# 9- 1 Lab: Approximations for hard problems

1. Consider the set of boolean expressions that consist of **or-clauses** (expressions with only boolean variables connected by ∨, and then these clauses connected by **“and”** ∧’s. Expressions of this types are said to be in **conjunctive normal form.**  A simple example with only one clause is (x ∨ y). As long as either x or y is true, then the expression would be true. Consider the example (w) ∧ ( ∨ ). Any assignment where z is true can be eliminated from consideration. It is an important problem (SAT) is to determine if such an expression can evaluate to **true** for some assignment of true / false values to the boolean variables x, y, z, w and their negations. For another example:

(w ∨ x∨ y ∨z) ∧ (w ∨ ) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( ∨ )

Use backtracking (where each stage of the state space tree represents an assignment of a truth value to a boolean variable. **Assign the values to variables in order w, x, y, z**. As you assign values, new subproblems will be formed since some clauses will become true and thus can be safely ignored or it may be clear that a clause will be forced to be false in which case there is no need to explore any further down that subtree since you know the entire expression will be false. Why?

Submit a complete picture of your search tree obtained using backtracking and the variable ordering given above.

1. Solve the following the minimum cost assignment of jobs to people using the greedy algorithm that at each step chooses the lowest cost person-job pair of all the remaining jobs that are feasible. The first person-job pair would be **P1-Job2** , then find the lowest cost assignment that does not conflict with any previous assignment.
2. Show the assignments in the order in which they are found by the greedy algorithm.

P1-Job2(2), P3-Job3(4), P4-Job1(6), P2-Job4(8) = 20

1. Write the pseudo code for the algorithm including any preprocessing but not reading in the problem instance.

Assume each cost in the table has a job and person value.

Sorted[] = Sort the costs in increasing order;

Assign[];

Assign[Sorted[1]];

For i = 2 to number of jobs do

Min cost = Find next cost in sorted that does not conflict with Assign[];

Assign[i] = min cost;

Return;

1. What is its complexity as a function of n = number of jobs? Count any preprocessing your algorithm may require. Do not include the computation required to read in the input.

The sorting takes O(nlogn) if we use quick sort. Then, looping through the jobs and finding the overall lowest non conflicting cost would take O(n2). So the overall complexity is O(n2).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Job1** | **Job 2** | **Job 3** | **Job 4** |
| P1 | 4 | 2 | 5 | 7 |
| P2 | 8 | 3 | 10 | 8 |
| P3 | 12 | 5 | 4 | 5 |
| P4 | 6 | 3 | 7 | 14 |

1. Use Branch and Bound to find the minimum cost assignment of jobs to people for the problem in #2 That is, draw the state space tree showing the order in which the nodes are created in the state space tree, Person 1, Person 2, etc. As in the screencast, for each interior node in the tree show the current assignment and the lower bound and for each leaf the assignment and the total cost.

Start

Lb = 2 + 3 + 3 + 4 = 12

P1 -> 1

Lb = 4 + 3 + 3 + 4 = 14

P1 -> 2

Lb = 2 + 4 + 5 + 6 = 17

P1 -> 3

Lb = 5 + 3 + 3 + 5 = 16

P1 -> 4

Lb = 7 + 3 + 3 + 4 = 17

P2 -> 2

Lb = 4 + 3 + 4 + 5 = 16

P2 -> 3

Lb = 4 + 10 + 3 + 3 = 20

P2 -> 4

Lb = 4 + 8 + 3 + 3 = 18

P3 -> 3

Lb = 4 + 3 + 4 + 14 = 25

P3 -> 4

Lb = 4 + 3 + 5 + 7 = 19

Solution

1. Apply a greedy algorithm based on ***greatest (vi / wi ) first*** to solve the Knapsack problem. Capacity of the Knapsack = 70

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Item1** | **Item2** | **Item3** | **Item4** |
| wi | 10 | 30 | 20 | 70 |
| vi | 11 | 30 | 19 | 63 |
| vi / wi | 1.1 | 1.0 | 0.95 | 0.9 |

Greedy Solution: Item1(11), Item2(30), Item3(19) = 60

1. Apply Branch and Bound to solve the **same** Knapsack problem using the bounding function discussed in class and the screencast.

W = 0, v = 0

Ub = 77

W = 10, v = 11

Ub = 71

W = 0, v = 0

Ub = 70

W = 40, v = 41

Ub = 69.5

W = 10, v = 11

Ub = 68

W = 60, v = 60

Ub = 69

W = 40, v = 41

Ub = 68

W = 30, v = 30

Ub = 68

W = 0, v = 0

Ub = 66.5

W = 0, v = 0

Ub = 63

W = 70, v = 63

Ub = 63

W = 130

Not Feasable

W = 60, v = 60

Ub = 60

W = 50, v = 49

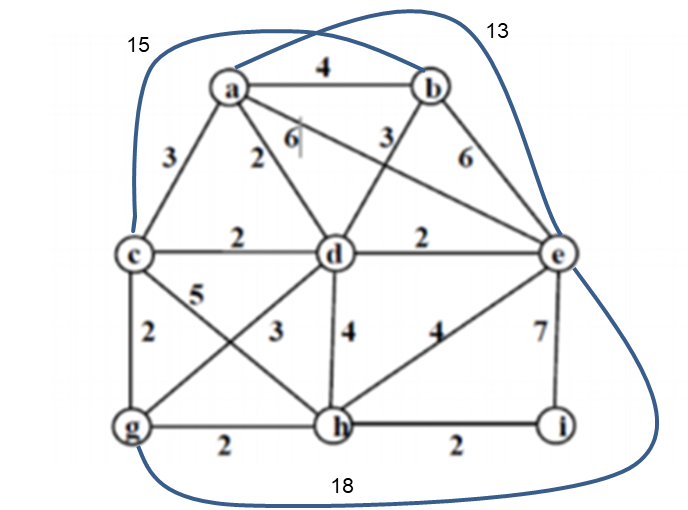
Ub = 67

W = 30, v = 30

Ub = 66

STOP HERE

1. Find an approximate solution to the TSP problem for the following graph using “Twice Around the Tree” starting from vertex a. Why might this not be a 2 approximation?



1. See Das Gupta – Vertex cover approximation using maximal matching