Approximation Algorithms - Homework 2

Ilay Menahem, and Aaron Ross

January 3, 2026

Problem 1: Weighted Max-Cut Local Search

(a) Polynomial Convergence

Overview

Proof of Validity

Complexity Analysis

(b) Improved Bound with Specific Initialization

Algorithm Modification

Complexity Analysis

Problem 2: Hitting Set

Algorithm: Recursive Local Ratio

Approximation Analysis

Problem 3: Multiple Choice Maximum Coverage

Problem 4: Generalized Steiner Forest (GSF)

Let $G = (V, E)$ be a connected undirected graph and $T_1, T_2, \dots, T_t \subseteq V$. Let $w : E \to \mathbb{R}_{\geq 0}$ be a non-negative weight function over the edges. A generalized Steiner forest is a subset $E' \subseteq E$, such that for every $u, v \in T_i$ $(1 \leq i \leq t)$ there is a path in E' connecting u and v. In the generalized Steiner forest problem the objective is to find a generalized Steiner forest of minimum weight. (a) Let $T = \bigcup_{i=1}^{t} T_i$. Assume $|T_i| \geq 2$ for every $1 \leq i \leq t$ and consider $w' : E \to \mathbb{R}$ defined as follows. For an edge (u, v), $w'(u, v) = |\{u, v\} \cap T|$. Show that any minimal generalized Steiner forest of G is a 2-approximation with respect to the weight function w'. Note: a generalized Steiner forest E' of G is minimal if there is no $F \subsetneq E'$ which is also a generalized Steiner forest of G. (b) Suggest a 2-approximation algorithm for the generalized Steiner forest problem using the local ratio technique.

(a) Minimal GSF is a 2-approximation for w'

Claim: Any minimal generalized Steiner forest E' of G is a 2-approximation with respect to the weight function w'.

Lemma 1. *For any T and E the following equality holds:*

$$w'(E) = \sum_{v \in T} \deg_E(v)$$

Proof.

$$w'(E) = \sum_{(u,v) \in E} w'(u, v) = \sum_{(u,v) \in E} |\{u, v\} \cap T| = \sum_{(u,v) \in E} (\mathbf{1}_{u \in T} + \mathbf{1}_{v \in T})$$
$$= \sum_{v \in T} |\{(u, v) \in E\}| = \sum_{v \in T} \deg_E(v)$$

□

Proof. **Lower Bound on OPT:** Consider a optimal solution E^*. Since $\forall 1 \leq i \leq t, |T_i| \geq 2$, we know that any $v \in T$ must be connected to at least one other vertex in T_i via some edge in E^*. Thus, each vertex in T contributes at least 1 to the total weight of E^*. Therefore, we have:

$$OPT = w'(E^*) = \sum_{v \in T} \deg_{E^*}(v) \geq \sum_{v \in T} 1 = |T|$$

Upper Bound on E': Let E' be a minimal generalized Steiner forest of G. Let $1 \leq k \leq t$ be the number of connected components in the subgraph $G' = (V, E')$, Since E' is minimal, removing any edge from any component would disconnect a path between some $u, v \in T_i$, Thus, each component is a tree. a connected subgraph of a tree is a tree, we will mark every connected component of vertices of T in the graph G' as $C_1, C_2, \dots, C_k$. For a tree with m vertices, the sum of degrees is $2(m - 1)$.

$$w'(E') = \sum_{u \in T} \deg_{E'}(u) = \sum_{C_i} 2(|C_i| - 1) \leq 2(|T| - 1) - k \leq 2|T|$$

Ratio: Combining the bounds, we have:

$$w'(E') \leq 2|T| \leq 2OPT$$

Thus, E' is a 2-approximation with respect to the weight function w'. $\square$

(b) Local Ratio Algorithm

we will use the local ratio technique to design a 2-approximation algorithm for the generalized Steiner forest problem, this will be done by recursively decomposing the weight functions into ones we can apply the result from part (a) on. in the end we will get a graph such that all edges connected to T have weight 0, and we can select any minimal generalized steiner forest of T.

Algorithm 1 2-Approximation for Generalized Steiner Forest using Local Ratio

```
1: function GSF($G = (V, E), T_1, T_2, \ldots, T_t, w$)
2:     if w is identically 0 then return $\emptyset$
3:     end if
4:     Define $\epsilon = \min_{(u,v)\in E|w(u,v)>0, u\in T \vee v\in T} w(u, v)$
5:     Define
6: end function
```

Problem 5: Knapsack with Partition Constraints

Consider the following variant of the Knapsack problem. The input consists of n items $I = \{1, \dots, n\}$, where item $i \in I$ has a weight $w_i \in \mathbb{Z}^+$ and a value $v_i \in \mathbb{Z}^+$. The items are partitioned into disjoint sets $S_1, S_2, \dots, S_m$; that is, $\bigcup_{j \in [m]} S_j = I$ and $S_j \cap S_\ell = \emptyset$ for all $j \neq \ell \in [m]$. Moreover, each set S_j has a bound $k(j)$. Also, we are given a knapsack capacity $B \in \mathbb{Z}^+$. A solution for the problem is a subset of items $P \subseteq I$ such that $\sum_{i \in P} w_i \leq B$ and $|P \cap S_j| \leq k(j)$ for all $j \in [m]$. The value of the solution P is $\sum_{i \in P} v_i$. The objective is to find a solution P of maximal value. Obtain an FPTAS for the problem.

Overview

The algorithm will be based on initiation of an array to follow dominant values for each weight up to the knapsack capacity B, we will go over each partition and update the array based on the items in the partition and their respective bounds. finally we will return the maximum value in the array.

For the FPTAS, we will lower the resultion of the values of the items by a factor simlar to the standard knapsack FPTAS.

Algorithm Description

here is the pseudocode for the extension of the knapsack dynamic programming algorithm to handle partition constraints:

Algorithm 2 Knapsack with Partition Constraints

```
1: function KNAPSACK-PARTITION($I, S_1, S_2, \dots, S_m, k, B$)
2:     Initialize array $DP[0 \dots B]$ with $DP[0] = 0$ and $DP[w] = -\infty$ for $w > 0$
3:     for each partition $S_j$ do
4:         Create a temporary array $TempDP[0 \dots k(j)][0 \dots B]$
5:         Initialize $TempDP[0] = DP$
6:         Initialize $TempDP[1 \dots k(j)][0 \dots B] = -\infty$
7:         for each item $i \in S_j$ do
8:             for $count$ from $k(j)$ down to 1 do
9:                 for $weight$ from $w_i$ to $B$ do
10:                    $TempDP[count][weight] = \max(TempDP[count][weight], TempDP[count-1][weight - w_i] + v_i)$
11:                end for
12:            end for
13:        end for
14:        for $weight$ from 0 to $B$ do
15:            $DP[weight] = \max_{0 \leq count \leq k(j)} TempDP[count][weight]$
16:        end for
17:    end for
18:    return $\max_{0 \leq w \leq B} DP[w]$
19: end function
```

without loss of generality, we can assume that $\forall i, k(i) \leq |S_i|$ otherwise we can just set $k(i) = |S_i|$ without changing the Correctness of the algorithm. the complexity of this algorithm is $O(m \cdot k_{max} \cdot n \cdot B) = O(mn^2B)$, where k_{max} is the maximum bound among all partitions. this algorithm runs in pseudo-polynomial time, and we can convert it to an FPTAS by scaling the values of the items appropriately.

Lemma 2. *After processing partition S_j, for every weight $0 \leq w \leq B$, $DP[w]$ contains the maximum value of any feasible selection of items from the first j partitions with total weight exactly w.*

Proof. **Base Case:** Before processing any partitions, $DP[0] = 0$ and $DP[w] = -\infty$ for $w > 0$, which correctly represents the maximum value for weight 0 (no items selected) and infeasibility for positive weights.

Inductive Step: we need to show that $DP[w]$ still includes the maximum value for weight w, and we respect the bound $k(j)$ for partition S_j. Assume that after processing partition S_{j-1}, $DP[w]$ correctly represents the maximum value for weight w using items from the first $j-1$ partitions. When processing partition S_j, we consider each item in S_j and update the temporary array $TempDP$ such that $TempDP[count][weight]$ holds the maximum value achievable with exactly *count* items from S_j and total weight *weight*. By iterating through all items in S_j and updating $TempDP$, we ensure that all combinations of items up to the bound $k(j)$ are considered. Finally, we update $DP[w]$ to be $\max_{0 \leq count \leq k(j)} TempDP[count][w]$, this yields that $DP[w]$ now correctly reflects the maximum value for weight w using items from the first j partitions while respecting the bound $k(j)$. □

Proof. **Correctness:** using the lemma above, after processing all partitions, $DP[w]$ contains the maximum value of any feasible selection of items from all partitions with total weight exactly w. Thus, returning $\max_{0 \leq w \leq B} DP[w]$ gives the maximum value of any feasible selection of items with total weight at most B. □

FPTAS Construction

to enable the FPTAS, we will use the standard scaling technique. let $K = \frac{\epsilon B}{n}$.

Algorithm 3 FPTAS for Knapsack with Partition Constraints

```
1: function FPTAS-KNAPSACK-PARTITION($I, S_1, S_2, \ldots, S_m, k, B$)
2:     Scale weights: for each item $i$, set $w'_i = \lfloor w_i/K \rfloor$
3:     Set new capacity $B' = \lfloor B/K \rfloor$
4:     return KNAPSACK-PARTITION($I, S_1, S_2, \ldots, S_m, k, B'$)
5: end function
```

Proof. **Time Complexity:** the time complexity is $O(mn^2B') = O(mn^2\frac{B}{K}) = O(mn^3/\epsilon)$, which is polynomial in n, m, and $1/\epsilon$.

Approximation Ratio: now we will show that the solution returned by the FPTAS is within a factor of $(1-\epsilon)$ of the optimal solution. Let P^* be the optimal solution for the original problem with total weight $W^* \leq B$ and value V^*. □