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

Gael Aglin, Alexander Gerniers, Yves Deville
November 2019

Guidelines

- This assignment is due on **Wednesday 11 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*.
- 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 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 on *INGInious* inside the *Assignment 4* task(s):
  - `binpacking_maxvalue.py`: your *maxvalue* local search for Knapsack problem, to submit on INGInious in the *Assignment4: Bin packing problem : maxvalue* task.
  - `binpacking_randomized_maxvalue.py`: your *randomized maxvalue* local search for Knapsack problem, to submit on INGInious in the *Assignment4: Bin packing randomized maxvalue* task.
  - `cgp_solver.py`: which contains `get_expression` method to solve the Color Grid problem, to submit on INGInious in the *Assignment4: Color Grid Problem* task.

1 The bin packing problem (13 pts)

In this assignment you will design an algorithm to solve the infamous bin packing problem. You are provided with a set of items, each having a specific size to pack into a certain number of bins having a maximal capacity. Your task is to find a good configuration of packing such that the number of bins needed is minimal. Figure ?? shows an example of a solution for bin packing problem.

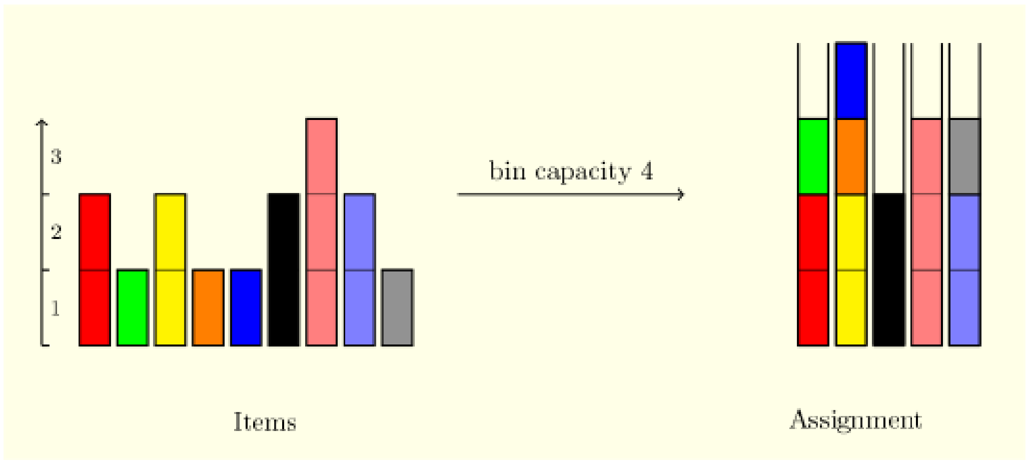

Figure 1: Example of a bin packing problem: Here it took 4 bins of capacity 4 to pack the 9 nine items
Source: https://scip.zib.de/doc/html/binpacking.png

Formally, the bin packing problem is defined as follows[1] : given a finite set $U = u_1, u_2, \ldots, u_n$ of items and a rational size $s(u) \in [0, 1]$ for each item $u \in U$, find a partition of U into disjoint subsets $U_1, U_2, \ldots, U_k$ such that the sum of the item sizes in each U_i is no more than 1 and k is as small as possible. Without loss of generality, the bin packing problem is considered to be the problem of packing a finite set of items of integer sizes $s(u_i) \in [0, C]$ into bins of capacity C to minimize the number of bins.

Objective function A solution to bin packing problem can be evaluated by using different functions provided in the literature, each with its particularity. The one provided here is called $Fitness$.[2]

$$Fitness = 1 - \left(\frac{\sum_{i=1}^{n} (fullness_i / C)^2}{k} \right)$$

where k = number of bins, $fullness_i$ = sum of all the pieces in bin i, and C = bin capacity. The function puts a premium on bins that are filled completely, or nearly so. It returns a value between zero and one, where lower is better, and a set of completely full bins would return a value of zero. In this case, the objective would be to minimize this function.

[1] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 29. wh freeman New York, 2002.

[2] Matthew Hyde et al. "A hyflex module for the one dimensional bin-packing problem". In: *School of Computer Science, University of Nottingham, Tech. Rep* (2009).

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 here two types of actions to build your neighborhood all based on swap moves. **The constraint of bins' capacity cannot be violated during building of the neighborhood.**

Swap two items This kind of action allows you to change positions of items into bins, then you modify the fullness of involving bins but without reducing the number of bins used for the solution. Figure 2 shows an example of this action.

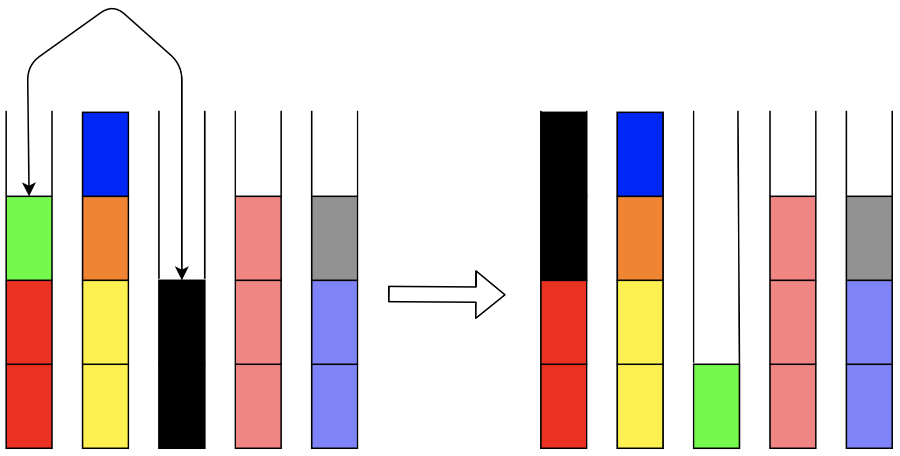

Figure 2: Swap of two items

Swap an item and a blank space Contrary to the previous action. This one can allow you to reduce the number of bins used from a configuration to another one since it allows to swap an item and a blank space : it is a kind of *move*. Figure 3 shows an example of this situation.

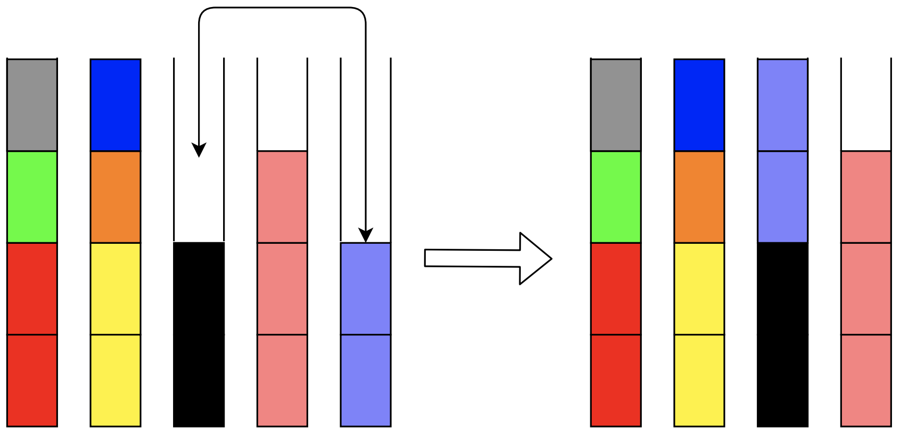

Figure 3: Swap of an item and a blank space (Move of an item)

You are not obliged to use the objective function as well as techniques for building the neighborhood. You can use your owns but you have to specify in you report the ones you use.

A bin packing input contains $n+1$ lines. The first line contains an integer which represent the number of items in the problem, n and the maximal capacity C of bins. The n following lines present the data for each of the items. Each line contains two integers. The first one is item's index i and the second is its size $s(u_i)$.

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

Input format

n	C
1	$s(u_1)$
2	$s(u_2)$
...	
n	$s(u_n)$

For this assignment, you will use *Local Search* to find good solutions to the bin packing 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

$U_1 = \{\cup u_i \mid u_i \in U\}$
...
$U_k = \{\cup u_i \mid u_i \in U\}$

Where k is total number of bins used to pack all the items.

For example, the input and an output for the bin packing problem of fig 1 could be:

Input format

```
9 4
1 2
2 1
3 2
4 1
5 1
6 2
7 3
8 2
9 1
```

Output format

```
1 2 9
8 6
7
4 5 3
```

1) Minimiser k tq tous les objets rentrent dans k trucs
2)celle donnée juste avant
3) swap/move ne dépasse pas le cylindre
4) faire n'importe quel move/swap augmente k ou le laisse pareil ?

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 pts)** Formulate the bin packing problem as a Local Search problem (problem, cost function, feasible solutions, optimal solutions).

2. **(5 pts)** You are given a template on Moodle: *binpacking.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*.

   (a) `maxvalue` chooses the best node (i.e., the node with minimum 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,callback=None)*.

   (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,callback=None)*.

   Describe in your report how you construct your initial solution and how your successor function works.

3. **(4 pts)** Compare the 2 strategies implemented in the previous question between each other and with `randomwalk`, defined in *search.py* on the given bin packing 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. 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:

- $(\neg A \lor C) \land (\neg B \lor C)$
- $(C \Rightarrow \neg A) \land \neg(B \lor C)$
- $(\neg A \lor B) \land \neg(B \Rightarrow \neg C) \land \neg(\neg D \Rightarrow A)$

Questions

1. **(1 pts)** 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 Color Grid Problem (6 pts)

The Color Grid Problem can be defined as follow. Given an square grid of shape $n\_rows \times n\_columns$, a grid coloring assigns a color to each cell, such that all cell on the same line (vertical, horizontal, diagonal) [redacted] erent color. A color grid using n_colors colors such that $n\_rows = n\_columns =$ The Color Grid Problem asks whether such a grid exists. Figures 4 below shows ple of a valid coloring (left) and an invalid coloring (right).

HELLO BEAUTIFUL
1,3
0,2
2,4

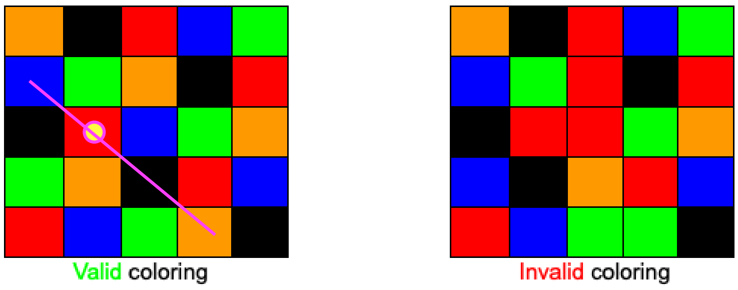

Figure 4: Example of graph coloring

Your task is to model this problem with propositional logic. We define $n\_rows \times n\_columns \times n\_colors$ variables: C_{ijk} is *true* iff cell at position (i, j) is assigned color k; *false* otherwise. The grid origin $(0, 0)$ is at left top corner.

Questions

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

On the Moodle site, you will find a zip `cgp.zip` containing the following files:

- `given_instances` are a few instances for you to test this problem.

- `solve.py` is a python file used to solve the color grid Problem.
- `cgp_solver.py` is the skeleton of a Python module to formulate the problem into an CNF.
- `minisat.py` is a simple Python module to interact with MiniSat.
- `clause.py` is a simple Python module to represent your clauses.
- `minisatLinux` is a pre-compiled MiniSat binary that should run on the machines in the computer labs and your machine if you have Linux.

To solve the Color Grid Problem, enter the following command in a terminal:

```
python3 solve.py INSTANCE_FILE
```

where `INSTANCE_FILE` is the color grid 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 colors of the grid are provided. It is described as follows:

Instance input format

$size$	n	
$p_1.i$	$p_1.j$	$p_1.k$
$p_2.i$	$p_2.j$	$p_2.k$
$\dots$		
$p_n.i$	$p_n.j$	$p_n.k$

where *size* represents the value of the three (03) parameters of the problem shape since the grid is square. n is the number of cells' information provided. The n following lines is composed of tree integers representing respectively values of x axis, y axis and color index, all starting from 0.

Example of input

```
5  3
0  0  2
3  2  3
4  1  0
```

Questions

3. **(2 pts)** Modify the function `get_expression(size)` in `cgp_solver.py` such that it outputs a list of clauses modeling the color grid problem for the given input. Submit your functions on INGInious inside the *Assignment4: Color Grid Problem* task.

   The file `cgp_solver.py` is the *only* file that you need to modify to solve this problem.