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

Jean-Baptiste Mairy, Cyrille Dejemeppe, Yves Deville
November 2013

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. Do not let the instructor discover it.
- Copying code 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 via the *SVN* tool. Only programs submitted via this procedure will be corrected. No report or program sent by email will be corrected.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as we will use fully-automated tests. When we provide a way to test your program, it *must* pass this test.

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 inside the `milestone4` directory of your SVN repository:
  - `knapsack.py`: the knapsack problem implementation (Section section 1)
  - `randomwalk.py`: knapsack algorithm of section 1.1
  - `maxvalue.py`: knapsack algorithm of section 1.1
  - `randomized_maxvalue.py`: knapsack algorithm of section 1.1
  - `play.py`: program for the package installation problem (Section section 2.2)
  - `Level_666.equipments`: the solution to the Level 666 equipment problem (Section section 2.2)

1 The Knapsack Problem (13 pts)

The knapsack problem, illustrated on Figure 1, is the following problem. A finite set of objects is available. Each object has a weight and a utility. The objective is to find the set of items maximizing the total utility without exceeding a given weight called the capacity. This problem has a lot of applications. For instance, in finance, for the selection of investments and portfolios, a refined version of the knapsack problem can be used. Another example is Santa Claus. Each year, he is able to select the presents maximizing the joy of the children without exceeding the weight capacity of his sleigh.

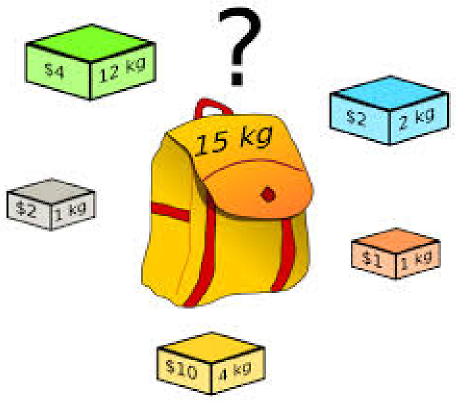

Figure 1: Illustration of the knapsack problem

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

File format

n

1	w_1	u_1
2	w_2	u_2
...		
n	w_n	u_n

C

In the instance files, n is the number of items, then for each item i, there is one line "$i \quad w_i \quad u_i$" where w_i is the weight of i and u_i its utility. Finally, C is the capacity. The goal is to find the subset S of items from $\{1, \dots, n\}$, maximizing $\sum_{i \in S} u_i$ such that $\sum_{i \in S} w_i \leq C$

For this assignment, you will use *Local Search* to find good solutions to the knapsack problem. In order to help you, some algorithms are already implemented in the `aima-python3` module on iCampus (still under Document > Assignments). The test instances can also be found on icampus under Document > Assignments > Assignment4.

Warning: the tests that you have to perform on the given instances require time. Do not wait the last minute to design and run them.

1.1 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 Knapsack problem as a Local Search problem. Implement, in a file named `knapsack.py`, your own extension of the *Problem* class from `aima-python3`. For this assignment, the moves considered in the algorithms are: add one item, remove one item and replace one item by one another. Of course, none of the moves returned by your successor function can violate the capacity constraint.
2. Implement the following strategies, each in their own file:
   (a) `randomwalk.py` chooses the next node in the neighborhood randomly (you can use the `random_walk` function for that),
   (b) `maxvalue.py` chooses the best node (i.e., the node with maximum value) in the neighborhood, even if it degrades the current solution,
   (c) `randomized_maxvalue.py` chooses the next node randomly among the 5 best neighbors (again, even if it degrades the current solution).

   To construct the initial solution, use a greedy algorithm maximizing the number of selected items: add the items in increasing weight order until no item can be added without violating the capacity constraint. The search shall stop after 100 steps and return the *best* solution found (which may be different from the last one). Each file is a Python program that takes the path to the instance to solve as only argument.
3. Compare the 3 strategies on the given knapsack instances. Report in a table the results of the tests. Interesting metrics to report are: 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). A good way to eliminate the effect of the randomness of some of the strategies is to run the computation multiple times and take the mean value of the runs. For the first and the third strategy, each instance should be tested 10 times.
4. 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 (2 pts)

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

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

Questions

1. For each sentence, give the number of models that satisfy it (considering the proposition variable A, B, C and D).

2.2 RPG Equipment Problem (5 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 three-way conflict. A conflict (equ_1, equ_2, equ_3) is such that buying two (or less) of these equipment pieces is allowed (buying (equ_1, equ_2), (equ_1, equ_3), (equ_2, equ_3), (equ_1), (equ_2) or (equ_3) is allowed) but you cannot buy the three (buying (equ_1, equ_2, equ_3) is not allowed).

Figure 2: Subset of available helmets

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 anymore (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 an 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, E_2 and E_3 are in conflict, the player cannot buy the three pieces together. However, the player is allowed to buy any subset of these three 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, Ice Brigand
Hood

...

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

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`.

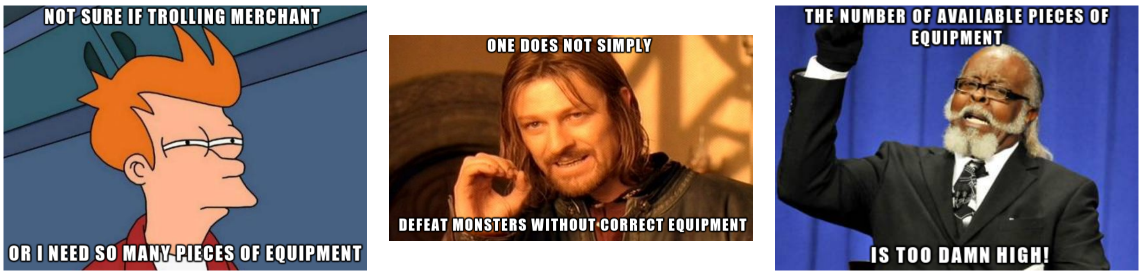
NOT SURE IF TROLLING MERCHANT
OR I NEED SO MANY PIECES OF EQUIPMENT
ONE DOES NOT SIMPLY
DEFEAT MONSTERS WITHOUT CORRECT EQUIPMENT
THE NUMBER OF AVAILABLE PIECES OF EQUIPMENT
IS TOO DAMN HIGH!

Figure 3: Reactions of the players not using propositional logic

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).

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_005.gz`, `Level_050.gz`, `Level_250.gz`, `Level_666.gz` are representations of levels containing respectively 5, 50, 250 and 666 enemies
- `rpg.py` is a Python module to load and manipulate the merchant and level files described above
- `play.py` is the skeletton 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.

Questions

3. Modify play.py which takes as first argument the merchant's filename (e.g., Merchant.gz) and as second argument the level's filename (e.g., Level_005.gz, Level_050.gz, Level_250.gz or Level_666.gz). The program should print the pieces of equipment that need to be purchased and the total number of purchases to the merchant. You only have to modify the #TODO part of the file.
4. What is the output of your program when simulating the level Level_05.gz with the merchant Merchant.gz? How many variables and how many clauses did you generate to get this result (this should appear in the output of the minisat program which is displayed in the output of play.py)?
5. Report in a table the number of clauses, variables and the number of equipment pieces needed when simulating the levels Level_005.gz, Level_050.gz, Level_250.gz and Level_666.gz with Merchant.gz. How does the number of clauses, variables and pieces of equipment needed evolve with the size of the level? The number of a level represents the number of enemies it contains (e.g. Level_005.gz contains 5 enemies).
6. Report in a file Level_666.equipments the solution when simulating the Level_666.gz with Merchant.py. Your solution file should list the pieces of equipment needed, one piece of equipment per line (only the name of a single piece of equipment should be printed per line and no additional character except the '\n' at the end of the line).