A Two Stage Heuristic Algorithm for Solving the Server Consolidation Problem with Item-Item and Bin-Item Incompatibility Constraints Section 3: Heuristic solution

Christopher Gonzalez

Linear Programming Formulation: (P1)

$$\text{Minimize} \quad \sum_i Y_i$$

$$\text{Subject to:}$$

$$Y_i \geq X_{ij} \qquad \forall i, j$$

$$\sum_i X_{ij} = 1 \qquad \forall j$$

$$Y_i * memory_i \geq \sum_j \widetilde{mem}_j X_{ij} \qquad \forall i$$

$$Y_i * cpu_i \geq \sum_j \widetilde{cpu}_j X_{ij} \qquad \forall i$$

$$Y_i * disk_i \geq \sum_j \widetilde{disk}_j X_{ij} \qquad \forall i$$

$$\sum_{j \in J_A} X_{ij} \leq 1 \qquad \forall i$$

$$\sum_{j \in J_B} X_{ij} \leq 1 \qquad \forall i$$

$$X_{i'j'} = 0 \qquad \exists i', j'$$

Heuristic solution

- The author solve the server(item) – target server(bin) mapping problem in two stages. In Summary:
- Step 1: Solve the set of “incompatibility constraints”. the solution to this step defines the clusters of servers such that servers belonging to the same cluster can be co-located onto a target server
- Step 2: Allocate servers (items) to the different target servers (bins)

Stage 1

- Stage 1 involves minimizing the problem subject to constraints 6 to 9. They call this limited version (P2)
- The optimal solution to (P2) gives us the heuristic estimate of the minimum number of target servers required for a given problem instance. Furthermore, solution to the problem (P2) organizes the servers into clusters.

$$\textit{Minimize} \sum_{i=1}^{I} Y_i$$

$$\textit{Subject to}$$

$$\sum_{j \in J_A} X_{ij} \leq 1 \qquad \forall i$$

$$\sum_{j \in J_B} X_{ij} \leq 1 \qquad \forall i \qquad (P2)$$

$$X_{i'j'} = 0 \qquad \exists i', j'$$

Stage 1

- The solution obtained at the end of stage-1 would have been optimal for the problem (P1) if the bins were un-capacitated. Stage 2 of the solution building process, refines the partial solution obtained in the previous step by considering the actual bin capacities and performing server – target-server mappings.

3.1 Solving the "Item-Item Incomp. Constraints"

- The authors consider a subset of the problem (P2) and call it (P3)
- A close look at the problem (P3) will reveal that the problem definition resembles that of the Graph Coloring Problem.
- The optimal solution to the problem (P3) helps determine the minimum number of target servers that will be required for the consolidation exercise.

3.1 Solving the "Item-Item Incomp. constraints"

Consider a subset of the problem $(P2)$:

$$Minimize \sum_{i=1}^{i} Y_i$$

$$\sum_{i} X_{ij} = 1 \qquad \forall j \in J_A, J_B$$

$$\sum_{j \in J_A} X_{ij} \leq 1 \qquad \forall i$$

$$\sum_{j \in J_B} X_{ij} \leq 1 \qquad \forall i \qquad (P3)$$

3.1 Solving the “Item-Item Incomp. Constraints”

- The authors introduce an undirected conflict graph (Fig 1) for the “Item-Item Incompatibility Constraints” where the nodes are items and the edges between them signify they are incompatible.
- Fig 1 is the conflict graph for example 1.

Example 1: Assume that we have two item-item incompatibility constraints:

$$X_{i1} + X_{i2} + X_{i3} + X_{i4} + X_{i5} \leq 1$$

$$X_{i1} + X_{i6} + X_{i7} \leq 1$$

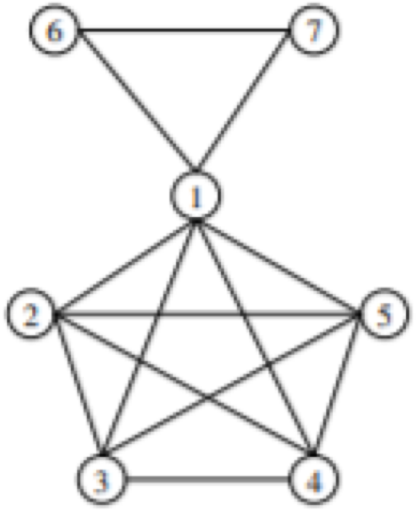


Figure 1: Conflict Graph for the "Item-Item Incompatibility Constraints" for Example 1.

Example 1 and Fig 1

3.1 Solving the “Item-Item Incomp. Constraints”

- The authors prove the theorem: Let G=(V,E) be a completely connected graph formed out of the set of “Item-Item incompatibility constraints” in (P2), and there exists a unique optimal solution to the problem (P3), then the optimal solution to the problem (P3) will represent the lower bound to the solution for the problem (P1).

3.1 Solving the “Item-Item Incomp. Constraints”

- Since, the number of target servers obtained as a result of solving the problem (P3) represents the minimal number of target servers that will be required for the problem in (P1), (P3) is solved first.
- Optimal solution to (P3) implicitly defines the clusters of conflicting items that can be grouped together such that item in one group is not in conflict with any other member of the group. However, the problem is known to be NP-hard

Solving the “Item-Item Incomp. constraints”

- The vertex coloring problem is well studied and a number of solutions have been proposed. The authors use the popular Welsh-Powell approximation algorithm.
- The algorithm assigns each vertex a color that is different from the colors of its neighbors. The algorithm is proven to use at most $\Delta(G) + 1$ colors, where $\Delta(G)$ is the maximum degree of the graph.

The Welsh-Powell algorithm

- The Welsh-Powell algorithm for solving the graph coloring problem uses a simple heuristic and is as follows:
- 1. Sort the vertices in decreasing order of degree. To begin with all the vertices are uncolored.
- 2. Traverse the vertices in the sorted list, and assign a vertex the color 1 if it is uncolored and in case the vertex does not yet have a neighbor having color 1.
- 3. Repeat this process with other colors until no vertex is uncolored.

3.2 Solving the “Bin-Item Incomp. Constraints”

- Continuing with the same example, the authors added three bin item constraints:

$$X_{A1} = 0$$

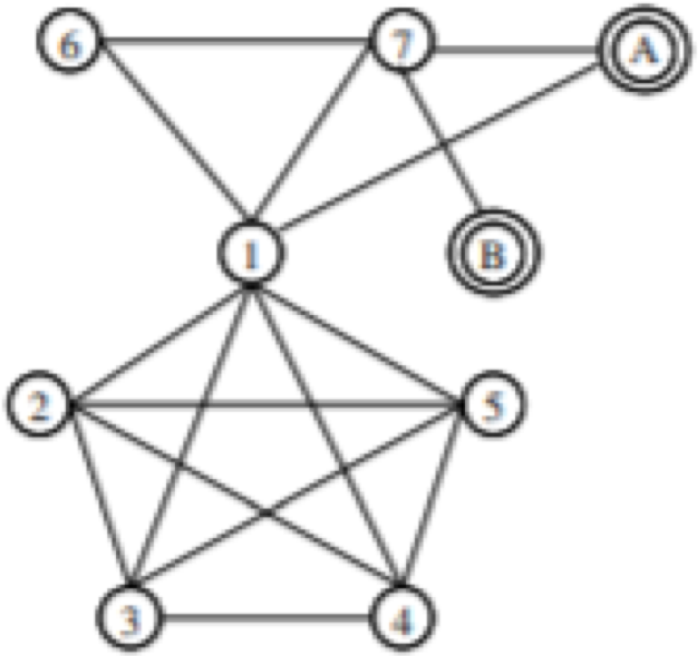
$$X_{A7} = 0$$

$$X_{B7} = 0$$

The existing graph is augmented by introducing ‘pre-colored’ dummy nodes into the graph. Each of the dummy nodes introduced will be colored differently and the number of such pre-colored dummy nodes depends on the number of bins that are in conflict with the items

Fig 2

- two pre-colored dummy nodes since there are only two bins – A and B that are not compatible for some of the items.
- Figure 2 shows these two newly introduced nodes with double circles. These doubly encircled nodes represent the bins
- A newly introduced ‘pre-colored’ bin-node is thereafter connected using an undirected edge to those item-nodes that cannot be assigned to this particular bin



3.2 Solving the “Bin-Item Incomp. Constraints”

- The problem is solved by modifying the graph coloring heuristic of Welsh-Powell suitably.
- Instead of sorting all the vertices in decreasing order of the degree, two sorted lists are maintained.
- The first one – called the pre-colored list, contains the sorted ‘pre-colored’ vertices in the decreasing order of the degree.
- The second – called the uncolored list, contains the sorted uncolored vertices in the decreasing order of the degree.
- Thereafter, apply the modified WelshPowell algorithm as follows

1. Select the next pre-colored vertex from the pre-colored list (At the start of the algorithm this is the first element in the pre-colored list).
2. Traverse the vertices in the uncolored list, and assign a vertex the color of the pre-colored vertex of step 1 if it is uncolored and in case the vertex does not yet have a neighbor having the same color. Go to Step-1.
3. For the vertices (in the uncolored list) that remain uncolored at the end of the traversal process of the pre-colored list, color them using the usual Welsh-Powell heuristic.

Stage 2: 3.3 Allocating servers to the target servers

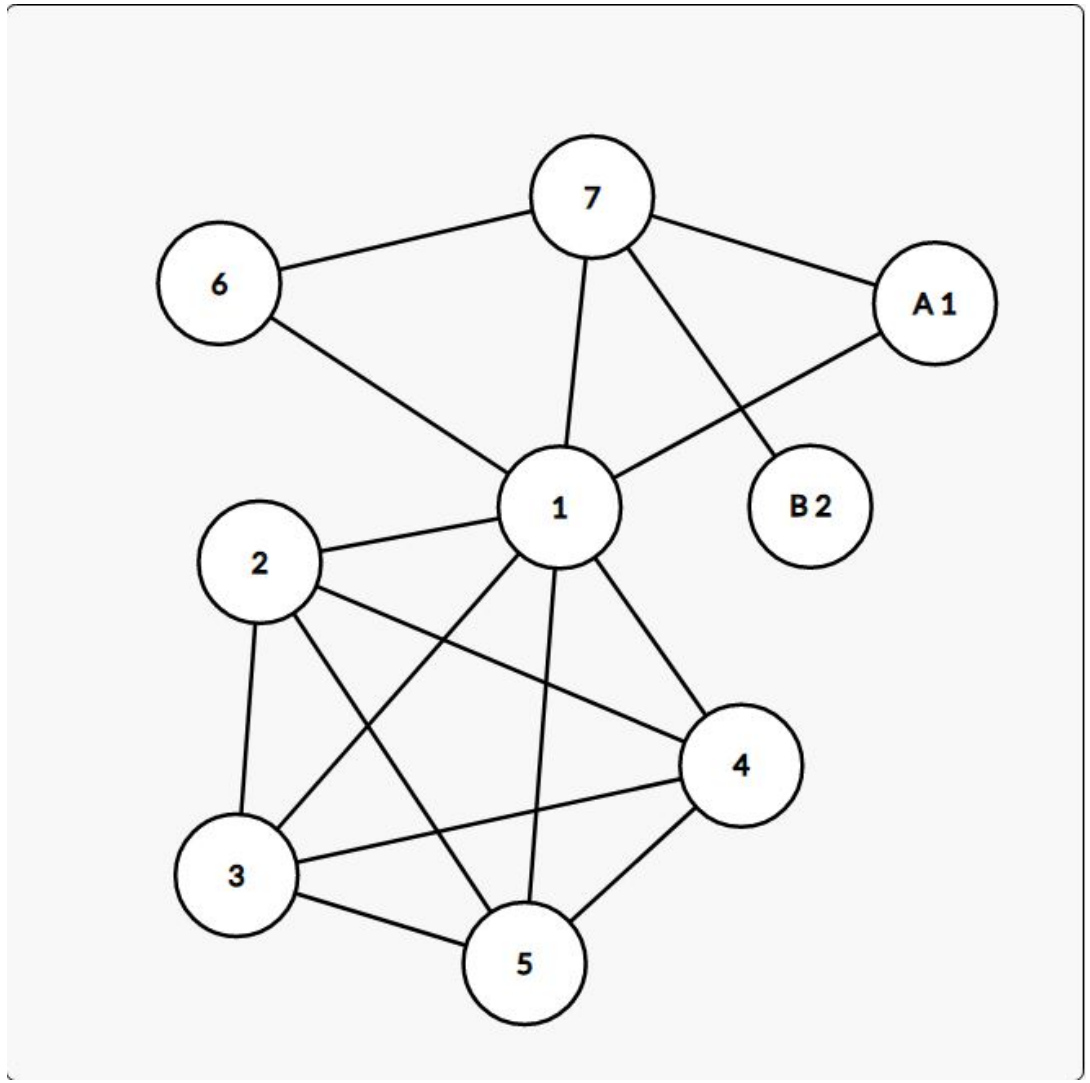
- Let N be the solution to the pre-colored graph coloring problem.
- The nodes having the same color define one cluster and hence denote the items that can be grouped together into a target server provided capacity constraints of the target server are not violated.
- these N clusters are labeled as n1, n2, …., nN. In addition to the items that belong to one of these N clusters, there are items which are not constrained by any of the incompatibility factors.
- These non-conflicting items can be associated with any of the N clusters. However, the authors decided to group such items into a separate cluster and associate a label nN+1 with the cluster.

Stage 2: 3.3 Allocating servers to the target servers

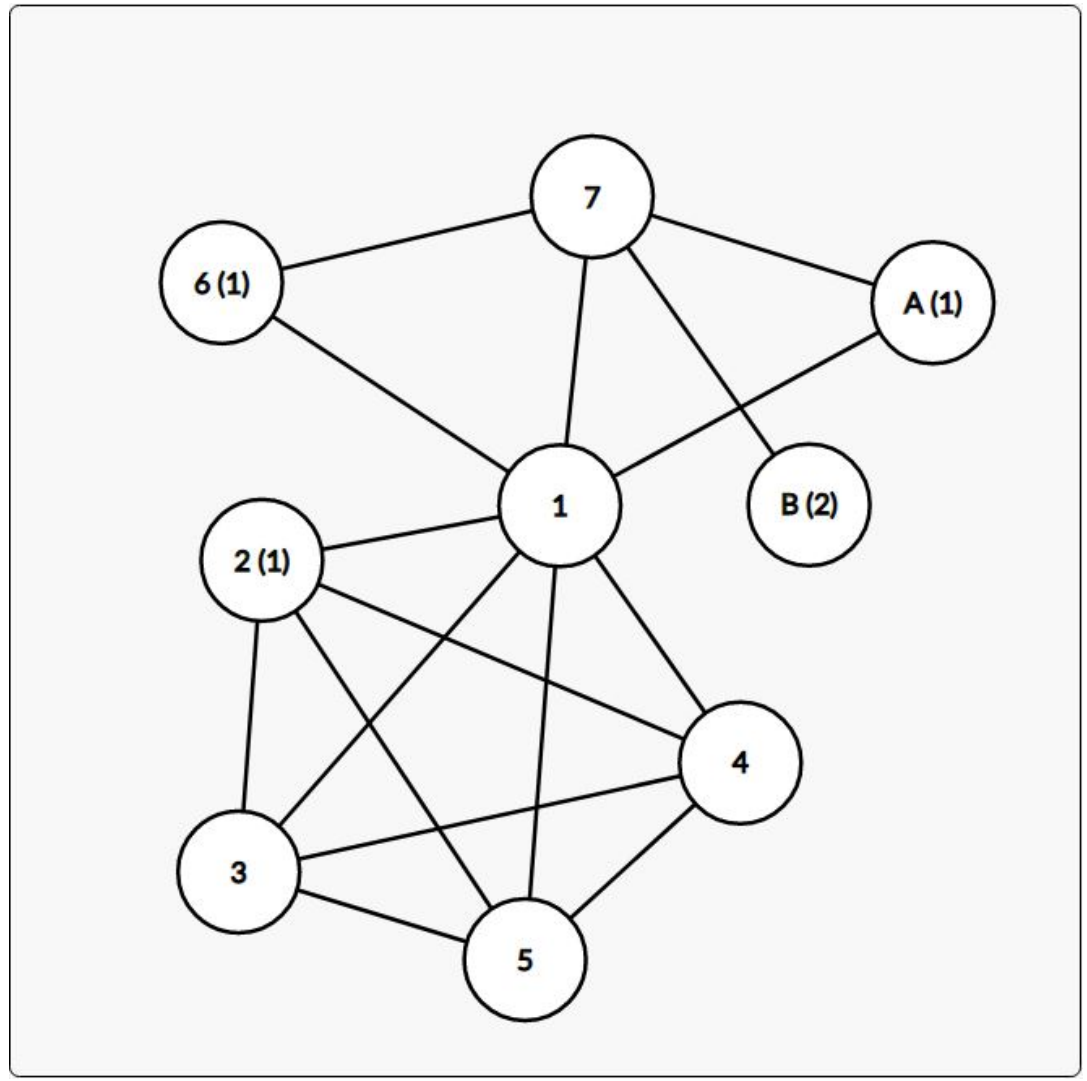
- This is FFD algorithm which packs items into a bin in descending order of size and adds a new bin for an item which cannot be accommodated in any of the already opened bins.
- In essence,they partition the items into N clusters of mutually non-conflicting items. The algorithm then solves a simple vector packing problem for each set.

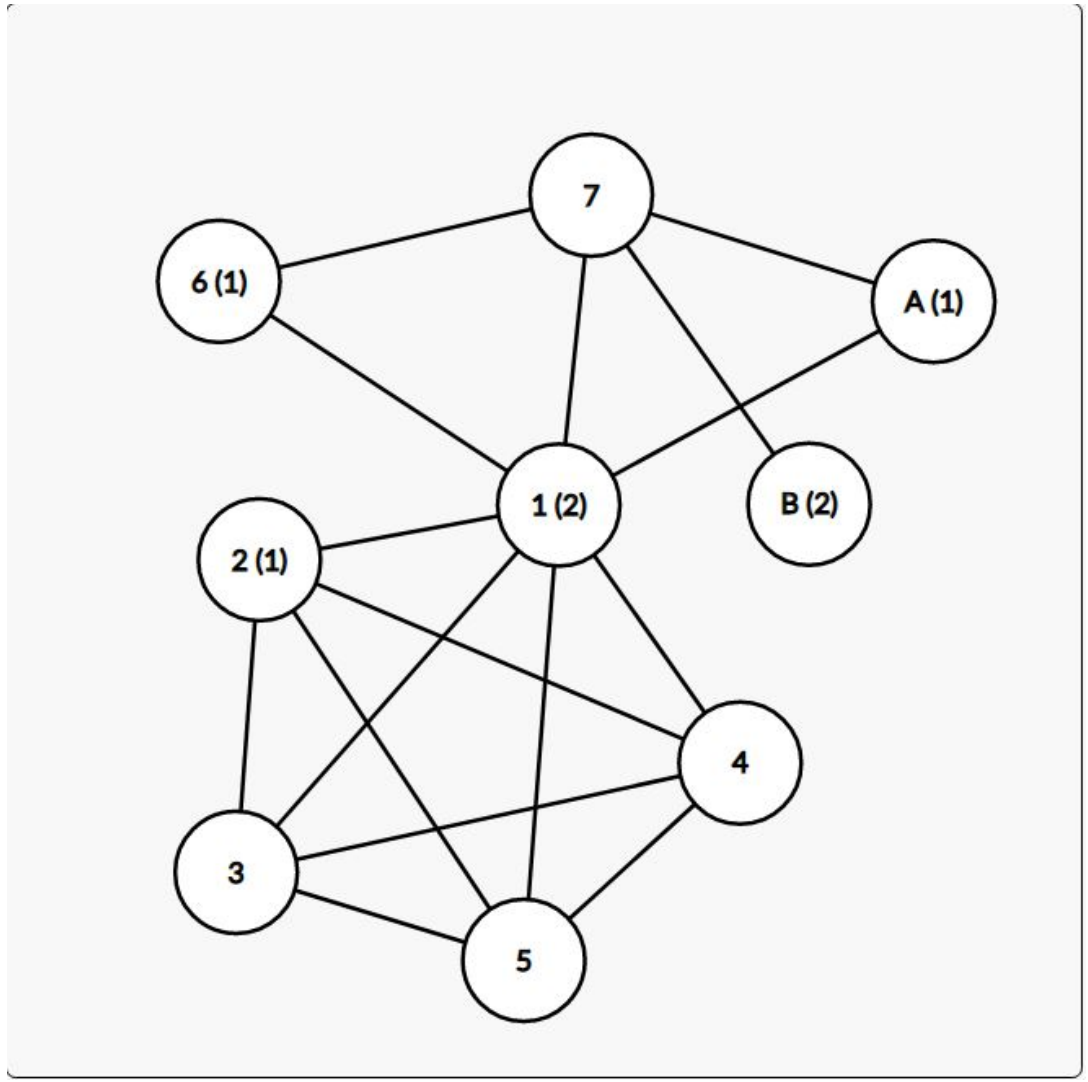
1. Sort the bins in the decreasing order of volumes (CPU*memory*disk). To begin with all the bins are empty. Only the bins that are involved in the Bin-Item Incompatibility Constraints have pre assigned label.
2. Sort the items in the decreasing order of volumes (CPU*memory*disk). Each of the items has an associated label.
3. Pop out the next element from the items list.
4. Let bin_counter = 1.
5. Check whether the item can be assigned to the bin identified by the bin_counter. If it can be assigned go to step 6 else go to step 8.
6. If the bin does not have any label assigned to it yet, it is assigned a label that is same as the label of the item being assigned to the bin. Thus, if the label of the item is n_1 (*i.e* the item belongs to the cluster n_1) then the bin is labeled as n_1 as well. This means that henceforth only items belonging to the group n_1 can be assigned to the bin. However, if the item belongs to the cluster n_{N+1}, no label change is required.
7. Remove the element from the list of sorted items and go to step 3.
8. Increment the bin_counter, if the bin_counter is less than or equal to the number of bins repeat step 5 else the item cannot be placed go to step 3. Since we are not placing any restriction on the number of bins, we are assuring that all the items get allocated.

Solving the Bin Item constraints


7
6
A 1
1
B 2
2
4
3
5

1. Select the next pre-colored vertex from the pre-colored list (At the start of the algorithm this is the first element in the pre-colored list).
2. Traverse the vertices in the uncolored list, and assign a vertex the color of the pre-colored vertex of step 1 if it is uncolored and in case the vertex does not yet have a neighbor having the same color. Go to Step-1.
3. For the vertices (in the uncolored list) that remain uncolored at the end of the traversal process of the pre-colored list, color them using the usual Welsh-Powell heuristic.

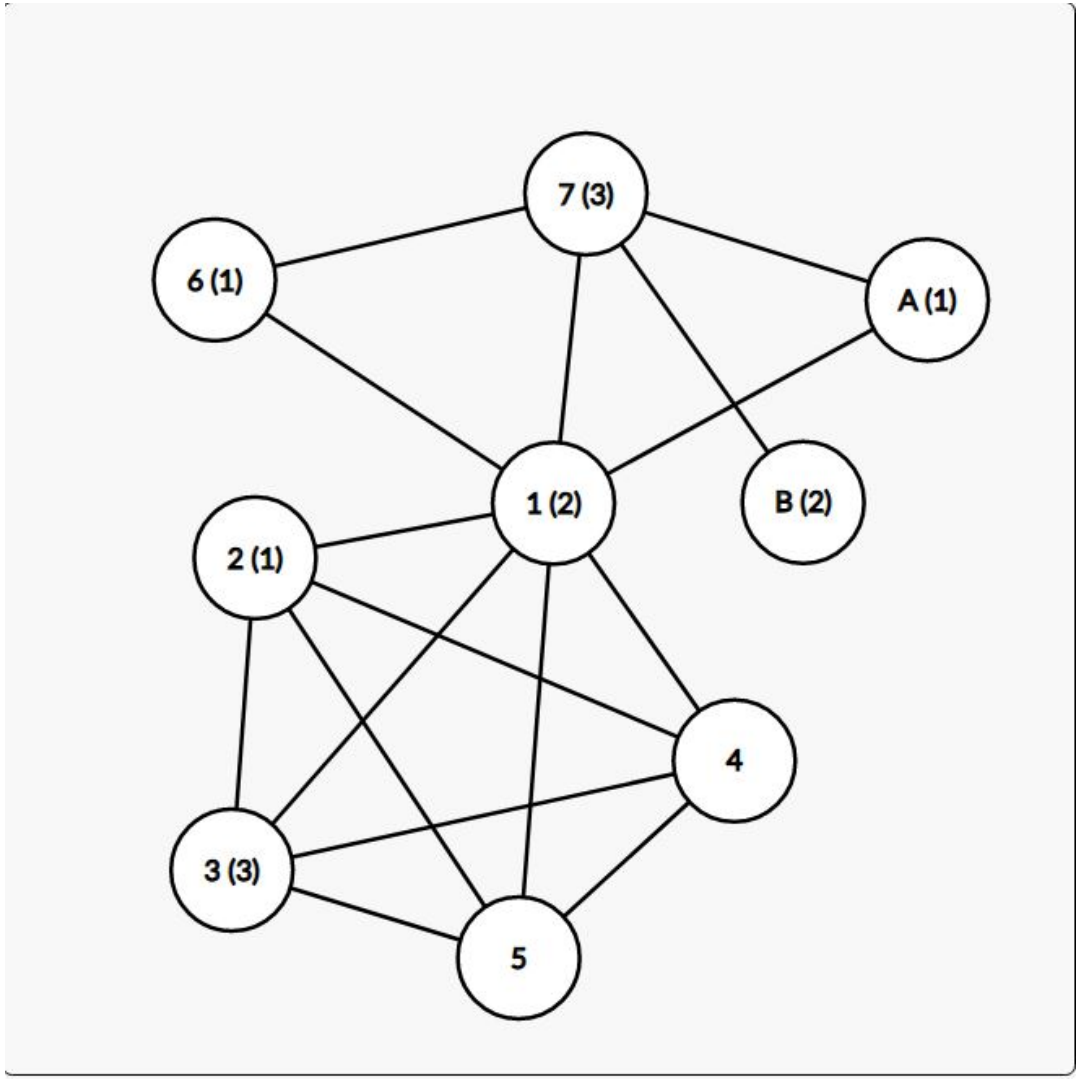

7
6 (1)
A (1)
1
B (2)
2 (1)
4
3
5

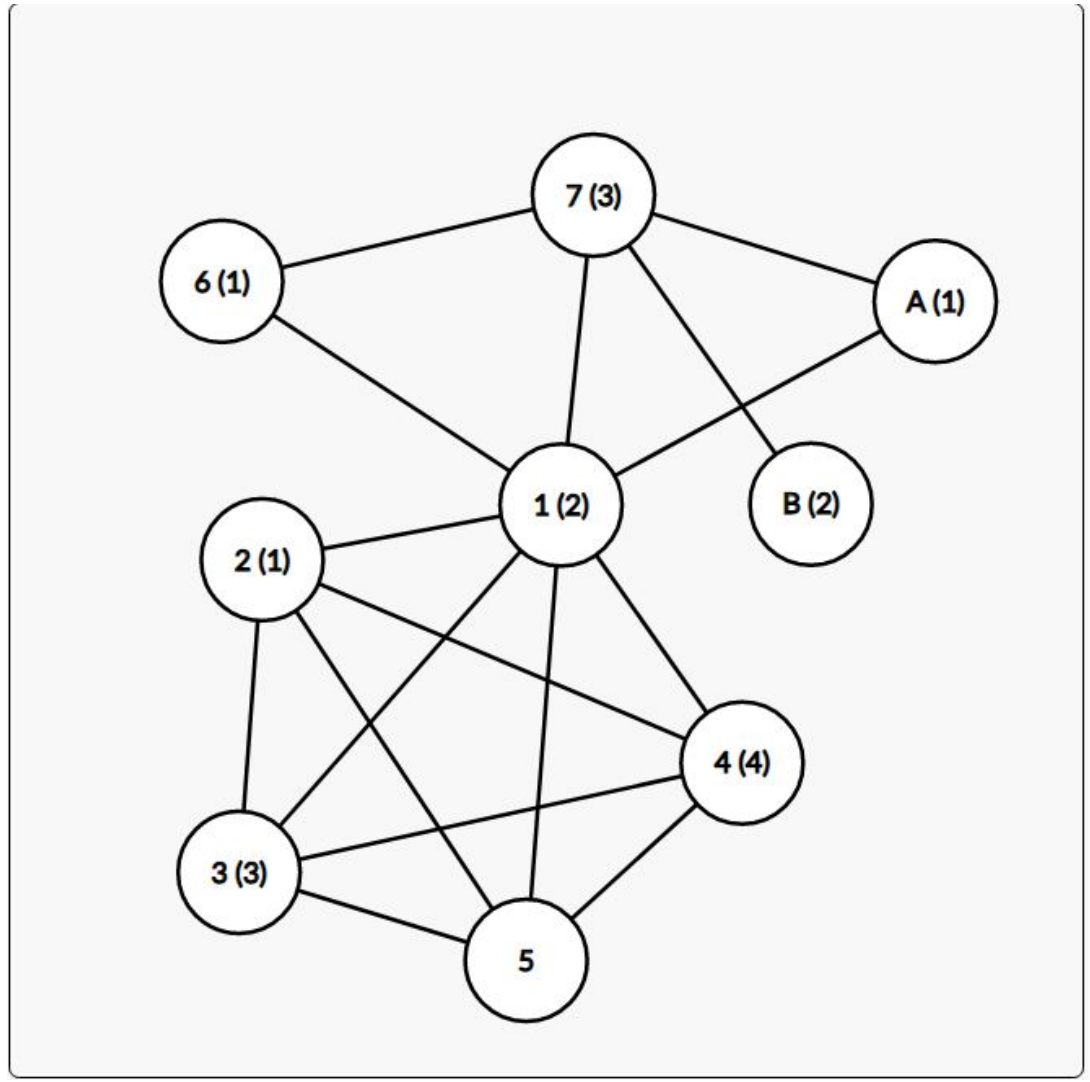

7
6 (1)
A (1)
1 (2)
B (2)
2 (1)
4
3
5

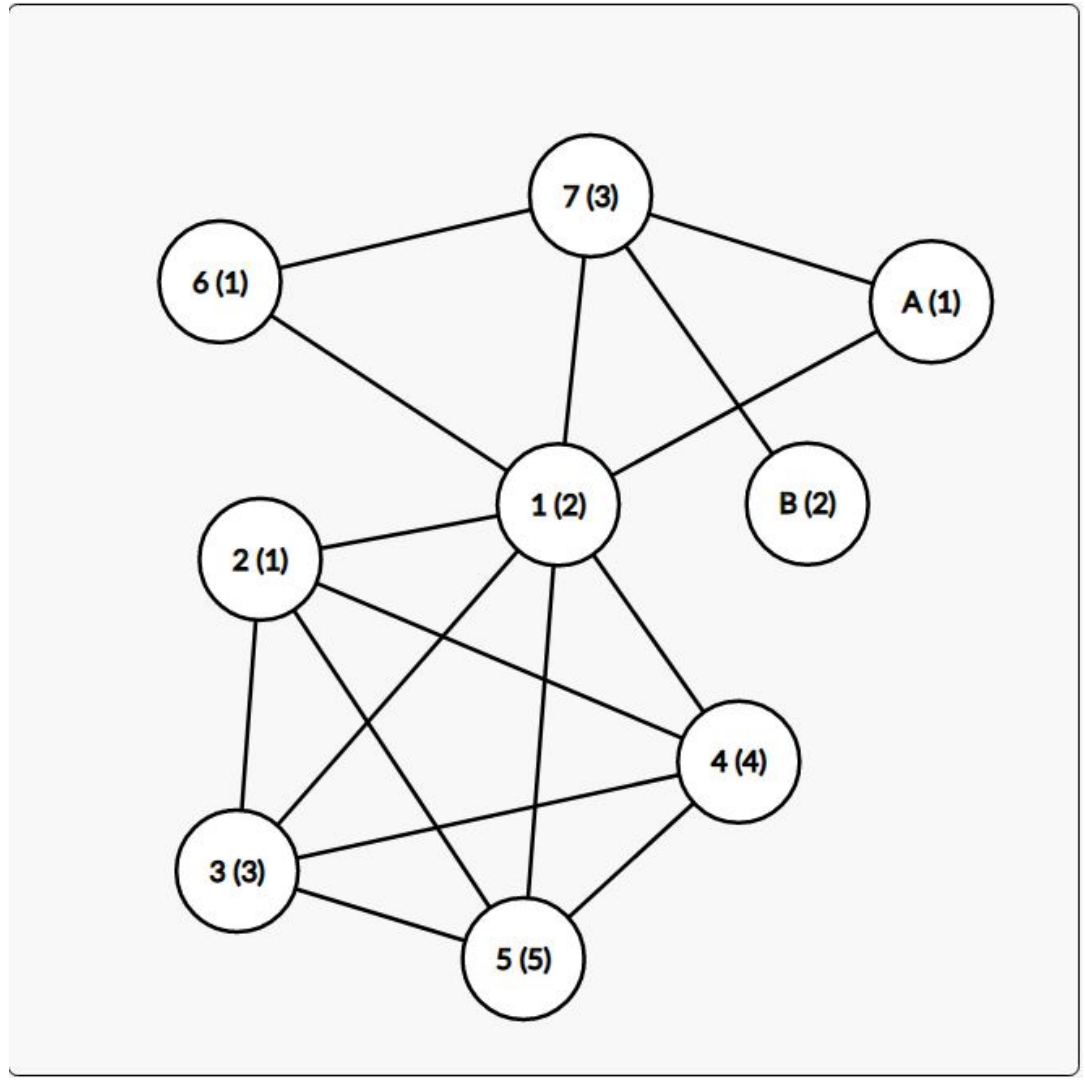
1. Sort the vertices in decreasing order of degree. To begin with all the vertices are uncolored.

2. Traverse the vertices in the sorted list, and assign a vertex the color 1 if it is uncolored and in case the vertex does not yet have a neighbor having color 1.

3. Repeat this process with other colors until no vertex is uncolored.


7 (3)
6 (1)
A (1)
1 (2)
B (2)
2 (1)
4
3 (3)
5


7 (3)
6 (1)
A (1)
1 (2)
B (2)
2 (1)
4 (4)
3 (3)
5


7 (3)
6 (1)
A (1)
1 (2)
B (2)
2 (1)
4 (4)
3 (3)
5 (5)