Monday, March 6, 2023
Lecture #21
MSSC 6000

Announcements

* HW 3 due Wednesday 11:59pm
* Midterm Exam, Wednesday in class
  (up to backtracking, no branch + bound)
* Friday, office hours 10am-10:50am in my office,
  no lecture
* Normal Office Hours this week, today 1pm-2pm
  in my office

Topic 8 - Branch and Bound

General Procedure:

search space

returns best sol in S OR best_sol, whichever is best (assume maximizing)

```
function bb(S, best_sol = None):
    if best_sol is None:
        best_score = -∞
    else:
        best_score = score(best_sol)

    if |S| = 1:        (no more branching, we're at a complete solution)
        candidate = the one thing in S
        value = score(candidate)
        if value > best_score:
            return candidate
        else:
            return best_sol

    S1, S2 = branch(S)        (could be more than 2)
    if bound(S1) > best_score:        (we have a chance of improving in S1)
        best_sol = bb(S1, best_sol)
        best_score = score(best_sol)
    if bound(S2) > best_score:
```

```
    best_sol = bb(S2, best_sol)
    best_score = score(best_sol)
return best_sol
```

If the # of branches varies (like Job Assignment Problem) you can do the last part in a loop.

```
branches = branch(S)
for branch in branches:
    if bound(branch) > best_score:
        best_sol = bb(branch, best_sol)
        best_score = score(best_sol)
```

Relaxation

Let's try to do B+B on the Knapsack prob.

items	weight	value
1	8	13
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

Capacity: 14

Need two things:
- Branching
- Bounding

Branching – same as with backtracking
Item 1 is in or out
Item 2 is in or out
⋮

items	weight	value
~~1~~	~~8~~	~~13~~
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

Bounding: Suppose we have put item 1 out and item 2 in. How can we find an upper bound on the best we could possibly do completing this solution?

Notes: * Greedy solutions are lower bounds.
* "Add up the values of all remaining items" is an upper bound but a pretty useless one (weak)
* We want our upper bound to be fast to compute.

The trick is relaxation: sometimes it's

easier to find an UB if you adjust the problem to be more permissive.

items	weight	value		
1	8	13	0.5	4/6.5
2	3	7	1	3/7
3	5	10	1	5/10
4	5	10	0.4	2/4
5	2	1	0	0
6	2	1	0	0
7	2	1	0	0
				14/27.5

Fractional Knapsack: You are allowed to take fractions of an item.

Theorem: An optimal (and greedy) solution to the Fractional Knapsack problem can be found by:

(1) order the items by $\frac{\text{value}}{\text{weight}}$

(2) take items from the top in full until you can't anymore

(3) take whatever fraction you can of the next item

We won't prove this, but you should think about it until you believe it.

Capacity = 14

items	weight	value	density			
1	8	13	1.625	4	12.5%	13/8
2	3	7	2.333	1	100%	7
3	5	10	2	2	100%	10
4	5	10	2	3	100%	10
5	2	1	0.5	5		
6	2	1	0.5	6		
7	2	1	0.5	7		

$\boxed{28.625}$

(If capacity = 10, you get 21 which beats the 20 for regular knapsack)

So Fractional Greedy = Fractional Optimal
≥ Regular Optimal

We can therefore get an UB for the regular knapsack by computing the greedy fractional solution on whatever items remain.

capacity = 10

items	weight	value	
~~1~~	~~8~~	~~13~~	
2	3	7	✓
3	5	10	✓
4	5	10	40%
5	2	1	
6	2	1	
7	2	1	

Partial solution:

Item 1 out

Item 2 in

capacity remaining = ~~7~~ ~~2~~ 0

value: 7 + 10 + 4 = 21

three different greedy runs

Greedy search: 14, 10, 18

best_score = ~~18~~ 20

(1) (2) (3) (4)