Algorithms: CSE 202 — Homework 4 Solutions

Problem 1: MaxCut for trees (KT 10.9)

Give a polynomial-time algorithm for the following problem. We are given a binary tree $T = (V, E)$ with an even number of nodes, and a nonnegative weight on each edge. We wish to find a partition of the nodes V into two sets of *equal* size so that the weight of the cut between the two sets is as large as possible (i.e., the total weight of edges with one end in each set is as large as possible). Note that the restriction that the graph is a tree is crucial here, but the assumption that the tree is binary is not. The problem is NP-hard in general graphs.

Solution: MaxCut for Trees (KT 10.9)

We root the tree at some node r and associate a subtree T_v with each node $v \in T$, and let n_v denote the number of nodes in the subtree T_v. We have $n = n_r$.

Dynamic programming definition: For vertex v of the tree and $k \geq 1$, let $OPT(v, k)$ be the maximum cut of the subtree T_v that splits the vertices of T_v into two parts where the side containing v has size k. We output $OPT(r, n/2)$ as the maximum weighted cut for T when the vertices are paritioned into two equal size parts.

Let $c(u, v)$ denote the cost of the edge $e = (u, v)$ in the tree. For a leaf v, T_v contains only one node and no edges. There is exactly one partition of the nodes and the part containing v has size 1. For this reason, we define $OPT(v, 1) = 0$.

Let v be any node of the tree other than a leaf node. Consider any partition of the nodes of T_v where the part containing v has size $k \geq 1$. Let A be the part containing the vertex v. Let $A' = A - \{v\}$. We have $|A'| = k - 1$. We consider two cases depending on the number of children of v.

Case I — v has one child: Let u be the child of v. If $u \in A$, then A' is a cut of T_u where the size of the part containing u is $k - 1$. Under these conditions, the maximum cut is exactly $OPT(u, k - 1)$.

If $u \notin A$, then A' is a cut of T_u where the size of the part containing u is $n_u - (k - 1)$. Under these conditions, the maximum cut is exactly $OPT(u, n_u - (k - 1)) + c(u, v)$.

Combining the two cases, we get that the maximum cut value is $\max(OPT(u, k - 1), OPT(u, n_u - (k - 1)) + c(u, v))$. This justifies the following recursive formulation when v has only one child u.

$$OPT(v, k) = \max(OPT(u, k - 1), OPT(u, n_u - (k - 1)) + c(u, v)).$$

Case II — v has two children: Let u_1 and u_2 be the children of v. The $k - 1$ nodes in A' are partitioned between the two subtrees T_{u_1} and T_{u_2}. Let l_i be the number of nodes in $A' \cap T_{u_i}$ for $i = 1$ and $i = 2$. For each pair (l_1, l_2) of values such that $0 \leq l_1, l_2 \leq k - 1$ and $l_1 + l_2 = k - 1$, we get four possibilities for the cut depending on whether u_1 and u_2 are in A'. For instance, consider the case that $u_1 \in A'$ and $u_2 \notin A'$. In this case, $T_{u_1} \cap A'$ is a cut of T_{u_1} where the part containing u_1 has size l_1. Among all such cuts of T_{u_1}, the maximum cut has value $OPT(u_1, l_1)$. For T_{u_2}, $A' \cap T_{u_2}$ is a cut where its complement contains u_2 and has size $n_{u_2} - l_2$. Among all such cuts, the maximum cut has value $OPT(u_2, n_{u_2} - l_2)$. Overall, the maximum cut $OPT(v, k)$ in this case is $c(v, u_2) + OPT(u_1, l_1) + OPT(u_2, n_{u_2} - l_2)$.

When we put all the cases together, we get the following recurrence.

$$OPT(v,k) = \max_{0 \le l_1 \le k-1, l_1 + l_2 = k-1} \begin{cases} OPT(u_1, l_1) + OPT(u_2, l_2) \\ OPT(u_1, l_1) + OPT(u_2, n_{u_2} - l_2) + c(v, u_2) \\ OPT(u_1, n_{u_1} - l_1) + c(v, u_1) + OPT(u_2, l_2) \\ OPT(u_1, n_{u_1} - l_1) + c(v, u_1) + OPT(u_2, n_{u_2} - l_2) + c(v, u_2) \end{cases}$$

Since $k \le n$, we have $O(n^2)$ subproblems. To compute $OPT(v,k)$, we need to consider at most $O(k)$ subproblems and it takes $O(k)$ time to look up the maximum cut value for these problems and find the maximum. Combining these observations, we get that the total time required is $O(n^3)$.

Problem 2: Heaviest first (KT 11.10)

Suppose you are given an $n \times n$ grid graph G, as in Figure 1 Associated with each node v is a weight $w(v)$,

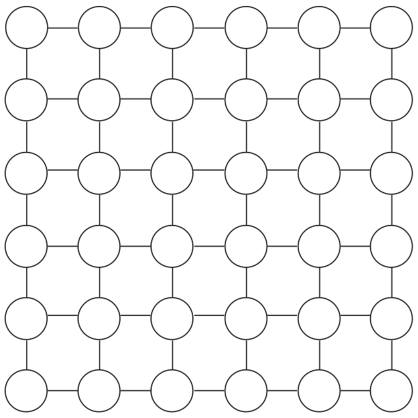
Figure 1: A grid graph

which is a nonnegative integer. You may assume that the weights of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible. (The sum of the weights of the nodes in S will be called its *total weight*.)

Consider the following greedy algorithm for this problem.

Algorithm 1: The "heaviest-first" greedy algorithm

```
Start with $S$ equal to the empty set
**while** some node remains in $G$ **do**
  Pick a node $v_i$ of maximum weight
  add $v_i$ to $S$
  Delete $v_i$ and its neighbors from $G$
**end while**
**return** $S$
```

1. Let S be the independent set returned by the "heaviest-first" greedy algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$,either $v \in S$,or there is a node $v' \in S$ so that $w(v) \le w(v')$ and (v, v') is an edge of G.
2. Show that the "heaviest-first" greedy algorithm returns an independent set of total weight at least $\frac{1}{4}$ times the maximum total weight of any independent set in the grid graph G.

Solution: Heaviest first (KT 11.10)

For any set A of nodes, let $w(A) = \sum_{v \in A} w(v)$. Let S be the independent set returned by the *heaviest-first* greedy algorithm. Let T be any independent set of the graph.

1. Let $v \in T$. If $v \notin S$, there must be a neighbor v' of v in S whose selection in some iteration of the greedy algorithm must have removed v from further consideration. According to the selection criterion, the weight of v' is at least as much as the weight of v. Otherwise, greedy algorithm would have chosen v instead of v' during the iteration. We thus have that either $v \in S$ or it has a neighbor v' in S such that $w(v') \geq w(v)$.

2. Let $R = S \cap T$ be the set of nodes common between S and T. Let $S' = S - R$ and $T' = T - R$ be the sets of vertices that are exclusive to S and T respectively. S' and R are disjoint and $S = S' \cup R$. Similarly, T' and R are disjoint and $T = T' \cup R$.

   From the earlier argument, we know that for $v \in T'$, there is a neighbor v' of v such that $v' \in S'$ and $w(v) \leq w(v')$. We use v' in S' to *cover* for v in T'. Any such v' in S' need to cover at most four of its neighbors v in T' where $w(v) \leq w(v')$. From this covering argument, we get $w(T') \leq 4w(S')$. Therefore, for any independent set T, we have

   $$\begin{aligned} w(S) &= w(R) + w(S') \\ &\geq w(R) + \frac{1}{4}w(T') \\ &\geq \frac{1}{4}(w(R) + w(T')) \\ &= \frac{1}{4}w(T) \end{aligned}$$

   Therefore, the "heaviest-first" greedy algorithm returns an independent set of total weight at least $\frac{1}{4}$ times the maximum total weight of any independent set in the grid graph G.

Problem 3: Bin packing

In the bin packing problem, we are given a collection of n items with weights $w_1, w_2, \ldots, w_n$. We are also given a collection of bins, each of which can hold a total of W units of weight. (We will assume that W is at least as large as each individual w_i.)

You want to pack each item in a bin; a bin can hold multiple items, as long as the total of weight of these items does not exceed W. The goal is to pack all the items using as few bins as possible. Doing this optimally turns out to be **NP**-complete, though you don't have to prove this.

Here's a merging heuristic for solving this problem: We start with each item in a separate bin and then repeatedly "merge" bins if we can do this without exceeding the weight limit. Specifically:

Merging Heuristic:

Algorithm 2: Merging heuristic

```
Start with each item in a different bin
while there exist two bins so that the union of their contents has total weight less than or equal to $W$ do
    Empty the contents of both bins
    Place all these items in a single bin.
end while
Return the current packing of items in bins.
```

Notice that the merging heuristic sometimes has the freedom to choose several possible pairs of bins to merge. Thus, on a given instance, there are multiple possible executions of the heuristic.

Example. Suppose we have four items with weights 1, 2, 3, 4, and W = 7. Then in one possible execution of the merging heuristic, we start with the items in four different bins; then we merge the bins containing the first two items; then we merge the bins containing the latter two items. At this point we have a packing using two bins, which cannot be merged. (Since the total weight after merging would be 10, which exceeds $W = 7$.)

1. Let's declare the size of the input to this problem to be proportional to

   $$n + \log W + \sum_{i=1}^{n} \log w_i$$

(In other words, the number of items plus the number of bits in all the weights.) Prove that the merging heuristic always terminates in time polynomial in the size of the input. (In this question, as in **NP**-complete number problems from class, you should account for the time required to perform any arithmetic operations.)

2. Give an example of an instance of the problem, and an execution of the merging heuristic on this instance, where the packing returned by the heuristic does not use the minimum possible number of bins.

3. Prove that in any execution of the merging heuristic, on any instance, the number of bins used in the packing returned by the heuristic is at most twice the minimum possible number of bins.

Solution: Bin packing

1.

By assumption $w_i \leq W$ for all $1 \leq i \leq n$ which implies $\sum_{i=1}^{n} \log(w_i) \leq n \log W = \texttt{poly}(n + \log W)$. We will show that the greedy heuristic can be implemented in polynomial in $n + \log W$.

We choose the strategy of merging the two smallest bins if the sum of their weights does not exceeed W.

In each iteration, the number of non-empty bins decreases by 1. The number of iterations is therefore bounded by n.

In each iteration, we need to check if there are two bins that can be merged. We use a heap to store the weights of the non-empty bins. We find the two smallest elements of the heap in $O(1)$ comparisons and add the merged bin to the heap $O(\log n)$ comparisons. We also need to add the two smallest weights and compare their the sum with W. Both addition and comparison are linear time operations in the bit length of the numbers involved. Since at any point the weight of the bins in bounded by W, both the addition and comparison operations take time $O(\log(W))$. The total time required to merge two bins is bounded by $O(\log n \log W)$ in each iteration.

Since the number of iterations is bounded by n, the total time complexity is $O(n \log n \log W) = \texttt{poly}(n + \log W)$.

2.

Let $W = 4$ and let items have weights $2, 2, 3$, and 1. The optimal solution uses two bins $(2 + 2, 3 + 1)$. However, the greedy solution merges an item with weight two with the item of weight 1, resulting in a solution with 3 bins, namely $(2 + 1, 3, 2)$.

3.

Let O be the number of bins in the optimal solution and let $o_1, o_2, \ldots, o_O$ be the weights of all nonempty bins in the optimal solution. Similarly, let G be the number of bins in the greedy solution and $g_1, \ldots, g_G$ be the weight of the nonempty bins.

Let $T = \sum_{i=1}^{n} w_i$ be the sum of all the weights. We have

$$\sum_{i=1}^{O} o_i = T = \sum_{i=1}^{G} g_i \tag{1}$$

Since $o_i \leq W$ for all i we also have

$$T \leq O \cdot W \tag{2}$$

Let $l := \min_{i=1}^{G} g_i$ be the weight of the smallest bin in the greedy solution. We have two cases:

Case 1 — $l \geq W/2$: In this case we have $g_i \geq W/2$ for all i and hence

$$T \geq \frac{G \cdot W}{2} \tag{3}$$

Combining equation 2 with 3 we get

$$O \cdot W \geq \frac{G \cdot W}{2} \tag{4}$$

and therefore

$$\frac{O}{G} \geq \frac{1}{2} \tag{5}$$

Case 2 — $l \leq W/2$: We can assume that $G \geq 2$, otherwise the greedy solution would only use one bin and is necessarily optimal. All bins other than the smallest have weight at least $W - l$, otherwise we could still merge two bins. We therefore have

$$\begin{aligned} T &\geq (G-1)(W-l) + l && (6)\\ &= GW - Gl - W + 2l && (7)\\ &\geq GW - \frac{GW}{2} - W + W && (8)\\ &= \frac{GW}{2} && (9) \end{aligned}$$

using $G \geq 2$ and $l \leq W/2$ for the penultimate step.

As in case 1, we conclude

$$\frac{O}{G} \geq \frac{1}{2} \tag{10}$$

Problem 4: Maximum coverage

The maximum coverage problem is the following: Given a universe U of n elements, with nonnegative weights specified, a collection of subsets of U, $S_1, \ldots, S_l$, and an integer k, pick k sets so as to maximize the weight of elements covered. Show that the obvious algorithms, of greedily picking the best set in each iteration until k sets are picked, achieves an approximation factor of $(1 - (1 - 1/k)^k) > (1 - 1/e)$.

Solution: Maximum coverage

For a set $S \subseteq U$, let

weight(S) denote the total weight of the elements in the set S.

Let $S_{o_1}, \ldots, S_{o_k}$ be an optimal collection of subsets that maximizes the sum of the weights of the covered elements. Let $O = \bigcup_{i=1}^{k} S_{o_i}$ be the set of elements covered by the optimal solution. For $1 \leq i \leq k$, let S_{g_i} be the set selected by the greedy algorithm during the i-th iteration. Let $G_0 = \varnothing$ and $G_i = \bigcup_{j=1}^{i} S_{g_j}$ be the set of elements covered by the greedy solution after the i-th iteration for $1 \leq i \leq k$. Note that G_k is the set of elements covered by the greedy solution.

Consider the difference of weights $\Delta_i :=$ weight$(O) -$ weight(G_i). $\Delta_i \geq 0$ since O is the set of elements covered by the optimal solution. We argue that Δ_i is decreasing at a rate of $(1 - 1/k)$ as i increases.

Claim 1. *For $1 \leq i \leq k$*

$$\Delta_i \leq \Delta_{i-1} - \frac{\Delta_{i-1}}{k} = (1 - 1/k)\Delta_{i-1} \tag{11}$$

Proof. We consider the set of elements that are covered by the optimal solution but not by the greedy solution after $i-1$ iterations and let $T_{i-1} := O - G_{i-1}$. We have $\text{weight}(T_{i-1}) \geq \Delta_{i-1}$. Since $T_{i-1} \subseteq O$ and O is the union of the k sets S_{o_j}, there must exist a set S_{o_j} for $1 \leq j \leq k$ such that $\text{weight}(S_{o_j} \cap T_{i-1}) \geq \text{weight}(T_{i-1})/k$. During the i-the iteration, the greedy algorithm has the opportunity to select S_{o_j} to improve the coverage by at least $\text{weight}(T_{i-1})/k \geq \Delta_{i-1}/k$. Since the greedy algorithm in each iteration selects a set that improves the coverage as much as possible, we get that the weight of G_i is more than that of G_{i-1} by at least Δ_{i-1}/k. Hence, $\Delta_i \leq \Delta_{i-1} - \frac{\Delta_{i-1}}{k} = (1-1/k)\Delta_{i-1}$. $\square$

To show we get the desired approximation ratio, we use our claim repeatedly to get

$$\text{weight}(O) - \text{weight}(G_k) = \Delta_k \leq (1-1/k)^k \Delta_0 = (1-1/k)^k \text{weight}(O) \tag{12}$$

and hence

$$\text{weight}(G_k) \geq (1-(1-1/k)^k)\text{weight}(O) \geq (1-1/e)\text{weight}(O) \tag{13}$$