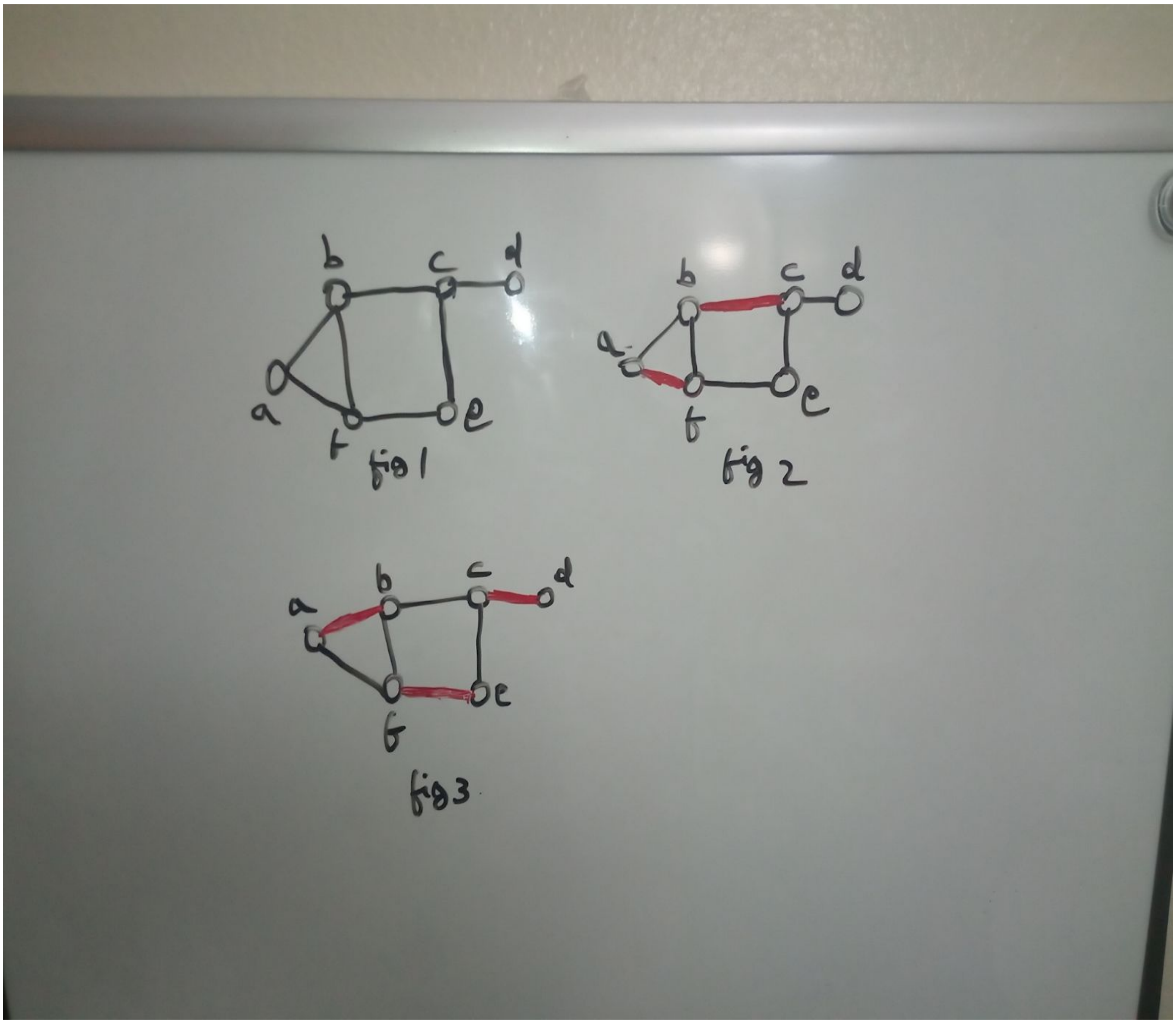
CSCI-5454 HW-5

References - Prof Bo Notes and wikipedia.
Collaboration - Abhilash,Nischal and Aakarsh.

Problem 1:
Part a:



- In the above Image fig.2 is one of the maximal matching because we cannot add anymore edges into the graph as it violates the property that all vertices appear at most once.

- Fig.3 is maximum matching. The maximum number of edges that can be added to the graph without violating above property are 3.

Part b

- Consider a graph G = [V, E]. Greedy approach to find maximal matching is as follows.
- Let M be the set of edges that form a maximal matching. Initialize M = {ϕ}
- For each edge e in E

```
        If e does not intersect any edge in M then
                M = M U {e}
```

- Return M

Part c

- Consider a Graph G=[V, E]. Approximation algorithm to find maximum matching is as follows:
- Let S = {Φ} initially
- Select an edge e(u, v) and add S = S U {e}
- Delete all the edges incident on vertices u and v.
- Repeat till no edges are left in the graph
- return S

Approximation proof:

- Let S^* be the maximum matching. Have to prove that our approximation algorithm is ∝ competitive. Such that |S|>= ∝$|S^*|$ where ∝=0.5
- **Lemma 1** : At the end of the algorithm every edge is removed from the graph. **Proof:** If this were not true then there would still be an edge e(u,v) with both vertices u and v not deleted from E and greedy would have considered it and added to S which is contradictory to our algorithm. Hence statement is true.
- **Lemma 2:** For each edge e added to S we delete at most 2 edges that are part of S^* from graph G. **Proof:** In greedy algorithm above for every edge e (u, v) that is added to S we remove all edges incident on it. S^* is optimal matching so it would have at most one edge with endpoint u and at most one edge with endpoint v. making it at most two edges.
- **Theorem:** The greedy algorithm satisfies the equation |S|>= 0.5*$|S^*|$.
- **Proof:** From Lemma 2 for every edge e(u,v) added to S we remove all incident edges on u and v which deletes at most 2 edges in S^* from graph G. From Lemma 1 we know that we finally end up deleting all the edges. Hence, $|S^*|$<= 2 |S| i.e |S|>= 0.5*$|S^*|$.

Problem 2
Part a
Lemma:
If S is a maximal matching in a graph,then the set of all endpoints of all edges in S is a vertex cover.
Proof by contradiction:

- S is a maximal matching.
- Suppose there is an edge an edge e(u,v) such that endpoints u∉S and v∉S. Since e does not share any endpoint with any of the vertices in S we can have a larger matching such that S=S U {e} but that contradicts our statement that S is maximal matching.
- From our Greedy algorithm for maximal matching we add edge e to S only if doesn’t intercept any edges already in S.
- It means that by the end of algorithm it must be the case that every edge e ε E must have at least one endpoint vertex in S.
- S satisfies the property of vertex cover that every edge e ε E must have at least one endpoint in vertex cover set. Hence, the set of all endpoints of all edges in S is a vertex cover.

Part b

Lemma:

If S is a maximal matching, then every vertex cover must contain at least |S|vertices.

Proof by contradiction:

- We find a maximal matching S with greedy algorithm and construct vertex cover C that consists of all endpoints of the edges in S.
- Suppose an edge e is not covered in C then S U {e} is a matching such that e ∉ S. Then S is not a maximal matching which is contradicting to our assumption.
- Furthermore, if e ε S then any vertex cover including the optimal vertex cover must contain u or v or both,otherwise e is not covered.
- Which means the optimal vertex cover contains at least one endpoint of each edge in S and set C is at most 2 times as large as optimal solution and at least as large as the size of |S|.
- Size of |S| is equal to the number of edges in S which means C has at least one endpoint vertex from each edge. Hence, C >=|S| and C<=c*OPT and c = 2

Part c

- Given graph G= (V, E) following is an approximation algorithm to find vortex cover C for graph G.
- Select any arbitrary edge e(u, v) and add vertices u and v to C.
- Delete all the edges from E that are either incident on u or v.
- Repeat until there are no edges in E.

Part a lemma - we know that if S is a maximal matching then set of endpoints of all edges in S is a vertex cover.
Part b lemma - We know that if S is a maximal matching then size of C is at least as much as size of S. We also know OPT has at least one endpoint of edge in S and C has at most all endpoints from S. Hence, 0.5|C|<=|OPT| or |C|<=2|OPT|

Part d

- Let’s start with W = Φ.
- As a new edge e(u, v) comes in, check if any of the endpoint u or v is already in W.
- If any endpoint of e exists in C discard edge u else add both endpoints u and v to W.
- Delete all the incident edges on either u or v.
- Repeat until there are no new incoming edges.
- Return W

Part a lemma - we know that if S is a maximal matching then set of endpoints of all edges in S is a vertex cover.

Part b lemma - We know that if S is a maximal matching then size of W is at least as much as size of W. We also know OPT has at least one endpoint of edge in S and W has at most all endpoints from S. Hence, 0.5|W|<=|OPT| or |W|<=2|OPT|

Problem 3

Part a

1) The value of the optimal solution is 9. Item in knapsack is item 3 with w_3 =10 and v_3 = 9

2)The value of greedy solution is 1. Item in Knapsack is item 1 with w_1 = 1 and v_1 = 1.

3)Ratio of Greedy/OPT = 1/9

Part b

- Let us modify above example a bit. $v_1 = 1, w_1 = 1$; $v_2 = 5/ε$, $w_2 = 10/ε$; $v_3 = 9/ε$, $w_3 = 10/ε$ and W = W/ε
- Bang per buck remains same so we have $a_1 >= a_2 >= a_3$ so greedy algorithm always ends up choosing item 1 with $v_1 = 1, w_1 = 1$.
- For any value of 0<ε<9 optimal will choose item 3 with value 9. So we have Greedy/OPT = 1/(9/ε) = ε/9<=ε
- For any value of ε>9 OPT and Greedy both will select item 1 with value 1. Greedy/OPT = 1<=ε.
- Hence the above inequality holds for any ε>0 ie Greedy/OPT<=ε

Part c:

- The value of the optimal solution is 9. Item in knapsack is item 3 with w_3 =10 and v_3 = 9
- For cheatingGreedy solution If we are allowed to include the last item in the loop, the one that does not fit in knapsack then we first select Items 1 and 3 which gives us value = v_1+v_3 = 10 and W' = 11 which is greater than W.
- Performance = cheatingGreedy/OPT = 10/9

Part d

- We choose items from 1..k+1 out of n to be part of knapsack using cheatingGreedy algorithm such that a_1>=a_2>=a_3...>=a_{k+1} where $a_i = v_i/w_i$ and sum of all weights is W'
- **Claim:** CheatingGreedy's performance is at least as large as Opt.
- **Proof by contradiction:**
- Suppose our cheatingGreedy is not optimal then there must be at least one item in current knapsack that can be replaced with an item that was not added to knapsack originally.
- Let us say an item i in knapsack can be replaced with j. But greedy had chosen item i to j because $a_i >= a_j$ i.e $v_i/w_i >= v_j/w_j$ $v_j<=(w_j/w_i)*v_i$
- We have two cases here $(w_j/w_i) = 1$ then $v_j = v_i$ It does not change the solution.
- If $(w_j/w_i) < 1$ it must be that $a_j>a_i$ in that case greedy would have chosen item j over i but chose i item which is contradictory to method of choosing items for cheatingGreedy hence it must be that $(w_j/w_i) > 1$.
- Hence the cheatingGreedy algorithm is atleast optimal for w' weight i.e cheatingGreedy>=OPT

Part e

- The value of the optimal solution is 9. Item in knapsack is item 3 with w_3 =10 and v_3 = 9.
- In Greedy we first add item 1 with v_1 = 1 and w_1 = 1 which makes V = 1 and we cannot add item 3 with v_3 = 9 and w_3=10 because weight exceeds 10.1.
- In this case careful greedy will just output value as 9 as w_3<W and not consider W_1.
- So performance = CarefulGreedy/OPT = 9/10

Part f

- For carefulGreedy we choose items into knapsack according to the rule mentioned in question part e. Let's just select item i because v_i>=V so value of carefulGreedy is v_i
- For cheatingGreedy we can add last item in the loop which does not fit the knapsack and violates weight constraint. So in the end we have V+v_i value for cheating greedy
- V can at max be v_i so value of cheatingGreedy is 2v_i
- Hence, 2*carefulGreedy>=cheatingGreedy i.e carefulGreedy>=0.5*cheatingGreedy
- From part d we know that cheatingGreedy>=OPT so we can write carefulGreedy>=0.5*OPT. Hence proved.

Problem 4

Algorithm:

- Sort the jobs in decreasing order of their processing time t_j.
- Assign job j to machine m which has smallest Load L.

Proof:

- Let M be the makespan of Greedy, and M' the optimal makespan.
- We assume that n > m. where n is the number of jobs and m is the number of machines.
- Since at least two of m+1 jobs that must be put on the same machine it must be that the optimal time $M^* >= 2 * t_{m+1}$.
- If machine i with final maximum load M runs job j only then solution is optimal because $M = t_j <= M^*$(from Lemma 4- class notes)
- We assigned job j to i because it had the minimum load so it must be the case that j>=m+1.
- So we have $t^j <= t^{m+1} <= 0.5M^*$ and we have $M - t_j <= M^*$. combining both the equations $M<=1.5M^*$

Problem 5

- Consider we have a bipartite graph G = (V, E) with two sets of vertices L and R such that there is no edge between vertices of the same set.
- We run maximum matching on graph G get a matching set S.
- L1 = L ∩ S; L2 = L - S; R1 = R ∩ S;R2 = R - S
- Let B be the vertices in R2 that have neighbours in L1
- C = L2 U R1 U B
- Output C

C is the minimum vertex cover for bipartite graph.

Proof:

- C covers all the edges that have their endpoint in either L2 or R1. Regarding edges in R2 and L1 we have B that covers vertices in R2 that have neighbours in L1. Hence all the edges have been covered and C is the minimum vertex cover.