Indian Institute of Technology, Ropar
Computer Science Department

Approximate Algorithms

Group 3
Ashutosh Rajput: 2023CSB1289
Nishant Sahni: 2023CSB1140
Aksh Sihag: 2023CSB1097
Tanisha Gupta: 2023CSB1170
N Yashwanth Bala: 2023CSB1135

Supervisor: Dr.Anil Shukla
Date: October 24, 2025

Contents

1. Introduction

Optimization problems are central to computer science and operations research. Many of these problems, such as those related to routing, covering, and resource allocation, are classified as **NP-hard**. This means that finding an exact, optimal solution in polynomial time is highly improbable under the widely accepted conjecture that P $\neq$ NP. In light of this computational intractability, the field of **Approximation Algorithms** provides a pragmatic approach. An approximation algorithm is a polynomial-time algorithm that, for a given optimization problem, is guaranteed to find a solution whose objective value is within a guaranteed factor of the optimal value. This guaranteed factor is known as the **approximation ratio** or **performance guarantee**.

This report explores the design and analysis of approximation algorithms for three fundamental NP-hard problems, each representing a distinct challenge and employing different algorithmic techniques:

1. **Maximum Satisfiability (MAX SAT):** An optimization variant of the classical Boolean Satisfiability problem. The goal is to find a truth assignment that maximizes the number of satisfied clauses in a given Conjunctive Normal Form (CNF) formula. We will examine approaches based on both randomized algorithms and the powerful technique of **Linear Programming (LP) Rounding**.
2. **k-Center Problem:** A facility location problem in a metric space. The objective is to select k points (centers) such that the maximum distance from any client point to its closest center (the covering radius) is minimized. The classical approach to this problem provides a tight 2-approximation ratio using a simple **greedy strategy**.
3. **Steiner Tree Problem:** A critical problem in network design. Given a graph with edge weights and a subset of required vertices (terminals), the aim is to find a minimum-weight tree that connects all the terminals, potentially using non-terminal vertices (Steiner points). We will detail the standard 2-approximation algorithm based on constructing a **Minimum Spanning Tree (MST)** in the metric closure.

Beyond the core algorithms, a significant focus is placed on the essential framework of **LP Rounding** as applied to the MAX SAT problem. The effectiveness of this technique relies fundamentally on the ability to solve the fractional Linear Program relaxation efficiently.

Appendix: The Ellipsoid Algorithm
The appendix of this report is dedicated to the **Ellipsoid Algorithm**. Although often slower in practice than the Simplex method, the Ellipsoid Algorithm is theoretically paramount because it was the first algorithm proven to solve Linear Programming problems in **polynomial time**. Its inclusion here is crucial to establishing the polynomial-time complexity of the LP relaxation step required for the $\frac{3}{4}$-approximation algorithm for MAX SAT.

2. MAX-SAT Problem

2.1. Definition

The **Maximum Satisfiability (MAX-SAT)** problem is the optimization counterpart to the Boolean Satisfiability (SAT) problem. While SAT asks if a formula is satisfiable, MAX-SAT seeks to find a truth assignment that satisfies the maximum number of clauses. The most general version of the problem is the **Weighted MAX-SAT (WMAX-SAT)** problem, which is formally defined as follows.

2.1.0.1 Input: The input is a tuple (X, C, w), where:

- $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of **Boolean variables**.
- $C = \{c_1, c_2, \ldots, c_m\}$ is a set of **clauses**. Each clause $c_j \in C$ is a disjunction (OR, $\vee$) of one or more literals. A literal is either a variable $x_i \in X$ or its negation $\neg x_i$.
- $w : C \to \mathbb{R}_{\geq 0}$ is a **weight function** that assigns a non-negative real weight $w(c_j)$ (or simply w_j) to each clause $c_j \in C$. For the unweighted MAX-SAT problem, all weights are considered to be 1.

2.1.0.2 Truth Assignment and Objective: A **truth assignment** is a function $\tau : X \to \{0, 1\}$. The goal is to find an optimal assignment τ^* that maximizes the total weight of satisfied clauses. Let $v(c_j, \tau)$ be an indicator function that is 1 if τ satisfies clause c_j and 0 otherwise. The objective is to find:

$$\tau^* = \arg \max_{\tau: X \to \{0,1\}} \left(\sum_{j=1}^{m} w_j \cdot v(c_j, \tau) \right)$$

2.1.0.3 Example: Consider the formula:

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2)$$

An assignment might not satisfy all clauses. The task is to find an assignment that satisfies as many as possible.

2.2. Contrast with the SAT Problem

The key difference lies in the objective:

- **SAT (Decision Problem):** Asks *if* there exists an assignment that satisfies *all* clauses. The answer is a simple YES or NO.
- **MAX-SAT (Optimization Problem):** Asks for an assignment that satisfies the *maximum number* of clauses. This is necessary when a formula is unsatisfiable.

2.3. Computational Complexity

The decision version of MAX-SAT ("Can at least k clauses be satisfied?") is **NP-complete**. The optimization problem is **NP-hard**, meaning there is no known polynomial-time algorithm to find the exact optimal solution for all instances, assuming P $\neq$ NP. This hardness motivates the development of approximation algorithms.

2.4. Variants of the MAX-SAT Problem

The standard MAX-SAT problem has several important variations:

- **Weighted MAX-SAT:** Each clause is assigned a weight, and the objective is to maximize the total weight of the satisfied clauses. This is useful for modeling problems where satisfying certain constraints is more important than others.
- **Partial MAX-SAT:** The clauses are divided into two sets: "hard" clauses that *must* be satisfied, and "soft" clauses that are desirable to satisfy. The goal is to satisfy all hard clauses while maximizing the number of satisfied soft clauses.
- **MAX-k-SAT:** A restricted version where every clause has exactly k literals. For example, MAX-2-SAT and MAX-3-SAT are well-studied variants. While 2-SAT is solvable in polynomial time, MAX-2-SAT is NP-hard.

2.5. Approximate Algorithms

Since finding the exact solution to MAX-SAT is computationally expensive, various techniques are used. These algorithms run in polynomial time and provide a solution that is provably close to the optimal one. Below are some of the most fundamental approaches.

2.5.1. 1/2-Approximation (Randomized Algorithm)

For each variable, assign it to be 'true' or 'false' with a probability of 0.5, independently. In expectation, this simple algorithm satisfies at least half of the total number of clauses.

2.5.2. $(1 - 1/e)$-Approximation (Randomized Rounding of a Linear Program)

A more sophisticated approach based on linear programming.

1. Formulate the MAX-SAT problem as an Integer Linear Program (ILP).
2. Relax the integer constraints to create a Linear Program (LP) and solve it to get fractional solutions (y_i^*) for each variable.
3. Round the fractional solution: set variable x_i to 'true' with probability y_i^*.

This method achieves an approximation ratio of approximately $1 - 1/e \approx 0.632$.

2.5.3. 3/4-Approximation (Combination of Algorithms)

This approach improves the approximation guarantee by running both the 1/2-approximation algorithm and the $(1 - 1/e)$-approximation algorithm, and then simply taking the better of the two results. This combined strategy is proven to achieve an approximation ratio of at least 3/4.

In the interest of minimizing notation, let us introduce common terminology for all three algorithms. Let the random variable W denote the total weight of satisfied clauses. For each clause c, let the random variable W_c denote the weight contributed by clause c to W. Thus, we have the relation:

$$W = \sum_{c \in C} W_c$$

The expected weight contributed by a single clause is given by:

$$\mathbb{E}[W_c] = w_c \cdot \Pr[c \text{ is satisfied}]$$

2.6. 1/2-Approximate Algorithm

2.6.1. Algorithm

Set each Boolean variable to be True independently with probability 1/2 and output the resulting truth assignment, say τ.

2.6.2. Proof of Approximation ratio 1/2

Lemma 1. *If a clause c has size k (i.e., size$(c) = k$), then the expected weight it contributes is $\mathbb{E}[W_c] = (1 - 2^{-k})w_c$.*

Proof. A clause c is not satisfied by a random assignment τ if and only if all of its k literals are set to False. Since each literal is effectively false with probability 1/2 independently, the probability of this event is $(1/2)^k = 2^{-k}$.

Therefore, the probability that clause c **is** satisfied is $1 - 2^{-k}$. The expected weight contributed by c is:

$$\mathbb{E}[W_c] = w_c \cdot \Pr[c \text{ is satisfied}] = w_c(1 - 2^{-k})$$

For any $k \geq 1$, the probability $(1 - 2^{-k}) \geq (1 - 1/2) = 1/2$.

By linearity of expectation, the total expected weight of the solution is:

$$\mathbb{E}[W] = \sum_{c \in C} \mathbb{E}[W_c] \geq \sum_{c \in C} \frac{1}{2} w_c = \frac{1}{2} \sum_{c \in C} w_c$$

The optimal solution, OPT, can be at most the total weight of all clauses. Thus, $\sum_{c \in C} w_c \geq \text{OPT}$. This gives us the approximation ratio:

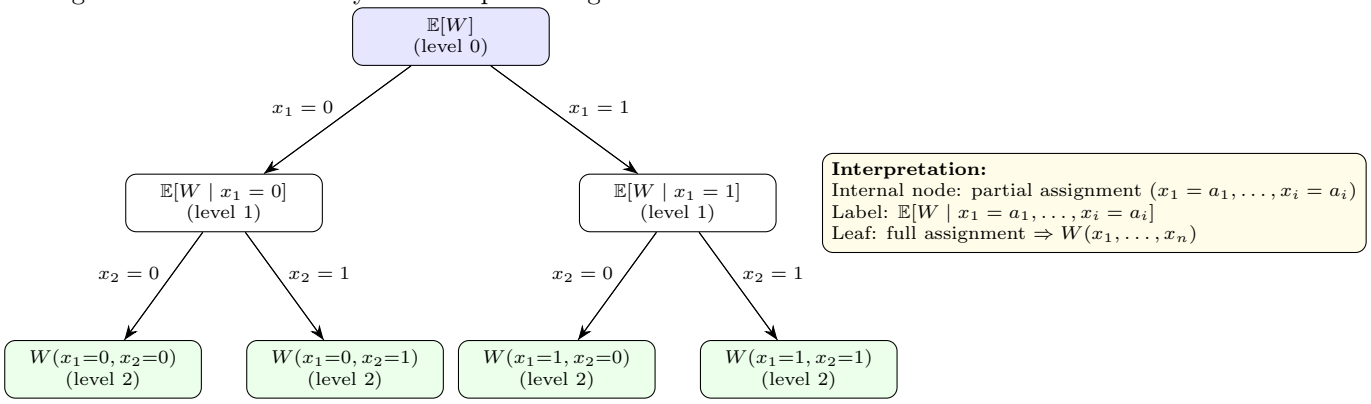
$$\mathbb{E}[W] \geq \frac{1}{2}\text{OPT}$$

A derandomized version of this procedure can deterministically compute a truth assignment such that the weight of satisfied clauses is at least $\mathbb{E}[W]$, guaranteeing a 1/2-approximation. ■

2.6.2.1 Observation Observe that the satisfaction probability, $\alpha_k = 1-2^{-k}$, increases as k increases. For example, if every clause has two or more literals ($k \geq 2$), then $\alpha_k \geq (1-2^{-2}) = 3/4$, and this algorithm provides a 3/4-approximation guarantee. Thus on formulas with larger clause size this algorithms tends to perform better.

2.6.3. Derandomizing using method of conditional expectation

Consider the self-reducibility tree, T, for a given formula f. Each internal node at a level i (where $0 \leq i < n$) corresponds to a partial truth assignment for the variables $x_1, \ldots, x_i$, while each leaf at level n represents a complete truth assignment. We label each node of T with its conditional expectation. For a node corresponding to a partial assignment $x_1 = a_1, \ldots, x_i = a_i$, its label is the value $\mathbb{E}[W \mid x_1 = a_1, \ldots, x_i = a_i]$. At a leaf node where $i = n$, this value is no longer an expectation; it is simply the total weight of clauses satisfied by that complete assignment.



Lemma 2. *The conditional expectation of any node in the self-reducibility tree T can be computed in polynomial time.*

Proof. Consider a node in the tree corresponding to a partial truth assignment $x_1 = a_1, \ldots, x_i = a_i$. The conditional expectation at this node is the sum of two components. The first is the total weight of clauses in the original formula f that are already satisfied by this partial assignment, which is a fixed value we can calculate. The second component is the expected weight from the remaining, undecided clauses. These clauses form a residual formula, let's call it ϕ, over the unassigned variables $x_{i+1}, \ldots, x_n$. The expected weight of ϕ under a random assignment can also be computed in polynomial time. Therefore, the total conditional expectation is the sum of the deterministic weight from the satisfied clauses and the expected weight from the residual formula ϕ. Since both parts are computed efficiently, the entire calculation is polynomial in time. ■

Theorem 1. *We can compute, in polynomial time, a path from the root to a leaf such that the conditional expectation of each node on this path is greater than or equal to the initial expected value, $\mathbb{E}[W]$.*

Proof. Let $\mathcal{A}_i$ denote the event of a partial assignment $x_1 = a_1, \ldots, x_i = a_i$. The conditional expectation of a node at level i is the average of the conditional expectations of its two children at level $i+1$. This is because the next variable, x_{i+1}, is set to true or false with equal probability (1/2).

Mathematically, this relationship is:

$$\mathbb{E}[W \mid \mathcal{A}_i] = \frac{1}{2} \cdot \mathbb{E}[W \mid \mathcal{A}_i, x_{i+1} = 0] + \frac{1}{2} \cdot \mathbb{E}[W \mid \mathcal{A}_i, x_{i+1} = 1]$$

Since the parent's value is the average of its two children's values, it is a mathematical certainty that at least one of the children must have a value greater than or equal to the parent's value. That is:

$$\max\left(\mathbb{E}[W \mid \mathcal{A}_i, x_{i+1} = 0], \mathbb{E}[W \mid \mathcal{A}_i, x_{i+1} = 1]\right) \geq \mathbb{E}[W \mid \mathcal{A}_i]$$

This establishes that a path of non-decreasing conditional expectations must exist from the root to a leaf.

By Lemma 16.3, the conditional expectation of each child can be computed in polynomial time. Therefore, we can find this path greedily by starting at the root and repeatedly choosing the child with the higher conditional expectation at each level. This entire process runs in polynomial time. ■

The deterministic algorithm follows as a corollary of Theorem 16.4. We simply output the truth assignment on the leaf node of the path computed. The total weight of clauses satisfied by it is>= E[W].

2.7. $(1 - 1/e)$-Approximation via Linear Programming

2.7.1. Integer Program Description

The following is an integer program for MAX-SAT. For each clause $c \in C$, let S_c^+ and S_c^- denote the sets of indices of Boolean variables occurring non-negated and negated in c, respectively. The truth assignment is encoded by variables y_i. Picking $y_i = 1$ ($y_i = 0$) denotes setting variable x_i to True (False). For each clause c, a variable z_c is introduced. The constraint for clause c ensures that z_c can be set to 1 only if at least one of the literals occurring in c is set to True, i.e., if clause c is satisfied by the picked truth assignment.

$$\text{Maximize} \quad \sum_{c \in C} w_c z_c \tag{1}$$

$$\text{Subject to:} \quad \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c, \quad \forall c \in C \tag{2}$$

$$y_i \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\}$$

$$z_c \in \{0, 1\}, \quad \forall c \in C$$

Now the previous Integer Linear Program is relaxed by allowing the variables y_i and z_c to take any real value between 0 and 1.

$$\text{Maximize} \quad \sum_{c \in C} w_c z_c \tag{3}$$

$$\text{Subject to:} \quad \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c, \quad \forall c \in C \tag{4}$$

$$0 \leq y_i \leq 1, \quad \forall i \in \{1, \ldots, n\} \tag{5}$$

$$0 \leq z_c \leq 1, \quad \forall c \in C \tag{6}$$

The algorithm is straightforward. First, solve the LP-relaxation defined in equations (3)–(6) and let (y^*, z^*) denote the optimal fractional solution obtained. Then, for each variable x_i (where $1 \leq i \leq n$), independently set x_i to `True` with probability y_i^*. The final resulting truth assignment, τ, is the output of the algorithm.

Note: Polynomial time algorithm for solving this relaxed linear programming problem **(i.e Ellipsoid algorithm)** is discussed in the Appendix I.

2.7.2. Proof of Approximation ratio $(1 - 1/e)$

Lemma 3. *If a clause c has size k (i.e., size$(c) = k$), then the expected weight it contributes, $\mathbb{E}[W_c]$, satisfies the following inequality with respect to the optimal LP solution value z_c^*:*

$$\mathbb{E}[W_c] \geq \beta_k \cdot w_c z_c^*$$

where $\beta_k = \left[1 - \left(1 - \frac{1}{k}\right)^k\right]$.

Proof. Let's prove the lemma for a general clause c of size k. Let S_c^+ be the set of indices for variables appearing as positive literals (x_i) and S_c^- be the set for variables appearing as negative literals ($\neg x_j$).

As you defined, the clause c is **not satisfied** if and only if every literal in it is set to `False`. The probability of this is the product of the probabilities of each of its literals being false:

$$\Pr[c \text{ is not satisfied}] = \left(\prod_{i \in S_c^+} (1 - y_i^*)\right) \cdot \left(\prod_{j \in S_c^-} y_j^*\right)$$

Now, we apply the AM-GM inequality to the k terms in the product. The geometric mean is less than

or equal to the arithmetic mean:

$$\begin{aligned}\Pr[c \text{ is not satisfied}] &\leq \left(\frac{\sum_{i\in S_c^+}(1-y_i^*)+\sum_{j\in S_c^-} y_j^*}{k}\right)^k \\ &= \left(\frac{k-\left(\sum_{i\in S_c^+} y_i^*+\sum_{j\in S_c^-}(1-y_j^*)\right)}{k}\right)^k \\ &= \left(1-\frac{\sum_{i\in S_c^+} y_i^*+\sum_{j\in S_c^-}(1-y_j^*)}{k}\right)^k \\ &\leq \left(1-\frac{z_c^*}{k}\right)^k\end{aligned}$$

The final step uses the main LP constraint (4). Therefore, the probability that the clause is satisfied is bounded by:

$$\Pr[c \text{ is satisfied}] \geq 1-\left(1-\frac{z_c^*}{k}\right)^k$$

Let the function $f(z) = 1-(1-z/k)^k$. This function is concave for $z \in [0,1]$ and $f(0) = 0$. For such a function, it is known that $f(z) \geq z \cdot f(1)$. We have $f(1) = 1-(1-1/k)^k = \beta_k$. Applying this with $z = z_c^*$:

$$\Pr[c \text{ is satisfied}] \geq f(z_c^*) \geq z_c^* \cdot f(1) = \beta_k z_c^*$$

Finally, the expected weight is $\mathbb{E}[W_c] = w_c \cdot \Pr[c \text{ is satisfied}]$, which leads to the desired result:

$$\mathbb{E}[W_c] \geq \beta_k w_c z_c^*$$ ■

2.7.3. Derandomizing using method of conditional expectation

The following algorithm converts the randomized rounding procedure into a deterministic one using the method of conditional expectations.

1. **Solve LP:** Solve the LP-relaxation to obtain the optimal fractional solution (y^*, z^*).
2. **Iterate and Choose Deterministically:** For $i = 1, \ldots, n$, perform the following steps:
   (a) Let the assignment for the first $i-1$ variables be fixed as $\mathcal{A}_{i-1} = (a_1, \ldots, a_{i-1})$.
   (b) Calculate the two conditional expectations for the next assignment:
   $$E_1 = \mathbb{E}[W \mid \mathcal{A}_{i-1}, x_i = 1]$$
   $$E_0 = \mathbb{E}[W \mid \mathcal{A}_{i-1}, x_i = 0]$$
   (c) Make the deterministic choice for x_i by selecting the outcome that maximizes the conditional expectation. Set $a_i := \arg\max(E_0, E_1)$.
3. **Output:** The final assignment is the deterministically constructed vector $\tau = (a_1, \ldots, a_n)$.

The correctness of the algorithm relies on the law of total expectation. The conditional expectation at step $i-1$ is a weighted average of the two possible expectations at step i:

$$\mathbb{E}[W \mid \mathcal{A}_{i-1}] = y_i^* \cdot E_1 + (1-y_i^*) \cdot E_0$$

Since a value cannot be smaller than both numbers it is an average of, it must be that $\max(E_1, E_0) \geq \mathbb{E}[W \mid \mathcal{A}_{i-1}]$. Our deterministic choice rule in step 2(c) leverages this fact to ensure that the conditional expectation of the outcome never decreases.

This creates a chain of non-decreasing expectations:

$$\mathbb{E}[W] \leq \mathbb{E}[W \mid x_1 = a_1] \leq \mathbb{E}[W \mid x_1 = a_1, x_2 = a_2] \leq \cdots \leq W(\tau)$$

After n steps, all variables are determined, so the final conditional expectation is simply the actual weight $W(\tau)$ of the assignment. This guarantees that the algorithm produces a solution with a weight of at least the initial expectation $\mathbb{E}[W]$. As established previously, each conditional expectation is computable in polynomial time, making the entire algorithm efficient.

2.8. 3/4-Approximation

We combine the two previous algorithms using a simple coin flip. Let b be the outcome of a fair coin; if $b = 0$, we run the first randomized algorithm (setting each variable to `True` with probability 1/2), and if $b = 1$, we run the second randomized algorithm (solving the LP and rounding based on the y_i^* values).

Lemma 4. *For the combined algorithm, the expected weight contributed by any clause c satisfies $\mathbb{E}[W_c] \geq \frac{3}{4} w_c z_c^*$.*

Proof. Let the size of clause c be k. By the law of total expectation, we average the outcomes of the two algorithms:

$$\mathbb{E}[W_c] = \frac{1}{2} \left(\mathbb{E}[W_c \mid b = 0] + \mathbb{E}[W_c \mid b = 1] \right)$$

From our previous lemmas, we have the following bounds:

- $\mathbb{E}[W_c \mid b = 0] = \alpha_k w_c$. Since $z_c^* \leq 1$, we have $\alpha_k w_c \geq \alpha_k w_c z_c^*$.
- $\mathbb{E}[W_c \mid b = 1] \geq \beta_k w_c z_c^*$.

Substituting these into the equation gives:

$$\mathbb{E}[W_c] \geq \frac{1}{2}(\alpha_k w_c z_c^* + \beta_k w_c z_c^*) = w_c z_c^* \left(\frac{\alpha_k + \beta_k}{2} \right)$$

The proof is complete by showing that $\alpha_k + \beta_k \geq 3/2$ for all $k \geq 1$.

- For $k = 1$: $\alpha_1 + \beta_1 = (1 - 1/2) + 1 = 3/2$.
- For $k = 2$: $\alpha_2 + \beta_2 = (1 - 1/4) + (1 - (1 - 1/2)^2) = 3/4 + 3/4 = 3/2$.
- For $k \geq 3$: We know $\alpha_k = 1 - 2^{-k} \geq 7/8$ and $\beta_k = 1 - (1 - 1/k)^k \geq 1 - 1/e$. Thus, $\alpha_k + \beta_k > 0.875 + 0.632 = 1.507 > 3/2$.

Since $\frac{\alpha_k + \beta_k}{2} \geq \frac{3}{4}$ for all k, the lemma holds. ■

By linearity of expectation, the total expected weight of the solution is:

$$\mathbb{E}[W] = \sum_{c \in C} \mathbb{E}[W_c] \geq \frac{3}{4} \sum_{c \in C} w_c z_c^* = \frac{3}{4} \mathrm{OPT}_f \geq \frac{3}{4} \mathrm{OPT}$$

where OPT_f is the optimal value of the LP relaxation. Derandomized form of the algorithm is simply run both the previous derandomized algorithms and pick the assignment with better results.

Theorem 2. *The deterministic algorithm that computes the outcomes of both randomized procedures and outputs the better of the two is a factor $\frac{3}{4}$ approximation algorithm for MAX-SAT.*

Proof. Let τ_1 and τ_2 be the assignments produced by derandomizing the first and second algorithms, respectively. The derandomization process guarantees that their weights are at least as large as their respective conditional expectations:

- $W(\tau_1) \geq \mathbb{E}[W \mid b = 0]$
- $W(\tau_2) \geq \mathbb{E}[W \mid b = 1]$

The algorithm outputs the better of these two assignments. The weight of this final solution is therefore $\max(W(\tau_1), W(\tau_2))$.

The total expected value of the combined randomized algorithm is $\mathbb{E}[W] = \frac{1}{2}\mathbb{E}[W \mid b = 0] + \frac{1}{2}\mathbb{E}[W \mid b = 1]$. Since $\mathbb{E}[W]$ is the average of the two conditional expectations, at least one of them must be greater than or equal to $\mathbb{E}[W]$. Therefore:

$$\max(W(\tau_1), W(\tau_2)) \geq \max(\mathbb{E}[W \mid b = 0], \mathbb{E}[W \mid b = 1]) \geq \mathbb{E}[W]$$

We have already shown that $\mathbb{E}[W] \geq \frac{3}{4}\mathrm{OPT}$, so the deterministic algorithm also achieves this performance guarantee. ■

3. Steiner Tree

3.1. Definition and Complexity

[Steiner Tree (graph version)] Let $G = (V, E)$ be an undirected graph with nonnegative edge costs $c : E \to \mathbb{R}_{\geq 0}$. Let $R \subseteq V$ be a designated set of *required* vertices (terminals). The *Steiner Tree* decision problem asks: given (G, c, R) and a bound $B \geq 0$, is there a connected subgraph of G of total cost at most B that contains every vertex of R? The optimization variant asks for a minimum-cost connected subgraph (typically taken to be a tree) spanning R and possibly some Steiner vertices $V \setminus R$.

[Metric Steiner Tree] The *metric Steiner Tree* problem is the Steiner Tree problem on instances where the graph is complete and costs satisfy the triangle inequality:

$$\forall u, v, w \in V : \quad c(u, v) \leq c(u, w) + c(w, v).$$

Given any graph Steiner instance one may form its *metric closure* (complete graph on same vertex set, with edge weights equal to shortest-path distances in G) and interpret the problem in that metric.

The decision version of Steiner Tree is in NP: given a candidate tree one can verify in polynomial time that it spans all required vertices and its cost is at most B. We now prove NP-hardness (completeness of the decision form) using a polynomial-time reduction from Set Cover.

3.2. NP-hardness and Inapproximability

3.2.1. Reduction from Set Cover to (undirected) Steiner Tree

We reduce from the well-known NP-complete problem Set Cover.

Set Cover (decision): Given a universe $U = \{e_1, \ldots, e_n\}$, a family of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ with costs $w_j \geq 0$, and a target integer k (or cost bound K), is there a subcollection of sets whose union is U with total cost at most K?

Theorem 3. *The Steiner Tree decision problem in undirected graphs is NP-complete.*

Proof. We give a polynomial-time reduction from Set Cover. Let $(U, \mathcal{S}, \{w_j\}, K)$ be an instance of Set Cover. We construct a Steiner Tree instance (G, c, R, B) as follows.

Construct a vertex r (a special root), a vertex s_j for each set $S_j \in \mathcal{S}$, and a vertex e_i for each element $e_i \in U$. Let the vertex set be

$$V = \{r\} \cup \{s_1, \ldots, s_m\} \cup \{e_1, \ldots, e_n\}.$$

Create undirected edges as follows:

- For each set node s_j, add an edge (r, s_j) of cost w_j.
- For each element e_i and each set S_j that contains e_i, add an edge (s_j, e_i) of cost 0.

Let the required set be $R = \{e_1, \ldots, e_n\}$ (all element vertices must be spanned). Let the cost bound be $B = K$.

This construction is polynomial-time.

Correctness:

- Suppose there exists a collection of sets $\mathcal{C} \subseteq \mathcal{S}$ with total cost $\sum_{S_j \in \mathcal{C}} w_j \leq K$ whose union covers U. Consider the subgraph that includes vertex r, includes every s_j with $S_j \in \mathcal{C}$, includes all edges (r, s_j) for $S_j \in \mathcal{C}$, and for each element e_i pick any set $S_j \in \mathcal{C}$ that contains e_i and include edge (s_j, e_i) (these edges cost 0). This subgraph connects every element e_i to r (via some chosen set vertex) and has total cost $\sum_{S_j \in \mathcal{C}} w_j \leq K$. Therefore there is a Steiner subgraph of cost at most B spanning all required vertices.
- Conversely, suppose there exists a connected subgraph H of total cost at most $B = K$ that spans all required vertices $\{e_i\}$. Because all edges incident to element vertices e_i have cost 0 and every element vertex must be connected to the rest of the subgraph, each e_i must be adjacent in H to some set vertex s_j. The only edges with positive cost in H are of the form (r, s_j); hence the set of set-vertices s_j that are connected to r in H corresponds to a collection of sets whose union covers all elements (since each element is adjacent to some chosen s_j). The total cost of the subgraph equals the sum of the costs of chosen (r, s_j) edges, which is the total cost of the corresponding set family, and this sum is at most K. Thus there is a set cover of cost at most K.

This reduction shows that Steiner Tree decision is NP-hard. Since the problem is also in NP (a candidate tree can be checked in polynomial time), it is NP-complete. ■

3.2.2. NP-hardness of Metric Steiner (via metric closure)

The metric closure (complete graph with shortest-path distances) provides an approximation-preserving, polynomial-time reduction from general Steiner instances to metric instances.

Theorem 4. *Metric Steiner Tree is NP-hard.*

Proof. Given an arbitrary Steiner Tree instance (G, c, R, B), form the metric closure $G' = (V, E')$ where for each pair $u, v \in V$ the cost $c'(u, v)$ is the shortest-path distance between u and v in G. Any solution in G maps to a solution in G' of no greater cost (each G-path maps to the corresponding G' edge), and any solution in G' can be realized in G by replacing G' edges by the corresponding shortest paths and deleting cycles. Hence there is a Steiner tree of cost $\leq B$ in G if and only if there is a Steiner tree of cost $\leq B$ in G'. The reduction is polynomial-time. Because the Steiner Tree decision problem in general graphs is NP-hard (previous theorem), metric Steiner (which includes these metric-closure images) is NP-hard as well. ■

The reduction also preserves approximation factors, so hardness results for metric Steiner follow from general-case hardness when appropriate.

3.3. 2-Approximation Algorithm

We present a simple MST-based factor-2 approximation for metric Steiner Tree. The algorithm (pseudocode) and its analysis follow.

Algorithm 1 MST-on-Required (Metric Steiner Tree, 2-Approximation)

Require: Graph $G = (V, E)$ with non-negative costs.
Require: Required set of terminals $R \subseteq V$.
Ensure: A Steiner Tree T spanning R with cost $w(T) \leq 2 \cdot w(T_{OPT})$.
1: **Metric Closure:** Compute all-pairs shortest path distances $d(u, v)$ in G for all $u, v \in R$.
2: **Construct K_R:** Form the complete graph K_R on the vertex set R, where the edge cost $\text{cost}(u, v)$ is set to the shortest-path distance $d(u, v)$.
3: **Minimum Spanning Tree (MST):** Compute a minimum spanning tree T_R of K_R.
4: **Lift to G:** Replace each edge (u, v) of T_R by a corresponding shortest u–v path in G.
5: **Prune:** Take the union of these paths and remove cycles to obtain a tree T that spans R.
6: **return** T =0

3.3.1. Proof of Factor-2 Approximation

Theorem 5. *The cost of the tree T output by MST-on-required is at most $2 \cdot \text{OPT}$, where OPT is the cost of an optimal Steiner tree.*

Proof. Let T^* be an optimal Steiner tree of cost OPT. Double every edge of T^* to obtain an Eulerian multigraph. An Euler tour of this multigraph has total length $2 \cdot \text{OPT}$. Traverse the Euler tour and record the order in which required vertices appear; by shortcutting repeated visits (possible because of triangle inequality in the metric closure), we obtain a traversal that visits each required vertex and whose total length is at most $2 \cdot \text{OPT}$. Removing one edge from this closed traversal yields a spanning tree on R (a candidate in K_R) of cost at most $2 \cdot \text{OPT}$. Since T_R is the minimum spanning tree on K_R, $\text{cost}(T_R) \leq 2 \cdot \text{OPT}$. Expanding edges back to G (replacing each K_R edge by its shortest-path realization) yields a Steiner tree in G of cost equal to $\text{cost}(T_R) \leq 2 \cdot \text{OPT}$. Hence the algorithm is a 2-approximation. ■

4. The Metric k-Center Problem

4.1. Definition and Complexity

The metric k-center problem is defined as follows:

Given a complete, undirected graph $G = (V, E)$ with n vertices, a positive integer k, and a cost function $\text{cost} : E \to \mathbb{R}_{\geq 0}$ satisfying:

- **Non-negativity:** $\text{cost}(u, v) \geq 0$, $\text{cost}(u, u) = 0$.
- **Symmetry:** $\text{cost}(u, v) = \text{cost}(v, u)$.
- **Triangle Inequality:** $\text{cost}(u, w) \leq \text{cost}(u, v) + \text{cost}(v, w)$.

The goal is to select k vertices $S \subseteq V$ (centers) minimizing the maximum distance from any vertex to its nearest center:

$$\min_{\substack{S \subseteq V \\ |S|=k}} \max_{v \in V} \min_{s \in S} \text{cost}(v, s)$$

The triangle inequality ensures a structured metric, allowing approximation algorithms. The optimal radius OPT equals one of the inter-vertex distances. Sorting edges as $\text{cost}(e_1) \leq \cdots \leq \text{cost}(e_m)$, for any radius r, define $G_r = (V, E_r)$ with edges $\leq r$. A set S is valid if every vertex is within distance r of some center; in G_r, this is equivalent to S being a dominating set. Thus, the problem reduces to finding the smallest r such that G_r has a dominating set of size at most k, connecting the geometric and combinatorial views.

4.2. NP-Hardness and Inapproximability

The connection to the dominating set problem provides both algorithmic insight and a basis for proving the computational hardness of the k-center problem. Specifically, achieving an approximation factor better than 2 is NP-hard.

Theorem 6. *Assuming $P \neq NP$, no polynomial-time algorithm can achieve an approximation factor of $2 - \epsilon$ for any $\epsilon > 0$ for the metric k-center problem.*

Proof. We reduce from the Dominating Set problem, which is NP-complete. Given an instance $(G' = (V, E'), k')$, construct a k-center instance $(G = (V, E), k)$ as follows:

- G is the complete graph on V, with $k = k'$.
- Edge costs:

$$\text{cost}(u, v) = \begin{cases} 1, & \text{if } (u, v) \in E' \text{ or } u = v \\ 2, & \text{otherwise} \end{cases}$$

This cost function satisfies the triangle inequality.

Case 1: G' has a dominating set of size $\leq k'$.

Choose this dominating set as the set of centers. Any vertex in the set has distance 0 to the nearest center, and any vertex outside the set is adjacent to at least one center in G', so its distance to the nearest center in G is 1. Hence, the maximum distance from any vertex to a center (the k-center cost) is $\text{OPT} = 1$.

Case 2: G' has no dominating set of size $\leq k'$.

For any set of k' centers, there is at least one vertex not adjacent to any center in G'. For such a vertex, the distance to the nearest center in G is 2. Therefore, the maximum distance to a center is $\text{OPT} = 2$.

A $(2 - \epsilon)$-approximation algorithm could distinguish these cases in polynomial time: it would return 1 for Case 1 and 2 for Case 2, effectively solving Dominating Set. This is impossible unless $P = NP$, so no such algorithm exists. ■

This demonstrates that the k-center problem's hardness arises from distinguishing instances where the optimal cost is 1 versus 2, highlighting its intrinsic NP-hard structure.

4.3. The Parametric Pruning Algorithm (Hochbaum & Shmoys)

Hochbaum and Shmoys' algorithm uses *parametric pruning*, converting the problem into a series of decision questions: "Does a valid k-center solution exist for radius r?" By testing candidate radii, it finds the smallest feasible radius and constructs a 2-approximate solution.

The algorithm exploits the combinatorial interpretation of k-center using dominating sets and introduces the *square of a graph* H^2, where vertices u and v are adjacent if there is a path of length at most 2 between them in H.

Algorithm 2 Parametric Pruning Algorithm for Metric k-Center

Require: Complete graph $G = (V, E)$ with edge costs $\text{cost}(u, v)$
Require: Integer k (maximum number of centers)
Ensure: Set of centers M_j with 2-approximation guarantee
1: Sort the unique edge costs: $\{c_1, \ldots, c_m\}$
2: **for** $i = 1$ to m **do**
3: Define $G_i = (V, E_i)$, where $E_i = \{(u, v) \mid \text{cost}(u, v) \leq c_i\}$
4: Construct G_i^2, the square of G_i
5: Compute a maximal independent set M_i in G_i^2
6: **if** $|M_i| \leq k$ **then**
7: $j \leftarrow i$ **break**
8: **end if**
9: **end for**
10: **return** M_j as the set of centers =0

4.3.1. Proof of 2-approximation

The proof has two parts: first, we show that the cost associated with the chosen index j is a lower bound on the optimal solution; second, we show that the constructed solution has cost at most twice this value.

Part 1: Lower Bound on the Optimal Cost

The lower bound relies on a relationship between independent sets in H^2 and dominating sets in H.

Lemma 5. *For any graph H, if I is an independent set in H^2, then the size of the minimum dominating set of H, dom(H), satisfies $|I| \leq$ dom(H).*

Proof. Let D be a minimum dominating set of H. Every vertex is either in D or adjacent to a vertex in D, so H can be covered by $|D|$ stars centered at vertices of D. In H^2, vertices connected by a path of length 2 in H become adjacent, turning each star into a clique. An independent set I in H^2 can have at most one vertex from each clique, so $|I| \leq |D| = \text{dom}(H)$. ■

Lemma 6. *For the index j found by the algorithm, $c_j \leq OPT$.*

Proof. Let i^* be the index such that $\text{OPT} = c_{i^*}$. By definition, i^* is the smallest index for which G_{i^*} has a dominating set of size at most k. For any $i < i^*$, $\text{dom}(G_i) > k$.

The algorithm finds the smallest j with $|M_j| \leq k$. For $i < j$, $|M_i| > k$ and by Lemma 2.1, $|M_i| \leq \text{dom}(G_i)$, so $\text{dom}(G_i) > k$. Hence, $i^* \geq j$, implying $c_j \leq c_{i^*} = \text{OPT}$. ■

Part 2: Upper Bound on the Solution Cost

Theorem 7. *The parametric pruning algorithm achieves a 2-approximation for the metric k-center problem.*

Proof. The algorithm returns M_j as centers. A maximal independent set is also a dominating set, so every vertex v is either in M_j or adjacent to a vertex in M_j in G_j^2.

By definition, an edge (u, v) in G_j^2 corresponds to a path of length at most 2 in G_j, and each edge in G_j has cost $\leq c_j$. By the triangle inequality, the distance from any vertex to its nearest center is at most $2c_j$.

Combining with Lemma 2.2 ($c_j \leq \text{OPT}$), the solution cost is at most $2 \cdot \text{OPT}$, proving the 2-approximation guarantee. ■

4.4. Gonzalez's Farthest-First Traversal Algorithm

An alternative, intuitive approach is the greedy algorithm by Teofilo Gonzalez. It constructs centers iteratively by selecting the point farthest from all previously chosen centers, spreading the centers effectively.

4.4.1. Algorithm Steps

1. Initialize S with an arbitrary vertex $s_1 \in V$.
2. For $i = 2, \ldots, k$:
   (a) Compute $d(v, S) = \min_{s \in S} \text{cost}(v, s)$ for each $v \in V$.
   (b) Select $s_i = \arg\max_{v \in V} d(v, S)$ and add it to S.
3. Return S.

The algorithm runs in $O(nk)$ time; $O(n^2)$ in the worst case if $k = \Theta(n)$.

4.4.2. Proof of 2-Approximation Guarantee

Theorem 8. *The farthest-first traversal algorithm is a 2-approximation for the metric k-center problem.*

Proof. Let $S = \{s_1, \ldots, s_k\}$ be the centers returned, r_{alg} their radius, and $S^* = \{c_1^*, \ldots, c_k^*\}$ be an optimal set with radius OPT. The optimal clusters $V_1^*, \ldots, V_k^*$ satisfy $\text{cost}(v, c_i^*) \leq \text{OPT}$ for $v \in V_i^*$. By the triangle inequality, $\text{cost}(u, v) \leq 2\text{OPT}$ for any u, v in the same cluster.

Assume $r_{\text{alg}} > 2\text{OPT}$. Let s_{k+1} be the hypothetical next point added. Then $\text{cost}(s_{k+1}, s_i) > 2\text{OPT}$ for all $s_i \in S$, and distances between any pair of centers in S are also $> 2\text{OPT}$.

Consider $S \cup \{s_{k+1}\}$ (size $k + 1$) and k optimal clusters. By the pigeonhole principle, some cluster contains two points p_1, p_2. Then $\text{cost}(p_1, p_2) \leq 2\text{OPT}$, contradicting that all pairwise distances $> 2\text{OPT}$. Hence, $r_{\text{alg}} \leq 2\text{OPT}$. ■

These two algorithms highlight two approaches to 2-approximation: Hochbaum & Shmoys' parametric pruning uses an indirect, value-based strategy with dominating sets, while Gonzalez's method greedily constructs a solution geometrically. The former is adaptable for complex variants, while the latter is simple and efficient.

4.5. The Metric Weighted k-Center Problem

This variant introduces economic constraints into the facility location model. In addition to locating centers to minimize service distance, the decision-maker must operate within a fixed budget.

[Metric Weighted k-Center] In addition to the standard metric k-center input, we are given a weight function $w : V \to \mathbb{R}^+$ for each vertex and a total weight budget $W \in \mathbb{R}^+$. The objective is to find a subset of centers $S \subseteq V$ such that its total weight $\sum_{s \in S} w(s) \leq W$, which minimizes the k-center radius $\max_{v \in V} d(v, S)$.

The parametric pruning framework can be elegantly adapted to solve this problem, yielding a 3-approximation.

4.5.1. The 3-Approximation Algorithm

This algorithm adapts the unweighted parametric pruning approach to handle weights:

1. Construct $G_i = (V, E_i)$ and their squares G_i^2 for each unique edge cost c_i.
2. Compute a maximal independent set M_i in each G_i^2.
3. For each $u \in M_i$, select its lightest neighbor in G_i (including itself) as $s_i(u)$.
4. Form the candidate center set $S_i = \{s_i(u) \mid u \in M_i\}$.
5. Choose the smallest index j such that the total weight $w(S_j) \leq W$.
6. Return S_j as the set of centers.

4.5.2. Proof of 3-Approximation Guarantee

Theorem 9. *The algorithm achieves a 3-approximation for the metric weighted k-center problem.*

Proof. **Lower Bound:** The index j chosen satisfies $c_j \leq \text{OPT}$, by the same reasoning as in the unweighted case using a weighted analogue of Lemma 2.1. Any index $i < i^*$ violates the budget constraint, ensuring $j \leq i^*$.

Upper Bound: For any vertex $v \in V$:

1. M_j is a maximal independent set in G_j^2, hence a dominating set.
2. There exists $u \in M_j$ with $\text{cost}(v, u) \leq 2c_j$.
3. The algorithm selects $s_j(u)$ as the actual center, with $\text{cost}(u, s_j(u)) \leq c_j$.
4. By the triangle inequality, $\text{cost}(v, s_j(u)) \leq \text{cost}(v, u) + \text{cost}(u, s_j(u)) \leq 2c_j + c_j = 3c_j$.

Thus, the solution radius is at most $3c_j \leq 3 \times \text{OPT}$. ■

This demonstrates the "price of generalization": enforcing weight constraints introduces an extra step in the distance calculation ($v \to u \to s_j(u)$), which increases the approximation factor from 2 to 3.

4.6. Application I: The Metric k-Cluster Problem

The k-cluster problem modifies the objective from minimizing service radius to minimizing the maximum diameter of clusters, focusing on internal cohesion rather than distance to a center.

[Metric k-Cluster] Given a complete metric graph $G = (V, E)$ and integer k, partition V into k disjoint clusters $V_1, \ldots, V_k$ to minimize

$$\max_{1 \leq i \leq k} \left\{ \max_{u,v \in V_i} \text{cost}(u, v) \right\},$$

i.e., the maximum cluster diameter.

4.6.1. 2-Approximation Algorithm and Proof

A 2-approximation can be achieved using Gonzalez's farthest-first traversal:

1. Select k centers $S = \{s_1, \ldots, s_k\}$ via the Gonzalez algorithm.
2. Assign each vertex $v \in V$ to the cluster of its nearest center $s_i \in S$.

Proof: Let OPT_{diam} be the optimal maximum diameter. For any u, v in the same cluster V_i^* of an optimal solution, $\text{cost}(u, v) \leq \text{OPT}_{\text{diam}}$. Any vertex v is within OPT_{diam} of its nearest selected center due to the farthest-first guarantee. For u, v assigned to the same center s_i, the triangle inequality gives:

$$\text{cost}(u, v) \leq \text{cost}(u, s_i) + \text{cost}(s_i, v) \leq 2 \times \text{OPT}_{\text{diam}}.$$

Hence, the algorithm produces a 2-approximation.

4.6.2. Hardness of Approximation

Claim: Approximating metric k-cluster within $2 - \epsilon$ is NP-hard.

Proof: Using the same reduction from Dominating Set as for k-center:

Case 1 (YES instance): If G' has a dominating set D of size at most k', we can form a clustering. For each $s \in D$, create a cluster consisting of s and all its neighbors. Any remaining vertices can be placed into these clusters or form their own singleton clusters (up to k' total clusters). Within each such cluster, all vertices are at distance 1 from each other (since they are all neighbors of s). Thus, the maximum diameter is 1. We can use up to k' clusters; if $k' < k$, the remaining $k - k'$ clusters can be empty or singletons. The optimal diameter is $\text{OPT}_{\text{diam}} = 1$.

Case 2 (NO instance): If G' has no dominating set of size at most k', consider any partition of V into k' clusters, $V_1, \ldots, V_{k'}$. If the diameter of every cluster were 1, it would mean that every cluster is a clique in G'. We could then pick one vertex from each cluster to form a dominating set of size k', which is a contradiction. Therefore, at least one cluster V_i must contain two vertices, u and v, that are

not adjacent in G'. The distance between them is $\text{cost}(u, v) = 2$. This means the diameter of cluster V_i is 2, and the cost of any such clustering is at least 2. The optimal diameter is $\text{OPT}_{\text{diam}} = 2$.

A $(2 - \epsilon)$-approximation would distinguish these cases, solving Dominating Set in polynomial time, a contradiction unless $P = NP$.

Remark: The same farthest-first heuristic bounds the radius, which via the triangle inequality also bounds the cluster diameter, illustrating the duality between minimizing radius and diameter.

4.7. Application II: The Fault-Tolerant k-Center Problem

This variant addresses redundancy in facility location: each client must be close to multiple centers to guard against failures.

Given a metric graph $G = (V, E)$ and integers k and $\alpha \leq k$, choose k centers S to minimize the radius r such that every vertex $v \in V$ has at least α centers within distance r. Formally, minimize $\max_{v \in V}\{d_\alpha(v, S)\}$, where $d_\alpha(v, S)$ is the distance to the α-th closest center.

4.7.1. Combinatorial Lemma

Let $D \subseteq V$ be an α-dominating set of H, where every vertex is adjacent to at least α vertices in D. Let I be an independent set in H^2.

Lemma 7. *$\alpha|I| \leq dom_\alpha(H)$.*

Proof. For distinct $u, v \in I$, their neighbors in H are disjoint. Each vertex in I contributes at least α unique dominators in D, so summing over I gives $\alpha|I| \leq |D| = \text{dom}_\alpha(H)$. ■

4.7.2. 3-Approximation Algorithm and Proof

Algorithm:

1. Construct graphs G_i and their squares G_i^2 for each unique edge cost c_i.
2. In G_i^2, find a maximal independent set M_i.
3. Select the smallest j such that $|M_j| \leq \lfloor k/\alpha \rfloor$.
4. For each $u \in M_j$, include u and its $\alpha - 1$ closest neighbors in G_j in S.
5. Return S.

Proof of 3-Approximation:

Lower bound: Let $\text{OPT} = c_{i^*}$, where G_{i^*} first has an α-dominating set of size $\leq k$. By Lemma 4.1, $|M_{i^*}| \leq \lfloor k/\alpha \rfloor$. The algorithm chooses $j \leq i^*$, so $c_j \leq \text{OPT}$.

Upper bound: For any vertex v, there exists $u \in M_j$ such that v is at most distance $2c_j$ from u in G_j^2. The solution set S includes u and $\alpha - 1$ neighbors. For any $s \in S_u$:

$$\text{cost}(v, s) \leq \begin{cases} 2c_j, & s = u \\ 2c_j + c_j = 3c_j, & s \neq u \end{cases}$$

Hence, v has α centers within $3c_j \leq 3\,\text{OPT}$. This confirms the 3-approximation guarantee.

A. The Ellipsoid Algorithm for Linear Programming

The Ellipsoid Algorithm, developed by **Shor (1970)** and **Yudin and Nemirovskii (1975)**, and later applied to derive the first polynomial-time algorithm for **Linear Programming (LP)** by **Khachiyan (1979)**, provides a key theoretical foundation for solving a large class of convex optimization problems. Although the algorithm is practically slower than the Simplex algorithm, it is theoretically superior, as Simplex has an exponential running time in the worst case.

The algorithm operates on convex sets and utilizes a polynomial-time separation procedure.

A.1. Definitions

- **Hyperplane**: The set of points satisfying the linear equation $ax = b$ where $a, x, b \in R^n$.
- **Convex Set ($K \subseteq R^n$)**: A set of points such that for all $x, y \in K$, $\lambda x + (1 - \lambda)y \in K$ where $\lambda \in [0, 1]$. The feasible solution space of an LP is the intersection of half-spaces, which is a convex set.
- **Ellipsoid**: A general ellipsoid in R^n can be represented by $(x - a)^T B(x - a) \leq 1$, where B is a positive semidefinite matrix.
- **Separating Oracle**: A polynomial-time procedure for a convex set K that, given a point p, either confirms that $p \in K$ or returns a **hyperplane (H)** separating p from K. Such a plane is guaranteed to exist if $p \notin K$.

A.2. LP Formulation

A general linear program is to:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0 \end{array}$$

where A is an $m \times n$ real constraint matrix, and $x, c \in R^n$.

A.3. Algorithm Strategy

A.3.1. Conversion to feasibility problem

The problem of optimizing an objective function can be reduced to a series of feasibility problems as follows. We start off with an estimate of the maximum value, say c_0, and check for the feasibility of the following system:

$$\begin{aligned} c^T x &\geq c_0 \\ Ax &\leq b \\ x &\geq 0 \end{aligned}$$

If the system is infeasible, we know that the optimum is lesser than c_0. We may now decrease c_0, say by a factor of 2, and check for feasibility again. If this is true, we know that the optimum lies in $[c_0/2, c_0)$. This is essentially a binary search to find the optimum with higher accuracies. We get the optimum in a number of steps polynomial in the input size, each step being a call to a feasibility checking algorithm.

The algorithm solves the optimization problem by reducing it to a series of **feasibility problems** using binary search. The Ellipsoid Algorithm determines if the convex feasible solution space, S, is non-empty.

A.3.2. Ellipsoid Algorithm

Now we can concentrate on the feasibility problem. The Ellipsoid Algorithm solves the feasibility problem in an ingenious way. Let us denote the convex set defined by the feasible solution space by S. Further, we assume that the constraints are non-degenerate, so that S is either empty or has a non-zero volume denoted by $Vol(S)$. In other words, we can find a lower bound V_l on $Vol(S)$.

We start off with an ellipsoid of volume V_u guaranteed to bound S if it is finite. If $Vol(S)$ is infinite, we start with a suitable V_u and we will eventually get to a feasible point anyway. In our case, the initial bounding ellipsoid is a sphere in $\mathbb{R}^n$.

A single step of the algorithm either finds a point in S, in which case we have proved feasibility, or finds another ellipsoid bounding S that has a volume that is substantially smaller than the volume of the previous ellipsoid. We iterate on this new ellipsoid.

A.4. Running time

In the worst case, we need to iterate until the volume of the bounding ellipsoid gets below V_l, in which case we can conclude that the system is infeasible. It turns out that only a polynomial number of iterations are required in the case of linear programming. This can be shown by using a non-trivial

Algorithm 3 Ellipsoid Algorithm for Feasibility

Require: Bounding ellipsoid E_0 for S, Lower bound V_l on $Vol(S)$.
Ensure: "yes" if the linear program is feasible, "no" otherwise.

```
i ← 0
while Vol(E_i) ≥ V_l do
    p ← Center of E_i
    (ans, H) ← SepOracle(p)
    if ans = yes then
        return "yes"
    else
        {Take the separating hyperplane H}
        E_{i+1} ← MinVolumeEllipsoid(E_i, H)
        i ← i + 1
    end if
end while
return "no" =0
```

geometrical lemma.

Lemma 5. The minimum volume ellipsoid surrounding a half ellipsoid (i.e., $E_i \cap H^+$ above) can be calculated in polynomial time and

$$Vol(E_{i+1}) \leq \left(1 - \frac{1}{2n}\right) Vol(E_i)$$

If the while loop in the algorithm runs t times, then by the above lemma

$$\left(1 - \frac{1}{2n}\right)^t \leq \frac{V_u}{V_l} \implies t = O\left(n \log\left(\frac{V_u}{V_l}\right)\right)$$

It can be shown that for a linear program which requires L bits to represent the input, we can choose $V_l = 2^{-c_1 nL}$ and $V_u = 2^{c_2 nL}$ for some constants c_1, c_2, which implies $t = O(n^2 L)$. Therefore, the above algorithm terminates after $O(n^2 L)$ iterations, each iteration taking polynomial time, which gives us an overall running time of $poly(n, m, L)$.

Now all that is required is a polynomial time **Separating Oracle**, which checks whether a point lies in S or not, and returns a separating hyperplane in the latter case. We can easily say that the oracle is polynomial time as it simply need to put the point in the hyperplane equation if any one fails then return no with that hyperplane, otherwise go on checking all the hyperplanes and return yes at the end, so this is polynomial in the size of input.

B. Implementation Repository

The complete implementations of all algorithms discussed in this report are available in the following GitHub repository:

https://github.com/Ashutosh-iitrpr/Approximate-Algorithms.git

This repository contains well-documented source code for each algorithm, organized according to the problem statements and corresponding approximation techniques presented in this report.