CS 170 Efficient Algorithms and Intractable Problems
Spring 2017 Prasad Raghavendra and Sanjam Garg

Final Project

Release date: April 7, 2017.

Summary

We present an NP-hard problem PICKITEMS for which you will design and implement an algorithm for heuristic solutions (since it's NP-hard, your algorithm doesn't need to be optimal). You must:

Phase I (due April 18): Submit three difficult input files. We will collect the resulting pool of student-submitted and staff-submitted input files, and release all of them publicly shortly after the due date.

Phase II (due May 2): Design an algorithm and run it on the entire pool of input files. Submit your output files along with your code and a one-page writeup explaining your team's strategy at a high level.

Grading will be assessed by a combination of your algorithm's performance, your writeup, and the difficulty of your input files (see section on grading below for more details). All due dates are at 11:59 PM, PST.

Gradewise, this project alone is worth two homeworks and cannot be dropped.

You may either work solo or with a partner.

Problem

You run the Prasad Corporation's merchanting business. Every week, you buy many items from the local Mart170, and re-sell everything online. One of your greatest business assets is how you prevent delivery fees – every week, you send Prasad to fill up a GargSack, along with an exact list of what items to buy.

Thanks to online sources, we have all the information we need about every item in stock at Mart170, including how much we can resell it for. It comes in the following format:

```
[P pounds]
[M dollars]
[N number of items]
[C number of constraints]
[item_name]; [class]; [weight]; [cost]; [resale value]
[item_name]; [class]; [weight]; [cost]; [resale value]
...
[incompatible_class1, incompatible_class2, incompatible_class3]
[incompatible_class4, incompatible_class5]
...
[end of file]
```

Unfortunately, there are multiple constraints on what to buy:

- Prasad cannot carry more than P pounds total (although, thankfully, the GargSack is imbued with the magic of infinite volume). This is specified in the first line of the input.
- Prasad brings exactly M dollars. This is to bound the worse case scenario of a robbery. This amount is specified as the second line of the input.
- Each item has a class to which it belongs, indexed by an integer between 0 and $N-1$. There are constraints on these classes of items: certain classes cannot be placed in the GargSack together, for they are incompatible. For example, suppose an item ”wood” belongs to class 1, and an item ”termites” belongs to class 2, and class 1 is specified to be incompatible with class 2. Putting *any* two items of incompatible types in the GargPack will result in the destruction of the GargSack, its contents, and anything within a 4-ft radius.

Here are some more detailed specifications on the input file:

- $P < 2^{32}$ will be specified up to 2 decimal places.
- $M < 2^{32}$ will be specified up to 2 decimal places.
- N will be a whole number, up to 200,000.
- C will be a whole number, up to 200,000.
- The item name must be unique and alphanumeric (underscores are also allowed).
- The class is specified as a number between 0 and $N-1$.
- The weight will be specified up to 2 decimal places.
- The cost will be specified up to 2 decimal places (in other words, rounded to the nearest cent).
- The resale value will be specified up to 2 decimal places (in other words, rounded to the nearest cent).
- Each incompatibility list will be separated by commas. These lists may have as many as 199,999 elements.
- No class can be incompatible with itself. This is to make the constraint definition as unambiguous as possible.
- The size of the input file may be no larger than 4 megabytes (MB).

Given all the information of what is in stock, the PICKITEMS output should be a list of items to buy:

```
item_name
item_name
item_name
item_name
```

Your goal is to maximize the total resale value sum under the given constraints. Any unspent cash is accounted for by directly adding it to the resale value sum. If you overspend your cash or buy items that weigh more than the maximum capacity of the GargSack, you receive zero points.

Grading

To reiterate: as a component of your final grade, the project alone is worth two homeworks and cannot be dropped. Your grade will be shared with your partner.

The breakdown of your grade will be:

- 60% algorithm: how much better did your outputs do compared to other submissions?
- 10% test cases: how difficult were the test cases you submitted? [measured by std deviation of scores]
- 30% writeup: how well can you communicate your strategy?

Phase I: Input files (due April 18)

Ideally, these files should be easy for your team to solve, but difficult for other teams. Recall that the input files will be pooled together into the evaluation set for Phase II.

Please submit to the Gradescope assignment *as a team of you and your partner*. You should drag and drop to upload three files with the following names:

```
problem1.in
problem2.in
problem3.in
```

Make sure that your submission matches these names exactly. Our auto-grader will check for any errors in the input files you submit. You will receive 1 pt if you submitted three valid files.

Phase II: Code, output files, writeup (due May 2)

After all input files have been submitted, we will release the entire pool in a zip file for you to run your algorithm on. We currently expect to receive around 1000 input files in the pool.

We have released starter code in the file `solver.py`, posted on Piazza. You do not have to use it; you may use any programming language you wish. However, your code must be able to run on only one machine and complete in a reasonable amount of time. If you use non-standard libraries, you need to explicitly cite which you used, and explain why you chose to use those libraries.

To make a submission, you need to include the following in a zip folder and drag and drop into Gradescope:

```
team_name.txt
code/
    [the code you want evaluated here]
writeup/
    [the writeup you want evaluated here]
output/
    problem1.out
    problem2.out
    ...
```

The auto-grader will score your output files immediately (it may take a while to run). We will maintain a live ladder showing how well your solutions perform compared to the rest of the class. The optional file `team_name.txt` should contain a single 80-character line which will be used to display your team's results (if not included, your team will be anonymous).

For your writeup, you need to clearly describe the strategy you are using in Phase II, as well as how you came up with difficult instances in Phase I. We encourage you to analyze its asymptotic running time, as well as any lower or upper bounds on performance.

Please make sure you *include your partner in your Gradescope submission*, and also include all your names and SIDs in your writeup.

We *strongly suggest* you start early – in particular, your code should be completed by no later than **April 30**. For a pool of 1000 input files at an average of 2 minutes per file, it would take **over 30 hours** to run!

Please note that we will NOT accept any late submissions.

Appendix: Example input and output files

Below is an example input file:

```
100
100
2
1
heavy_worthless_rock; 0; 99; 1; 5
gem_on_sale; 1; 2; 50; 200
0, 1
```

And the output file corresponding to its optimal solution is:

```
gem_on_sale
```