# 9- 1 Lab: Approximations for hard problems

**Do not submit the problems on this page**. Make sure you check the solutions posted since problems similar to these may be on the final exam.

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.

A. Show the assignments in the order in which they are found by the greedy algorithm.

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

C. 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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.
2. 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 |

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

**Submit the following problems: 1-4, 6. Clearly labeling each**

1. Suppose an algorithm, KA is a 2-approximation for the 0-1 Knapsack problem. You have problem instance X and you run KA on X and get a set of items that has a value of 100. What does this tell you about the value of the optimal solution. That is:
   1. What are the range of possible values for the optimal solution?

S\* / 100 ≤ 2 100 ≤ S\* ≤ 200

* 1. Justify you answer using the definition of a 2-approximation.

Since we know that the Knapsack problem is a maximization problem, we know that the accuracy ratio must be less than or equal a c-approximation. In this case, c is equal to 2. Therefore, S\* / 100 ≤ 2, where S\* is the optimal solution. Then by multiplying 100 to both sides, we get that S\* can be at most 2 times the KA solution (S\* ≤ 200).

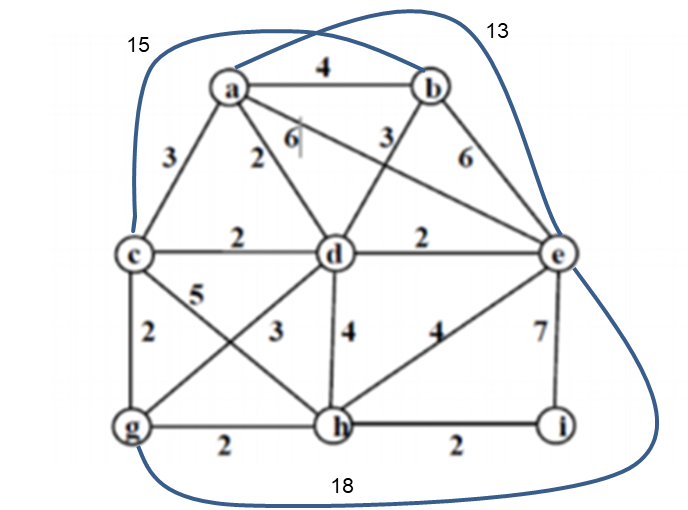
1. Suppose an algorithm, TA is a 2-approximation for the Assignment problem. You have problem instance X and you run TA on X and get a matching of jobs to people that has a cost of 100. What does this tell you about the value of the optimal solution. That is:
   1. What are the range of possible values for the optimal solution?

50 ≤ S\* ≤ 100

* 1. Justify you answer using the definition of a 2-approximation.

Since we know that the Assignment problem is a minimization problem, we know that the accuracy ratio must be less than or equal a c-approximation. In this case, c is equal to 2. Therefore, 100 / S\* ≤ 2, where S\* is the optimal solution. Then by solving for S\*, we get that S\* must be at least half of the TA solution (S\* ≥ 50).

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?



a

b

e

d

c

i

hj

g

The approximate is 39. This is not a metric TSP so it may not be a 2 approximation. The triangle inequality does not hold for this TSP. For example, the triangle formed by vertices a, d, and e has two edges that add up to 4 but this is not greater than the third edge of length 6.

1. Warm up for the oral final: Prove Twice around the tree is a 2 approximation without consulting you notes? Study it now and decide if you have any questions. Work with your group. Take turns presenting the proof. Write it out tonight.

First, we know that the value of the MST must be less than the length of the optimal tour

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, CNF.**  A simple example with only one clause is (x ∨ y). If either x or y is true, then the expression would be true. It is an important problem (SAT) is to determine if such a CNF expression can evaluate to **true** for some assignment of true / false values to the boolean variables x, y, z, w … and their negations. Consider an expression in CNF form that contains a subexpression … (w) ∧ ( ∨ ) … . Any assignment where z is **true** shows that the entire expression will be false and thus z is **true** can be eliminated from consideration.
   1. Why?
   2. Consider: (x∨ y ∨z) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( z ∨ ). What must the values of x, y and z be? Explain why?
2. Given (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 for that subtree.

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