

Searching in Graphs

Michał Szyfelbein

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology, Poland

November 3, 2025

Binary Search

Binary Search – a classical strategy used to efficiently locate a hidden **target** element x in a linearly ordered set S using $O(\log n)$ comparison operations.

Binary Search

Binary Search – a classical strategy used to efficiently locate a hidden **target** element x in a linearly ordered set S using $O(\log n)$ comparison operations.

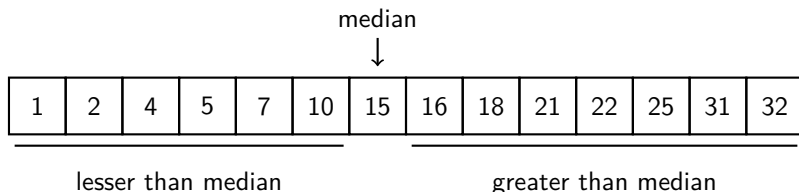


Figure: Example of a sorted array containing 14 elements.

Searching in Graphs

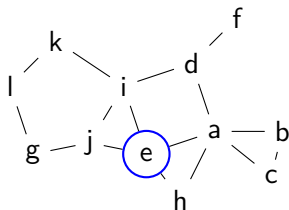
Easy to generalize to graphs:

A **query** to a vertex v returns information whether v is the target x , and if not, which connected component of $G - v$ contains x .

Searching in Graphs

Easy to generalize to graphs:

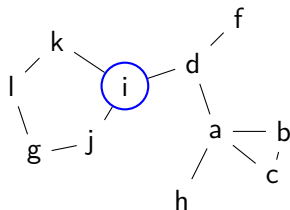
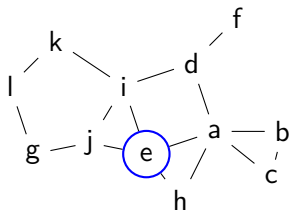
A **query** to a vertex v returns information whether v is the target x , and if not, which connected component of $G - v$ contains x .



Searching in Graphs

Easy to generalize to graphs:

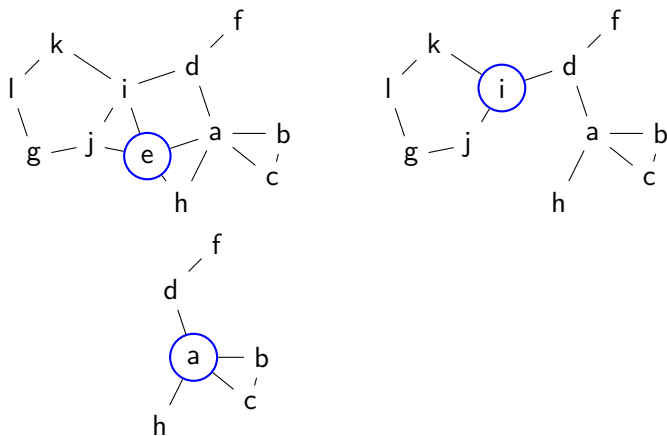
A **query** to a vertex v returns information whether v is the target x , and if not, which connected component of $G - v$ contains x .



Searching in Graphs

Easy to generalize to graphs:

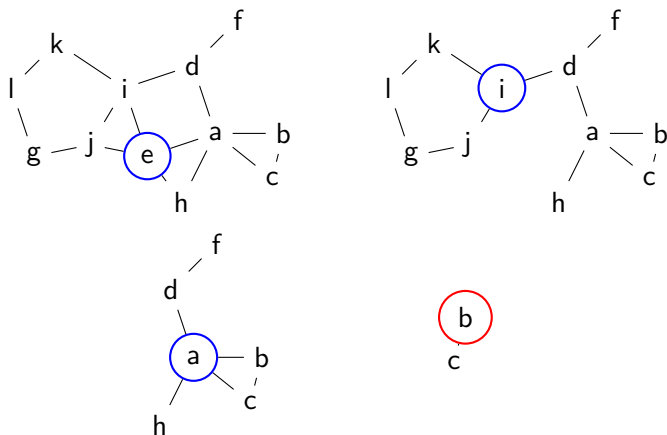
A **query** to a vertex v returns information whether v is the target x , and if not, which connected component of $G - v$ contains x .



Searching in Graphs

Easy to generalize to graphs:

A **query** to a vertex v returns information whether v is the target x , and if not, which connected component of $G - v$ contains x .



Strategy of searching

We wish to find a strategy of searching.

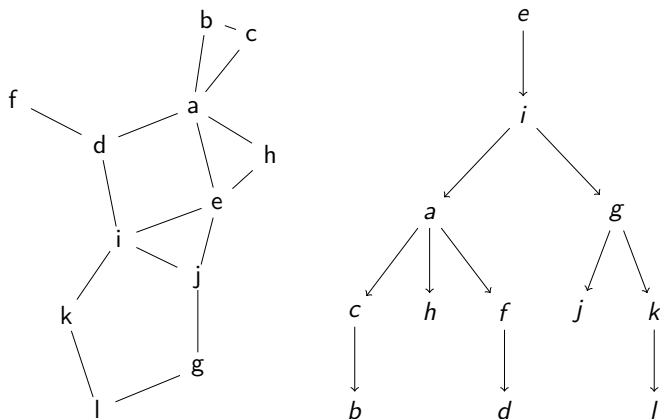


Figure: Sample input graph a decision tree for it.

Additional parameters

We introduce additional information about the input:

- ▶ To each vertex v we assign an arbitrary **cost** $c(v)$, which denotes a cost of performing a query to v .

Additional parameters

We introduce additional information about the input:

- ▶ To each vertex v we assign an arbitrary **cost** $c(v)$, which denotes a cost of performing a query to v .
- ▶ Additionally, to each vertex v we also assign an arbitrary **weight** $c(v)$, which denotes the importance of v .

Additional parameters

We introduce additional information about the input:

- ▶ To each vertex v we assign an arbitrary **cost** $c(v)$, which denotes a cost of performing a query to v .
- ▶ Additionally, to each vertex v we also assign an arbitrary **weight** $w(v)$, which denotes the importance of v .
- ▶ For any $S \subseteq V(G)$, we denote $c(S) = \sum_{v \in S} c(v)$ and $w(S) = \sum_{v \in S} w(v)$.

Our setup

What is the best strategy of searching in a graph?

Graph Search Problem (GSP)

Input: Graph G , a query cost function $c: V(G) \rightarrow \mathbb{N}$ and a weight function $w: V(G) \rightarrow \mathbb{N}$.

Our setup

What is the best strategy of searching in a graph?

Graph Search Problem (GSP)

Input: Graph G , a query cost function $c: V(G) \rightarrow \mathbb{N}$ and a weight function $w: V(G) \rightarrow \mathbb{N}$.

Output: A decision tree D minimizing the weighted average search cost:

$$c_G(D) = \sum_{x \in V(G)} w(x) \cdot c(Q_G(D, x))$$

where $Q_G(D, x)$ denotes the set of queries performed along the unique path in D from the root $r(D)$ to x .

Decision Tree

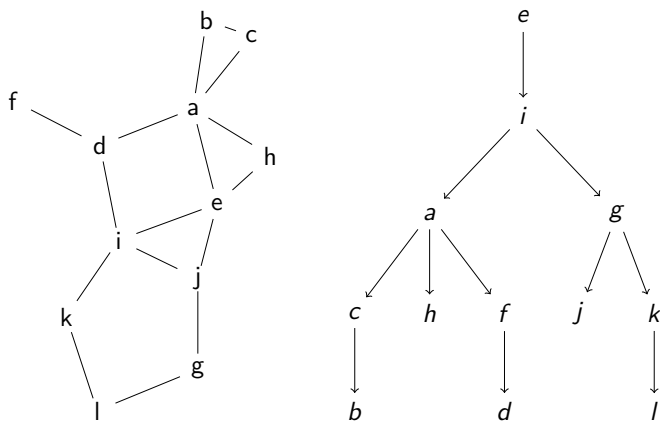


Figure: Sample input graph a decision tree for it.

Why do we care?

Useful in:

- ▶ Automated bug detection in computer code,
- ▶ Hierarchical clustering of data.

Why do we care?

Useful in:

- ▶ Automated bug detection in computer code,
- ▶ Hierarchical clustering of data.

Related to:

- ▶ Scheduling of parallel database join operations,
- ▶ Parallel Cholesky factorization of matrices,
- ▶ Parallel assembly of multi-part products from their components.

Why do we care?

Why our setup?:

1. **Average case** – it is natural to assume that the search strategies we design are intended to be used repeatedly.

Why do we care?

Why our setup?:

1. **Average case** – it is natural to assume that the search strategies we design are intended to be used repeatedly.
2. **Weight function** – some vertices may serve as targets more frequently than the others.

Why do we care?

Why our setup?:

1. **Average case** – it is natural to assume that the search strategies we design are intended to be used repeatedly.
2. **Weight function** – some vertices may serve as targets more frequently than the others.
3. **Query costs** – performing a query may require significant resources, such as time or money.

Why do we care?

Why our setup?:

1. **Average case** – it is natural to assume that the search strategies we design are intended to be used repeatedly.
2. **Weight function** – some vertices may serve as targets more frequently than the others.
3. **Query costs** – performing a query may require significant resources, such as time or money.
4. **Have not yet been investigated.**

Other variants

Notation: **search space** || **query model** || **objective function**:

Other variants

Notation: **search space** || **query model** || **objective function**:

- ▶ $T || V || \sum C_i$:
 - ▶ $(1 + \epsilon)$ in $O\left((1/\epsilon)^{2/\log_2 3} \cdot n^{1+4/\log_2 3} \cdot \log^2(n/\epsilon)\right)$ time.
 - ▶ Every optimal decision tree has height at most $O(\log w(T))$.
 - ▶ Weighted centroid decision tree is 2-approximate.

Other variants

Notation: **search space** || **query model** || **objective function**:

- ▶ $T || V || \sum C_i$:
 - ▶ $(1 + \epsilon)$ in $O\left((1/\epsilon)^{2/\log_2 3} \cdot n^{1+4/\log_2 3} \cdot \log^2(n/\epsilon)\right)$ time.
 - ▶ Every optimal decision tree has height at most $O(\log w(T))$.
 - ▶ Weighted centroid decision tree is 2-approximate.
- ▶ $T || V || \sum C_{\max}$:
 - ▶ Solvable in $O(n)$ time, $\lfloor \log n \rfloor + 1$ queries always suffice.

Other variants

Notation: **search space** || **query model** || **objective function**:

- ▶ $T || V || \sum C_i$:
 - ▶ $(1 + \epsilon)$ in $O\left((1/\epsilon)^{2/\log_2 3} \cdot n^{1+4/\log_2 3} \cdot \log^2(n/\epsilon)\right)$ time.
 - ▶ Every optimal decision tree has height at most $O(\log w(T))$.
 - ▶ Weighted centroid decision tree is 2-approximate.
- ▶ $T || V || \sum C_{max}$:
 - ▶ Solvable in $O(n)$ time, $\lfloor \log n \rfloor + 1$ queries always suffice.
- ▶ $G || V || \sum C_{max}$
 - ▶ NP-hard, best known approximation is $O(\log^{3/2} n)$.

Other variants

Notation: **search space** || **query model** || **objective function**:

- ▶ $T || V || \sum C_i$:
 - ▶ $(1 + \epsilon)$ in $O\left((1/\epsilon)^{2/\log_2 3} \cdot n^{1+4/\log_2 3} \cdot \log^2(n/\epsilon)\right)$ time.
 - ▶ Every optimal decision tree has height at most $O(\log w(T))$.
 - ▶ Weighted centroid decision tree is 2-approximate.
- ▶ $T || V || \sum C_{max}$:
 - ▶ Solvable in $O(n)$ time, $\lfloor \log n \rfloor + 1$ queries always suffice.
- ▶ $G || V || \sum C_{max}$
 - ▶ NP-hard, best known approximation is $O(\log^{3/2} n)$.
- ▶ $T || V, c || \sum C_{max}$
 - ▶ NP-hard, best known approximation is $O(\sqrt{\log n})$.

Other variants

Notation: **search space** || **query model** || **objective function**:

- ▶ $T || V || \sum C_i$:
 - ▶ $(1 + \epsilon)$ in $O\left((1/\epsilon)^{2/\log_2 3} \cdot n^{1+4/\log_2 3} \cdot \log^2(n/\epsilon)\right)$ time.
 - ▶ Every optimal decision tree has height at most $O(\log w(T))$.
 - ▶ Weighted centroid decision tree is 2-approximate.
- ▶ $T || V || \sum C_{max}$:
 - ▶ Solvable in $O(n)$ time, $\lfloor \log n \rfloor + 1$ queries always suffice.
- ▶ $G || V || \sum C_{max}$
 - ▶ NP-hard, best known approximation is $O(\log^{3/2} n)$.
- ▶ $T || V, c || \sum C_{max}$
 - ▶ NP-hard, best known approximation is $O(\sqrt{\log n})$.
- ▶ $G || V, c || \sum C_{max}$
 - ▶ Reducible to the uniform cost version.

Many names

- ▶ Binary Search [Knu73; OP06; Der+17],
- ▶ Tree Search Problem [Jac+10; Cic+16],
- ▶ Binary Identification Problem [Cic+12],
- ▶ Ranking Colorings [LY98; Der06],
- ▶ Ordered Colorings [KMS95],
- ▶ Elimination Trees [Pot88],
- ▶ Hub Labeling [Ang18],
- ▶ Tree-Depth [NO06],
- ▶ Partition Trees [Høg24],
- ▶ Hierarchical Clustering [Das16; Coh+19; CC17],
- ▶ Search Trees on Trees [BK22; Ber+22],
- ▶ LIFO-Search [GHT12].

How to tackle the problem

Bad news: The Graph Search Problem is **NP-hard** even when restricted to bounded degree trees and bounded diameter trees.

How to tackle the problem

Bad news: The Graph Search Problem is **NP-hard** even when restricted to bounded degree trees and bounded diameter trees. We want to find an algorithm providing a good **approximation** for the problem:

- ▶ $(4 + \epsilon)$ -approximation for **trees**.
- ▶ $O(\sqrt{\log n})$ -approximation for **general graphs**.

Weighted α -Separator Problem

Weighted α -Separator Problem

Input: Graph G , a cost function $c: V \rightarrow \mathbb{N}$, a weight function $w: V \rightarrow \mathbb{N}$ and a real number α .

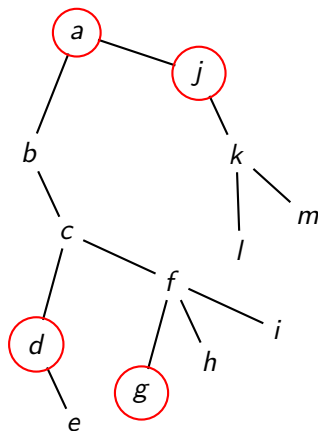
Weighted α -Separator Problem

Weighted α -Separator Problem

Input: Graph G , a cost function $c: V \rightarrow \mathbb{N}$, a weight function $w: V \rightarrow \mathbb{N}$ and a real number α .

Output: A set $S \subseteq V(G)$ called **separator** such that for every $H \in G - S$, $w(H) \leq w(G)/\alpha$ and $c(S)$ is minimized.

Example of a separator



	$w(v)$	$c(v)$
<i>a</i>	2	3
<i>b</i>	1	4
<i>c</i>	3	6
<i>d</i>	2	2
<i>e</i>	4	1
<i>f</i>	0	3
<i>g</i>	1	1
<i>h</i>	4	3
<i>i</i>	2	3
<i>j</i>	5	2
<i>k</i>	1	2
<i>l</i>	2	3
<i>m</i>	3	4

Figure: Sample input tree T and a weighted 3-separator (the circled vertices) of cost 8.

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well.

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well. But there exists a bicriteria **FPTAS** for trees:

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well. But there exists a bicriteria **FPTAS** for trees:

Theorem

Let S be an optimal weighted α -separator for (T, c, w, α) . For any $\delta > 0$ there exists an algorithm `SeparatorFPTAS`, which returns a separator S' , such that:

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well. But there exists a bicriteria **FPTAS** for trees:

Theorem

Let S be an optimal weighted α -separator for (T, c, w, α) . For any $\delta > 0$ there exists an algorithm `SeparatorFPTAS`, which returns a separator S' , such that:

1. $c(S') \leq c(S)$.

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well. But there exists a bicriteria **FPTAS** for trees:

Theorem

Let S be an optimal weighted α -separator for (T, c, w, α) . For any $\delta > 0$ there exists an algorithm SeparatorFPTAS, which returns a separator S' , such that:

1. $c(S') \leq c(S)$.
2. $w(H) \leq \frac{(1+\delta) \cdot w(T)}{\alpha}$ for every $H \in T - S'$.

How to find the separator

Bad news again: The Weighted α -separator Problem is **NP-hard** as well. But there exists a bicriteria **FPTAS** for trees:

Theorem

Let S be an optimal weighted α -separator for (T, c, w, α) . For any $\delta > 0$ there exists an algorithm `SeparatorFPTAS`, which returns a separator S' , such that:

1. $c(S') \leq c(S)$.
2. $w(H) \leq \frac{(1+\delta) \cdot w(T)}{\alpha}$ for every $H \in T - S'$.
3. The algorithm runs in $O(n^3/\delta^2)$ time.

Notation

To connect searching and separating we need the following notation:

- ▶ $\mathcal{R}_D(G)$ – the family of all candidate subsets of D in G .

Notation

To connect searching and separating we need the following notation:

- ▶ $\mathcal{R}_D(G)$ – the family of all candidate subsets of D in G .
- ▶ D^* – optimal decision tree.

Notation

To connect searching and separating we need the following notation:

- ▶ $\mathcal{R}_D(G)$ – the family of all candidate subsets of D in G .
- ▶ D^* – optimal decision tree.
- ▶ \mathcal{L}_k^* – the k -th **level** of $\text{OPT}(G)$: the subset of $\mathcal{R}_{D^*}(G)$ consisting of all maximal elements H of $\mathcal{R}_{D^*}(G)$ with $w(H) \leq k$.

Notation

To connect searching and separating we need the following notation:

- ▶ $\mathcal{R}_D(G)$ – the family of all candidate subsets of D in G .
- ▶ D^* – optimal decision tree.
- ▶ \mathcal{L}_k^* – the k -th **level** of $\text{OPT}(G)$: the subset of $\mathcal{R}_{D^*}(G)$ consisting of all maximal elements H of $\mathcal{R}_{D^*}(G)$ with $w(H) \leq k$.
- ▶ $S_k^* = V(G) - \mathcal{L}_k^*$ – vertices belonging to the separator at the level \mathcal{L}_k^* . S_k^* forms a weighted $w(G)/k$ -separator of G .

Example of the connection

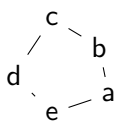


Figure: G .



Figure: D^* .

Example of the connection

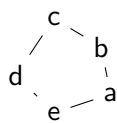


Figure: G .

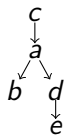


Figure: D^* .

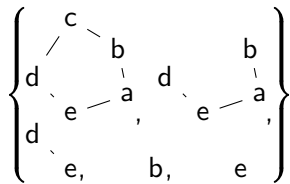


Figure: $\mathcal{R}_{D^*}(G)$.

Example of the connection

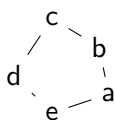


Figure: G .



Figure: D^* .

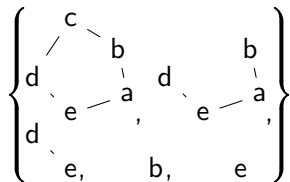


Figure: $\mathcal{R}_{D^*}(G)$.

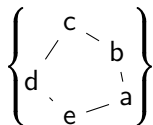


Figure: $\mathcal{L}_5^*, S_5^* = \{\}$.

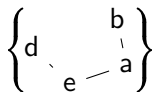


Figure: $\mathcal{L}_4^*, S_4^* = \{c\}$.

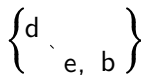


Figure: $\mathcal{L}_3^*, S_3^* = \{a, c\}$.

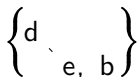


Figure: $\mathcal{L}_2^*, S_2^* = \{a, c\}$.



Figure:



Figure:

$\mathcal{L}_1^*, S_1^* = \{a, b, c, d, e\}$.

Basic lemmas

Lemma

Let $G_{D,v}$ be the candidate subgraph of G in which v is queried when using D . Then, $c_G(D) = \sum_{v \in V(G)} w(G_{D,v}) \cdot c(v)$.

Basic lemmas

Lemma

Let $G_{D,v}$ be the candidate subgraph of G in which v is queried when using D . Then, $c_G(D) = \sum_{v \in V(G)} w(G_{D,v}) \cdot c(v)$.

Lemma

$$OPT(G) = \sum_{k=0}^{w(G)-1} c(S_k^*).$$

Basic lemmas

Lemma

Let $G_{D,v}$ be the candidate subgraph of G in which v is queried when using D . Then, $c_G(D) = \sum_{v \in V(G)} w(G_{D,v}) \cdot c(v)$.

Lemma

$$OPT(G) = \sum_{k=0}^{w(G)-1} c(S_k^*).$$

Proof.

Consider any vertex v . For every $0 \leq k < w(G_{D^*,v})$, $v \notin \bigcup_{H \in \mathcal{L}_k^*} H$, so $v \in S_k^*$ and the contribution of v to the cost is $w(G_{D^*,v}) \cdot c(v)$:

$$\sum_{k=0}^{w(G)-1} c(S_k^*) = \sum_{v \in V(G)} \sum_{k=0}^{w(G_{D^*,v})-1} c(v) = OPT(G).$$

Basic lemmas

Lemma

$$2 \cdot OPT(G) = 2 \cdot \sum_{k=0}^{w(G)-1} c(S_k^*) \geq \sum_{k=0}^{w(G)} c(S_{\lfloor k/2 \rfloor}^*).$$

Basic lemmas

Lemma

$$2 \cdot \text{OPT}(G) = 2 \cdot \sum_{k=0}^{w(G)-1} c(S_k^*) \geq \sum_{k=0}^{w(G)} c(S_{\lfloor k/2 \rfloor}^*).$$

Lemma

Let \mathcal{G} be any subgraph of G and $0 \leq \beta \leq 1$. Then:

$$\beta \cdot w(\mathcal{G}) \cdot c(S_{\lfloor w(\mathcal{G})/2 \rfloor}^* \cap \mathcal{G}) \leq \sum_{k=(1-\beta)w(\mathcal{G})+1}^{w(\mathcal{G})} c(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{G}).$$

Searching in Trees

We will iteratively use the **FPTAS** for the Weighted α -separator problem to create an $(4 + \epsilon)$ -approximation algorithm for the Tree Search Problem:

Searching in Trees

We will iteratively use the **FPTAS** for the Weighted α -separator problem to create an $(4 + \epsilon)$ -approximation algorithm for the Tree Search Problem:

Theorem

For any $\epsilon > 0$ there exists an $(4 + \epsilon)$ -approximation algorithm for the Tree Search Problem running in $O(n^4/\epsilon^2)$ time.

The algorithm

proc DecisionTree(T, c, w, ϵ):

1. $S_T \leftarrow \text{SeparatorFPTAS}\left(T, c, w, \alpha = 2, \delta = \frac{\epsilon}{4+\epsilon}\right)$.
2. $D_T \leftarrow$ arbitrary partial decision tree for T , built from vertices of S_T .
3. For each $H \in T - S_T$:
 - 3.1 $D_H \leftarrow \text{DecisionTree}(H, c, w, \epsilon)$.
 - 3.2 Hang D_H in D_T below the last query to $v \in N_T(H)$.
4. Return D_T .

Structure of the solution

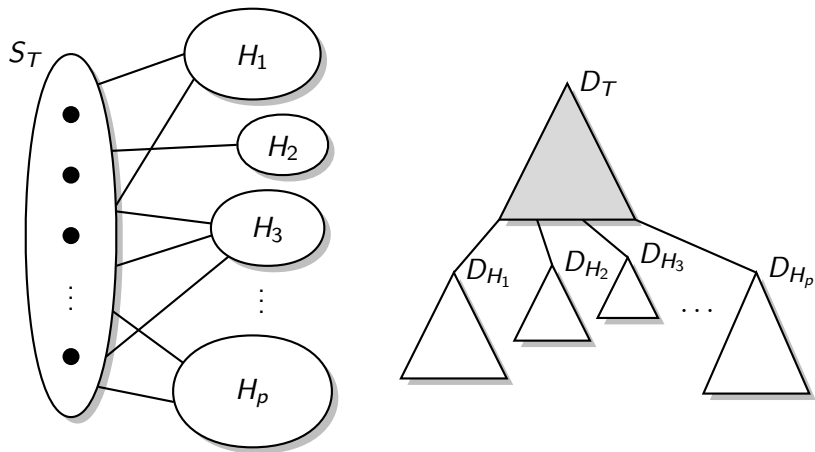


Figure: The separator S_T produced by the algorithm and the structure of the decision tree built using S_T .

Sketch of the proof

- ▶ \mathcal{T} – subtree of T for which the procedure was called.

Sketch of the proof

- ▶ \mathcal{T} – subtree of T for which the procedure was called.
- ▶ Let $S_{\mathcal{T}}^* = S_{\lfloor w(\mathcal{T})/2 \rfloor}^* \cap \mathcal{T}$. We have $c(S_{\mathcal{T}}) \leq c(S_{\mathcal{T}}^*)$.

Sketch of the proof

- ▶ \mathcal{T} – subtree of T for which the procedure was called.
- ▶ Let $S_{\mathcal{T}}^* = S_{\lfloor w(\mathcal{T})/2 \rfloor}^* \cap \mathcal{T}$. We have $c(S_{\mathcal{T}}) \leq c(S_{\mathcal{T}}^*)$.
- ▶ We have that the contribution of the decision tree $D_{\mathcal{T}}$ is bounded by:

Sketch of the proof

- ▶ \mathcal{T} – subtree of T for which the procedure was called.
- ▶ Let $S_{\mathcal{T}}^* = S_{\lfloor w(\mathcal{T})/2 \rfloor}^* \cap \mathcal{T}$. We have $c(S_{\mathcal{T}}) \leq c(S_{\mathcal{T}}^*)$.
- ▶ We have that the contribution of the decision tree $D_{\mathcal{T}}$ is bounded by:

$$\begin{aligned} w(\mathcal{T}) \cdot c(S_{\mathcal{T}}) &\leq w(\mathcal{T}) \cdot c(S_{\mathcal{T}}^*) \\ &\leq \frac{2}{1-\delta} \cdot \sum_{k=\frac{1+\delta}{2} \cdot w(\mathcal{T})+1}^{w(\mathcal{T})} c\left(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{T}\right). \end{aligned}$$

Sketch of the proof

Lemma

$$\sum_{\mathcal{T}} \sum_{k=\frac{1+\delta}{2} \cdot w(\mathcal{T})+1}^{w(\mathcal{T})} c\left(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{T}\right) \leq \sum_{k=0}^{w(T)} c\left(S_{\lfloor k/2 \rfloor}^*\right).$$

Let $\beta = \frac{1-\delta}{2}$. We have:

$$\begin{aligned} c_T(D) &\leq \frac{2}{1-\delta} \cdot \sum_{\mathcal{T}} \sum_{k=\frac{1+\delta}{2} \cdot w(\mathcal{T})+1}^{w(\mathcal{T})} c\left(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{T}\right) \\ &\leq \frac{2}{1-\delta} \cdot \sum_{k=0}^{w(T)} c\left(S_{\lfloor k/2 \rfloor}^*\right) \leq (4 + \epsilon) \cdot \text{OPT}(T). \end{aligned}$$

Min-Ratio Vertex Cut Problem

Min-Ratio Vertex Cut Problem

Input: Graph $G = (V(G), E(G))$, the cost function $c: V \rightarrow \mathbb{N}$ and the weight function $w: V \rightarrow \mathbb{N}$.

Min-Ratio Vertex Cut Problem

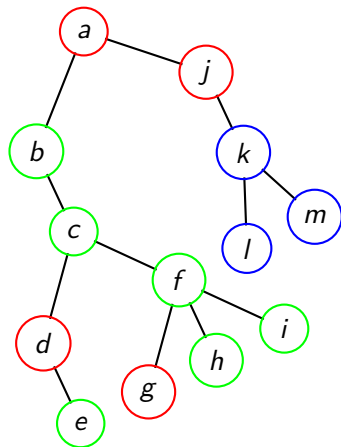
Min-Ratio Vertex Cut Problem

Input: Graph $G = (V(G), E(G))$, the cost function $c: V \rightarrow \mathbb{N}$ and the weight function $w: V \rightarrow \mathbb{N}$.

Output: A partition (A, S, B) of $V(G)$ called **vertex-cut**, such that there are no $u \in A$ and $v \in B$ for which $uv \in E(G)$, minimizing the ratio:

$$\alpha_{c,w}(A, S, B) = \frac{c(S)}{w(A \cup S) \cdot w(B \cup S)}.$$

Example of a vertex cut



	$w(v)$	$c(v)$
a	2	3
b	1	4
c	3	6
d	2	2
e	4	1
f	0	3
g	1	1
h	4	3
i	2	3
j	5	2
k	1	2
l	2	3
m	3	4

Figure: A tree and a vertex cut of ratio

$$\alpha_{c,w}(A, S, B) = \frac{c(S)}{w(A \cup S) \cdot w(B \cup S)} = \frac{8}{(6+10) \cdot (14+10)} = \frac{8}{384} = \frac{1}{48}.$$

How to find the cut

Min-Ratio Vertex Cut Problem is also **NP-hard**. However, we use the following result of [FHL05]:

Theorem

Given a graph $G = (V(G), E(G))$, the cost function $c : V \rightarrow \mathbb{N}$ and the weight function $w : V \rightarrow \mathbb{N}$, there exists a polynomial-time algorithm, which computes a partition (A, S, B) , such that:

$$\alpha_{c,w}(A, S, B) = O\left(\sqrt{\log n}\right) \cdot \alpha_{c,w}(G).$$

Searching in Graphs

We will iteratively use the f_n -approximation algorithm for the Min-Ratio Vertex Cut Problem to create an $O(f_n)$ -approximation algorithm for the Graph Search Problem:

Searching in Graphs

We will iteratively use the f_n -approximation algorithm for the Min-Ratio Vertex Cut Problem to create an $O(f_n)$ -approximation algorithm for the Graph Search Problem:

Theorem

Let f_n be the approximation ratio of any polynomial time algorithm for the Min-Ratio Vertex Cut Problem. Then, there exists an $O(f_n)$ -approximation algorithm for the Graph Search Problem, running in polynomial time.

The algorithm

proc DecisionTree(T, c, w, ϵ):

1. $A_G, S_G, B_G \leftarrow \text{AlgorithmMinCut}(G, c, w)$.
2. $D_G \leftarrow$ arbitrary partial decision tree for G , built from vertices of S_G .
3. For each $H \in G - S_G$:
 - 3.1 $D_H \leftarrow \text{DecisionTree}(H, c, w)$.
 - 3.2 Hang D_H in D_G below the last query to $v \in N_G(H)$.
4. Return D_G .

Sketch of the proof

- ▶ \mathcal{G} – any subgraph of G , for which the procedure was called.

Sketch of the proof

- ▶ \mathcal{G} – any subgraph of G , for which the procedure was called.
- ▶ Let $S_{\mathcal{G}}^* = S_{\lfloor w(\mathcal{G})/2 \rfloor}^* \cap \mathcal{G}$.

Sketch of the proof

- ▶ \mathcal{G} – any subgraph of G , for which the procedure was called.
- ▶ Let $S_{\mathcal{G}}^* = S_{\lfloor w(\mathcal{G})/2 \rfloor}^* \cap \mathcal{G}$.

Lemma

Let $\mathcal{H} = \mathcal{G} - S_{\mathcal{G}}^$. Then, we can partition \mathcal{H} into two sets, \mathcal{A} and \mathcal{B} such that for $A = \bigcup_{H \in \mathcal{A}} V(H)$ and $B = \bigcup_{H \in \mathcal{B}} V(H)$, we have:*

$$w(A \cup S_{\mathcal{G}}^*) \cdot w(B \cup S_{\mathcal{G}}^*) \geq w(\mathcal{G})^2 / 11.$$

Sketch of the proof

- ▶ \mathcal{G} – any subgraph of G , for which the procedure was called.
- ▶ Let $S_{\mathcal{G}}^* = S_{\lfloor w(\mathcal{G})/2 \rfloor}^* \cap \mathcal{G}$.

Lemma

Let $\mathcal{H} = \mathcal{G} - S_{\mathcal{G}}^*$. Then, we can partition \mathcal{H} into two sets, \mathcal{A} and \mathcal{B} such that for $A = \bigcup_{H \in \mathcal{A}} V(H)$ and $B = \bigcup_{H \in \mathcal{B}} V(H)$, we have:

$$w(A \cup S_{\mathcal{G}}^*) \cdot w(B \cup S_{\mathcal{G}}^*) \geq w(\mathcal{G})^2 / 11.$$

The partition $(A, S_{\mathcal{G}}^*, B)$ in the above lemma is a vertex cut of \mathcal{G} , so we have:

$$\alpha_{c,w}(\mathcal{G}) \leq \frac{c(S_{\mathcal{G}}^*)}{w(A \cup S_{\mathcal{G}}^*) \cdot w(B \cup S_{\mathcal{G}}^*)} \leq \frac{11 \cdot c(S_{\mathcal{G}}^*)}{w(\mathcal{G})^2}.$$

Sketch of the proof

- Let $(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}})$, be the partition returned by the algorithm:

$$\alpha_{c,w}(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}}) = \frac{c(S_{\mathcal{G}})}{w(A_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}})} \leq f_n \cdot \frac{11 \cdot c(S_{\mathcal{G}}^*)}{w(\mathcal{G})^2}.$$

Sketch of the proof

- Let $(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}})$, be the partition returned by the algorithm:

$$\alpha_{c,w}(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}}) = \frac{c(S_{\mathcal{G}})}{w(A_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}})} \leq f_n \cdot \frac{11 \cdot c(S_{\mathcal{G}}^*)}{w(\mathcal{G})^2}.$$

- Let $\beta = w(B_{\mathcal{G}} \cup S_{\mathcal{G}}) / w(\mathcal{G})$, so that $(1 - \beta) \cdot w(\mathcal{G}) = w(A_{\mathcal{G}})$.

Sketch of the proof

- ▶ Let $(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}})$, be the partition returned by the algorithm:

$$\alpha_{c,w}(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}}) = \frac{c(S_{\mathcal{G}})}{w(A_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}})} \leq f_n \cdot \frac{11 \cdot c(S_{\mathcal{G}}^*)}{w(\mathcal{G})^2}.$$

- ▶ Let $\beta = w(B_{\mathcal{G}} \cup S_{\mathcal{G}}) / w(\mathcal{G})$, so that $(1 - \beta) \cdot w(\mathcal{G}) = w(A_{\mathcal{G}})$.
- ▶ Assume w. l. o. s. that $w(A_{\mathcal{G}}) \geq w(B_{\mathcal{G}})$.

Sketch of the proof

- ▶ Let $(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}})$, be the partition returned by the algorithm:

$$\alpha_{c,w}(A_{\mathcal{G}}, S_{\mathcal{G}}, B_{\mathcal{G}}) = \frac{c(S_{\mathcal{G}})}{w(A_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}})} \leq f_n \cdot \frac{11 \cdot c(S_{\mathcal{G}}^*)}{w(\mathcal{G})^2}.$$

- ▶ Let $\beta = w(B_{\mathcal{G}} \cup S_{\mathcal{G}}) / w(\mathcal{G})$, so that $(1 - \beta) \cdot w(\mathcal{G}) = w(A_{\mathcal{G}})$.
- ▶ Assume w. l. o. s. that $w(A_{\mathcal{G}}) \geq w(B_{\mathcal{G}})$.
- ▶ The contribution of the decision tree $D_{\mathcal{G}}$ is bounded by:

$$\begin{aligned} w(\mathcal{G}) \cdot c(S_{\mathcal{G}}) &\leq 11 \cdot f_n \cdot \frac{w(A_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}})}{w(\mathcal{G})} \cdot c(S_{\mathcal{G}}^*) \\ &\leq 11 \cdot f_n \cdot w(B_{\mathcal{G}} \cup S_{\mathcal{G}}) \cdot c(S_{\mathcal{G}}^*) \\ &\leq 11 \cdot f_n \cdot \sum_{k=w(A_{\mathcal{G}})+1}^{w(\mathcal{G})} c(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{G}) \end{aligned}$$

Sketch of the proof

Let D be the decision tree returned by the procedure. We have:

$$\begin{aligned}c_G(D) &\leq \sum_{\mathcal{G}} w(\mathcal{G}) \cdot c(S_{\mathcal{G}}) \\&\leq 11 \cdot f_n \cdot \sum_{\mathcal{G}} \sum_{k=w(A_{\mathcal{G}})+1}^{w(\mathcal{G})} c\left(S_{\lfloor k/2 \rfloor}^* \cap \mathcal{G}\right) \\&\leq 11 \cdot f_n \cdot \sum_{k=0}^{w(G)} c\left(S_{\lfloor k/2 \rfloor}^*\right) \\&\leq 22 \cdot f_n \cdot \text{OPT}(G)\end{aligned}$$

What's next?

- ▶ The results for the **average case** variant of the Graph Search Problem far **outperform** those for the **worst case** variant:
 - ▶ For trees: $(4 + \epsilon)$ vs. $O(\sqrt{\log n})$,
 - ▶ For general graphs: $O(\sqrt{\log n})$ vs $O(\log^{3/2} n)$.

What's next?

- ▶ The results for the **average case** variant of the Graph Search Problem far **outperform** those for the **worst case** variant:
 - ▶ For trees: $(4 + \epsilon)$ vs. $O(\sqrt{\log n})$,
 - ▶ For general graphs: $O(\sqrt{\log n})$ vs $O(\log^{3/2} n)$.
- ▶ **Greedy** algorithms do not work well in the **worst case** setup. However we think there is a different way of connecting searching and separating which may yield a constant factor approximation for trees.

What's next?

- ▶ The results for the **average case** variant of the Graph Search Problem far **outperform** those for the **worst case** variant:
 - ▶ For trees: $(4 + \epsilon)$ vs. $O(\sqrt{\log n})$,
 - ▶ For general graphs: $O(\sqrt{\log n})$ vs $O(\log^{3/2} n)$.
- ▶ **Greedy** algorithms do not work well in the **worst case** setup. However we think there is a different way of connecting searching and separating which may yield a constant factor approximation for trees.
- ▶ For general graphs we are working on an $O(\log n)$ approximation.

Thank you for your attention!

Questions?

Bibliography I

- [Ang18] Haris Angelidakis. “Shortest path queries, graph partitioning and covering problems in worst and beyond worst case settings”. In: *ArXiv abs/1807.09389* (2018). URL: <https://api.semanticscholar.org/CorpusID:51718679>.
- [BK22] Benjamin Berendsohn and László Kozma. “Splay trees on trees”. In: Jan. 2022, pp. 1875–1900. ISBN: 978-1-61197-707-3. DOI: 10.1137/1.9781611977073.75.
- [Ber+22] Benjamin Berendsohn et al. *Fast approximation of search trees on trees with centroid trees*. Sept. 2022. DOI: 10.48550/arXiv.2209.08024.
- [CC17] Moses Charikar and Vaggos Chatziafratis. “Approximate hierarchical clustering via sparsest cut and spreading metrics”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '17. Barcelona, Spain: Society for Industrial and Applied Mathematics, 2017, pp. 841–854.
- [Cic+12] Ferdinando Cicalese et al. “The binary identification problem for weighted trees”. In: *Theoretical Computer Science* 459 (2012), pp. 100–112. ISSN: 0304-3975. DOI: <https://doi.org/10.1016/j.tcs.2012.06.023>.
- [Cic+16] Ferdinando Cicalese et al. “On the tree search problem with non-uniform costs”. In: *Theoretical Computer Science* 647 (2016), pp. 22–32. ISSN: 0304-3975. DOI: <https://doi.org/10.1016/j.tcs.2016.07.019>.
- [Coh+19] Vincent Cohen-addad et al. “Hierarchical Clustering: Objective Functions and Algorithms”. In: *J. ACM* 66.4 (June 2019). ISSN: 0004-5411. DOI: 10.1145/3321386. URL: <https://doi.org/10.1145/3321386>.

Bibliography II

- [Das16] Sanjoy Dasgupta. "A cost function for similarity-based hierarchical clustering". In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '16. Cambridge, MA, USA: Association for Computing Machinery, 2016, pp. 118–127. ISBN: 9781450341325. DOI: 10.1145/2897518.2897527. URL: <https://doi.org/10.1145/2897518.2897527>.
- [Der06] Dariusz Dereniowski. "Edge ranking of weighted trees". In: *Discrete Applied Mathematics* 154.8 (2006), pp. 1198–1209. ISSN: 0166-218X. DOI: <https://doi.org/10.1016/j.dam.2005.11.005>.
- [Der+17] Dariusz Dereniowski et al. "Approximation Strategies for Generalized Binary Search in Weighted Trees". In: *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Ed. by Ioannis Chatzigiannakis et al. Vol. 80. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 84:1–84:14. ISBN: 978-3-95977-041-5. DOI: 10.4230/LIPIcs.ICALP.2017.84.
- [FHL05] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. "Improved approximation algorithms for minimum-weight vertex separators". In: *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA: Association for Computing Machinery, 2005, pp. 563–572. ISBN: 1581139608. DOI: 10.1145/1060590.1060674. URL: <https://doi.org/10.1145/1060590.1060674>.
- [GHT12] Archontia C. Giannopoulou, Paul Hunter, and Dimitrios M. Thilikos. "LIFO-search: A min–max theorem and a searching game for cycle-rank and tree-depth". In: *Discrete Applied Mathematics* 160.15 (2012), pp. 2089–2097. ISSN: 0166-218X. DOI: <https://doi.org/10.1016/j.dam.2012.03.015>. URL: <https://www.sciencedirect.com/science/article/pii/S0166218X12001199>.

Bibliography III

- [Høg24] Svein Høgemo. "Tight Approximation Bounds on a Simple Algorithm for Minimum Average Search Time in Trees". In: *ArXiv abs/2402.05560* (2024). URL: <https://api.semanticscholar.org/CorpusID:267547530>.
- [Jac+10] Tobias Jacobs et al. "On the Complexity of Searching in Trees: Average-Case Minimization". In: *Automata, Languages and Programming*. Ed. by Samson Abramsky et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 527–539. ISBN: 978-3-642-14165-2.
- [KMS95] Meir Katchalski, William McCuaig, and Suzanne Seager. "Ordered colourings". In: *Discrete Mathematics* 142.1 (1995), pp. 141–154. ISSN: 0012-365X. DOI: [https://doi.org/10.1016/0012-365X\(93\)E0216-Q](https://doi.org/10.1016/0012-365X(93)E0216-Q). URL: <https://www.sciencedirect.com/science/article/pii/0012365X93E0216Q>.
- [Knu73] Donald Knuth. *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973, pp. 391–392.
- [LY98] Tak Wah Lam and Fung Ling Yue. "Edge ranking of graphs is hard". In: *Discrete Applied Mathematics* 85.1 (1998), pp. 71–86. ISSN: 0166-218X. DOI: [https://doi.org/10.1016/S0166-218X\(98\)00029-8](https://doi.org/10.1016/S0166-218X(98)00029-8).
- [NO06] Jaroslav Nešetřil and Patrice Ossona de Mendez. "Tree-depth, subgraph coloring and homomorphism bounds". In: *European Journal of Combinatorics* 27.6 (2006), pp. 1022–1041. ISSN: 0195-6698. DOI: <https://doi.org/10.1016/j.ejc.2005.01.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0195669805000570>.
- [OP06] Krzysztof Onak and Pawel Parys. "Generalization of Binary Search: Searching in Trees and Forest-Like Partial Orders". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 379–388. DOI: 10.1109/FOCS.2006.32.

Bibliography IV

- [Pot88] Alex Pothén. *The Complexity of Optimal Elimination Trees*. Technical Report CS-88-16. Penn State University CS-88-16. Pennsylvania State University, Department of Computer Science, Apr. 1988.