Homework 7 Solutions
NP Problems
Problems and Solutions listed in a different order than assigned

1. **Set Cover Problem**: Solution is at the bottom of Page 2 of enclosed document. Please note that the sets S can be arbitrary and it may not be possible to map them on Euclidean plane with some distance metric.

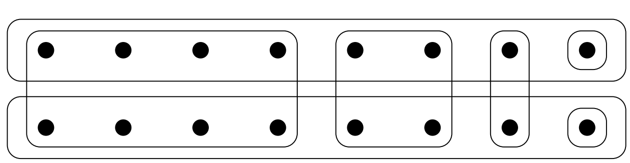
2. **k-Clustering Problem**: Solution is provided in DPV, Section 9.2.2, Pages 280-281.

3. **Greedy-Set-Cover Problem**: Easy solution. [thread, lost, shun, arid]

4. **Bin-Packing Problem**: Solution is provided later in the enclosed document.

5. **Graph-Coloring Problem**: (i) Solution is provided later in the enclosed document. (ii) Solution is on Page 487 of KT with 6 nodes.

6. **0-1 Knapsack Problem**: Why dynamic programming algorithm is not a polynomial algorithm? What is a proper way of describing the computational complexity of the dynamic programming solution to the 0-1 knapsack problem?: Page 491 of KT. Essentially, W is 2^{log w}. It is exponential in the input w bits.

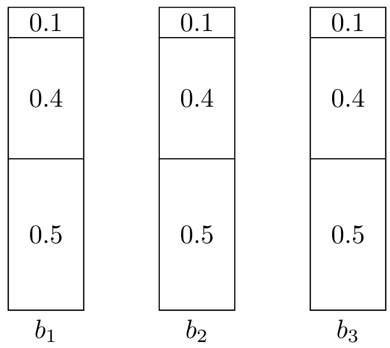


3 Bin packing

Definition 5 (Bin packing problem)**.** *Given a set $\mathcal{S}$ with n items where each item i has $size(i) \in (0, 1]$, find the minimum number of unit-sized (size 1) bins that can hold all n items.*

For any problem instance I, let $OPT(I)$ be a optimum bin assignment and $|OPT(I)|$ be the corresponding minimum number of bins required. One can see that $\sum_{i=1}^{n} size(i) \leq |OPT(I)|$.

Example Consider an instance where $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$, where $|\mathcal{S}| = n = 9$. Since $\sum_{i=1}^{n} size(i) = 3$, at least 3 bins are needed. One can verify that 3 bins suffices: $b_1 = b_2 = b_3 = \{0.5, 0.4, 0.1\}$. Hence, $|OPT(\mathcal{S})| = 3$.



3.1 First-fit: A 2-approximation algorithm for bin packing

Algorithm 2 FIRSTFIT($\mathcal{S}$)

B $\rightarrow \emptyset$ ▷ Collection of bins
for $i \in \{1, \dots, n\}$ **do**
  if $size(i) \leq size(b)$ for some bin $b \in B$ **then**
    $size(b) \leftarrow size(b) - size(i)$ ▷ Put item i to existing bin b
  else
    $B \leftarrow B \cup \{b'\}$, where $size(b') = 1 - size(x_i)$ ▷ Put item i into a fresh bin b'
  end if
end for
return B

Algorithm 2 shows the First-Fit algorithm which processes items one-by-one, creating new bins if an item cannot fit into existing bins.

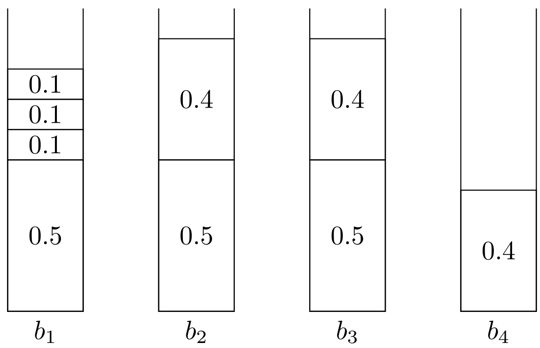
Lemma 6. *Using First-Fit, at most one bin is less than half-full. That is,* $|\{b \in B : size(b) \leq \frac{1}{2}\}| \leq 1$.

Proof. Suppose, for a contradiction, that there are two bins b_i and b_j such that $i < j$, $size(i) \leq \frac{1}{2}$ and $size(j) \leq \frac{1}{2}$. Then, First-Fit could have put all items in b_j into b_i, and not create b_j. Contradiction. □

Theorem 7. *First-Fit is a 2-approximation algorithm for bin packing.*

Proof. Suppose First-Fit terminates with $|B| = m$ bins. By lemma above, $\sum_{i=1}^{n} size(i) > \frac{m-1}{2}$. Since $\sum_{i=1}^{n} size(i) \leq |OPT(I)|$, we have $m - 1 < 2\sum_{i=1}^{n} size(i) \leq 2 \cdot |OPT(I)|$. That is, $m \leq 2 \cdot |OPT(I)|$. □

Recall the example where $\mathcal{S} = \{0.5, 0.1, 0.1, 0.1, 0.5, 0.4, 0.5, 0.4, 0.4\}$. First-Fit will use 4 bins: $b_1 = \{0.5, 0.1, 0.1, 0.1\}, b_2 = b_3 = \{0.5, 0.4\}, b_4 = \{0.4\}$. As expected, $4 = |\text{FIRSTFIT}(\mathcal{S})| \leq 2 \cdot |OPT(\mathcal{S})| = 6$.



2 Graph Coloring

2.1 Scheduling Final Exams

Every semester, the registrar must schedule final exams in a way so that no student must take two exams at the same time. We can begin modeling this situation by constructing a graph $G = (V, E)$. In this case, the vertex set V corresponds to the set of all classes (one vertex per class). The set E is defined as follows: there is an edge between u and v if there exists at least one student taking both class u and class v. The goal is to assign each vertex to a time slot so that the two endpoints of every edge are assigned to different slots.

The simplest solution is to assign each final exam to a unique time slot. If there are n classes, this results in n time slots, but this may lead to an extremely long final exam period. So we want a solution that is not only feasible, but also minimizes the number of total time slots.

One way to visualize the notion of "assigning a time slot" is to consider each time slot as a color. In other words, given G, we must color every vertex of G so that the two endpoints of every edge receive different colors. Fig. 1 gives an example of coloring a graph with 5 vertices using 3 colors, and Fig. 2 colors the same graph using only two colors.

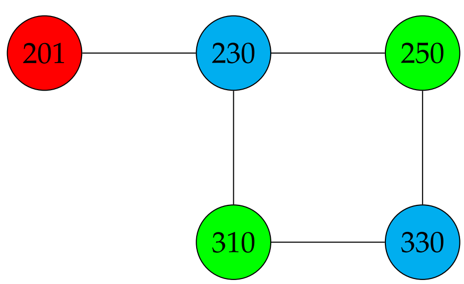


Figure 1: A graph G with 5 vertices corresponding to 5 classes. Each of the three colors represents a distinct time slot.

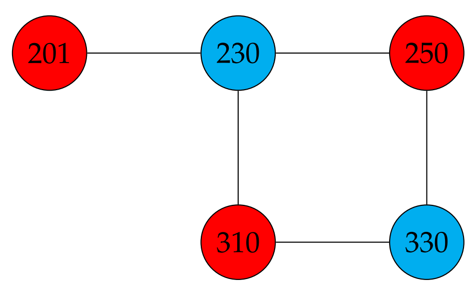


Figure 2: The same graph from Fig. 1, colored using only two colors.

Definition 1. *Let k be a positive integer. A graph is* k-colorable *if it is possible to assign each vertex to one of k colors such that the two endpoints of every edge are assigned different colors.*

When a coloring properly assigns the endpoints of an edge to different colors, we often say that the coloring *respects* the edge. If a coloring respects every edge of the graph, then the coloring is *proper* or *valid*.

The graph shown in Fig. 2 is 2-colorable, since every edge has a red endpoint and a blue endpoint. Notice that Fig. 1 shows that the same graph is 3-colorable—in general, if a graph is k-colorable, then it is also ℓ-colorable for any $\ell \geq k$. We will now prove a simple observation regarding graphs that are 2-colorable.

Observation 1. *Let G be a graph. Then G is 2-colorable if and only if G is bipartite.*

Proof. Let G be a 2-colorable graph, which means we can color every vertex either red or blue, and no edge will have both endpoints colored the same color. Let A denote the subset of vertices colored red, and let B denote the subset of vertices colored blue. Since all vertices of A are red, there are no edges within A, and similarly for B. This implies that every edge has one endpoint in A and the other in B, which means G is bipartite.

Conversely, suppose G is bipartite, that is, we can partition the vertices into two subsets V_1, V_2 every edge has one endpoint in V_1 and the other in V_2. Then coloring every vertex of V_1 red and every vertex of V_2 blue yields a valid coloring, so G is 2-colorable. $\square$

Thus, Observation 1 tells us that the graph in Fig. 2 is bipartite. Indeed, by observing Fig. 3, it becomes even clearer that this graph is bipartite.

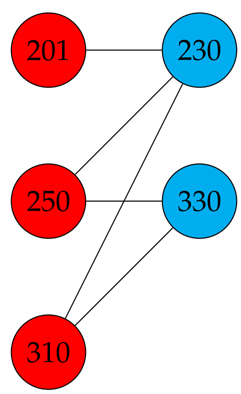


Figure 3: The same graph and coloring from Fig. 2, with the vertices both colored and rearranged to further illustrate that it's bipartite.