Approximation Algorithms - Homework 2

Ilay Menahem, and Aaron Ross

January 27, 2026

Problem 1: Weighted Max-Cut Local Search

auxiliary definitions:

- $(S, \bar{S})$: a partition of the vertices of G s.t $S = V \setminus \bar{S}$.
- $W(S, \bar{S})$: the total weight of edges crossing the cut $(S, \bar{S})$. $W(S, \bar{S}) = \sum_{(u,v)\in E, u\in S, v\in \bar{S}} w(u, v)$.
- W_k: the weight of the cut after k iterations.
- W_{total}: the total weight of all edges in the graph. $W_{total} = \sum_{(u,v)\in E} w(u, v)$.

(a) Termination in Polynomial Time

Problem Statement

for a given graph $G = (V, E)$ with non-negative edge weights $w : E \to \mathbb{R}_{\geq 0}$ and a parameter $\epsilon \in (0, 1)$, show that the number of local improvement iterations until the Modified $LS_{Max-Cut}$ algorithm reaches an approximate local optimum is polynomial in the input size.

Proof. In the Modified $LS_{Max-Cut}$ algorithm, an iteration consists of moving a vertex v from one side of the partition to the other only if the weight of the cut increases by a factor of more than $(1 + \frac{\epsilon}{n})$.

Lemma 1. *For any $k \geq 0$,*

$$W_k \geq \left(1 + \frac{\epsilon}{n}\right)^k W_0 \tag{1}$$

Proof. By the definition of the algorithm, for any iteration $i \geq 0$, the new weight W_{i+1} satisfies:

$$W_{i+1} \geq \left(1 + \frac{\epsilon}{n}\right) W_i \tag{2}$$

By induction on k.

- Base case: $k = 0$, $W_0 = W_0$.
- Inductive step: assume $W_k \geq \left(1 + \frac{\epsilon}{n}\right)^k W_0$, then $W_{k+1} \geq \left(1 + \frac{\epsilon}{n}\right) W_k \geq \left(1 + \frac{\epsilon}{n}\right)^{k+1} W_0$.

□

The weight of any cut is bounded from above by the total weight of all edges in the graph. the final weight W_{final} satisfies $W_{final} \leq W_{total}$. Combining this with inequality (1), we have:

$$W_{total} \geq W_k \geq \left(1 + \frac{\epsilon}{n}\right)^k W_0$$

$$\ln\left(\frac{W_{total}}{W_0}\right) \geq k \ln\left(1 + \frac{\epsilon}{n}\right)$$

$$k \leq \frac{\ln(W_{total}/W_0)}{\ln\left(1 + \frac{\epsilon}{n}\right)}$$

To bound the denominator, we use the known inequality $\ln(1 + x) \geq \frac{x}{1+x}$, Substituting $x = \frac{\epsilon}{n}$:

$$\ln\left(1 + \frac{\epsilon}{n}\right) \geq \frac{\frac{\epsilon}{n}}{1 + \frac{\epsilon}{n}}$$

Substituting this back into inequality:

$$k \leq \frac{\ln(W_{total}/W_0)}{\ln\left(1 + \frac{\epsilon}{n}\right)} \leq \frac{\ln(W_{total}/W_0)}{\frac{\frac{\epsilon}{n}}{1+\frac{\epsilon}{n}}} = \left(1 + \frac{\epsilon}{n}\right) \frac{n}{\epsilon} \ln\left(\frac{W_{total}}{W_0}\right)$$

Since $\epsilon \in (0, 1)$, we have $(1 + \frac{\epsilon}{n}) < 2$ for $n \geq 1$. Thus:

$$k < 2\frac{n}{\epsilon} \ln\left(\frac{W_{total}}{W_0}\right)$$

Complexity Analysis

To prove k is polynomial in the input size, we analyze the term $\ln(W_{total})$. Let $\langle \text{input} \rangle$ denote the size of the input in bits.

- The term $\frac{n}{\epsilon}$ is clearly polynomial in the input parameters n and $\frac{1}{\epsilon}$.
- Let w_{max} be the maximum edge weight. The value W_{total} is at most $|E| \cdot w_{max} \leq n^2 w_{max}$.
- The number of bits required to represent w_{max} is part of the input size. Thus, $\ln(w_{max})$ is proportional to the number of bits and is linear in the input size.
- Therefore, $\ln(W_{total}) \leq \ln(n^2) + \ln(w_{max}) = 2\ln(n) + \ln(w_{max})$, which is polynomial in the input size.

Conclusion: Since both $\frac{n}{\epsilon}$ and $\ln(W_{total}/W_0)$ are polynomial in the input size (specifically, polynomial in n, $1/\epsilon$, and the number of bits representing the weights), the total number of iterations k is polynomial. □

(b) Strongly Polynomial Time Analysis

Problem Statement: Suppose we change Step 1 of the algorithm to select the initial cut $(S, \bar{S})$ where $S = \{v^*\}$ and $v^* = \arg\max_{v \in V} w(E(\{v\}, V \setminus \{v\}))$. Show that the number of local improvement iterations is strongly polynomial (dependent only on n and ϵ).

Proof. Let $d(v) = w(E(\{v\}, V \setminus \{v\}))$ denote the weighted degree of vertex v.

The new initialization step selects the vertex v^* with the maximum weighted degree. Thus, the weight of the initial cut is $W_0 = d(v^*)$.

Step 1: Lower bound on the initial cut W_0 We know that the sum of the weighted degrees of all vertices equals twice the total edge weight (by the Handshaking Lemma generalized for weighted graphs):

$$\sum_{v \in V} d(v) = 2W_{total}$$

Since v^* is the vertex with the maximum weighted degree, $d(v^*)$ must be at least the average weighted degree:

$$d(v^*) \geq \frac{1}{n} \sum_{v \in V} d(v) = \frac{2W_{total}}{n}$$

Thus, we have the lower bound:

$$W_0 \geq \frac{2W_{total}}{n} \tag{3}$$

Step 2: Upper bound on the ratio W_{total}/W_0 Using the inequality (3), we can bound the ratio of the maximum possible weight to the initial weight:

$$\frac{W_{total}}{W_0} \leq \frac{W_{total}}{\frac{2W_{total}}{n}} = \frac{n}{2}$$

Step 3: Bounding the number of iterations From part (a), we established that the number of iterations k is bounded by:

$$k < 2\frac{n}{\epsilon} \ln\left(\frac{W_{total}}{W_0}\right)$$

Substituting the bound from Step 2 into this inequality:

$$k < 2\frac{n}{\epsilon} \ln\left(\frac{n}{2}\right)$$

Conclusion: The bound on the number of iterations k is now $O(\frac{n}{\epsilon} \ln n)$ □

Problem 2: Hitting Set

Input: A collection of non-empty sets $\mathcal{C} = \{S_1, \ldots, S_m\}$ over a universe U, and a non-negative weight function $w : U \to \mathbb{R}_{\geq 0}$.
Output: A subset $H \subseteq U$ such that $H \cap S_i \neq \emptyset$ for all i, minimizing $\sum_{u \in H} w(u)$.
Parameter: Let $S_{max} = \max_{S \in \mathcal{C}} |S|$.

The Algorithm

Algorithm 1 LocalRatioHittingSet$(\mathcal{C}, U, w)$

1: $U_0 \leftarrow \{u \in U \mid w(u) = 0\}$
2: **if** $\forall S \in \mathcal{C}$, $S \cap U_0 \neq \emptyset$ **then**
3: **return** U_0
4: **end if**
5: Select any $S \in \mathcal{C}$ such that $S \cap U_0 = \emptyset$
6: $\epsilon \leftarrow \min_{u \in S} w(u)$
7: Define $w_1(u) = \begin{cases} \epsilon & \text{if } u \in S \\ 0 & \text{otherwise} \end{cases}$
8: Define $w_2(u) = w(u) - w_1(u)$ for all $u \in U$
9: $H \leftarrow$ LocalRatioHittingSet$(\mathcal{C}, U, w_2)$
10: **return** H

Proof of Validity

!!! TODO: Prove the algorithm returns a valid hitting set. !!!

Approximation Analysis

Theorem 1. *The algorithm produces an S_{max}-approximation for the Weighted Hitting Set problem.*

Proof. We proceed by induction on the size of the set $U^+ = \{u \in U \mid w(u) > 0\}$, which is the number of elements with strictly positive weight.

Base Case: If the algorithm terminates at Step 2, it returns U_0. For all $u \in U_0$, $w(u) = 0$. Thus, the total cost is $w(U_0) = 0$. Since the optimal cost is non-negative, $w(U_0) \leq S_{max} \cdot OPT$, satisfying the approximation ratio.

Inductive Step: Let $k = |U^+|$ be the number of elements with strictly positive weight in w. Assume the approximation holds for any instance where the number of positive-weight elements is strictly less than k.

Consider the weight function w_2 defined in Step 5. By our choice of $\epsilon = \min_{u \in S} w(u)$, there is at least one element $u^* \in S$ such that $w(u^*) = \epsilon$. Consequently, for this element, $w_2(u^*) = w(u^*) - \epsilon = 0$. Since $w_2(u) \leq w(u)$ for all u, and at least one element (u^*) has

transitioned from positive weight in w to zero weight in w_2, the number of positive-weight elements in w_2 is strictly less than k.

Therefore, we can apply the inductive hypothesis to the recursive call on $(\mathcal{C}, U, w_2)$. This call returns a set H satisfying:

$$w_2(H) \leq S_{max} \cdot OPT(w_2) \tag{4}$$

where $OPT(w_2)$ is the weight of the optimal hitting set under w_2.

We analyze the approximation with respect to the "easy" weight function w_1:

1. **Cost of H under w_1:** Since $w_1(u) = \epsilon$ for $u \in S$ and 0 otherwise:

$$w_1(H) = \sum_{u \in H} w_1(u) = \sum_{u \in H \cap S} \epsilon = |H \cap S| \cdot \epsilon$$

   Since H is a valid hitting set, it must hit S. Regardless of which elements it picks, we know $|H \cap S| \leq |S|$. Furthermore, by definition $S_{max} \geq |S|$. Thus:

$$w_1(H) \leq |S| \cdot \epsilon \leq S_{max} \cdot \epsilon$$

2. **Cost of OPT under w_1:** Any feasible hitting set H^* must hit the specific set S selected in Step 3. Therefore, $H^* \cap S \neq \emptyset$. The minimum cost to hit S under w_1 is exactly ϵ (by picking any single element $u \in S$). Thus:

$$OPT(w_1) \geq \epsilon$$

3. **Ratio for w_1:** Combining the above:

$$w_1(H) \leq S_{max} \cdot \epsilon \leq S_{max} \cdot OPT(w_1)$$

Total Weight: The total weight of the solution H under w is:

$$w(H) = w_1(H) + w_2(H)$$

Using the inductive hypothesis (4) and the bound for w_1:

$$w(H) \leq S_{max} \cdot OPT(w_1) + S_{max} \cdot OPT(w_2)$$

$$w(H) \leq S_{max} \cdot (OPT(w_1) + OPT(w_2))$$

Since any optimal solution H^* for w is also a valid hitting set for w_1 and w_2, we have $OPT(w_1) + OPT(w_2) \leq w_1(H^*) + w_2(H^*) = w(H^*) = OPT(w)$. Therefore:

$$w(H) \leq S_{max} \cdot OPT(w)$$

□

Complexity Analysis

- **Recursive Depth:** In every recursive step, we select a set S that is *not* currently hit by U_0. We calculate $\epsilon = \min_{u \in S} w(u)$. After subtracting w_1, at least one element $u^* \in S$ (specifically the one with weight ϵ) will have its weight reduced to 0 in w_2.
- Consequently, the number of elements with positive weight strictly decreases in each recursive call. Therefore, there are at most $|U| = n$ recursive calls.
- **Per-Step Cost:** Finding a set unhit by U_0 takes $O(m \cdot S_{max})$. Updating weights takes $O(S_{max})$ because we only update weights for one set.
- **Total Complexity:** The algorithm runs in polynomial time, specifically $O(n \cdot m \cdot S_{max})$.

Problem 3: Multiple Choice Maximum Coverage

Problem Statement

Let the universe of elements be Ω. We are given k collections $\mathcal{C}_1, \ldots, \mathcal{C}_k$. Let $J = (j_1, \ldots, j_k)$ be the indices of the sets chosen by the **local search** algorithm. Let $S_{local} = \bigcup_{i=1}^{k} S_{i,j_i}$ be the set of elements covered by the local solution. Let $O = (o_1, \ldots, o_k)$ be the indices of the sets chosen by the **optimal** solution. Let $S_{opt} = \bigcup_{i=1}^{k} S_{i,o_i}$ be the set of elements covered by the optimal solution.

Definitions from Hint

For each $i \in \{1, \ldots, k\}$, let U_{-i} denote the union of all sets chosen by the local solution *except* the one from collection i:

$$U_{-i} = \bigcup_{r \in \{1,\ldots,k\} \setminus \{i\}} S_{r,j_r}$$

Using this notation, we define the sets A_i and B_i as given in the hint:

$$A_i = S_{i,j_i} \setminus U_{-i}$$
$$B_i = S_{i,o_i} \setminus U_{-i}$$

- A_i represents the **unique contribution** of the local set S_{i,j_i} to the current cover S_{local}.
- B_i represents the **potential contribution** of the optimal set S_{i,o_i} if we were to swap it into the local solution at index i.

Proof of Validity

!!! TODO: Prove the algorithm returns a valid solution. !!!

Approximation Analysis

Lemma 2. *For every $i \in \{1, \ldots, k\}$, $|B_i| \leq |A_i|$.*

Proof. Since $J = (j_1, \ldots, j_k)$ is returned by the algorithm, it is a local optimum. This means that changing any single index j_i to another index (specifically o_i) cannot strictly increase the size of the union.

The size of the current local solution is:

$$|S_{local}| = |S_{i,j_i} \cup U_{-i}| = |U_{-i}| + |S_{i,j_i} \setminus U_{-i}| = |U_{-i}| + |A_i|$$

Consider the neighbor solution J' where we replace j_i with o_i. The set covered by this new solution is $S' = S_{i,o_i} \cup U_{-i}$. The size of this new solution is:

$$|S'| = |S_{i,o_i} \cup U_{-i}| = |U_{-i}| + |S_{i,o_i} \setminus U_{-i}| = |U_{-i}| + |B_i|$$

By the local optimality condition, $|S'| \leq |S_{local}|$. Therefore:

$$|U_{-i}| + |B_i| \leq |U_{-i}| + |A_i| \implies |B_i| \leq |A_i|$$

□

Theorem 2. *The local search algorithm is a $\frac{1}{2}$-approximation. That is, $|S_{local}| \geq \frac{1}{2}|S_{opt}|$.*

Proof. We can decompose the optimal solution size into two parts: elements that are already covered by our local solution, and elements that are not.

$$|S_{opt}| = |S_{opt} \cap S_{local}| + |S_{opt} \setminus S_{local}|$$

Bounding the first term: Trivially, $|S_{opt} \cap S_{local}| \leq |S_{local}|$.

Bounding the second term: Consider an element $x \in S_{opt} \setminus S_{local}$. Since $x \in S_{opt}$, there must exist some index i such that $x \in S_{i,o_i}$. Since $x \notin S_{local}$, it means x is not covered by any set in the local solution. Specifically, $x \notin S_{i,j_i}$ and $x \notin U_{-i}$.

Because $x \in S_{i,o_i}$ and $x \notin U_{-i}$, by definition $x \in B_i$. Therefore, every element in $S_{opt} \setminus S_{local}$ must belong to at least one set B_i:

$$S_{opt} \setminus S_{local} \subseteq \bigcup_{i=1}^{k} B_i$$

Taking the cardinality:

$$|S_{opt} \setminus S_{local}| \leq \sum_{i=1}^{k} |B_i|$$

Using Lemma 2, we know $\sum |B_i| \leq \sum |A_i|$.

$$|S_{opt} \setminus S_{local}| \leq \sum_{i=1}^{k} |A_i|$$

Now, observe the sets A_i. By definition, A_i consists of elements in S_{i,j_i} that are *not* in any other S_{r,j_r}. Therefore, the sets $A_1, \ldots, A_k$ are pairwise disjoint subsets of S_{local}.

$$\sum_{i=1}^{k} |A_i| = \left| \bigcup_{i=1}^{k} A_i \right| \leq |S_{local}|$$

Combining the bounds:

$$\begin{aligned} |S_{opt}| &= |S_{opt} \cap S_{local}| + |S_{opt} \setminus S_{local}| \\ &\leq |S_{local}| + \sum_{i=1}^{k} |A_i| \\ &\leq |S_{local}| + |S_{local}| \\ &= 2|S_{local}| \end{aligned}$$

Thus, $|S_{local}| \geq \frac{1}{2}|S_{opt}|$. □

Complexity Analysis

Let $n = |\Omega|$ be the total number of elements in the universe. Let $M = \sum_{i=1}^{k} \ell_i$ be the total number of sets available across all k collections.

- **Monotonic Improvement:** In each iteration of the local search, the algorithm only updates the current solution if the total value $V(j_1, \ldots, j_k)$ strictly increases.
- **Bounded Iterations:** Since the value corresponds to the number of covered elements, it is an integer bounded between 0 and n. Therefore, the value can increase at most n times. This implies the 'while' loop runs at most n times.
- **Cost per Iteration:** In each iteration, the algorithm checks every possible single-swap neighbor. There are exactly $M - k$ such neighbors. Calculating the coverage of a neighbor takes polynomial time in the input size.
- **Conclusion:** The total running time is bounded by $O(n \cdot M \cdot \text{poly(input size)})$, which is polynomial.

Problem 4: Generalized Steiner Forest (GSF)

Let $G = (V, E)$ be a connected undirected graph and $T_1, T_2, \ldots, T_t \subseteq V$. 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 3. *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, and consider the forest $G' = (V, E')$. Since E' is minimal, no connected component of G' that contains terminals can contain a cycle (otherwise we could remove an edge on the cycle and preserve all required terminal-to-terminal connections inside that component). Hence, every such component is a tree.

Fix a connected component H of G' that contains at least one terminal. Let $T_H := T \cap V(H)$ and let $S_H := V(H) \setminus T_H$ be the non-terminals of H. We claim that no vertex of S_H can be a leaf in the tree H: if some $x \in S_H$ had degree 1 in H, then removing its

unique incident edge would not disconnect any pair $u, v \in T_i$ (since x is not a terminal), contradicting minimality. Therefore, every $x \in S_H$ satisfies $\deg_H(x) \geq 2$, so

$$\sum_{x \in S_H} \deg_H(x) \;\geq\; 2|S_H|.$$

Since H is a tree, $\sum_{v \in V(H)} \deg_H(v) = 2(|V(H)| - 1) = 2(|T_H| + |S_H| - 1)$. Thus,

$$\sum_{v \in T_H} \deg_H(v) = 2(|T_H| + |S_H| - 1) - \sum_{x \in S_H} \deg_H(x) \leq 2(|T_H| + |S_H| - 1) - 2|S_H| = 2|T_H| - 2.$$

Summing over all terminal-containing components (say there are $c \geq 1$ of them) gives

$$w'(E') = \sum_{v \in T} \deg_{E'}(v) \;\leq\; \sum_{H} (2|T_H| - 2) \;=\; 2|T| - 2c \;\leq\; 2|T|.$$

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'. □

(b) Local Ratio Algorithm

Algorithm

here are three primitive procedures that are used in the algorithm:

- isGsf($F, \{T_1, \ldots, T_k\}$): This procedure checks if the forest F is a generalized Steiner forest.

Algorithm 2 isGsf($F, \{T_1, \ldots, T_k\}$)

```
G_F ← (V, F)
for i ∈ {1, ..., k} do
    if not isConnected(G_F, T_i) then
        return false
    end if
end for
return true
```

The complexity of this algorithm is $O(k \cdot (|V| + |F|))$.

- UpdateActiveSet($e = (u, v), T$): This procedure propagates the "active" status to new nodes (the "infection" step).
  - If $u \in T$ and $v \notin T$, update $T \leftarrow T \cup \{v\}$.
  - If $v \in T$ and $u \notin T$, update $T \leftarrow T \cup \{u\}$.

– If both or neither are in T, do nothing.

The complexity of this algorithm is $O(1)$.

- PRUNEFOREST$(F, S, \{T_1, T_2, \ldots, T_t\})$: This procedure makes the forest F minimal by removing any edge whose deletion does not break the generalized Steiner property.

Algorithm 3 PruneForest$(F, S, \{T_1, T_2, \ldots, T_t\})$

while $S \neq \emptyset$ **do**
  $e \leftarrow \text{POP}(S)$
  if ISGSF$(F \setminus \{e\}, \{T_1, \ldots, T_t\})$ **then**
    $F \leftarrow F \setminus \{e\}$
  end if
end while

The complexity of this algorithm is $O(|S| \cdot k(|V| + |F|))$ in the worst case.

Algorithm 4 2-Approximation for Generalized Steiner Forest using Local Ratio

function GSF$(G = (V, E), \{T_1, T_2, \ldots, T_t\}, w)$
  $F \leftarrow \emptyset$
  $\mathcal{T} \leftarrow \bigcup_{i=1}^{t} T_i$
  $S \leftarrow \mathit{Stack}(\emptyset)$
  while NOT ISGSF$(F, \{T_1, \ldots, T_t\})$ **do**
    $E_{active} \leftarrow \{(u, v) \in E \mid \{u, v\} \cap \mathcal{T} \neq \emptyset\}$
    $w'(e) \leftarrow \begin{cases} |\{u, v\} \cap \mathcal{T}| & \text{if } e = (u, v) \in E_{active} \\ 0 & \text{otherwise} \end{cases}$
    $\epsilon \leftarrow \min_{e \in E_{active}} \frac{w(e)}{w'(e)}$
    $w(e) \leftarrow w(e) - \epsilon \cdot w'(e)$
    $E^* \leftarrow \{e \in E_{active} \mid w(e) = 0\} \setminus F$
    $F \leftarrow F \cup E^*$
    Push each $e \in E^*$ onto S
    for each $e \in E^*$ do: UPDATEACTIVESET$(e, \mathcal{T})$
  end while
  PRUNEFOREST$(F, S, \{T_1, T_2, \ldots, T_t\})$
  return F
end function

Proof of Correctness

We claim that assuming the graph is connected, the algorithm terminates and returns a valid Generalized Steiner Forest F.

before we prove the correctness of the algorithm, we should prove the following

Lemma 4. *For any input for which the graph is connected, the loop runs at most $|E|$ times.*

Proof. In each iteration, we select $\epsilon = \min_{e \in E_{active}} \frac{w(e)}{w'(e)}$. This ensures at least one edge e^* in E_{active} reaches weight 0. Edges with zero weight are added to F (if not already present) and never removed during the loop. Since $|E|$ is finite and each iteration effectively processes at least one new edge towards inclusion in F, the loop runs at most $|E|$ times. □

1. **Termination:** By the Lemma above, the **while** loop runs at most $|E|$ times. Since each iteration takes finite time, the loop must terminate. Furthermore, since the graph G is connected, a valid GSF exists (e.g., the spanning tree). The loop continues adding edges until the condition $\textsc{isGsf}(F, \{T_i\})$ is met. In the worst case, F grows to include all edges in E, which is definitely a GSF. Thus, the algorithm terminates with a valid solution.

2. **Feasibility (Phase 1):** The **while** loop terminates only when $\textsc{isGsf}(F, \{T_i\})$ returns true. Therefore, the set F obtained after the loop satisfies the Generalized Steiner Forest property.

3. **Feasibility (Phase 2 - Pruning):** The pruning phase iterates through the edges in S and removes an edge e from F only if $\textsc{isGsf}(F \setminus \{e\}, \{T_i\})$ remains true. This explicitly maintains the invariant that F is a valid Generalized Steiner Forest. Thus, the final set F returned by the algorithm is a valid solution.

Approximation Analysis

Theorem 3. *Let F be the forest returned by the algorithm and let OPT be an optimal generalized Steiner forest for the original weights w. Then $w(F) \leq 2 \cdot w(OPT)$.*

Proof. We apply the local ratio technique.

Step 1: Weight Decomposition. The algorithm decomposes the weight function as:

$$w = \sum_{j=1}^{k} \epsilon_j w'_j + w^{(k)}$$

where each $\epsilon_j > 0$, each $w'_j \geq 0$, and $w^{(k)}$ is the residual after k iterations.

Lemma 5. *For any j, $w^{(k)}(e) \geq 0$ for all $e \in E$.*

Proof. By induction on the number of iterations.

- Base case: $k = 0$. $w^{(0)}(e) = w(e) \geq 0$
- Inductive step: assume $w^{(k)}(e) \geq 0$ for all $e \in E$, then since $\epsilon_j \cdot w'_j(e) \leq w(e)$, we have $w^{(k+1)}(e) = w(e) - \epsilon_j \cdot w'_j(e) \geq 0$

□

Step 2: Cost of the Solution. Edges are added to F only when they become tight (weight reaches 0). Since weights only decrease, every edge in F satisfies $w^{(k)}(e) = 0$. Pruning only removes edges, so:

$$w(F) = \sum_{j=1}^{k} \epsilon_j \cdot w'_j(F)$$

Step 3: Lower Bound on OPT. For any feasible solution X, and $X = OPT$ in particular:

$$w(X) = \sum_{j=1}^{k} \epsilon_j \cdot w'_j(X) + w^{(k)}(X) \geq \sum_{j=1}^{k} \epsilon_j \cdot w'_j(X)$$

Step 4: Applying Part (a). We show that for each iteration j: $w'_j(F) \leq 2 \cdot w'_j(OPT)$.

In each iteration, $w'_j(u, v) = |\{u, v\} \cap \mathcal{T}_j|$ where $\mathcal{T}_j \supseteq T = \bigcup_i T_i$ is the current active set. Since the pruning step ensures F is a *minimal* GSF, we apply part (a): the returned forest F is a 2-approximation with respect to w'_j.

Note: The active set $\mathcal{T}_j$ may grow to include non-terminal vertices via the infection step. However, part (a)'s proof relies only on:

- Lower bound: Each original terminal $v \in T$ must have $\deg_{OPT}(v) \geq 1$, so $w'_j(OPT) \geq |T|$.
- Upper bound: After pruning, F is a forest with no non-terminal leaves, so $w'_j(F) \leq 2|T|$ by the degree-sum argument in part (a).

Thus $w'_j(F) \leq 2|T| \leq 2 \cdot w'_j(OPT)$ holds regardless of the infected non-terminals.

Step 5: Conclusion.

$$w(F) = \sum_{j=1}^{k} \epsilon_j \cdot w'_j(F) \leq \sum_{j=1}^{k} 2\epsilon_j \cdot w'_j(OPT) = 2 \sum_{j=1}^{k} \epsilon_j \cdot w'_j(OPT) \leq 2 \cdot w(OPT)$$

□

Complexity Analysis

The algorithm runs in polynomial time.

1. **Number of Iterations:** As proven in the Lemma, the **while** loop executes at most $|E|$ times.
2. **Cost per Iteration:** Inside the loop, the most expensive operations are checking isGsf and updating edge weights. isGsf takes $O(k(|V| + |E|))$ time. Updating weights and finding ϵ takes $O(|E|)$.
3. **Pruning Phase:** The pruning step iterates through the stack S, which contains at most $|E|$ edges. thus the complexity of the pruning step is $O(|S| \cdot k(|V| + |F|)) = O(|E| \cdot k(|V| + |E|))$.

Since the number of iterations is bounded by $|E|$ and the work per iteration is polynomial, the total running time is polynomial. Specifically, a loose upper bound is $O(|E| \cdot k \cdot (|V| + |E|))$, which is polynomial in the input size.

Problem 5: Knapsack with Partition Constraints

Consider the following variant of the Knapsack problem. The input consists of n items $I = \{1, \ldots, n\}$, 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 $S_1, S_2, \ldots, S_m$; 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

we will maintain a *domination list* (a list of nondominated (weight, value) pairs). A pair (w, v) *dominates* (w', v') if $w \leq w'$ and $v \geq v'$. Dominated pairs can be discarded without affecting optimality.

For the FPTAS, we will apply the standard scaling technique (as in the usual knapsack FPTAS) and run the same domination-list DP on the scaled instance.

Algorithm Description

Domination-list primitives.

- $\textsc{Shift}(L, \Delta w, \Delta v) = \{(w + \Delta w, v + \Delta v) : (w, v) \in L\}$.
- $\textsc{Prune}(L)$ removes all dominated pairs from L (and may also remove pairs with $w > B$). One implementation: sort pairs by increasing weight; sweep, keeping a pair only if its value is strictly larger than the maximum value seen so far. Its complexity is $O(|L| \log |L|)$ due to sorting.
- $\textsc{Convolve}(L_1, L_2)$ computes all pairwise sums of pairs from L_1 and L_2:

$$\textsc{Convolve}(L_1, L_2) = \{(w_1 + w_2, v_1 + v_2) : (w_1, v_1) \in L_1, (w_2, v_2) \in L_2\}$$

. Its complexity is $O(|L_1| \cdot |L_2|)$.

Algorithm 5 Knapsack with Partition Constraints (Domination Lists)

function KNAPSACK-PARTITION($I, S_1, S_2, \ldots, S_m, k, B$)
  $L \leftarrow \{(0,0)\}$ ▷ global nondominated set of feasible (weight,value) pairs
  for each partition S_j **do**
    ▷ Compute nondominated options using items from S_j with at most $k(j)$ picks
    $L^{(0)} \leftarrow \{(0,0)\}$
    for $c = 1$ to $k(j)$ **do** $L^{(c)} \leftarrow \emptyset$
    end for
    for each item $i \in S_j$ **do**
      for c from $k(j)$ down to 1 **do**
        $L^{(c)} \leftarrow \textsc{Prune}\Big(L^{(c)} \cup \textsc{Shift}(L^{(c-1)}, w_i, v_i)\Big)$
      end for
    end for
    ▷ Combine the chosen items from S_j with the global list
    $L_{\text{next}} \leftarrow \emptyset$
    for $c = 0$ to $k(j)$ **do**
      $L_{\text{next}} \leftarrow \textsc{Prune}(L_{\text{next}} \cup \textsc{Convolve}(L, L^{(c)}))$
    end for
    $L \leftarrow L_{\text{next}}$
  end for
  return $\max\{v : (w,v) \in L\}$
end function

Proof of Correctness

Without loss of generality, assume $\forall j,\ k(j) \le |S_j|$ (otherwise replace $k(j)$ by $|S_j|$).

Lemma 6. *After processing partitions $S_1, \ldots, S_j$, the list L contains exactly the nondominated pairs (w, v) such that there exists a feasible selection of items from the first j partitions of total weight w and total value v (respecting all bounds).*

Proof. **Base case ($j = 0$):** $L = \{(0,0)\}$ is the unique feasible pair (select nothing), and it is nondominated.

Inductive step: Assume the claim holds after $S_1, \ldots, S_{j-1}$. The lists $L^{(c)}$ are constructed with an explicit counter c, and PRUNE only removes dominated pairs, hence each $L^{(c)}$ represents exactly the nondominated achievable pairs using exactly c items from S_j. When convolving $L^{(c)}$ with L_{new}, we consider all ways to add c items from S_j to previously selected items (from partitions 1 to $j-1$). The final pruning step ensures that only nondominated pairs remain in L_{new}. Thus, after processing S_j, L contains exactly the nondominated pairs corresponding to feasible selections from the first j partitions. □

Theorem 4. *The algorithm returns the optimal objective value for the knapsack with partition constraints problem.*

Proof. After processing all partitions, every feasible solution corresponds to some pair in the (unpruned) constructed set, and pruning preserves at least one representative of every

attainable optimum value (for some weight). Thus $\max\{v : (w, v) \in L\}$ equals the optimal objective value under capacity B and the partition bounds. □

Time Complexity

We will note $V = \sum_{i \in I} v_i$, and $k_{\max} = \max_{j \in [m]} k(j) \leq n$. The outer loop runs m times (once per partition).

Inside, we have two main parts:

- Updating $L^{(c)}$ for each item in S_j: we do $|S_j| \leq n$ iterations, each involving at most $k_{\max}$ shifts and prunes. Each prune takes $O(|L^{(c)}| \log |L^{(c)}|)$ time, and $|L^{(c)}| \leq \min(B, V)$ (since weights are at most B and values at most V). Thus this part takes $O(n \cdot k_{\max} \cdot \min(B, V) \log(\min(B, V)))$ time per partition.
- Combining $L^{(c)}$ into L_{next}: we do at most $k_{\max}+1$ convolutions and prunes. Each convolution takes $O(|L| \cdot |L^{(c)}|) \leq O(\min(B, V)^2)$ time, and each prune takes $O(|L_{\text{next}}| \log |L_{\text{next}}|) \leq O(\min(B, V) \log(\min(B, V)))$ time. Thus this part takes $O(k_{\max} \cdot \min(B, V)^2)$ time per partition.

Thus the total complexity is $O(m \cdot n \cdot k_{\max} \cdot \min(B, V)^2) = O(m \cdot n^2 \cdot \min(B, V)^2)$, which is polynomial in n, m, and $\min(B, V)$.

FPTAS Construction

To enable the FPTAS, we will use the standard scaling technique. Let $V_{\max} = \max_{i \in I} v_i$, $K = \frac{\epsilon V_{\max}}{n}$.

Algorithm 6 FPTAS for Knapsack with Partition Constraints

```
function FPTAS-Knapsack-Partition(I, S_1, S_2, ..., S_m, k, B)
    Scale values: for each i ∈ I, set v'_i = ⌊v_i / K⌋
    return Knapsack-Partition(I, S_1, S_2, ..., S_m, k, B)
end function
```

Proof. **Time Complexity:** The time complexity is $O\left(m \cdot n^2 \cdot \min(B, V')^2\right)$

$$\min(B, V') \leq V' = \sum_{i \in I} v'_i \leq \sum_{i \in I} \frac{v_i}{K} = \frac{\sum_{i \in I} v_i}{K} \leq \frac{nV_{\max}}{K} = \frac{nV_{\max}}{\epsilon V_{\max}/n} = \frac{n^2}{\epsilon}$$

Thus the time complexity is $O\left(m \cdot n^2 \cdot \left(\frac{n^2}{\epsilon}\right)^2\right) = O\left(\frac{m \cdot n^6}{\epsilon^2}\right)$, which is polynomial in n, m, and $\frac{1}{\epsilon}$. □

Proof. **Approximation Ratio:** For the optimal set P^*, the difference between the true value and the scaled value is:

$$\sum_{i \in P^*} v_i - K \cdot \sum_{i \in P^*} v'_i < \sum_{i \in P^*} K = |P^*|K \leq nK = \epsilon V_{\max} \leq \epsilon \cdot OPT$$

Thus, the solution returned is at least $(1 - \epsilon) \cdot OPT$. □