0, \cdots, k-1\} : n_a(s) = n_a(N) \tag{3}$$

where $n_a(A)$ is the number of occurrence of value a in set A.

1.2 Neighborhood

In local search, we have to build at each decision time a neighborhood in which a solution is picked depending on a specific criterion. We suggest to only consider local solutions with exactly $|V|$ vertices, and to use a simple swap action to build the neighborhood. This action consists of the swap of two vertices u and v where $s_u \neq s_v$.

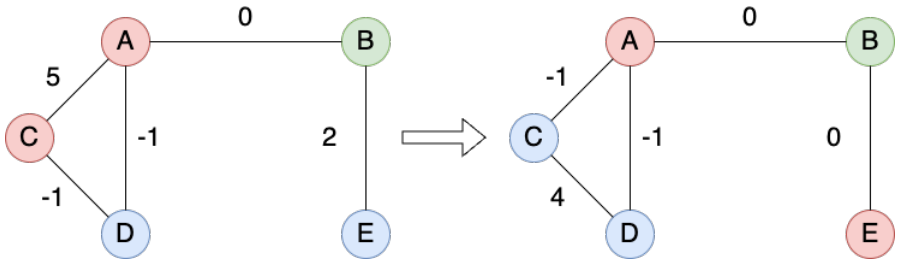

Figure 2: Swap of two vertices

You are not obliged to use the proposed objective function as well as techniques for building the neighborhood. You can use your owns but you have to specify in your report the ones you used. However, your solution has to contains the required number of atoms of each type. Moreover, note that your result will be evaluated with the function described in 1.

An atom placement instance contains $2 + k + |F| + 3$ lines. The instance is divided in four sections, separated by a blank line The first line contains 3 integers :the number of atoms/sites $n = |N|$, the number of types k and the number of edges $f = |F|$. The second section contains the number of atoms of each types (n_l is the number of atoms of type l). The following section, containing k lines, gives the H matrix. Each line contains k integers. The ith value of the jth row is the energie between type i and j. And the last section, which contains f lines, present each edge in F : each line contains two integers that are the indexes of the nodes that are linked by this edge.

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

Input format

$n \quad k \quad f$

$n_0 \quad n_1 \quad \ldots \quad n_{k-1}$

$H_{0,0} \quad H_{0,1} \quad \ldots \quad H_{0,k-1}$
$H_{1,0} \quad H_{1,1} \quad \ldots \quad H_{1,k-1}$
$\ldots$
$H_{k-1,0} \quad H_{k-1,1} \quad \ldots \quad H_{k-1,k-1}$

$u_0 \quad v_0$
$u_1 \quad v_1$
$\ldots$
$u_{f-1} \quad v_{f-1}$

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

Output format

$s_0 \; s_1 \; \ldots \; s_{n-1}$

Where $s_0 \; s_1 \; \ldots \; s_{n-1}$ are the types of atom placed in each site.
For example, the input and an output for the atom placement problem of Figure 1 could be:

Input format

5 3 5

2 1 2

5 0 −1
0 7 2
−1 2 4

0 1
0 2
0 3
1 4
2 3

Output format

0 0 1 2 2

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. **(1 pt)** Formulate the atom placement problem as a Local Search problem (problem, cost function, feasible solutions, optimal solutions).
2. **(5 pts)** You are given a template on Moodle: *atomplacement.py*. Implement your own extension of the *Problem* class from `aima-python3`. Implement the `maxvalue` and `randomized maxvalue` strategies. To do so, you can get inspiration from the *randomwalk* function in *search.py*. Submit your code on INGInious inside the *Assignment 4 : Atom Placement* task. Your program will be evaluated on 14 instances (during 1 minute) of which 4 are hidden. We expect you to find a good enough solution on at least 12 out the 14.
   (a) `maxvalue` chooses the best node (i.e., the node with maximum value) in the neighborhood, even if it degrades the quality of the current solution. The `maxvalue` strategy should be defined in a function called *maxvalue* with the following signature: `maxvalue(problem,limit=100)`.
   (b) `randomized maxvalue` chooses the next node randomly among the 5 best neighbors (again, even if it degrades the quality of 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)`.
3. **(3 pts)** Compare the 2 strategies implemented in the previous question and `randomwalk` defined in *search.py* on the given atom placement instances. Run the strategies during 100 steps, and report, in a table, the computation time, the value of the best solution and the number of steps needed to reach the best result. For the randomized max value and the random walk, each instance should be tested 10 times to reduce the effects of the randomness on the result. When multiple runs of the same instance are executed, report the mean of the quantities.
4. **(4 pts)** Answer the following questions:
   (a) What is the best strategy?
   (b) Why do you think the best strategy beats the other ones? What conclusions can be drawn on the atom placement problem ?
   (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 pt)

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

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

Questions

1. **(1 pt)** 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 Tapestry Problem (6 pts)

The Tapestry Problem can be defined as follow. Given a tapestry of shape $n\_rows \times n\_columns$, you have to place n_shapes shapes on the chessboard such as each shape is present exactly once per row and column. You also have to give a color among n_colors colors to each shape in the tapestry; once again each color needs to present just one time in each row and column. But there is a last constraint : each combination of shape and color must be unique in the entire tapestry : as an example, if a given cell contains a green circle, each other circle in the tapestry must have a different color and each other green shape can't be a circle. A Tapestry problem uses n_shapes shapes and n_colors colors such that $n\_rows = n\_columns = n\_shapes = n\_colors$. The Tapestry Problem asks whether such a placement exists. Figures 3 below shows an example of a valid solution (A) and an invalid solution (B).

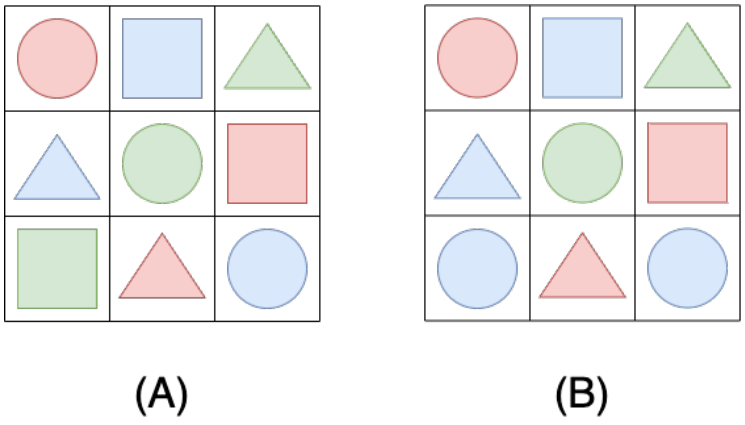

Figure 3: Example of Tapestry Problem solutions. Solution (A) is a valid solution that fit all the rules of the problem, and (B) is an invalid solution as it breaks the three rules : There is a blue circle in two different cells, there is two circle in the first column and the last row and the color blue is present twice in the first column and the last row.

Your task is to model this problem with propositional logic. We define $n\_rows \times n\_columns \times n\_shapes \times n\_colors$ variables: C_{ijab} is *true* iff there is a shape a of color b

at position (i, j); *false* otherwise. The chessboard origin $(0, 0)$ is at left top corner.

Questions

1. **(2 pts)** Explain how you can express this problem with propositional logic. For each sentence, give its meaning.
2. **(2 pts)** Translate your model into Conjunctive Normal Form (CNF).

On the Moodle site, you will find an archive `tapestry.zip` containing the following files:

- `instances` is a directory containing a few instances for you to test this problem.
- `solve_linux.py / solve_mac.py` is a python file used to solve the tapestry Problem, whether you machine is running on Linux or Mac.
- `tapestry_solver.py` is the skeleton of a Python module to formulate the problem into a CNF.
- `minisat.py` is a simple Python module to interact with MiniSat.
- `clause.py` is a simple Python module to represent your clauses.
- `minisatLinux / minisatMac` is a pre-compiled MiniSat binary that should run on the machines in the computer labs or your machine depending on the os.

To solve the tapestry Problem for instance on Linux machine, enter the following command in a terminal:

```
python3 solve_linux.py INSTANCE_FILE
```

where `INSTANCE_FILE` is the tapestry instance file. To have different versions of a problem of specific size, the instance file is not only composed of the size of the problem but some cells already has given symbol and color. It is described as follows:

Instance input format

n k
$c_1.i$ $c_1.j$ $c_1.a$ $c_1.b$
$c_2.i$ $c_2.j$ $c_2.a$ $c_2.b$
...
$c_k.i$ $c_k.j$ $c_k.a$ $c_k.b$

where n represents the value of the three (03) parameters of the problem shape since the chessboard is square. k is the number of cell's information provided. The k following lines represent information about the fixed cells and are composed of four integers representing respectively values of y axis, x axis, shape and color, all starting from 0.

Questions

3. **(2 pts)** Modify the function `get_expression(size)` in `tapestry_solver.py` such that it outputs a list of clauses modeling the tapestry problem for the given input. Submit your code on INGInious inside the *Assignment4: Tapestry Problem* task. The file `tapestry_solver.py` is the *only* file that you need to modify to solve

this problem. Your program will be evaluated on 10 instances of which 5 are hidden. We expect you to solve all the instances.