GDAŃSK UNIVERSITY OF TECHNOLOGY

Experimental analysis of binary search models in graphs

Supervised by: prof. dr. hab. inż. Dariusz Dereniowski

Michał Szyfelbein

October 6, 2025

Basic information

The aim: Experimental analysis of selected generalized binary search problems. The aim of the analysis is to verify the hypotheses regarding the effectiveness of search algorithms.

Schedule:

- 2024.01 – 2024.06: Researching and selection of the query model.
- 2024.06 – 2025.05: Development and formal analysis of the proposed algorithms.
- 2025.06 – 2025.10: Selection of models and algorithms for comparison, implementation, thesis writing.
- 2025.11 – 2025.12: Execution of experiments, analysis and interpretation of the results, finalization of the thesis.

Basic information

The aim: Experimental analysis of selected generalized binary search problems. The aim of the analysis is to verify the hypotheses regarding the effectiveness of search algorithms.

Schedule:

- 2024.01 – 2024.06: Researching and selection of the query model.
- 2024.06 – 2025.05: Development and formal analysis of the proposed algorithms.
- 2025.06 – 2025.10: Selection of models and algorithms for comparison, implementation, thesis writing.
- 2025.11 – 2025.12: Execution of experiments, analysis and interpretation of the results, finalization of the thesis.

State of the work

- Implementation, about 60% complete:
  - Language: **python**,
  - Libraries: **networkx**,
  - Environment: **PyCharm**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Binary-Search.
- Thesis, about 80% complete:
  - Language: **LaTeX**,
  - Environment: **Visual Studio Code**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Papers.
- What is left:
  - Experiments,
  - Advanced data generation,
  - Optimization,
  - Bug fixing,
  - Data visualization.

State of the work

- Implementation, about 60% complete:
  - Language: **python**,
  - Libraries: **networkx**,
  - Environment: **PyCharm**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Binary-Search.
- Thesis, about 80% complete:
  - Language: **LaTeX**,
  - Environment: **Visual Studio Code**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Papers.
- What is left:
  - Experiments,
  - Advanced data generation,
  - Optimization,
  - Bug fixing,
  - Data visualization.

State of the work

- Implementation, about 60% complete:
  - Language: **python**,
  - Libraries: **networkx**,
  - Environment: **PyCharm**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Binary-Search.
- Thesis, about 80% complete:
  - Language: **LaTeX**,
  - Environment: **Visual Studio Code**,
  - Versioning: **git** + **github**, https://github.com/MSzyfel/Papers.
- What is left:
  - Experiments,
  - Advanced data generation,
  - Optimization,
  - Bug fixing,
  - Data visualization.

median

$\downarrow$

1	2	4	5	7	10	15	16	18	21	22	25	31	32

lesser than median

greater than median

Figure: Example of a sorted array containing 14 elements.

Generalized binary search

Definition

A **searcher** is required to find a hidden **target** vertex x in a graph G. To do so, they iteratively perform **queries** to an **oracle**, each about a chosen vertex v. After each such call, the oracle responds whether the target was found and if not, the searcher receives as a reply the connected component of $G - v$ containing the target.

A further generalization is to associate with each vertex a **cost** function $c : V(G) \to \mathbb{R}_{\geq 0}$ representing the time required to query a given vertex.

Generalized binary search

Definition

A **searcher** is required to find a hidden **target** vertex x in a graph G. To do so, they iteratively perform **queries** to an **oracle**, each about a chosen vertex v. After each such call, the oracle responds whether the target was found and if not, the searcher receives as a reply the connected component of $G - v$ containing the target.

A further generalization is to associate with each vertex a **cost** function $c : V(G) \to \mathbb{R}_{\geq 0}$ representing the time required to query a given vertex.

Example

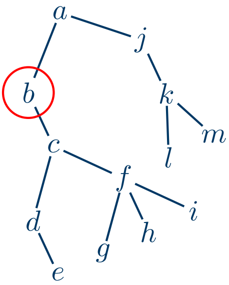


Figure: Query to b.

Figure: Query to f.

Figure: Query to d.

Figure: Query to c.

Example

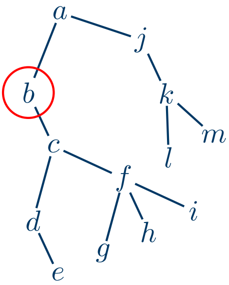


Figure: Query to b.

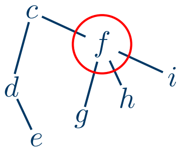


Figure: Query to f.

Figure: Query to d.

Figure: Query to c.

Example

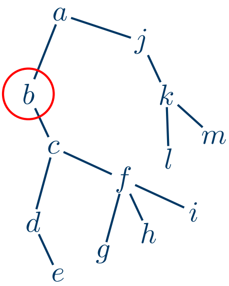


Figure: Query to b.

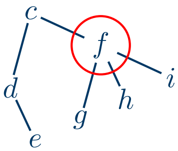


Figure: Query to f.

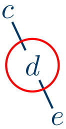


Figure: Query to d.

Figure: Query to c.

GDAŃSK UNIVERSITY OF TECHNOLOGY

Example

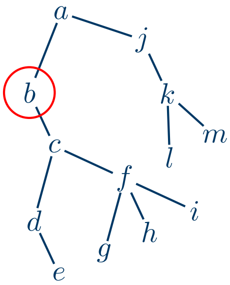


Figure: Query to b.

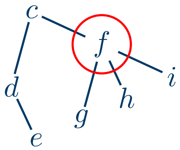


Figure: Query to f.

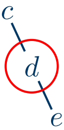


Figure: Query to d.

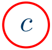


Figure: Query to c.

Classes of graphs considered

There are three main classes of graphs to be considered:

- **Paths** - equivalent to searching in a sorted array.
- **Trees** - The most extensively studied model. **Our choice**.
- **General graphs** - Computationally hardest.

Classes of graphs considered

There are three main classes of graphs to be considered:

- **Paths** - equivalent to searching in a sorted array.
- **Trees** - The most extensively studied model. **Our choice**.
- **General graphs** - Computationally hardest.

Classes of graphs considered

There are three main classes of graphs to be considered:

- **Paths** - equivalent to searching in a sorted array.
- **Trees** - The most extensively studied model. **Our choice**.
- **General graphs** - Computationally hardest.

Definition

Decision tree:

- $D = (V(D), E(D))$, $V(D) = V(T)$ are vertices and $E(D)$ are edges of D.
- $Q_D(T, x)$ - sequence of queries performed in order to find x.
- **Cost** of D in (T, c):

$$\mathtt{COST}_D(T, c) = \max_{x \in V(T)} \left\{ \sum_{q \in Q_D(T,x)} c(q) \right\}.$$

- $\mathtt{OPT}(T, c) = \min_D \{\mathtt{COST}_D(T, c)\}$ - **optimal cost**.

Definition

Decision tree:

- $D = (V(D), E(D))$, $V(D) = V(T)$ are vertices and $E(D)$ are edges of D.
- $Q_D(T, x)$ - sequence of queries performed in order to find x.
- **Cost** of D in (T, c):

$$\mathtt{COST}_D(T, c) = \max_{x \in V(T)} \left\{ \sum_{q \in Q_D(T,x)} c(q) \right\}.$$

- $\mathtt{OPT}(T, c) = \min_D \{\mathtt{COST}_D(T, c)\}$ - **optimal cost**.

Definition

Decision tree:

- $D = (V(D), E(D))$, $V(D) = V(T)$ are vertices and $E(D)$ are edges of D.
- $Q_D(T, x)$ - sequence of queries performed in order to find x.
- **Cost** of D in (T, c):

$$\mathtt{COST}_D(T, c) = \max_{x \in V(T)} \left\{ \sum_{q \in Q_D(T,x)} c(q) \right\}.$$

- $\mathtt{OPT}(T, c) = \min_D \{\mathtt{COST}_D(T, c)\}$ - **optimal cost**.

Definition

Decision tree:

- $D = (V(D), E(D))$, $V(D) = V(T)$ are vertices and $E(D)$ are edges of D.
- $Q_D(T, x)$ - sequence of queries performed in order to find x.
- **Cost** of D in (T, c):

$$\texttt{COST}_D(T, c) = \max_{x \in V(T)} \left\{ \sum_{q \in Q_D(T,x)} c(q) \right\}.$$

- $\texttt{OPT}(T, c) = \min_D \{\texttt{COST}_D(T, c)\}$ - **optimal cost**.

Example of decision tree

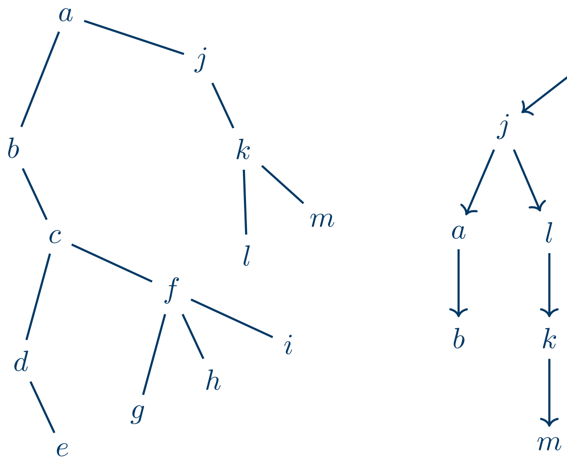


Figure: Sample input tree and a decision tree for it.

Problem statement

Definition

Given a tree T and weight function c, the **Tree Search Problem** consists of finding a decision tree D, such that $\texttt{COST}_D(T,c) = \texttt{OPT}(T,c)$.

Unluckily, the Tree Search Problem is **strongly NP-Hard** even when restricted to binary trees and spiders of diameter at most 6. However, one can find **approximate** solutions.

Problem statement

Definition

Given a tree T and weight function c, the **Tree Search Problem** consists of finding a decision tree D, such that $\texttt{COST}_D\left(T,c\right)=\texttt{OPT}\left(T,c\right)$.

Unluckily, the Tree Search Problem is **strongly NP-Hard** even when restricted to binary trees and spiders of diameter at most 6. However, one can find **approximate** solutions.

Theorem

Let $c(v) = 1$ for every $v \in V(T)$. There exists an exact algorithm called `RankingBasedDT` *for the Tree Search Problem running in linear time, such that the resulting decision tree uses at most $\lfloor \log n \rfloor + 1$ queries.*

Theorem

Fix $0 < \epsilon \leq 35$. There exists a $(1+\epsilon)$-approximation algorithm for the Tree Search Problem running in $n^{O(\log n/\epsilon^2)}$ time.

Theorem

There exists a polynomial time $O\left(\sqrt{\log n}\right)$-approximation algorithm for the Tree Search Problem.

Claims

Theorem

Let $c(v) = 1$ for every $v \in V(T)$. There exists an exact algorithm called `RankingBasedDT` *for the Tree Search Problem running in linear time, such that the resulting decision tree uses at most $\lfloor \log n \rfloor + 1$ queries.*

Theorem

Fix $0 < \epsilon \leq 35$. There exists a $(1 + \epsilon)$-approximation algorithm for the Tree Search Problem running in $n^{O(\log n / \epsilon^2)}$ time.

Theorem

There exists a polynomial time $O\left(\sqrt{\log n}\right)$-approximation algorithm for the Tree Search Problem.

Claims

Theorem

Let $c(v) = 1$ *for every* $v \in V(T)$. *There exists an exact algorithm called* `RankingBasedDT` *for the Tree Search Problem running in linear time, such that the resulting decision tree uses at most* $\lfloor \log n \rfloor + 1$ *queries.*

Theorem

Fix $0 < \epsilon \leq 35$. *There exists a* $(1 + \epsilon)$*-approximation algorithm for the Tree Search Problem running in* $n^{O(\log n / \epsilon^2)}$ *time.*

Theorem

There exists a polynomial time $O\left(\sqrt{\log n}\right)$*-approximation algorithm for the Tree Search Problem.*

Implementation results I

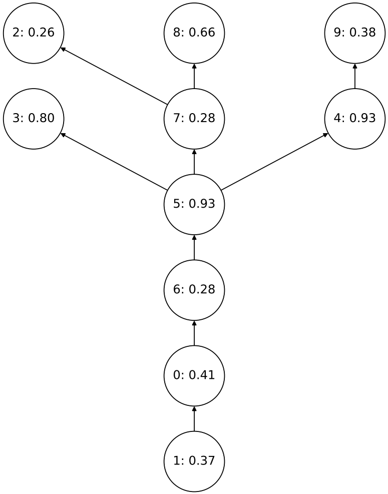


Figure: Input tree of size 10.

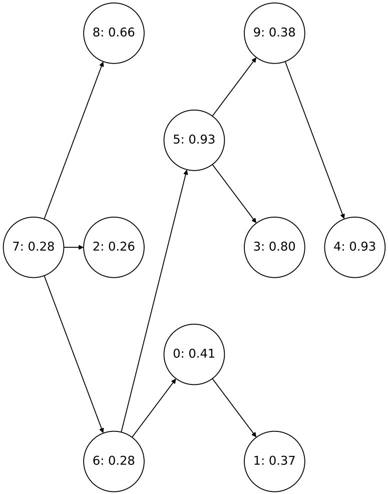


Figure: Decision tree of cost 2.8.

Implementation results II

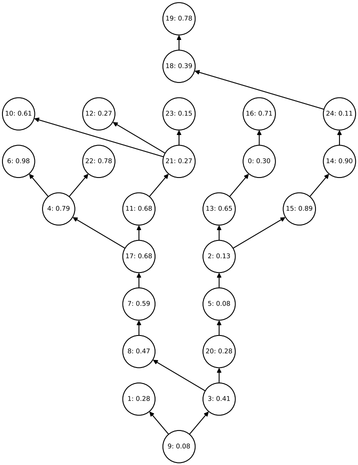


Figure: Input tree of size 25.

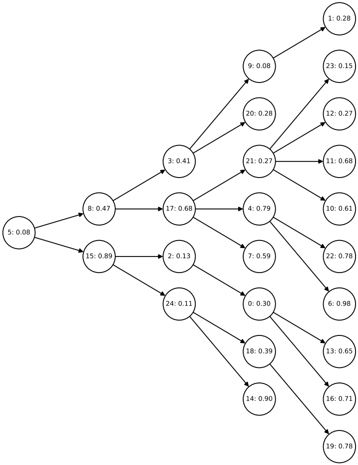


Figure: Decision tree of cost 3.

GDAŃSK UNIVERSITY OF TECHNOLOGY | Implementation results III

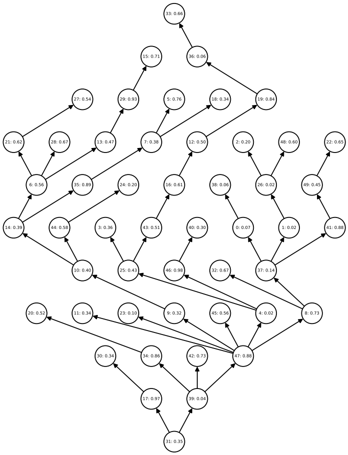


Figure: Input tree of size 50.

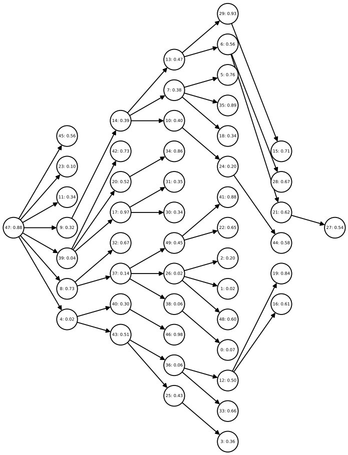


Figure: Decision tree of cost 3.78.

Implementation results IV

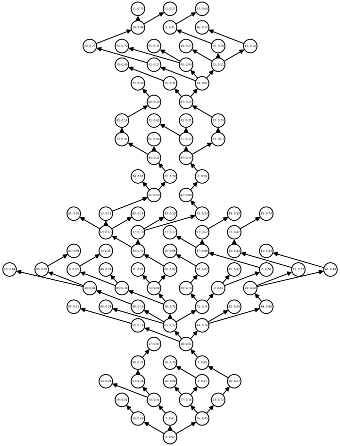


Figure: Input tree of size 100.

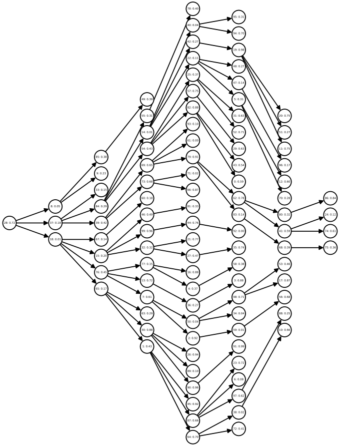


Figure: Decision tree of cost 4.85.

Implementation results V

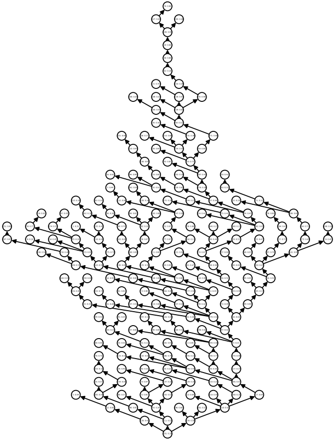


Figure: Input tree of size 200.

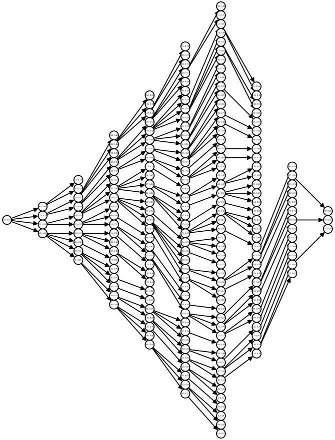


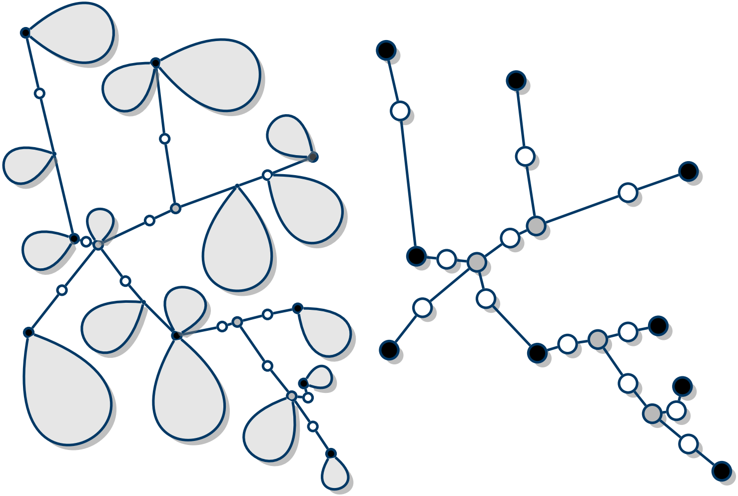
Figure: Decision tree of cost 4.66

Basic steps

proc `ApproxDT` (T):

1. Set a global parameter $k = 2^{\lfloor \sqrt{\log n} \rfloor + 2}$.
2. **if** $n(T) \leq k$ **then**: **return** `QPTAS` $(T, \epsilon = 1)$.
3. **else**: find a set $\mathcal{X} \subseteq V(T)$, such that $|\mathcal{X}| \leq k$ and for every $H \in T - \mathcal{X}$, $n(H) \leq n(T) / 2^{\sqrt{\log n}}$.
4. $\mathcal{Y} \leftarrow \mathcal{X} \cup$ all branching vertices in $T\langle X \rangle$.
5. $\mathcal{Z} \leftarrow \mathcal{Y} \cup$ all lightest vertices on paths between vertices of Y.
6. $D \leftarrow D_{\mathcal{Z}} \leftarrow$ `QPTAS` $(T_{\mathcal{Z}}, \epsilon = 1)$, where $T_{\mathcal{Z}}$ is a tree built on $\mathcal{Z}$.
7. **for** $H \in T - \mathcal{X}$: hang $D_H \leftarrow$ `ApproxDT` (H) in $D_{\mathcal{Z}}$.
8. **return** D.

Auxiliarry tree structure



Solution structure

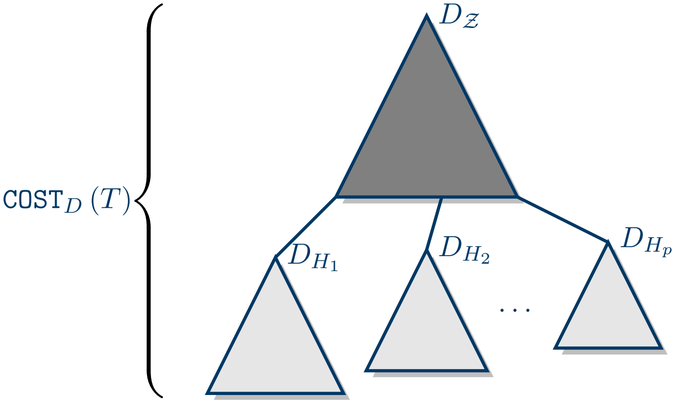


Figure: Structure of the decision tree D.

Approximation guarantee

Since $\texttt{COST}_{D_{\mathcal{Z}}} \leq 2 \cdot \texttt{OPT}(T)$, we have:

$$\begin{aligned}
\texttt{COST}_D(T) &\leq \texttt{COST}_{D_{\mathcal{Z}}} + \max_{H \in T - \mathcal{Z}} \{\texttt{COST}_{D_H}(H)\} \\
&\leq 2 \cdot \log_{2^{\sqrt{\log n}}}(n) \cdot \texttt{OPT}(T) \\
&= \frac{2 \log n}{\sqrt{\log n}} \cdot \texttt{OPT}(T) \\
&= 2\sqrt{\log n} \cdot \texttt{OPT}(T).
\end{aligned}$$

Rounding trick

- Let $p \in \mathbb{N}$.
- Let $k = a/pn$, for some $a \in \mathbb{N}$.
- We define new cost function c', called **aligned cost function**:

$$c'(v) = \begin{cases} \lceil c(v) \rceil_k, & \text{if } c(v) > pk, \textbf{ heavy vertex}, \\ \lceil c(v) \rceil_{\frac{1}{pn}}, & \text{otherwise, } \textbf{light vertex}. \end{cases}$$

Rounding trick

- Let $p \in \mathbb{N}$.
- Let $k = a/pn$, for some $a \in \mathbb{N}$.
- We define new cost function c', called **aligned cost function**:

$$c'(v) = \begin{cases} \lceil c(v) \rceil_k, & \text{if } c(v) > pk, \textbf{ heavy vertex}, \\ \lceil c(v) \rceil_{\frac{1}{pn}}, & \text{otherwise, } \textbf{light vertex}. \end{cases}$$

Lemma

$\mathtt{OPT}(T, c') \leq \left(1 + \frac{2}{p}\right) \cdot \mathtt{OPT}(T, c)$.

Lemma

There exists a decision tree D for (T, c'), such that:

1. $\mathtt{COST}_D(T, c') \leq \left(1 + \frac{3}{p}\right) \cdot \mathtt{OPT}(T, c')$.
2. *Starting point of each heavy query is aligned to a multiple of k.*
3. *Starting point of each light query is aligned to a multiple of $\frac{1}{pn}$.*

We call a decision tree fulfiling the requirements 2. and 3. of the above lemma an **aligned decision tree**.

Lemmas

Lemma

$\mathtt{OPT}(T, c') \leq \left(1 + \frac{2}{p}\right) \cdot \mathtt{OPT}(T, c)$.

Lemma

There exists a decision tree D for (T, c'), such that:

1. $\mathtt{COST}_D(T, c') \leq \left(1 + \frac{3}{p}\right) \cdot \mathtt{OPT}(T, c')$.
2. *Starting point of each heavy query is aligned to a multiple of k.*
3. *Starting point of each light query is aligned to a multiple of $\frac{1}{pn}$.*

We call a decision tree fulfiling the requirements 2. and 3. of the above lemma an **aligned decision tree**.

Lemma

$\texttt{OPT}(T, c') \leq \left(1 + \frac{2}{p}\right) \cdot \texttt{OPT}(T, c)\,.$

Lemma

There exists a decision tree D for (T, c'), such that:

1. $\texttt{COST}_D\,(T, c') \leq \left(1 + \frac{3}{p}\right) \cdot \texttt{OPT}(T, c')$.
2. *Starting point of each heavy query is aligned to a multiple of k.*
3. *Starting point of each light query is aligned to a multiple of $\frac{1}{pn}$.*

We call a decision tree fulfiling the requirements 2. and 3. of the above lemma an **aligned decision tree**.

Directions of the responses

- x - the target.
- T - current candidate subtree.
- $q \in V(D)$ next query to be made.
- Let $T' \in T - q$, such that $x \in T'$.
- If $r(T) \in V(T')$, then the response is called an **up** response, the subtree D_u of D associated with T' is called **left** subtree and u is a **left** child.
- If otherwise, then the response is called an **down** response, the subtree D_u of D associated with T' is called **right** subtree and u is a **right** child.

Directions of the responses

- x - the target.
- T - current candidate subtree.
- $q \in V(D)$ next query to be made.
- Let $T' \in T - q$, such that $x \in T'$.
- If $r(T) \in V(T')$, then the response is called an **up** response, the subtree D_u of D associated with T' is called **left** subtree and u is a **left** child.
- If otherwise, then the response is called an **down** response, the subtree D_u of D associated with T' is called **right** subtree and u is a **right** child.

GDAŃSK UNIVERSITY OF TECHNOLOGY | Directions of the responses

- x - the target.
- T - current candidate subtree.
- $q \in V(D)$ next query to be made.
- Let $T' \in T - q$, such that $x \in T'$.
- If $r(T) \in V(T')$, then the response is called an **up** response, the subtree D_u of D associated with T' is called **left** subtree and u is a **left** child.
- If otherwise, then the response is called an **down** response, the subtree D_u of D associated with T' is called **right** subtree and u is a **right** child.

Directions of the responses

- x - the target.
- T - current candidate subtree.
- $q \in V(D)$ next query to be made.
- Let $T' \in T - q$, such that $x \in T'$.
- If $r(T) \in V(T')$, then the response is called an **up** response, the subtree D_u of D associated with T' is called **left** subtree and u is a **left** child.
- If otherwise, then the response is called an **down** response, the subtree D_u of D associated with T' is called **right** subtree and u is a **right** child.

Example

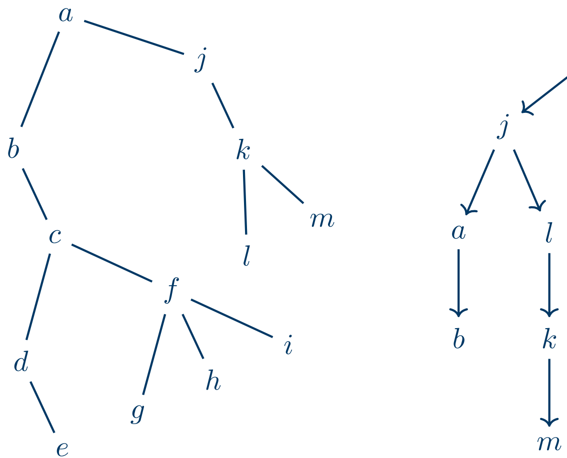


Figure: Sample input tree and a decision tree for it.

Aligned cost

For any vertex $v \in V(T)$ and query $q \in Q_D(T, v)$ the **contribution** $\kappa_{T,c',k}(q, v)$ is defined as:

$$\kappa_{T,c',k}(q, v) = \begin{cases} 0, & \text{if } q \text{ is light and the response is down,} \\ c'(q), & \text{otherwise.} \end{cases}$$

Then, the **aligned cost** of D is defined as:

$$\texttt{COST}'_D(T, c', k) = \max_{v \in V(T)} \left\{ \sum_{q \in Q_D(T,v)} \kappa_{T,c',k}(q, v) \right\}.$$

$\texttt{OPT}'(T, c', k)$ - the **optimal aligned cost**.

Aligned cost

For any vertex $v \in V(T)$ and query $q \in Q_D(T, v)$ the **contribution** $\kappa_{T,c',k}(q, v)$ is defined as:

$$\kappa_{T,c',k}(q, v) = \begin{cases} 0, & \text{if } q \text{ is light and the response is down,} \\ c'(q), & \text{otherwise.} \end{cases}$$

Then, the **aligned cost** of D is defined as:

$$\mathtt{COST}'_D(T, c', k) = \max_{v \in V(T)} \left\{ \sum_{q \in Q_D(T,v)} \kappa_{T,c',k}(q, v) \right\}.$$

$\mathtt{OPT}'(T, c', k)$ - the **optimal aligned cost**.

Heavy module contraction

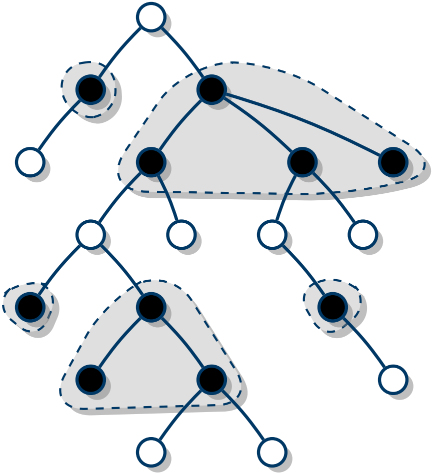


Figure: Input tree.

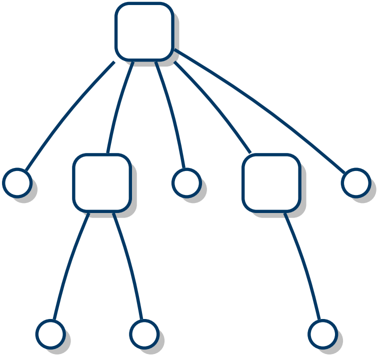


Figure: Input tree with heavy modules contracted.

Propositions

Proposition

Let T be a tree, c' an aligned cost function, $p \in \mathbb{N}$, k the box size and $d \in \mathbb{N}$ be the depth. There exists a DP procedure, which (if it exists) calculates an aligned decision tree D for (T, c') of cost at most $\mathtt{COST}'_D(T, c', k) \leq kd$, running in $(pn)^{O(d)}$ time.

Proposition

Let T be a tree, c' be an aligned cost function, $p \in \mathbb{N}$, $k \in \mathbb{R}_{>0}$ be the box size, D_A be a decision tree for T and F_C be forest of decision trees for T with all heavy modules contracted. There exists a polynomial time MergeDTs procedure which returns a decision tree of cost at most:

$$\mathtt{COST}_D(T, c', k) \leq \mathtt{COST}'_{D_A}(T, c', k) + 2pk \cdot \mathtt{COST}_{F_C}(T, 1).$$

Proposition

Let T be a tree, c' an aligned cost function, $p \in \mathbb{N}$, k the box size and $d \in \mathbb{N}$ be the depth. There exists a DP procedure, which (if it exists) calculates an aligned decision tree D for (T, c') of cost at most $\mathtt{COST}'_D(T, c', k) \leq kd$, running in $(pn)^{O(d)}$ time.

Proposition

Let T be a tree, c' be an aligned cost function, $p \in \mathbb{N}$, $k \in \mathbb{R}_{>0}$ be the box size, D_A be a decision tree for T and F_C be forest of decision trees for T with all heavy modules contracted. There exists a polynomial time MergeDTs procedure which returns a decision tree of cost at most:

$$\mathtt{COST}_D\left(T, c', k\right) \leq \mathtt{COST}'_{D_A}\left(T, c', k\right) + 2pk \cdot \mathtt{COST}_{F_C}\left(T, 1\right).$$

proc $\texttt{QPTAS}\,(T, \epsilon)$:

1. $p \leftarrow \lfloor 59/\epsilon \rfloor$,
2. $d \leftarrow p^2 \cdot (\lfloor \log n \rfloor + 1)$.
3. $k \leftarrow 0$.
4. **while** $D = \emptyset$:
   1. $k \leftarrow k + \frac{1}{pn}$.
   2. **for** $v \in V\,(T)$: **if** $c\,(v) > pk$ **then**: $c'\,(v) \leftarrow \lceil c\,(v) \rceil_k$, **else**: $c'\,(v) \leftarrow \lceil c\,(v) \rceil_{\frac{1}{pn}}$.
   3. $D_A \leftarrow \texttt{DP}\,(T, c', p, k, n, d)$.
   4. **if** $D_A \neq \emptyset$ **then**:
      1. $T_C \leftarrow T$ with all heavy modules contracted.
      2. $D_C \leftarrow \texttt{RankingBasedDT}\,(T_C)$.
      3. $D \leftarrow \texttt{MergeDTs}\,(T, D_A, D_C)$.
5. **return** D.

Quality guarantee

Let k' be the value of k for which D was found. Since:
$k' \leq \frac{\mathtt{OPT}'(T,c',k')}{d} + \frac{1}{pn} \leq \frac{2\cdot\mathtt{OPT}'(T,c')}{p^2\cdot(\lfloor \log n\rfloor+1)}$, we have that:

$$
\begin{aligned}
\mathtt{COST}_D\left(T,c'\right) &\leq \mathtt{OPT}'\left(T,c'\right) + 2pk' \cdot \left(\lfloor \log n \rfloor + 1\right) \\
&\leq \mathtt{OPT}'\left(T,c'\right) + 2p \cdot \left(\lfloor \log n \rfloor + 1\right) \cdot \frac{2 \cdot \mathtt{OPT}'\left(T,c'\right)}{p^2 \cdot \left(\lfloor \log n \rfloor + 1\right)} \\
&\leq \left(1 + \frac{4}{p}\right) \cdot \mathtt{OPT}'\left(T,c'\right) \\
&\leq \left(1 + \frac{2}{p}\right) \cdot \left(1 + \frac{3}{p}\right) \cdot \left(1 + \frac{4}{p}\right) \cdot \mathtt{OPT}\left(T,c\right) \\
&\leq \left(1 + \frac{59}{p}\right) \cdot \mathtt{OPT}\left(T,c\right) = \left(1 + \frac{59}{\left\lceil \frac{59}{\epsilon} \right\rceil}\right) \cdot \mathtt{OPT}\left(T,c\right) \\
&\leq \left(1 + \epsilon\right) \cdot \mathtt{OPT}\left(T,c\right).
\end{aligned}
$$

Quality guarantee

Let k' be the value of k for which D was found. Since:
$k' \leq \frac{\mathtt{OPT}'(T,c',k')}{d} + \frac{1}{pn} \leq \frac{2\cdot\mathtt{OPT}'(T,c')}{p^2\cdot(\lfloor\log n\rfloor+1)}$, we have that:

$$\begin{aligned}
\mathtt{COST}_D\left(T,c'\right) &\leq \mathtt{OPT}'\left(T,c'\right) + 2pk'\cdot\left(\lfloor\log n\rfloor+1\right)\\
&\leq \mathtt{OPT}'\left(T,c'\right) + 2p\cdot\left(\lfloor\log n\rfloor+1\right)\cdot\frac{2\cdot\mathtt{OPT}'\left(T,c'\right)}{p^2\cdot\left(\lfloor\log n\rfloor+1\right)}\\
&\leq \left(1+\frac{4}{p}\right)\cdot\mathtt{OPT}'\left(T,c'\right)\\
&\leq \left(1+\frac{2}{p}\right)\cdot\left(1+\frac{3}{p}\right)\cdot\left(1+\frac{4}{p}\right)\cdot\mathtt{OPT}\left(T,c\right)\\
&\leq \left(1+\frac{59}{p}\right)\cdot\mathtt{OPT}\left(T,c\right) = \left(1+\frac{59}{\left\lceil\frac{59}{\epsilon}\right\rceil}\right)\cdot\mathtt{OPT}\left(T,c\right)\\
&\leq \left(1+\epsilon\right)\cdot\mathtt{OPT}\left(T,c\right).
\end{aligned}$$

`MergeDTs` procedure

proc `MergeDTs` (T, D_A):

1. **if** F_C is connected **then**: $r = r\,(F_C)$.
2. **else**: $r = r\,(D_A)$.
3. $D \leftarrow$ decision tree with root r.
4. **for** $T' \in T - r$:
    1. $F'_C \leftarrow F_C$ restricted to T'_C
    2. $D'_A \leftarrow D$ restricted to T'.
    3. $D' \leftarrow$ `MergeDTs` (T', D'_A, F'_C).
    4. Hang D' below r in D.
5. **return** D.

Definition

Boxed decision tree:

1. $D = (V(D), E(D), u, l)$, $V(D)$ - nodes of D, called **boxes**, $E(D)$ - edges of D, $u : V(T) \times V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **usage** function and $l : V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **load** function.
2. For every $b \in V(D)$, $Q(b) = \{v \in V(T) \,|\, u(v, b) > 0\}$ - **query assignment**, vertices of T, such that queries to them overlap with b.
3. All boxes containing v form a left path. Additionally, for each interior box b of this path, $l(b) = 0$ and $u(v, b) = k$.
4. For every $b \in V(D)$, $l(b) + \sum_{v \in V(T)} u(v, b) \leq k$ and for every $v \in V(T)$, either $\sum_{b \in V(D)} u(v, b) = 0$ or $\sum_{b \in V(D)} u(v, b) = c'(v)$.
5. Every heavy query is aligned.

Boxed decision tree

Definition

Boxed decision tree:

1. $D = (V(D), E(D), u, l)$, $V(D)$ - nodes of D, called **boxes**, $E(D)$ - edges of D, $u : V(T) \times V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **usage** function and $l : V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **load** function.
2. For every $b \in V(D)$, $Q(b) = \{v \in V(T) \,|\, u(v, b) > 0\}$ - **query assignment**, vertices of T, such that queries to them overlap with b.
3. All boxes containing v form a left path. Additionally, for each interior box b of this path, $l(b) = 0$ and $u(v, b) = k$.
4. For every $b \in V(D)$, $l(b) + \sum_{v \in V(T)} u(v, b) \leq k$ and for every $v \in V(T)$, either $\sum_{b \in V(D)} u(v, b) = 0$ or $\sum_{b \in V(D)} u(v, b) = c'(v)$.
5. Every heavy query is aligned.

Boxed decision tree

Definition

Boxed decision tree:

1. $D = (V(D), E(D), u, l)$, $V(D)$ - nodes of D, called **boxes**, $E(D)$ - edges of D, $u : V(T) \times V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **usage** function and $l : V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **load** function.
2. For every $b \in V(D)$, $Q(b) = \{v \in V(T) \,|\, u(v, b) > 0\}$ - **query assignment**, vertices of T, such that queries to them overlap with b.
3. All boxes containing v form a left path. Additionally, for each interior box b of this path, $l(b) = 0$ and $u(v, b) = k$.
4. For every $b \in V(D)$, $l(b) + \sum_{v \in V(T)} u(v, b) \leq k$ and for every $v \in V(T)$, either $\sum_{b \in V(D)} u(v, b) = 0$ or $\sum_{b \in V(D)} u(v, b) = c'(v)$.
5. Every heavy query is aligned.

Boxed decision tree

Definition

Boxed decision tree:

1. $D = (V(D), E(D), u, l)$, $V(D)$ - nodes of D, called **boxes**, $E(D)$ - edges of D, $u : V(T) \times V(D) \rightarrow \{0, 1/pn, 2/pn, \ldots, k\}$ - **usage** function and $l : V(D) \rightarrow \{0, 1/pn, 2/pn, \ldots, k\}$ - **load** function.
2. For every $b \in V(D)$, $Q(b) = \{v \in V(T) | u(v, b) > 0\}$ - **query assignment**, vertices of T, such that queries to them overlap with b.
3. All boxes containing v form a left path. Additionally, for each interior box b of this path, $l(b) = 0$ and $u(v, b) = k$.
4. For every $b \in V(D)$, $l(b) + \sum_{v \in V(T)} u(v, b) \leq k$ and for every $v \in V(T)$, either $\sum_{b \in V(D)} u(v, b) = 0$ or $\sum_{b \in V(D)} u(v, b) = c'(v)$.
5. Every heavy query is aligned.

Boxed decision tree

Definition

Boxed decision tree:

1. $D = (V(D), E(D), u, l)$, $V(D)$ - nodes of D, called **boxes**, $E(D)$ - edges of D, $u : V(T) \times V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **usage** function and $l : V(D) \to \{0, 1/pn, 2/pn, \ldots, k\}$ - **load** function.
2. For every $b \in V(D)$, $Q(b) = \{v \in V(T) | u(v, b) > 0\}$ - **query assignment**, vertices of T, such that queries to them overlap with b.
3. All boxes containing v form a left path. Additionally, for each interior box b of this path, $l(b) = 0$ and $u(v, b) = k$.
4. For every $b \in V(D)$, $l(b) + \sum_{v \in V(T)} u(v, b) \leq k$ and for every $v \in V(T)$, either $\sum_{b \in V(D)} u(v, b) = 0$ or $\sum_{b \in V(D)} u(v, b) = c'(v)$.
5. Every heavy query is aligned.

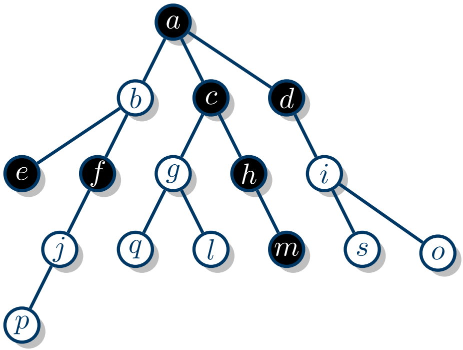


Figure: Input tree.

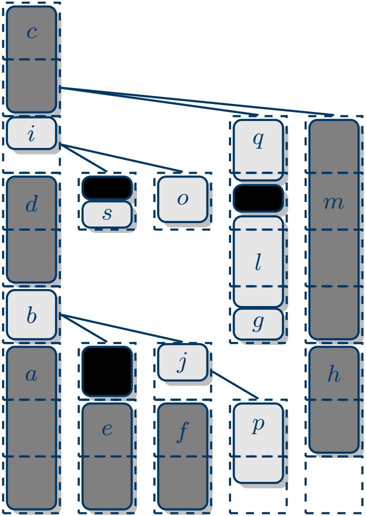


Figure: Boxed decision tree.

Boxline

Definition

Boxline: $B = \langle (b_1, \tau_1), (b_2, \tau_2), \ldots, (b_d, \tau_d) \rangle$, b_j - box, such that $Q(b_j) = \emptyset$, τ_j - boolean flag.

Definition

Left box-path of D:

1. $B_D = \langle (q_1, f_1), (q_2, f_2), \ldots, (q_h, f_h) \rangle$, q_j - box f_j - boolean flag, obtained by traversing boxes of D towards left. For each such box b_j, $q_j = l(b) + \sum_{v \in Q(b)} q(v, b)$, whereas f_j denotes whether there exists a **transcending** query in $Q(b_j)$, i. e.: $v \in Q(b_j)$, such that $v \in Q(b_{j+1})$.
2. Decision tree D with a left box-path B_D is **box-compatible** with boxline B ($h \leq d$), if $l(q_j) \geq l(b_j)$ and $\tau_j \implies f_j$.

Definition

Boxline: $B = \langle (b_1, \tau_1), (b_2, \tau_2), \ldots, (b_d, \tau_d) \rangle$, b_j - box, such that $Q(b_j) = \emptyset$, τ_j - boolean flag.

Definition

Left box-path of D:

1. $B_D = \langle (q_1, f_1), (q_2, f_2), \ldots, (q_h, f_h) \rangle$, q_j - box f_j - boolean flag, obtained by traversing boxes of D towards left. For each such box b_j, $q_j = l(b) + \sum_{v \in Q(b)} q(v, b)$, whereas f_j denotes whether there exists a **transcending** query in $Q(b_j)$, i. e.: $v \in Q(b_j)$, such that $v \in Q(b_{j+1})$.
2. Decision tree D with a left box-path B_D is **box-compatible** with boxline B $(h \leq d)$, if $l(q_j) \geq l(b_j)$ and $\tau_j \implies f_j$.

Useful operations I

- Puting a query to vertex v at s-th slot of a box b:
  1. $\sigma(v) \leftarrow c(v)$:
  2. **while** $\sigma(v) > 0$:
     1. $u(v,b) \leftarrow \min\{k - s/pn, \sigma(v)\}$.
     2. $\sigma(v) \leftarrow \sigma(v) - u(v,b)$.
     3. $b \leftarrow$ left child of b.

If such operation violates the definition of D or query to v transcendents any box b_j, such that τ_j we mark D as **conflicted**.

Useful operations II

- Building a decision tree D based on boxline B:
  1. $D \leftarrow \emptyset$.
  2. **for** $1 \leq j \leq |B|$:
     1. Create box q_j in D.
     2. $l(j) \leftarrow b_j$.
     3. $Q(q_j) \leftarrow \emptyset$.
     4. **if** $j > 1$ **then**: hang q_j as the left child of q_{j-1}.
  3. **return** D.
- Bipartitioning of B. A bipartition of a boxline B consists of a pair of boxlines (B_1, B_2) such that:
  - $|B| = |B_1| = |B_2|$.
  - $l(b_{1,j}) + l(b_{2,j}) - k = l(b_j)$.
  - $(\tau_{1,j} \land \tau_{2,j} \iff \tau_j)$.
  - $(\tau_{1,j} \lor \tau_{2,j})$.

Useful operations III

- Rotating a decision tree D around vertex v:
  1. $q_h \leftarrow$ the box containing the end of the query to v.
  2. Sort vertices whose queries start in $Q\left(q_h\right)$ according to c'.
  3. Create box q.
  4. Move queries from $Q\left(q_h\right)$ to $Q\left(q\right)$, so that all queries after v are in $Q\left(q\right)$.
  5. Hang q'_h as a right child of q_h.
  6. Rehang left child of q_h as the left child of q.

Useful operations IV

- Aligning D_1 and D_2 by their left paths to create new decision tree D:
  1. $D \leftarrow \emptyset$.
  2. **for** $1 \leq j \leq |B|$:
     1. Create box q_j in D.
     2. $l(j) \leftarrow l(q_{1,j}) + l(q_{2,j}) - k$.
     3. **for** $v \in V(T)$: $u(v, q_j) \leftarrow \max\{u(v, q_{1,j}), u(v, q_{2,j})\}$.
     4. Hang all right children of $q_{1,j}$ and $q_{2,j}$ below q_j.
     5. **if** $j > 1$ **then**: hang q_j as the left child of q_{j-1}.
  3. **return** D.

Figure: No children.

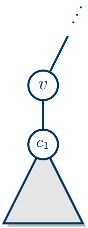


Figure: One child.

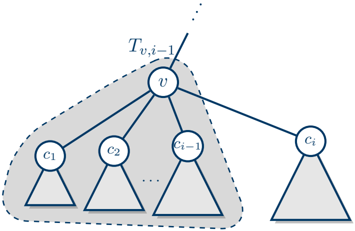


Figure: Many children.

No children

proc $\texttt{DPNoChildren}\left(T_{v,i}, c', B\right)$:

1. **for** $1 \leq b \leq d$ and $0 \leq s \leq \left(k/pn\right.$ **if** $c\left(v\right) > pk$ **else** $\left.0\right)$:
    1. $D \leftarrow$ a decision tree based on B.
    2. Put query to v at the s-th slot of q_b.
    3. **if** D is not conflicted **then**:
        1. **if** $\texttt{COST}'_D\left(T_{v,i}, c', k\right) \leq dk$ **then**: **return** D, **else**: **return** $\emptyset$.
2. **return** $\emptyset$.

proc $\texttt{DPOneChild}\,(T_{v,i}, c', B)$:

1. $\mathcal{D} \leftarrow \emptyset$.
2. **for** $1 \leq b \leq d$ and $0 \leq s \leq (k/pn$ **if** $c\,(v) > pk$ **else** $0)$:
   1. $D \leftarrow$ a decision tree based on B.
   2. Put query to v at the s-th slot of q_b.
   3. **if** D is not conflicted and $\texttt{COST}'_D\,(T_{v,i}, c', k) \leq dk$ **then**:
      1. $B' \leftarrow$ left box-path of D.
      2. $h \leftarrow$ index of the last box q_h occupied by the query to v.
      3. **for** $h < j \leq d$: $b'_j \leftarrow 0$, $t'_j \leftarrow$ **false**.
      4. $D' \leftarrow \texttt{DP}\,(T_{c_1}, c', B')$.
      5. Put query to v at the s-th slot of q_b.
      6. Rotate D' around v.
      7. $\mathcal{D} \leftarrow \mathcal{D} \cup \{D$ and D' with their left paths aligned$\}$.
3. **return** $\arg\min_{D \in \mathcal{D}} \{\texttt{COST}'_D\,(T_{v,i}, c', k)\}$.

Many children

proc `DPManyChildren` $(T_{v,i}, c', B)$:

1. $\mathcal{D} \leftarrow \emptyset$.
2. **for** bipartition (B_1, B_2) of B:
   1. $D_1 \leftarrow \mathtt{DP}\,(T_{v,i-1}, c', B_1)$.
   2. $h \leftarrow$ index the last box $q_{1,h}$ occupied by the query to v.
   3. **for** $h \leq j \leq d$: $b_{2,j} \leftarrow 0$, $t_{2,j} \leftarrow$ **false**.
   4. $D_2 \leftarrow \mathtt{DP}\,(T_{c_i}, c', B_2)$.
   5. Rotate D_2 around v.
   6. $\mathcal{D} \leftarrow \mathcal{D} \cup \{D_1$ and $D_2$with their left paths aligned$\}$.
3. **return** $\arg\min_{D \in \mathcal{D}} \{\mathtt{COST}'_D\,(T_{v,i}, c', k)\}$.

Visualization

Figure: Boxline B.

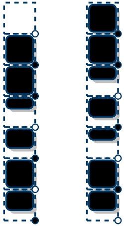


Figure: Bipartition of B.

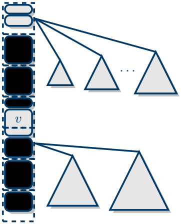


Figure: Decision tree D_1.

Figure: Boxline B_2.

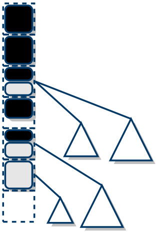


Figure: Decision tree D_2.

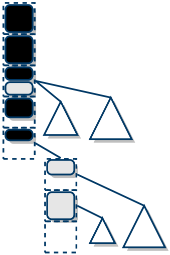


Figure: Rotating step.

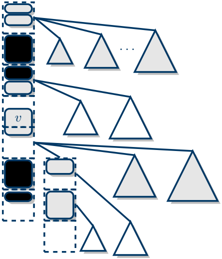


Figure: Resulting decision tree D.