Homework 3
Integer Programming and Combinatorial Auctions

CMU 15-780: Graduate AI (Spring 2016)
OUT: Feb 17, 2016
DUE: Feb 28, 2016 11:59pm

Instructions

Collaboration/Academic Policy

You may discuss assignments with other students as you work through them, but writeups must be done alone. NO DOWNLOADING OR COPYING OF CODE OR OTHER ANSWERS IS ALLOWED. If you use a string of at least 5 words from some source, you must cite the source; however you cannot just copy the solution from some source but instead you should write it up in your own words. If you want to use a 3rd party python library, please let us know before Friday, Feb 19, 2016 so that we can ensure it is available to the autograder on autolab.

Submission

Please create a tar archive of your answers and submit to Homework 3 on autolab. You should have two files in your archive: a completed `problems.py` for the programming portion, and a PDF for your answers to the written component. Your completed functions will be autograded by running through several test cases and their return values will be compared to the reference implementation. There is a `sample.py` that contains sample inputs and outputs for reference. Please put your Andrew ID somewhere on the first page of your written answers.

You have 8 late days for homeworks over the semester, and can use at most 3 for one homework.

TAs

If you need help, the names beside the questions are the names of the TAs who came up with them, and are more likely to be familiar with the topics.

1 Written: Integer Programming and SAT [Guillermo; 30 points]

We deal with conjunctive normal form (CNF) formulas here, as in homework 1. The n variables are labeled $x_0, \ldots, x_{n-1}$ like before. Here is an example 3 variable formula:

$$x_0 \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1 \vee x_2)\,.$$

1. [20 pts] Reformulate SAT as an Integer Program. Specifically, state the variables, constraints and objective. Show that if there is a satisfying assignment for the SAT then there is a solution to the IP, and vice versa.

2. [10 pts] To solve SAT, We can use the LP relaxation of the integer program in a similar fashion as BCP. We take the solution of that program and assign all the variables that are integers. Then we will use the solution to pick a new variable to branch on. Let's compare running the relaxed LP and BCP given the same formula and the same partial assignment. Will BCP assign more variables than the relaxed LP or less? Or does it depend? Justify.

2 Programming: Combinatorial Auctions [Guillermo/Daniel; 70 points]

You will be implementing parts of a winner determination algorithm for combinatorial auctions as shown in class. Recall that a combinatorial auction is made up of n items, and there are m bids. Each bid is for a subset of the items, at a particular price. The winner determination algorithm will find a set of non-overlapping bids with the highest total price.

In python, the n items of an auction are represented by their indices: `[0, 1, ..., n-1]`. Each bid is a tuple with a list of items and a price. For example, `([0,4], 10)` is the bid that is for items 0 and 4 with price 10. An instance of an auction is then represented by a tuple with the number of items and a list of bids. For example, `(5, [([0,4], 10), ([1], 4)])` is an auction with 5 items and two bids; the bids are $[0, 4]$ for a price of 10 and [1] for a price of 4. Empty bids i.e. bids with no items are not allowed. Each bid can contain at most one copy of an item. Also, no two bids will be for the exact same subset of items. Auctions do not require that all items are bid on. Bid prices are positive integers.

The algorithm we will use is Branch by Bid with A*. A node in the search tree is represented by which bids have been considered along the path to get there. To generate the successors, we must first pick the next bid to branch on; then the successors of a node are the two branches of taking or not taking the next bid. See Figure 2 for an example.

The $g(...)$ value for A* will be the sum of the prices of the bids that we have already taken. The $h(...)$ value should be an upper bound (because we are maximizing) on the best total price that we can achieve from the remaining bids not considered yet.

You will not be coding A*, but will only code up parts of it. To help test your code, you can use the functions and code in `sample.py`, which include the A* search and a simple random auction generator. See the code for more details.

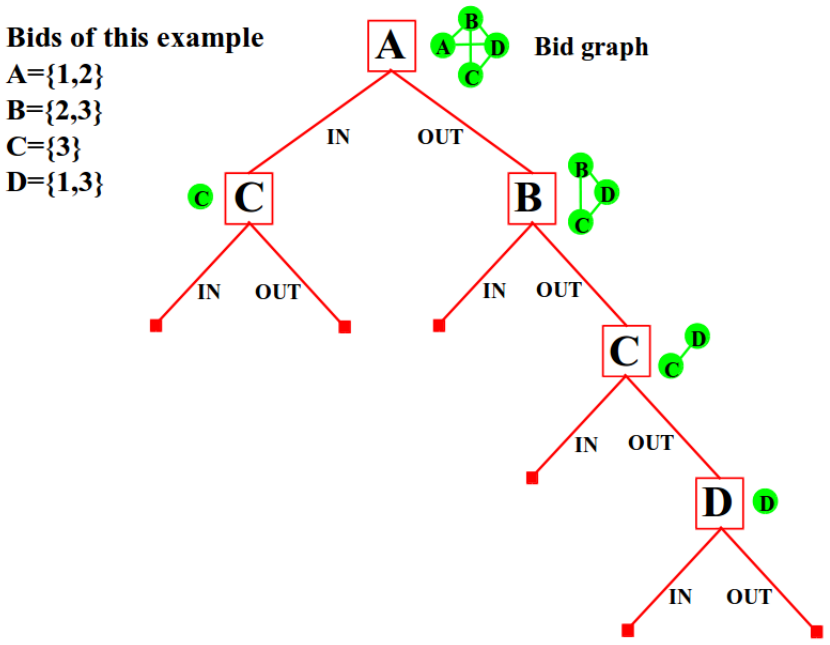

Figure 1: For this example (taken from slides). Suppose there are only 3 items. Suppose we index the bids in order such that 0=A, 1=B, 2=C, 3=D. Since the prices aren't given, let's set them to be A=3, B=2, C=1, D=1. Then we can represent this auction by `(3, [([0,1], 3), ([1,2], 2), ([2], 1), ([0,2], 1)])` The node in the search tree labeled with D would be represented by `[(0, False), (1, False), (2, False)]`. The successors of that node would branch on the bid D, resulting in two new nodes `[(0, False), (1, False), (2, False), (3, True)]` and `[(0, False), (1, False), (2, False), (3, False)]`

.

2.1 Variable Ordering [10 pts]

You will be implementing a function that picks the next bid to branch on given the current node in the search tree i.e. determining the successors of a node. You should pick the next bid that hasn't been considered yet with the lowest index, and does not overlap with any of the bids already taken.

You will implement this in the `get_next_bid(...)` function in problems.py. The autograder will check that you pick the correct next bid.

2.2 LP Heuristic [30 pts]

You will be implementing the heuristic function $h(...)$ which solves the LP relaxation of the IP of winner determination. You will also be hot-starting the LP programs of successor nodes with the dual solution of the parent node (as described in class). You should use your simplex solvers from Homework 2 to solve the LPs, and specifically your `add_constraint` function to hot-start. If you had trouble with Homework 2, you can use the reference solution which will be released 3 days after the deadline of Homework 2 (to account for grace days).

Here are some suggested guidelines for constructing your LP: your LP should have m basic variables corresponding to the m bids; you should have at most n constraints corresponding to the n items (to make sure no item is in more than one bid taken); if an item is not included by any bids, you don't need a constraint. Since these are inequality constraints, you will need to introduce a new slack variable for every constraint

to turn them into equality constraints. See the slide with title 'Linear program of the winner determination problem' and related slides for reference. At the start node (the root node), you will need to solve the LP fully, most likely using your primal simplex algorithm. Note that there is a simple primal feasible index set which concerns the slack variables that you can use as an initial point.

Note that for a search node, the solution to your LP will probably be an upper bound on the total price (i.e. f-value). Your heuristic function should return an upper bound on the value of the remaining bids that have not yet been considered (i.e. h-value).

You will implement this in the `auction_heuristic(...)` function in problems.py. The autograder will check that your heuristic value is close enough to the reference solution.

2.3 Gomory [20 pts]

You will be finding Gomory cuts to the LP relaxation of the IP of winner determination.

The first line in the slide titled "Derivation of Gomory mixed integer cut" is derived from looking at a row of the final Simplex tableau. Let x be the non-slack variables and v be all the variables (non-slack and slack). Let I be the index set. Assume the constraints are $Av = b$ and $v \geq 0$. The final tableau is $A_I^{-1}Av = A_I^{-1}b$. Going back to the slides, the term $a_{i,0} = \left(A_I^{-1}b\right)_k$ and the terms $a_{i,j} = \left(A_I^{-1}A\right)_{k,j}$ where $i = I_k$.

When we pick the row we want to look at the smallest index $i \in I$ such that v_i is a fractional, basic, non-slack, integer variable. If there is no such i, output None.

Assume that the constraints we give use integer coefficients.

You will implement this in the `gomory_cut(...)` function in problems.py. The autograder will check that cut is close enough to the reference solution.

2.4 Variable Ordering Better [10 pts]

You will be coming up with your own rule for picking the next bid to branch on in the search tree for Branch by Bid. Be creative.

You will implement this in the `get_next_bid_better(...)` function in problems.py. The autograder will measure your performance by how many fewer nodes are expanded by A* compared with the simple variable ordering rule above (baseline). We will use your LP relaxation as the h-value. This will be entered into a leaderboard, and as long as you do better than the baseline, you will receive 5 points. The other 5 points will be a linear function of your placement on the leaderboard and will be computed after the deadline.