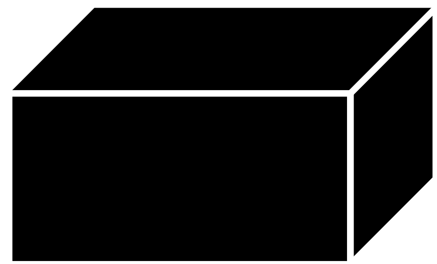
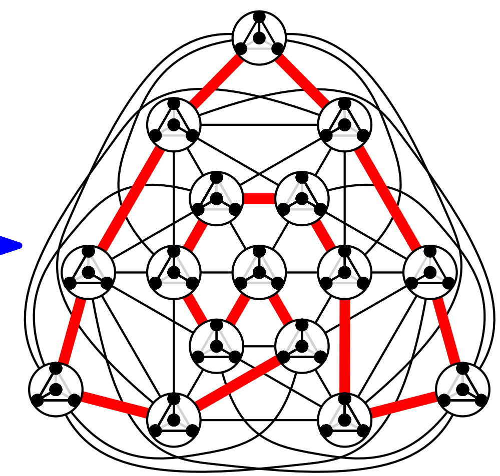
Traversing Combinatorial 0/1-Polytopes

$$
\begin{aligned}
\min \quad & d_H(x^*, x) \\
\text{s.t.} \quad & x \in \mathcal{X}, \\
& x_i = 1 \quad \forall i \in I_P, \\
& x_i = 0 \quad \forall i \in I_F.
\end{aligned}
$$

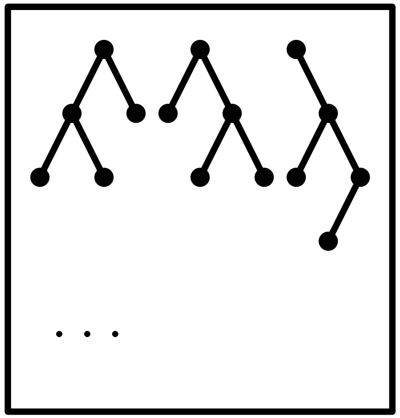




Arturo Merino
Saarland University & Max Planck Institute for Informatics
Joint work with Torsten Mütze

Introduction

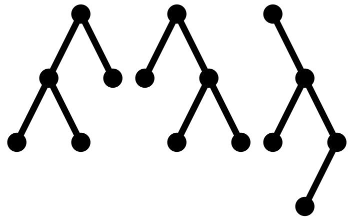
- Many different classes of combinatorial objects



binary trees

Introduction

- Many different classes of combinatorial objects



...

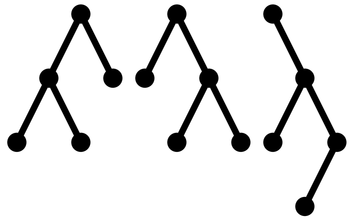
binary trees

123
132
312
321
...

permutations

Introduction

- Many different classes of combinatorial objects



. . .

binary trees

123
132
312
321
. . .

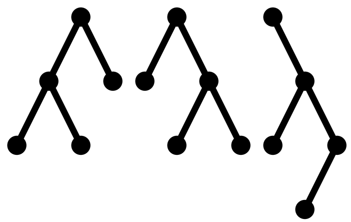
permutations

000
001
010
011
. . .

bitstrings

Introduction

- Many different classes of combinatorial objects



. . .

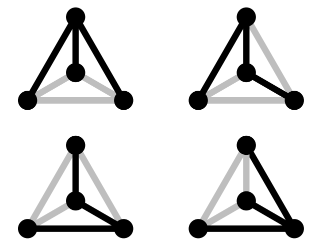
binary trees

123
132
312
321
. . .

permutations

000
001
010
011
. . .

bitstrings

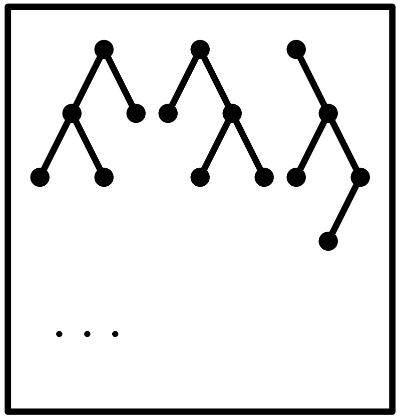


. . .

spanning trees

Introduction

- Many different classes of combinatorial objects



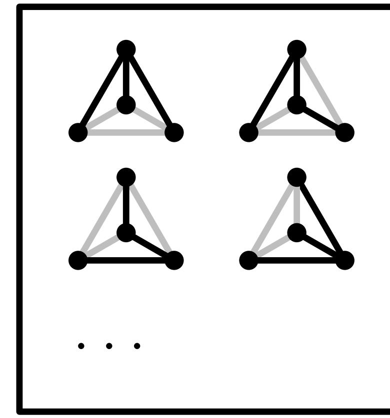
binary trees

123
132
312
321
...

permutations

000
001
010
011
...

bitstrings

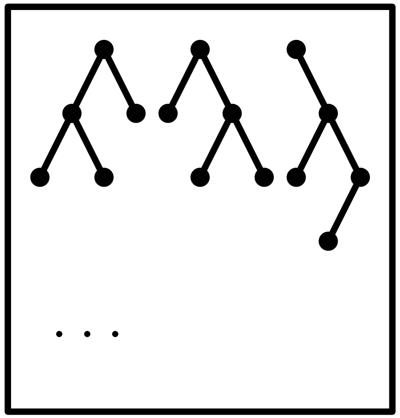


spanning trees

- fundamental algorithmic tasks:

Introduction

- Many different classes of combinatorial objects



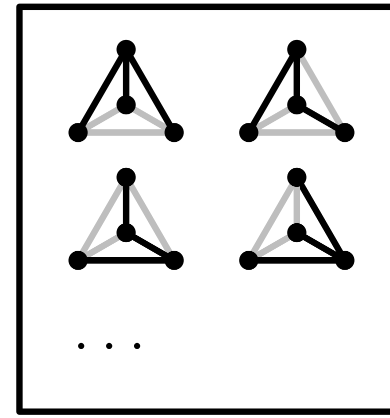
binary trees

123
132
312
321
...

permutations

000
001
010
011
...

bitstrings

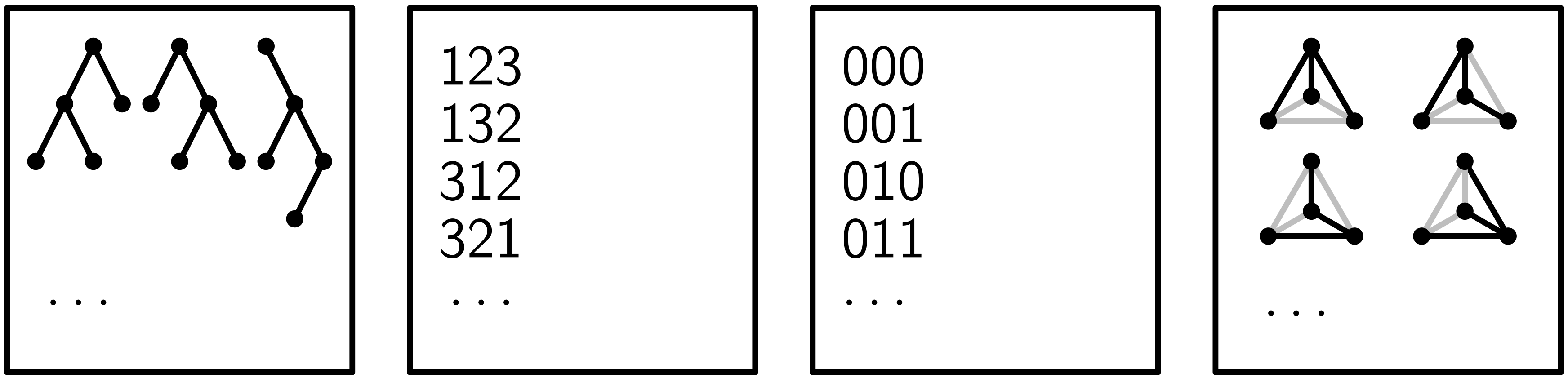


spanning trees

- fundamental algorithmic tasks:
  - counting,

Introduction

- Many different classes of combinatorial objects

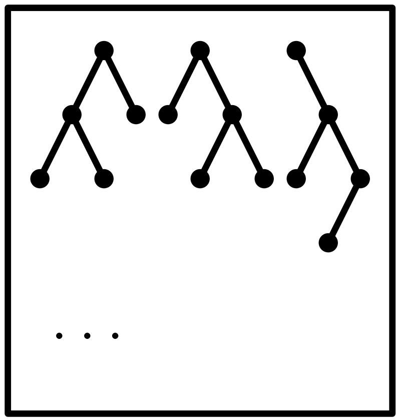


binary trees | permutations | bitstrings | spanning trees

- fundamental algorithmic tasks:
  - counting,
  - random sampling,

Introduction

- Many different classes of combinatorial objects



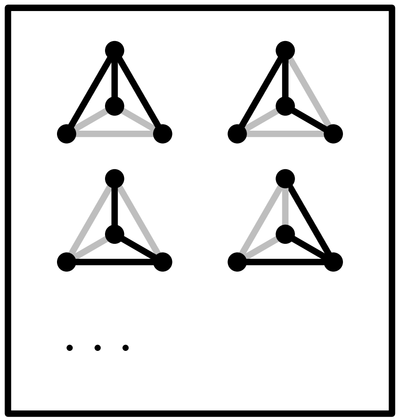
binary trees

123
132
312
321
...

permutations

000
001
010
011
...

bitstrings

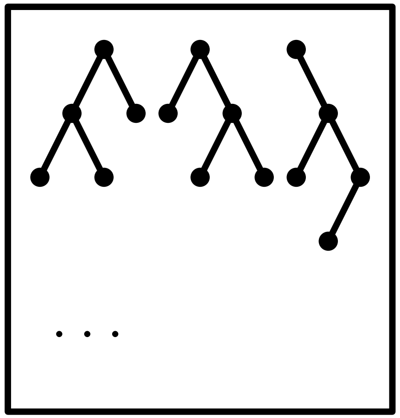


spanning trees

- fundamental algorithmic tasks:
  - counting,
  - random sampling,
  - combinatorial optimization,

Introduction

- Many different classes of combinatorial objects



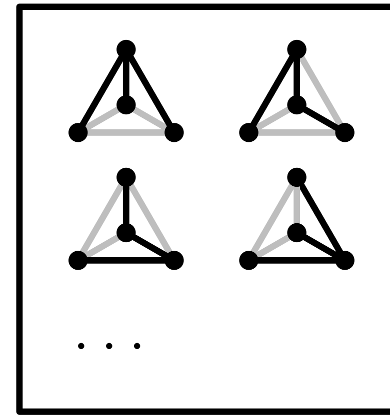
binary trees

123
132
312
321
...

permutations

000
001
010
011
...

bitstrings



spanning trees

- fundamental algorithmic tasks:
    - counting,
    - random sampling,
    - combinatorial optimization,
    - **combinatorial generation** **[Knuth TAOCP Vol. 4A]**.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

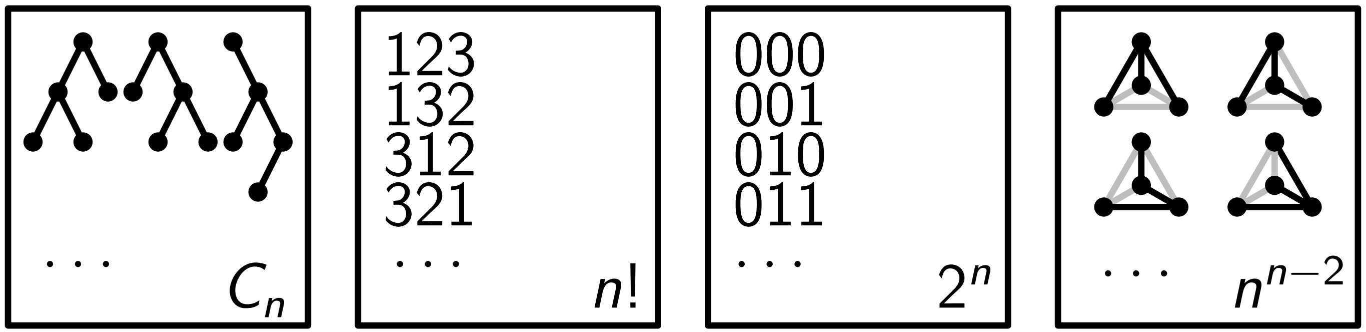
Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
    - *visit* each object exactly once.
- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

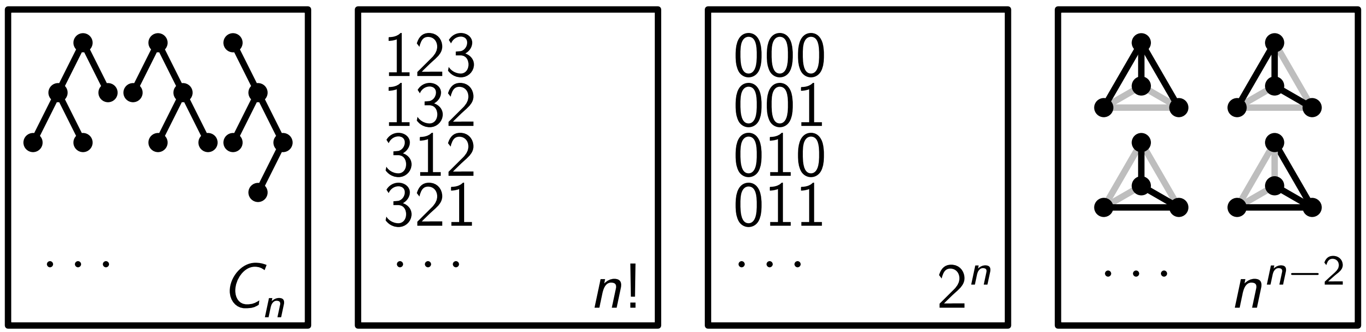
- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.



Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

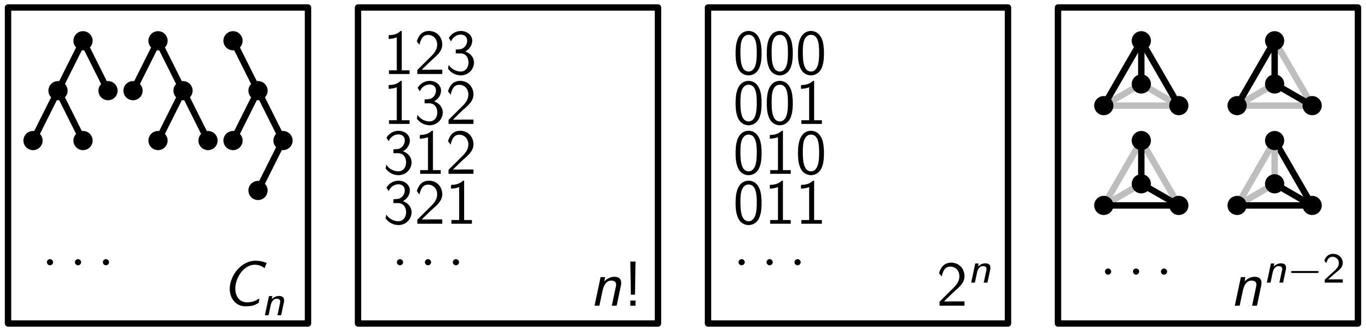
- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.



- Different measures:
  - **polynomial total time**: $\mathrm{poly}(|\texttt{In}|) \cdot \mathrm{poly}(|\texttt{Out}|)$.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.
- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.

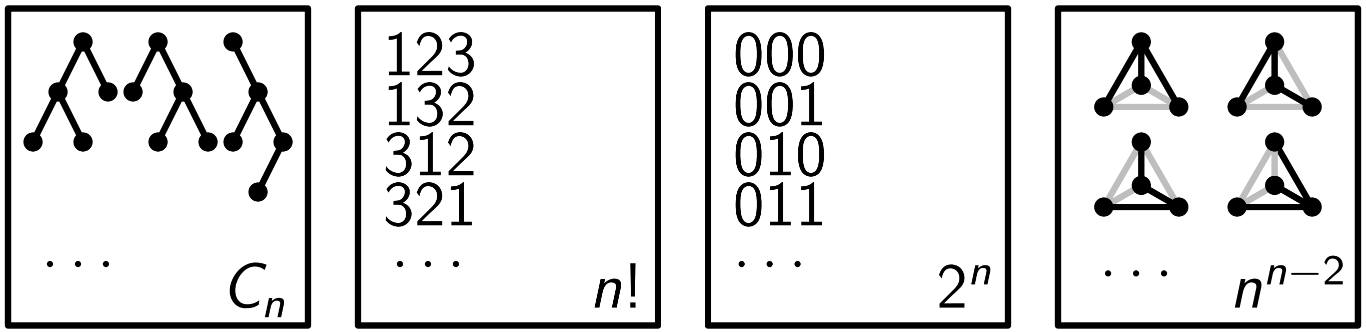


- Different measures:
  - **polynomial total time**: $\mathrm{poly}(|\texttt{In}|) \cdot \mathrm{poly}(|\texttt{Out}|)$.
  - **polynomial amortized delay**: $\mathrm{poly}(|\texttt{In}|) \cdot |\texttt{Out}|$.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.

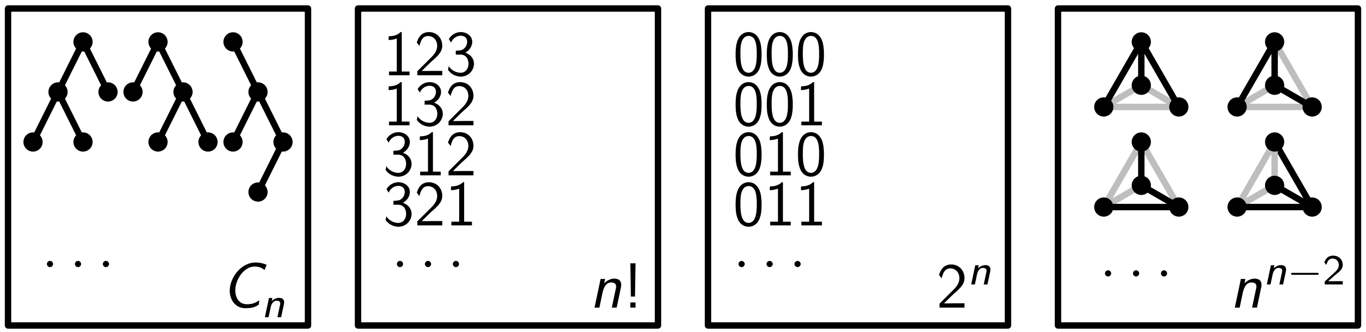


- Different measures:
  - **polynomial total time**: $\text{poly}(|\texttt{In}|) \cdot \text{poly}(|\texttt{Out}|)$.
  - **polynomial amortized delay**: $\text{poly}(|\texttt{In}|) \cdot |\texttt{Out}|$.

    $\Longrightarrow$ each new object in $\text{poly}(|\texttt{In}|)$ time on average.

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.
- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.



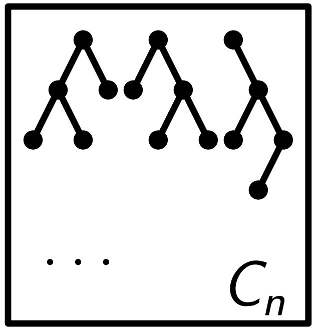
- Different measures:
  - **polynomial total time**: $\mathrm{poly}(|\texttt{In}|) \cdot \mathrm{poly}(|\texttt{Out}|)$.
  - **polynomial amortized delay**: $\mathrm{poly}(|\texttt{In}|) \cdot |\texttt{Out}|$.

    $\Longrightarrow$ each new object in $\mathrm{poly}(|\texttt{In}|)$ time on average.
  - **polynomial delay**: each new object in $\mathrm{poly}(|\texttt{In}|)$ time.

Combinatorial generation

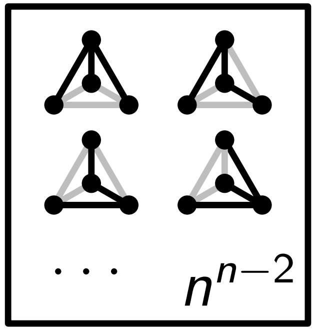
- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.



123
132
312
321
. . .
$n!$

000
001
010
011
. . .
2^n



- Different measures:
  - **polynomial total time**: $\mathrm{poly}(|\texttt{In}|) \cdot \mathrm{poly}(|\texttt{Out}|)$.
  - **polynomial amortized delay**: $\mathrm{poly}(|\texttt{In}|) \cdot |\texttt{Out}|$.

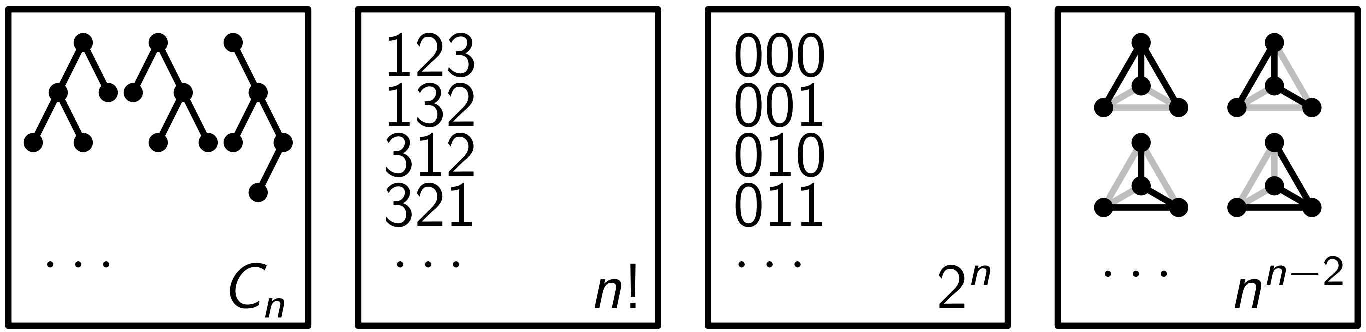
    $\Longrightarrow$ each new object in $\mathrm{poly}(|\texttt{In}|)$ time on average.
  - **polynomial delay**: each new object in $\mathrm{poly}(|\texttt{In}|)$ time.

- **Dream:** each new object in **constant time** (or constant **delay**)

Combinatorial generation

- **Goal:** generate all objects of a combinatorial class efficiently.
  - *visit* each object exactly once.

- Meaning of efficiency: $|\texttt{Out}|$ usually exponential.



- Different measures:
  - **polynomial total time**: $\mathrm{poly}(|\texttt{In}|) \cdot \mathrm{poly}(|\texttt{Out}|)$.
  - **polynomial amortized delay**: $\mathrm{poly}(|\texttt{In}|) \cdot |\texttt{Out}|$.

    $\Longrightarrow$ each new object in $\mathrm{poly}(|\texttt{In}|)$ time on average.
  - **polynomial delay**: each new object in $\mathrm{poly}(|\texttt{In}|)$ time.

- **Dream:** each new object in **constant time** (or constant **delay**)
  - need that consecutive objects differ in a "local change".

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"**
  $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"** $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"**
  $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Examples

- binary trees on n nodes by **rotations** **[Lucas, van Baronaigien, Ruskey 93]**

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"**
  $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Examples

- binary trees on n nodes by **rotations** **[Lucas, van Baronaigien, Ruskey 93]**
- n-permutations by **adjacent transpositions**
  (SJT algorithm) **[Steinhaus 58], [Johnson 64], [Trotter 62]**

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"**
  $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Examples

- ◦ binary trees on n nodes by **rotations** **[Lucas, van Baronaigien, Ruskey 93]**
- ◦ n-permutations by **adjacent transpositions**
  (SJT algorithm) **[Steinhaus 58], [Johnson 64], [Trotter 62]**
- ◦ bitstrings of length n by **bitflips** (BRGC) **[Gray 53]**

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"**
  $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Examples

- binary trees on n nodes by **rotations** **[Lucas, van Baronaigien, Ruskey 93]**
- n-permutations by **adjacent transpositions**
  (SJT algorithm) **[Steinhaus 58], [Johnson 64], [Trotter 62]**
- bitstrings of length n by **bitflips** (BRGC) **[Gray 53]**
- spanning trees of a fixed graph by **edge exchanges**
  [Cummings 66]

Combinatorial Gray codes

- Listing where consecutive objects differ by a **"local change"** $\rightarrow$ **Gray code** (recent survey **[Mütze 22]**)
- Combinatorial notion that aids in exhaustive generation.

Examples

- binary trees on n nodes by **rotations** **[Lucas, van Baronaigien, Ruskey 93]**
- n-permutations by **adjacent transpositions** (SJT algorithm) **[Steinhaus 58], [Johnson 64], [Trotter 62]**
- bitstrings of length n by **bitflips** (BRGC) **[Gray 53]**
- spanning trees of a fixed graph by **edge exchanges** **[Cummings 66]**

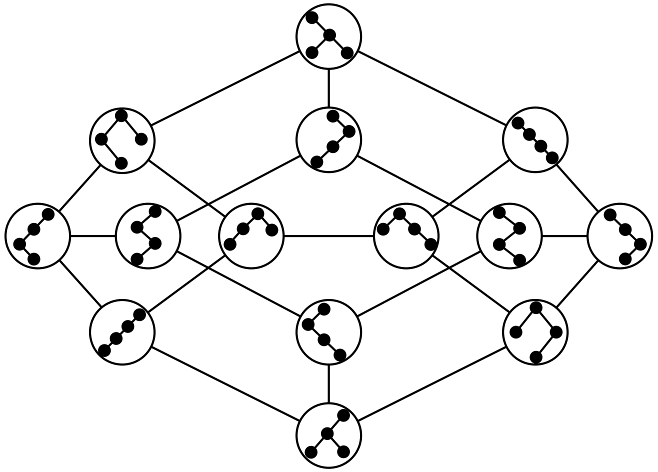
- All these examples lead to constant (amortized) delay generation algorithms.

Gray Codes and Flip Graphs

- **Flip graph:** vertices are combinatorial objects, edges capture change operations.

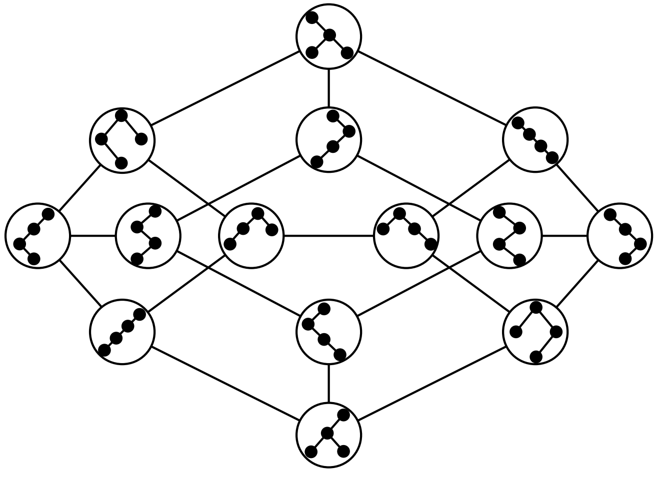
Gray Codes and Flip Graphs

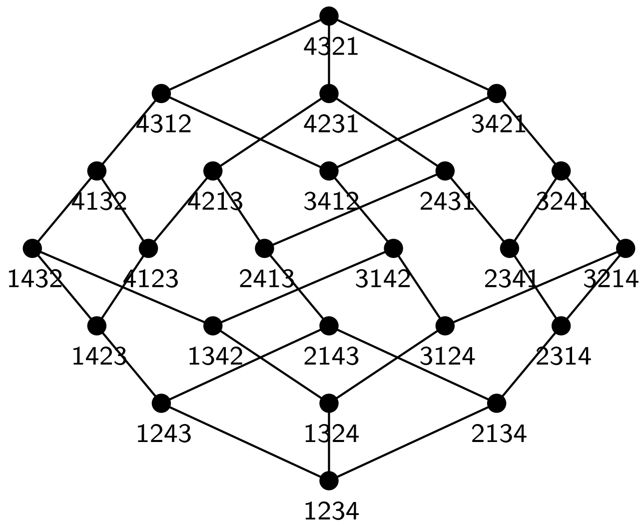
- **Flip graph:** vertices are combinatorial objects, edges capture change operations.



Gray Codes and Flip Graphs

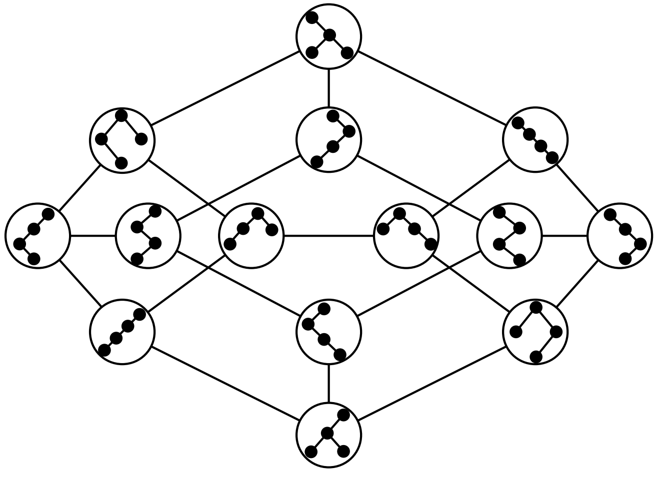
- **Flip graph:** vertices are combinatorial objects, edges capture change operations.

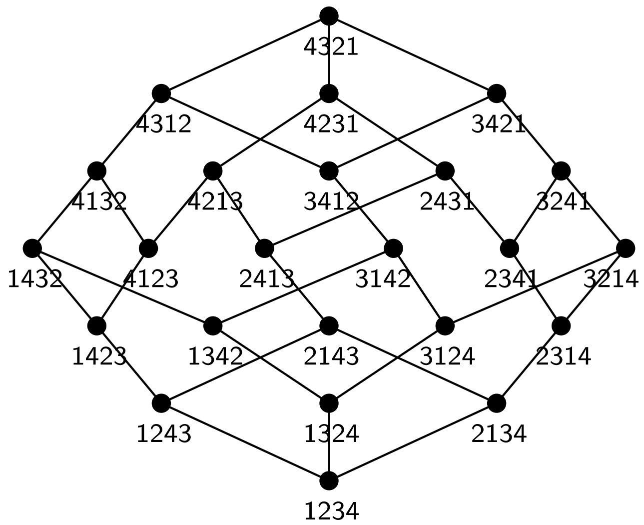


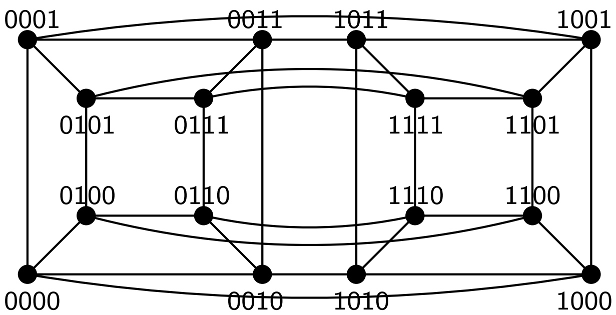


Gray Codes and Flip Graphs

- **Flip graph:** vertices are combinatorial objects, edges capture change operations.

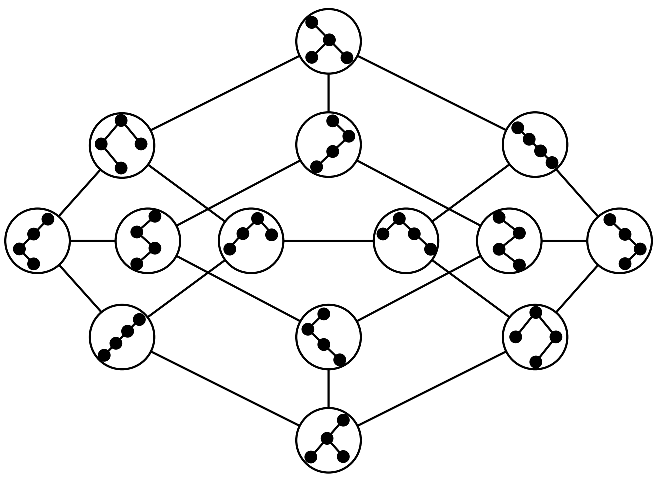


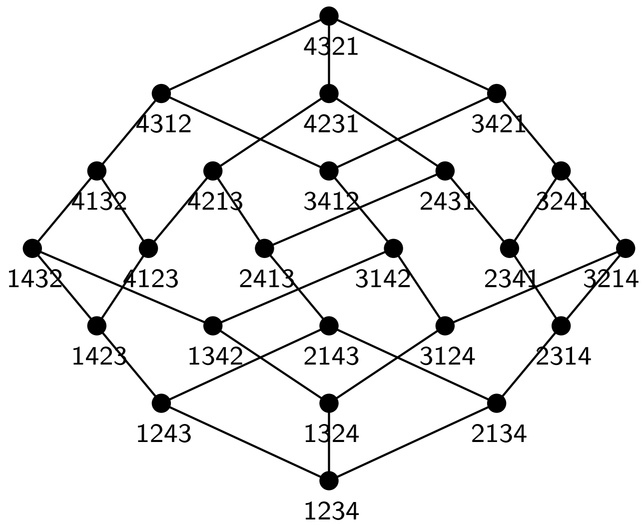


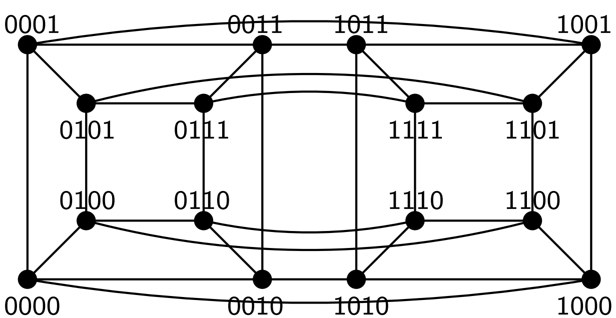


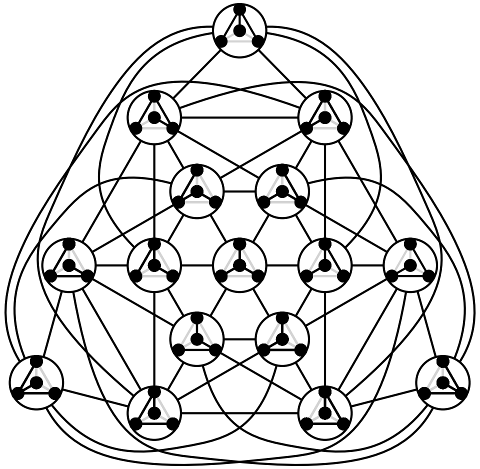
Gray Codes and Flip Graphs

- **Flip graph:** vertices are combinatorial objects, edges capture change operations.



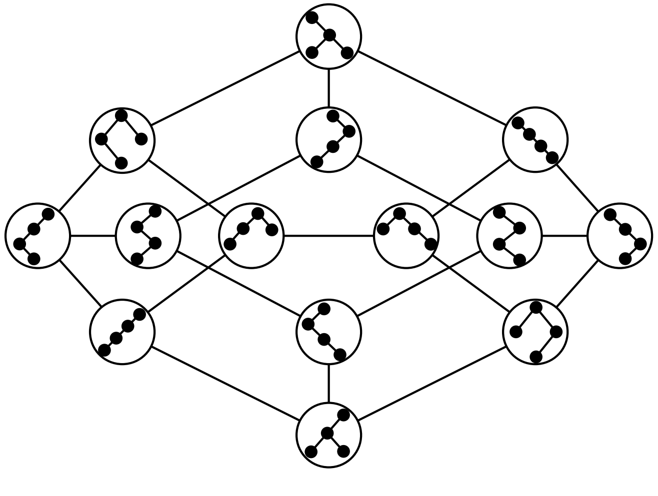


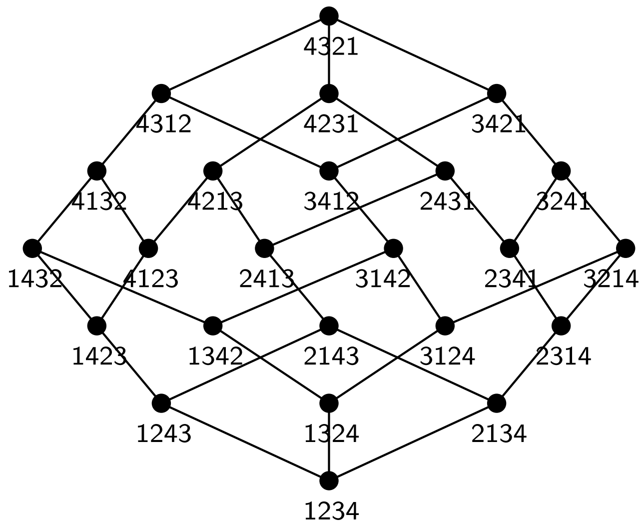


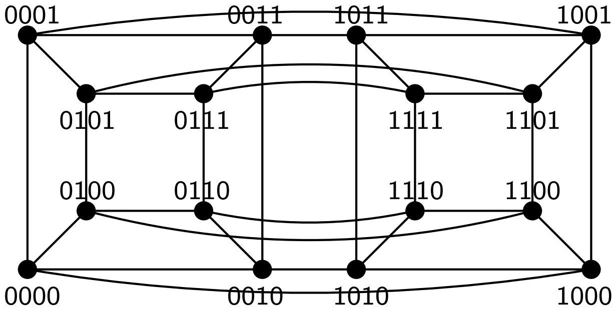


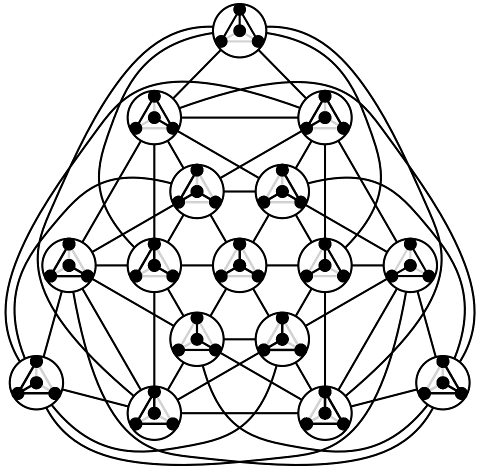
Gray Codes and Flip Graphs

- Many flip graphs can be realized as polytopes and posets.



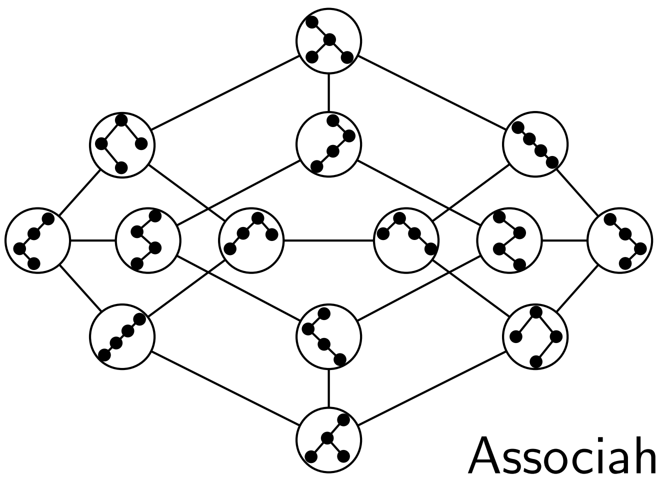






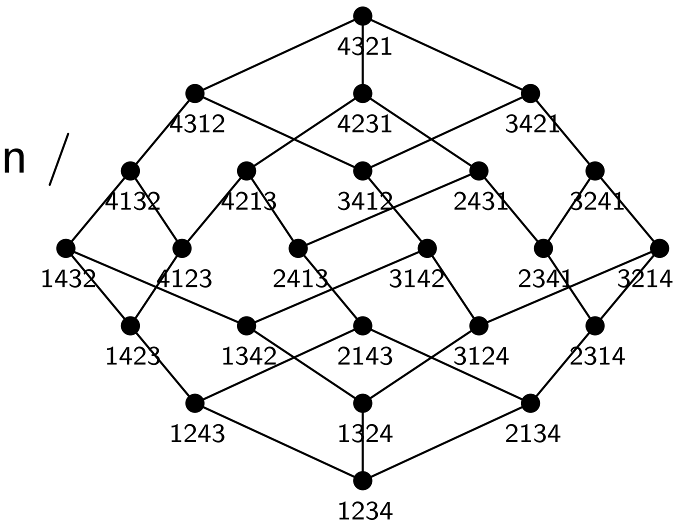
Gray Codes and Flip Graphs

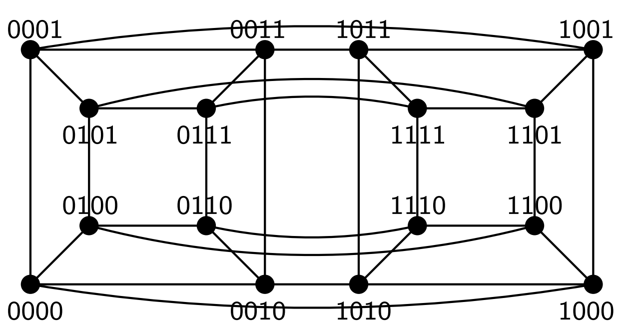
- Many flip graphs can be realized as polytopes and posets.



Associahedron / Tamari lattice

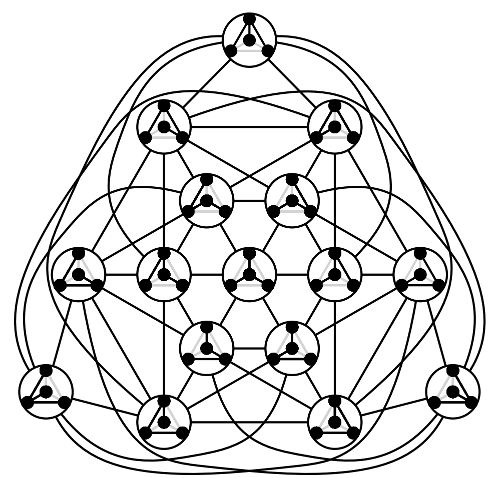
Π_4 Permutahedron / weak order





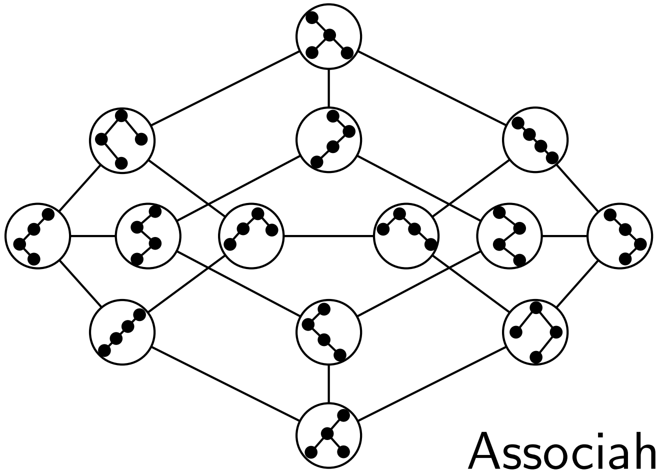
Q_4 Hypercube / Boolean lattice

Base polytope



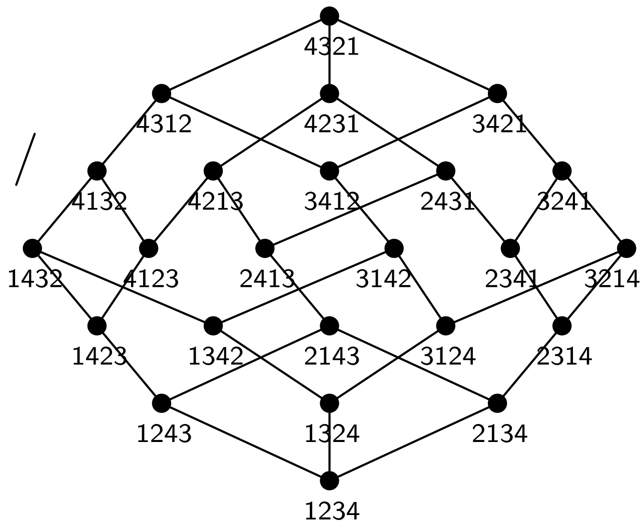
Gray Codes and Flip Graphs

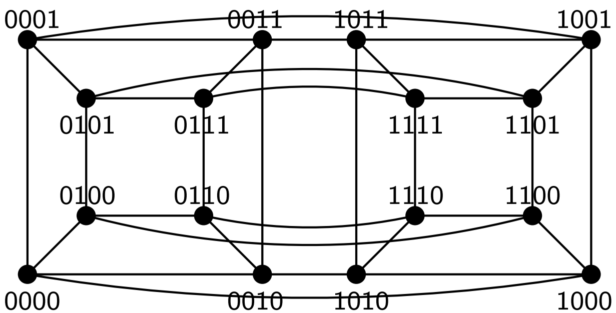
- Gray code → **Hamilton path/cycle** on the flip graph



Associahedron / Tamari lattice

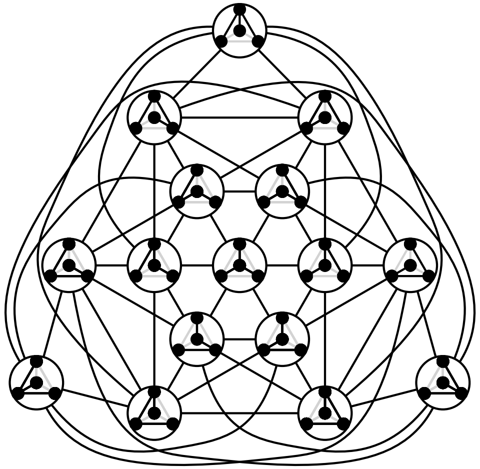
Π_4 Permutahedron / weak order





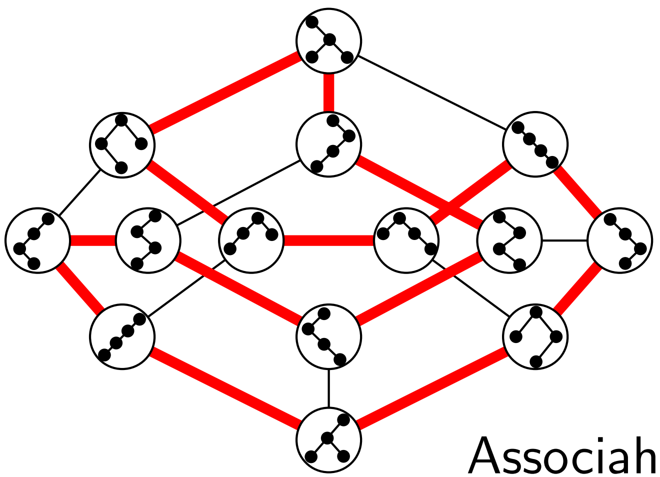
Q_4 Hypercube / Boolean lattice

Base polytope



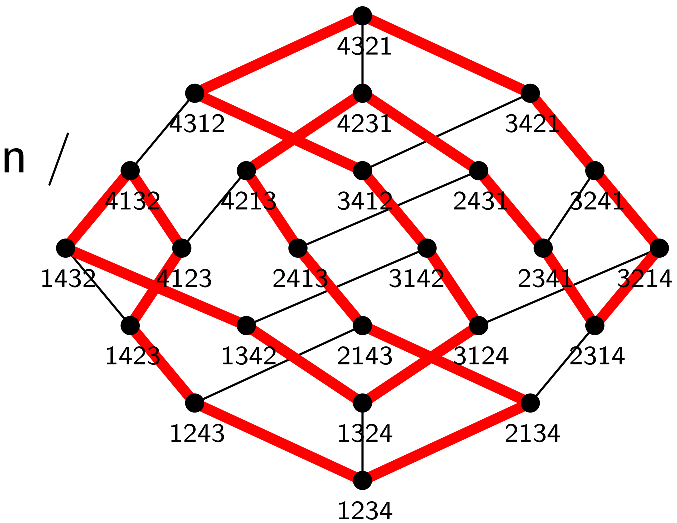
Gray Codes and Flip Graphs

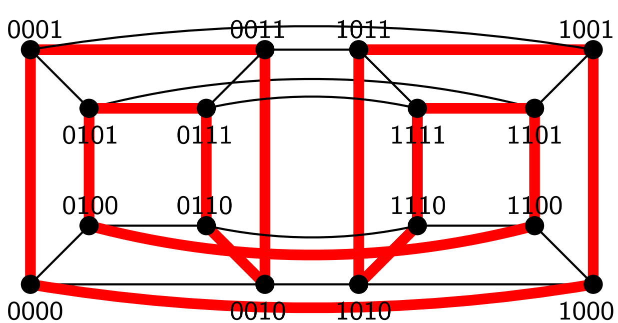
- Gray code → **Hamilton path/cycle** on the flip graph



Associahedron / Tamari lattice

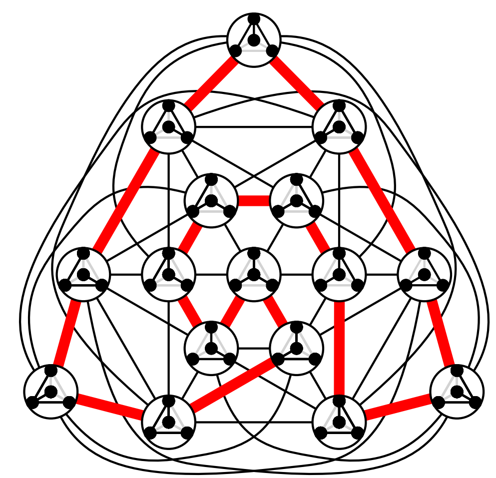
Π_4 Permutahedron / weak order





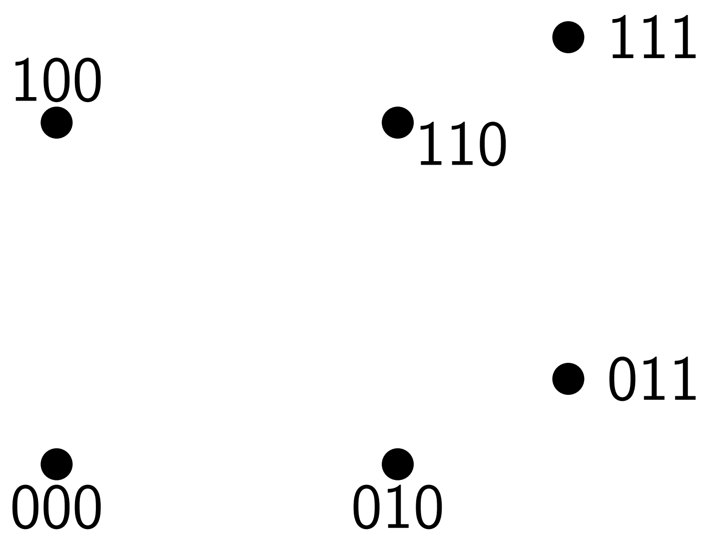
Q_4 Hypercube / Boolean lattice

Base polytope



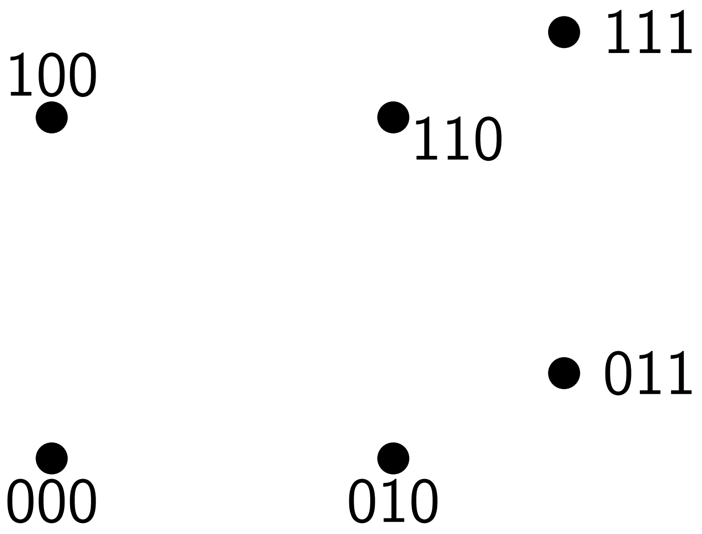
Binary sets and 0/1 polytopes

- **This work:** Gray codes for binary sets $\mathcal{X} \subseteq \{0, 1\}^n$.



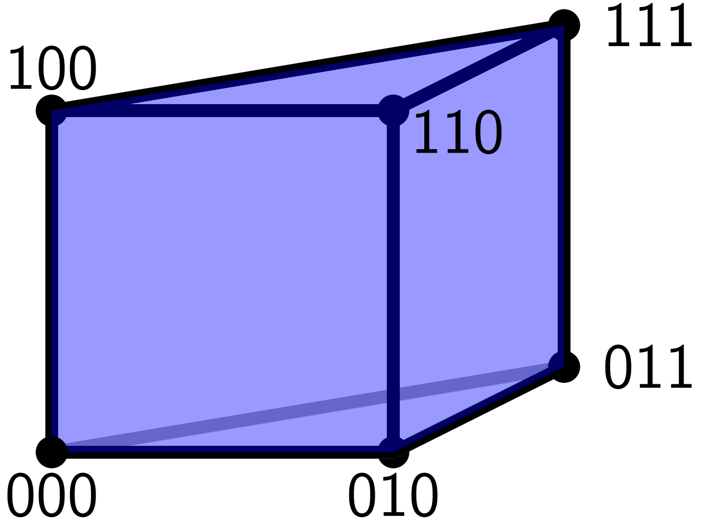
Binary sets and 0/1 polytopes

- **This work:** Gray codes for binary sets $\mathcal{X} \subseteq \{0,1\}^n$.
- **Natural polytope realization:**
  $\rightarrow$ conv$(\mathcal{X})$ is a polytope with $\mathcal{X}$ as vertices.



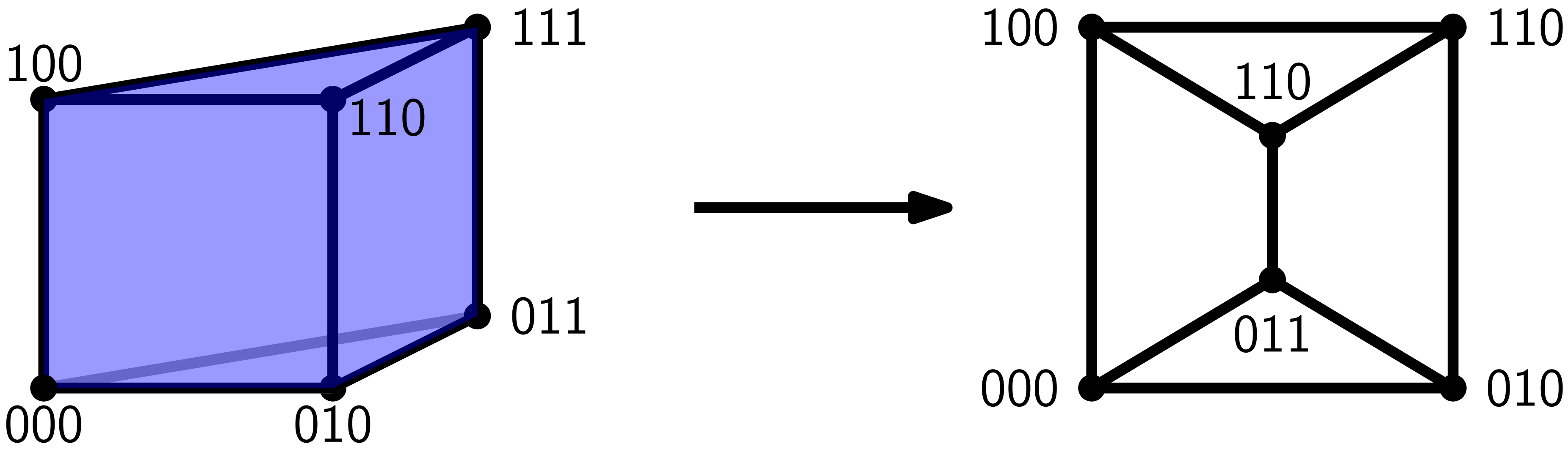
Binary sets and 0/1 polytopes

- **This work:** Gray codes for binary sets $\mathcal{X} \subseteq \{0,1\}^n$.
- **Natural polytope realization:**
  $\rightarrow$ $\mathrm{conv}(\mathcal{X})$ is a polytope with $\mathcal{X}$ as vertices.



Binary sets and 0/1 polytopes

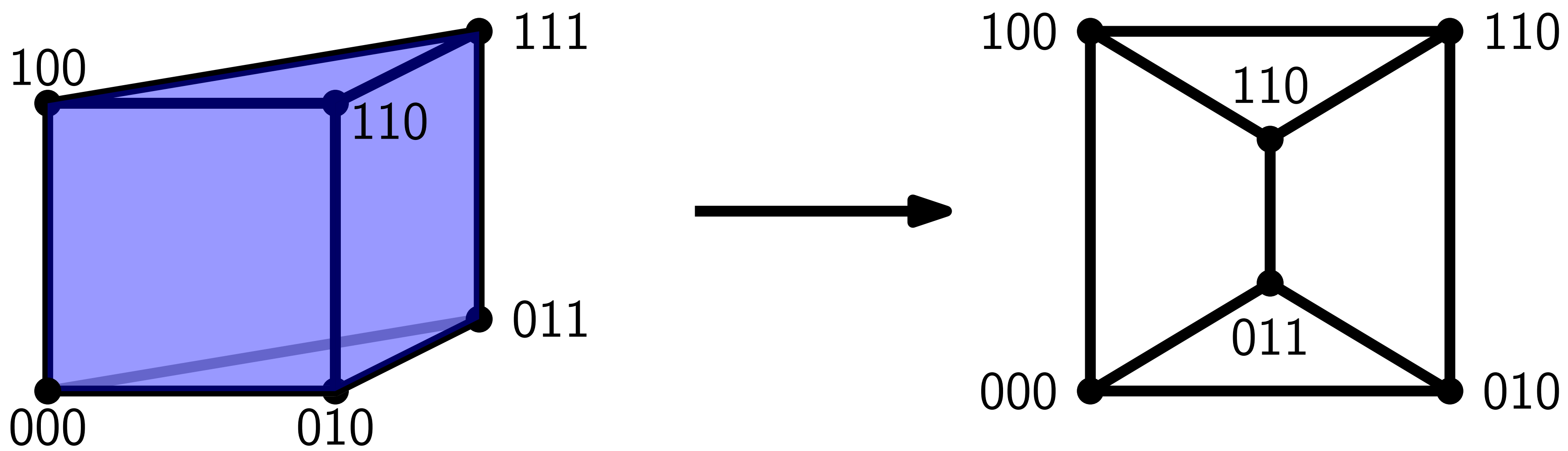
- **This work:** Gray codes for binary sets $\mathcal{X} \subseteq \{0,1\}^n$.
- **Natural polytope realization:**
  $\rightarrow$ conv($\mathcal{X}$) is a polytope with $\mathcal{X}$ as vertices.



- The edges of conv($\mathcal{X}$) capture "local changes" betwen elements of $\mathcal{X}$.

Binary sets and 0/1 polytopes

- **This work:** Gray codes for binary sets $\mathcal{X} \subseteq \{0,1\}^n$.
- **Natural polytope realization:**
  $\rightarrow$ conv($\mathcal{X}$) is a polytope with $\mathcal{X}$ as vertices.



- The edges of conv($\mathcal{X}$) capture "local changes" betwen elements of $\mathcal{X}$.
- Hamilton paths in conv($\mathcal{X}$) are combinatorial Gray codes.

Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.

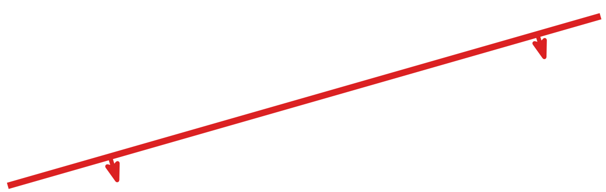
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

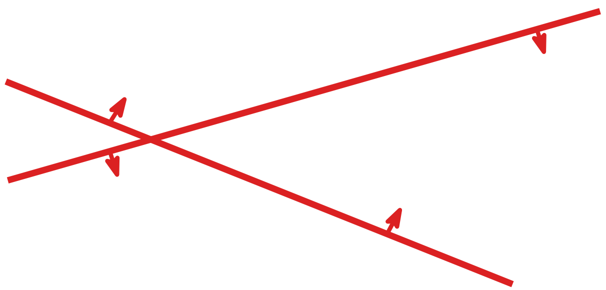
- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem
Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.
Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



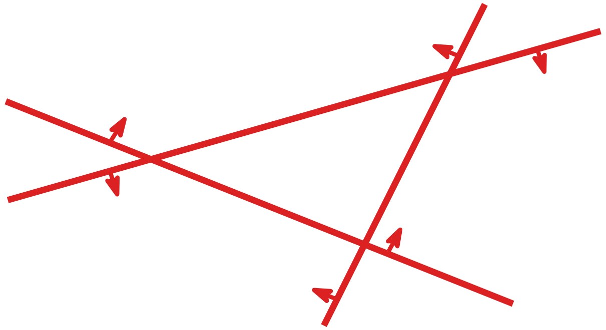
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



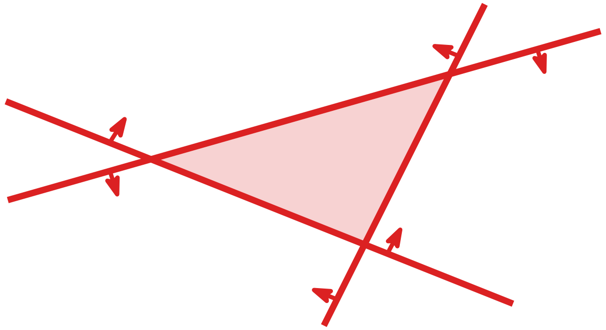
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



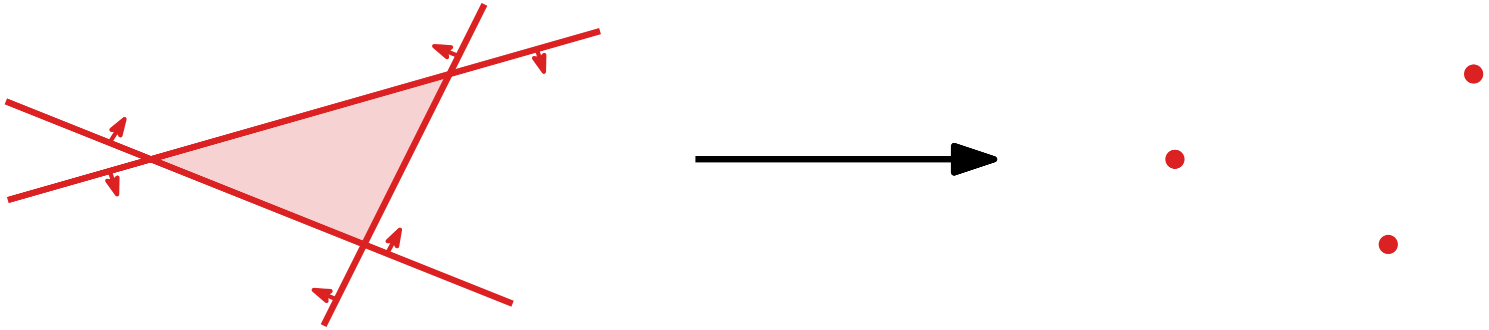
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

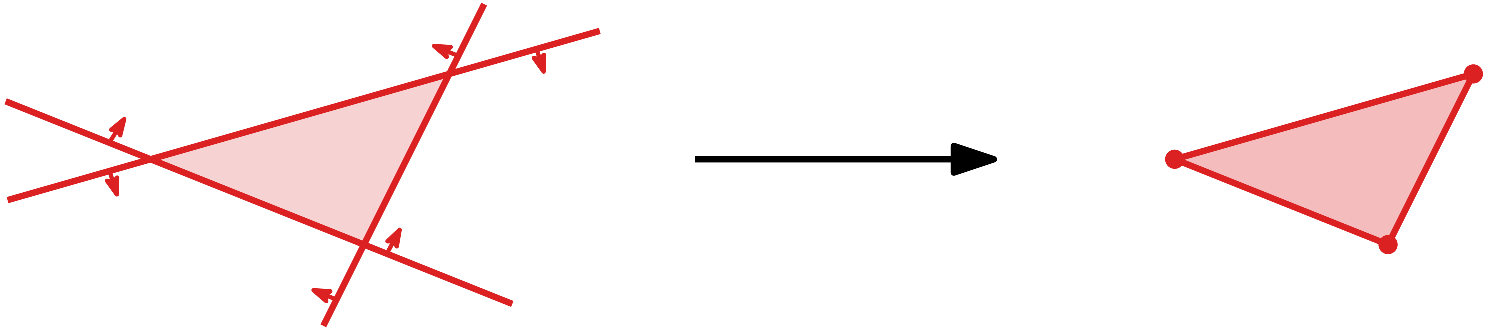
- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem
Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.
Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



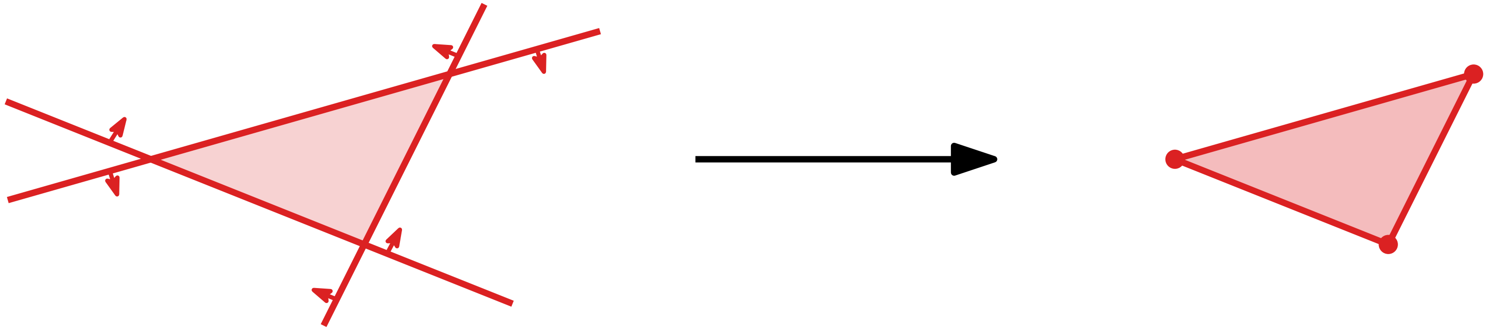
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



	Generation	Gray codes
Avg. delay Delay		

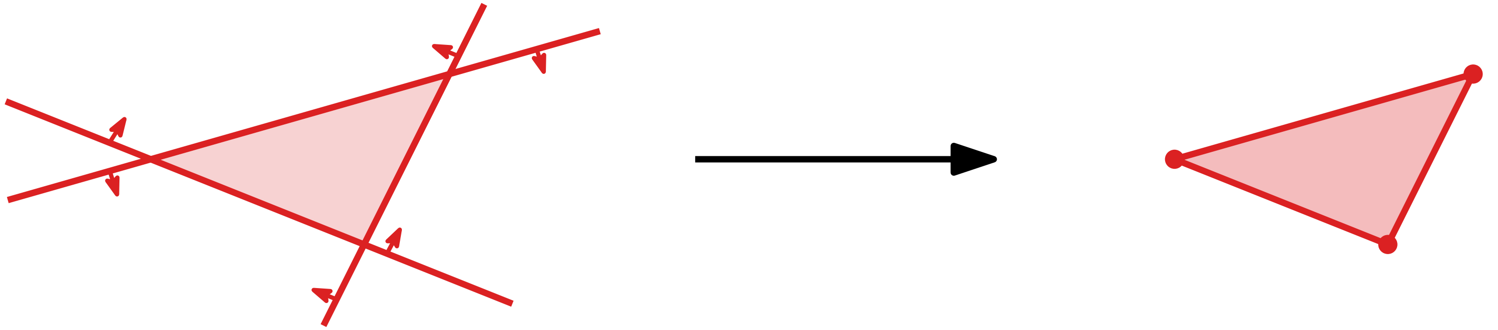
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



	Generation	Gray codes
Avg. delay Delay	$\mathcal{O}(nT_{LP})$ **[Bussieck, Lübbecke 99]**	

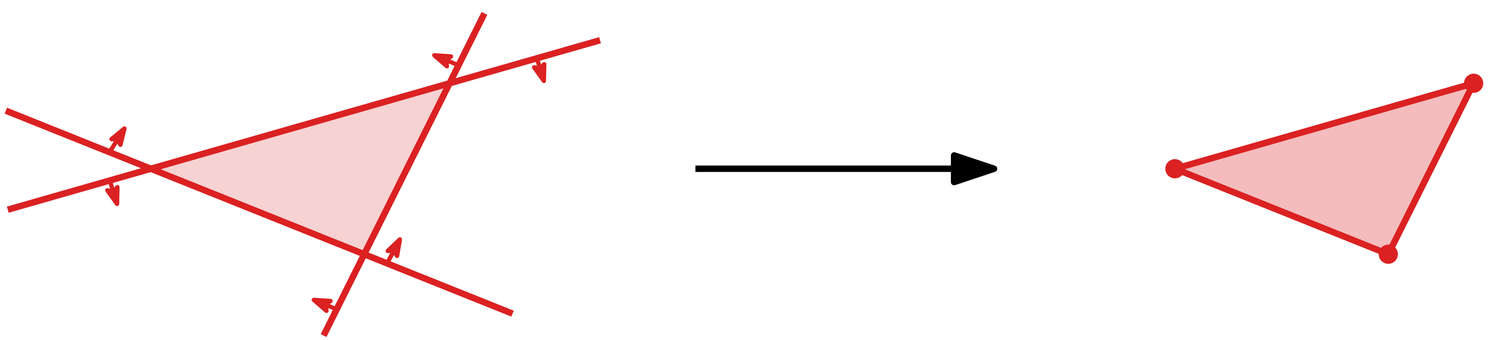
Four 0/1 problems: Vertex enumeration for 0/1 polytopes

0/1 Vertex enumeration problem

Given: Inequalities $Ax \leq b$ that define a 0/1 polytope.

Compute: all vertices of $\{x \in \mathbb{R}^n : Ax \leq b\}$

- Interesting problem from computational geometry
  $\rightarrow$ Change from $\mathcal{H}$-representation to $\mathcal{V}$-representation.



	Generation	Gray codes
Avg. delay Delay	$\mathcal{O}(nT_{LP})$ **[Bussieck, Lübbecke 99]**	???

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0,1\}^m : T \text{ is a spanning tree of } G\}$

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes
Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.
Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0,1\}^m : T \text{ is a spanning tree of } G\}$
- Vertices of $\text{conv}(\mathcal{X}) \leftarrow$ Spanning trees of G.

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0, 1\}^m : T \text{ is a spanning tree of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ Spanning trees of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ Spanning trees which differ by an edge exchange.

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0,1\}^m : T \text{ is a spanning tree of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ Spanning trees of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ Spanning trees which differ by an edge exchange.

	Generation	Gray codes
Avg. delay		
Delay		

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes
Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.
Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0,1\}^m : T \text{ is a spanning tree of } G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ Spanning trees of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ Spanning trees which differ by an edge exchange.

	Generation	Gray codes
Avg. delay	$\mathcal{O}(1)$ **[Shioura, Tamura 95]**	
Delay	$\mathcal{O}(m \log n (\log \log n)^3)$ **[M, Mütze, Williams 22]**	

Four 0/1 problems: Spanning tree Gray codes

Spanning tree Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all spanning trees of G s.t. consecutives ones differ by an edge-exchange.

- $\mathcal{X} := \{\chi_T \in \{0,1\}^m : T \text{ is a spanning tree of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ Spanning trees of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ Spanning trees which differ by an edge exchange.

	Generation	Gray codes
Avg. delay	$\mathcal{O}(1)$ **[Shioura, Tamura 95]**	$\mathcal{O}(1)$ **[Smith 96]**
Delay	$\mathcal{O}(m \log n (\log \log n)^3)$ **[M, Mütze, Williams 22]**	

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of $\text{conv}(\mathcal{X}) \leftarrow$ perfect matchings of G.

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ perfect matchings of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ perfect matchings which differ by an alternating cycle.

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ perfect matchings of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ perfect matchings of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay		
Delay		

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ perfect matchings of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay	$\mathcal{O}(n)$ **[Fukuda, Matsui 94]**	
Delay	$\mathcal{O}(m)$ **[Fukuda, Matsui 94]**	

Four 0/1 problems: Perfect matching Gray codes

Perfect matching Gray codes

Given: A graph $G = (V, E)$, $n = |V|$, $m = |E|$.

Compute: all perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a perfect matching of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ perfect matchings of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay	$\mathcal{O}(n)$ **[Fukuda, Matsui 94]**	???
Delay	$\mathcal{O}(m)$ **[Fukuda, Matsui 94]**	

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a min-weight perfect matching of } G\}$

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a min-weight perfect matching of } G\}$
- Vertices of $\text{conv}(\mathcal{X}) \leftarrow$ min-weight perfect matchings of G.

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M$ is a min-weight perfect matching of $G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ min-weight perfect matchings of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ min-weight perfect matchings which differ by an alternating cycle.

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a min-weight perfect matching of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a min-weight perfect matching of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay		
Delay		

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M$ is a min-weight perfect matching of $G\}$
- Vertices of conv$(\mathcal{X}) \leftarrow$ min-weight perfect matchings of G.
- Edges of conv$(\mathcal{X}) \leftarrow$ min-weight perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay	$\mathcal{O}(m+n)$ **[Fukuda, Matsui 92,94]**	
Delay	???	

Four 0/1 problems: Min-weight perfect matchings Gray codes

Min-weight perfect matching Gray codes

Given: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}$.

Output: all minimum weight perfect matchings of G s.t. consecutive ones differ by an alternating cycle.

- $\mathcal{X} := \{\chi_M \in \{0,1\}^m : M \text{ is a min-weight perfect matching of } G\}$
- Vertices of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings of G.
- Edges of conv($\mathcal{X}$) $\leftarrow$ min-weight perfect matchings which differ by an alternating cycle.
- Only the bipartite version known:

	Generation	Gray codes
Avg. delay	$\mathcal{O}(m+n)$ **[Fukuda, Matsui 92,94]**	???
Delay	???	

Big Hammers

- Solutions to the four previous problems with ad hoc methods.

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.
- Combinatorial generation:

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.
- Combinatorial generation:
  - Reverse-search **[Avis, Fukuda 96]**

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.
- Combinatorial generation:
  - Reverse-search **[Avis, Fukuda 96]**
  - Backtracking **[Folklore]**

Big Hammers

- Solutions to the four previous problems with ad hoc methods.

- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.

- Combinatorial generation:
  - Reverse-search **[Avis, Fukuda 96]**
  - Backtracking **[Folklore]**

- Gray codes:

Big Hammers

- Solutions to the four previous problems with ad hoc methods.
- Fundamental algorithmic tasks:
  - Counting $\leftarrow$ generating functions,
  - Random sampling $\leftarrow$ markov chains,
  - Optimization $\leftarrow$ linear programming, dynamic programming, greedy.
- Combinatorial generation:
  - Reverse-search **[Avis, Fukuda 96]**
  - Backtracking **[Folklore]**
- Gray codes:
  - Jump framework **[Hartung, Hoang, Mütze, Williams 20]**

Main results

Thm.

Optimization over $\mathcal{X} \subseteq \{0,1\}^n$ can be solved in time $\mathcal{O}(T)$

$$\Longrightarrow$$

A Hamilton path in $\mathrm{conv}(\mathcal{X})$ can be computed in $\mathcal{O}(T \operatorname{polylog} n)$-delay.

Main results

Thm.

Optimization over $\mathcal{X} \subseteq \{0,1\}^n$ can be solved in time $\mathcal{O}(T)$

$$\Longrightarrow$$

A Hamilton path in $\mathrm{conv}(\mathcal{X})$ can be computed in $\mathcal{O}(T \operatorname{polylog} n)$-delay.

- **Weighted version:** Let $c \in \mathbb{R}^n$.

Main results

Thm.

Optimization over $\mathcal{X} \subseteq \{0,1\}^n$ can be solved in time $\mathcal{O}(T)$
$$\Longrightarrow$$
A Hamilton path in $\text{conv}(\mathcal{X})$ can be computed in $\mathcal{O}(T \operatorname{polylog} n)$-delay.

- **Weighted version:** Let $c \in \mathbb{R}^n$.

Thm.

Optimization over $\mathcal{X} \subseteq \{0,1\}^n$ can be solved in time $\mathcal{O}(T)$
$$\Longrightarrow$$
Ham. path in $\text{conv}(\arg\min_{x \in \mathcal{X}} cx)$ can be computed in $\mathcal{O}(T \operatorname{polylog} n)$-delay.

Black box corollaries

- **0/1 Vertex enumeration:**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay $+$ Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.

Best delay for generation algorithms!

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay $+$ Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.

Best amortized delay for generation algorithms!

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.

Results in non-bipartite graphs!

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.
- **Super versatile!**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.
- **Super versatile!**
- **Many more new Gray codes and applications:**

Black box corollaries

- **0/1 Vertex enumeration:**
  - $\mathcal{O}(\log n T_{LP})$ delay + Hamilton path in the polytope.
- **Spanning tree Gray codes:**
  - $\mathcal{O}(m \log n)$ delay.
- **Perfect matching Gray codes:**
  - $\mathcal{O}(m\sqrt{n}(\log n)^{5/2})$ delay.
- **Min weight perfect matching Gray codes:**
  - $\mathcal{O}((mn + n^2 \log n) \log n)$ delay.
- **Super versatile!**
- **Many more new Gray codes and applications:**
  - Forests, matchings, matroids, matroid intersection, etc...

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,
- ◦ among those, y differs to x in the shortest possible prefix,

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- y is unvisited,
- among those, y differs to x in the shortest possible prefix,
- among those, choose y such that $d_H(x, y)$ is as small as possible.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- y is unvisited,
- among those, y differs to x in the shortest possible prefix,
- among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

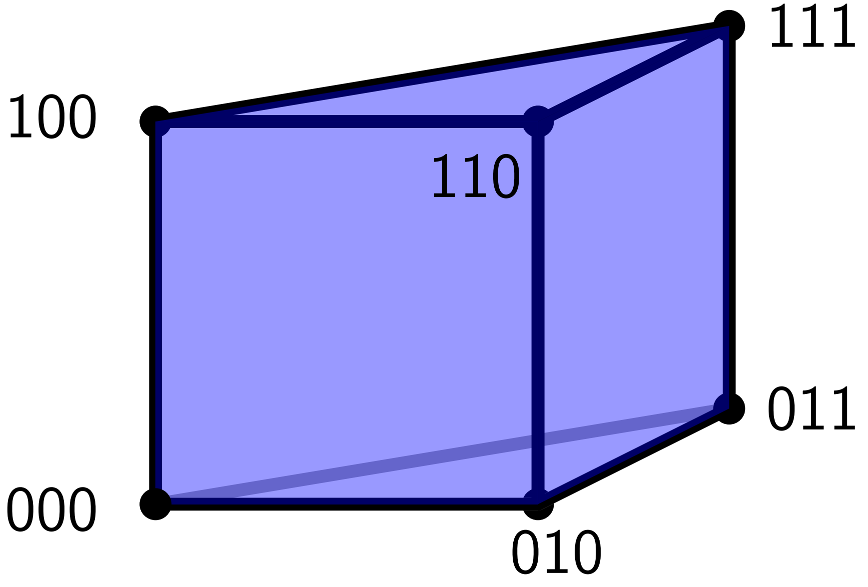
P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

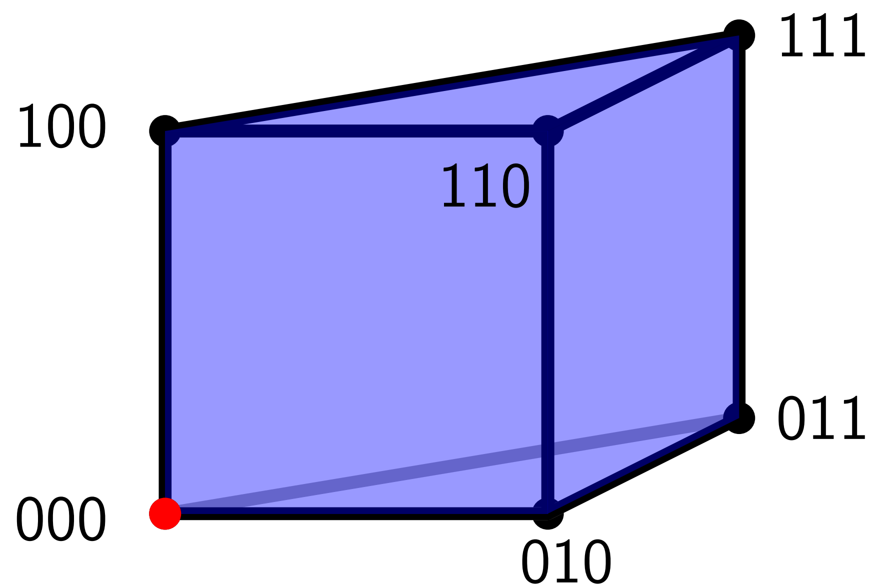
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

∘ y is unvisited,

∘ among those, y differs to x in the shortest possible prefix,

∘ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

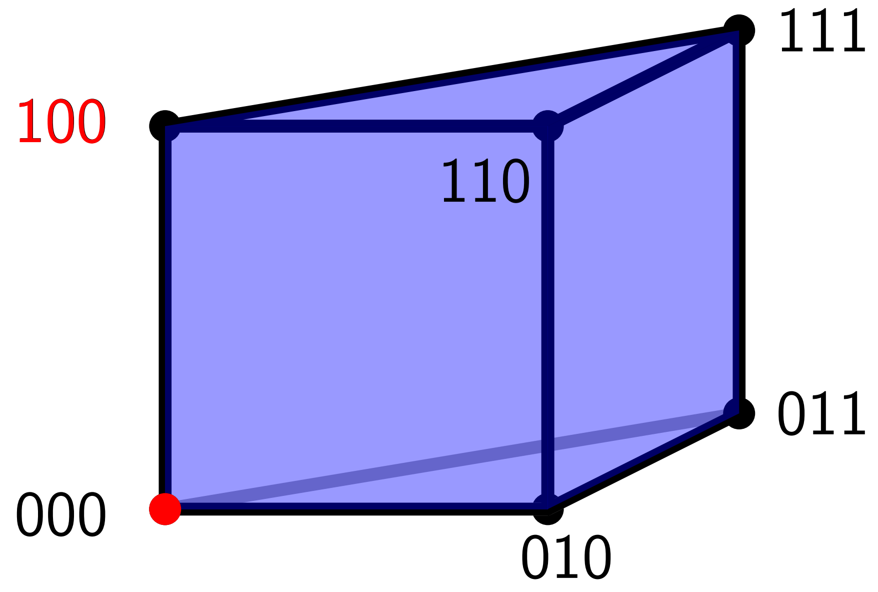
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

∘ y is unvisited,

∘ among those, y differs to x in the shortest possible prefix,

∘ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

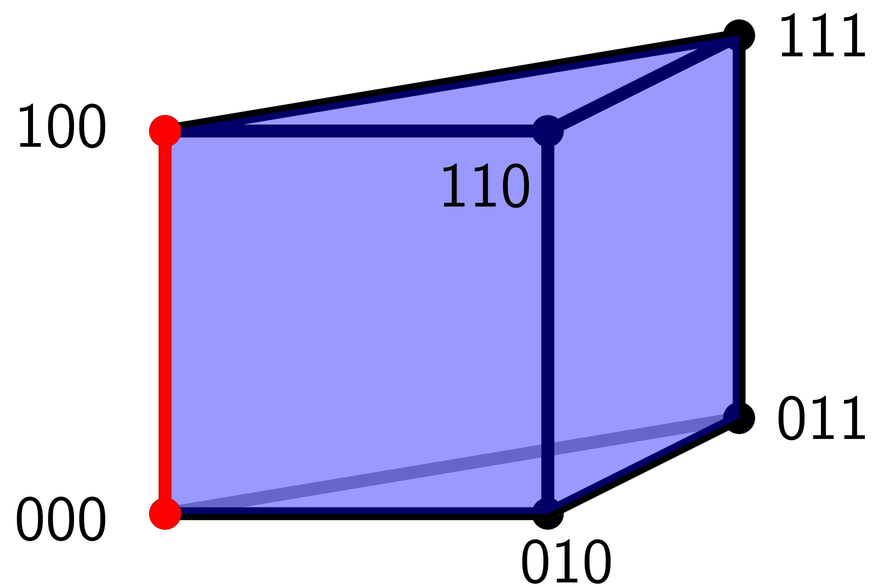
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

○ y is unvisited,

○ among those, y differs to x in the shortest possible prefix,

○ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

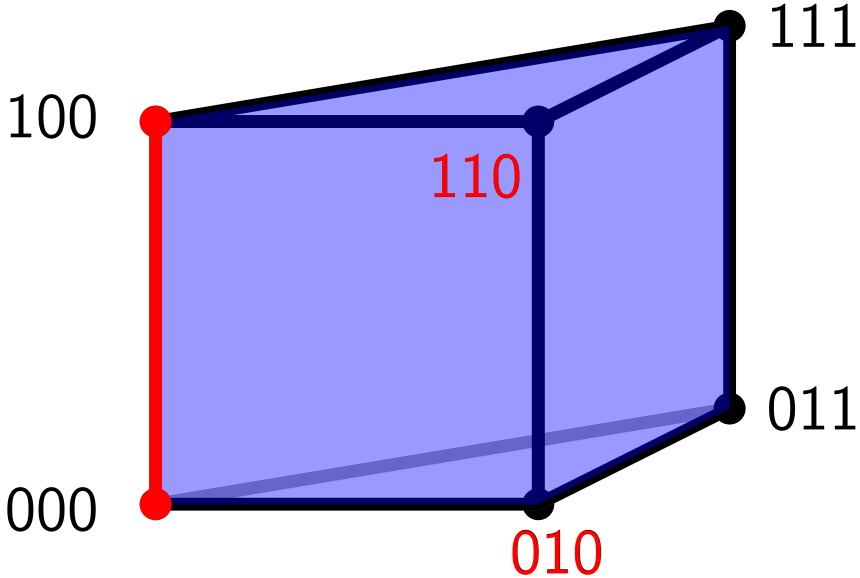
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

∘ y is unvisited,

∘ among those, y differs to x in the shortest possible prefix,

∘ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

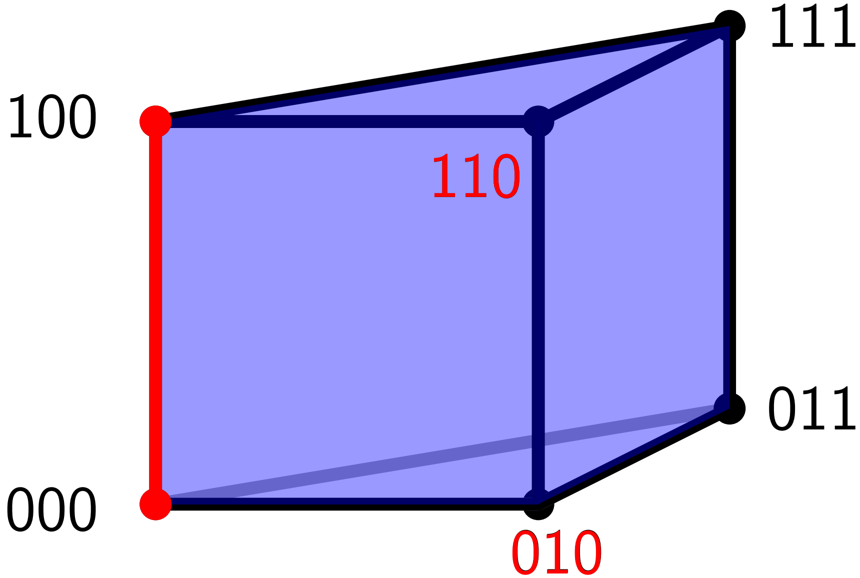
P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



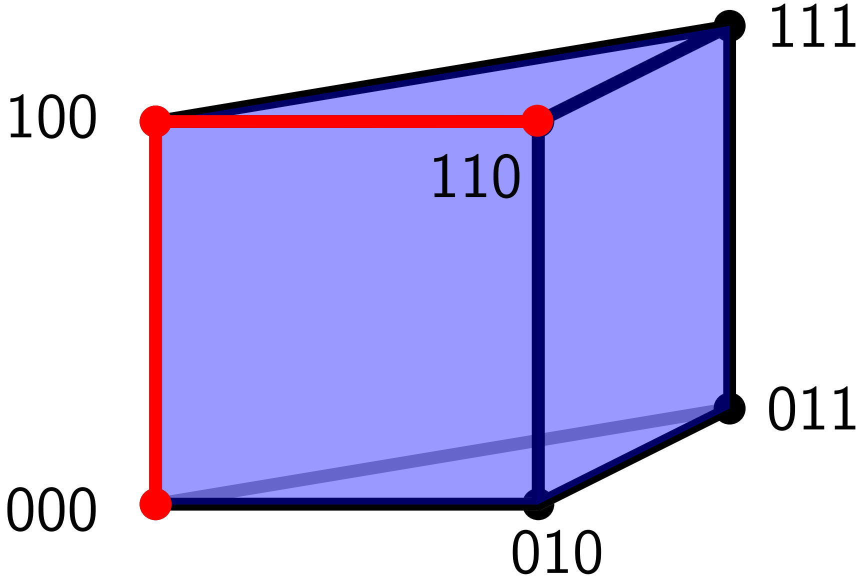
The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

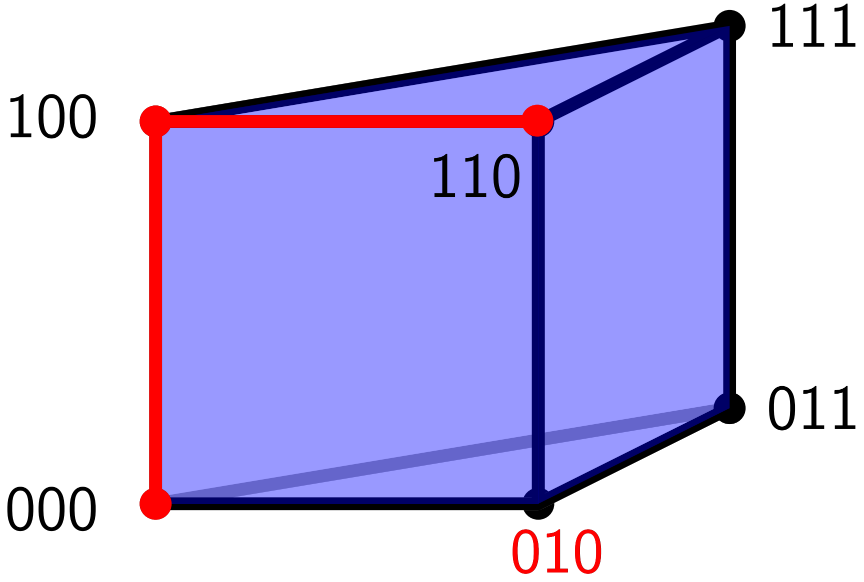
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

○ y is unvisited,

○ among those, y differs to x in the shortest possible prefix,

○ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



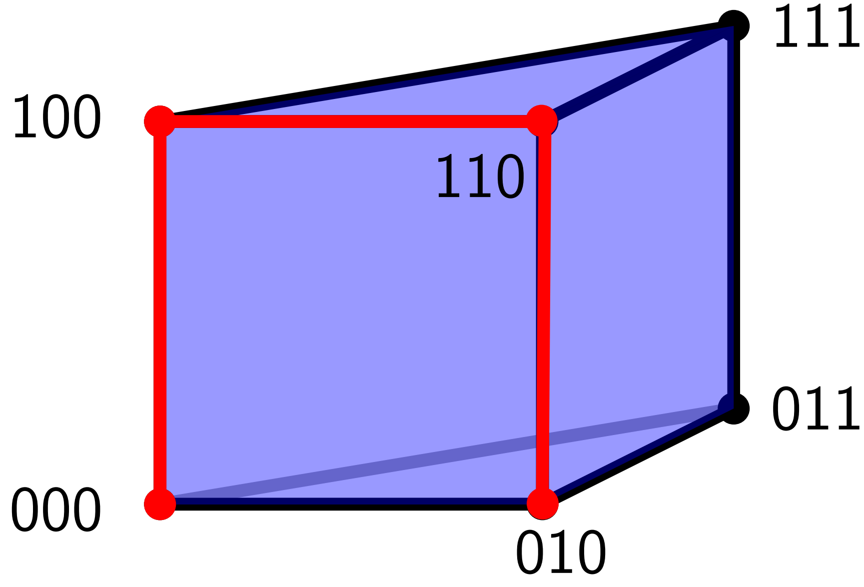
The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

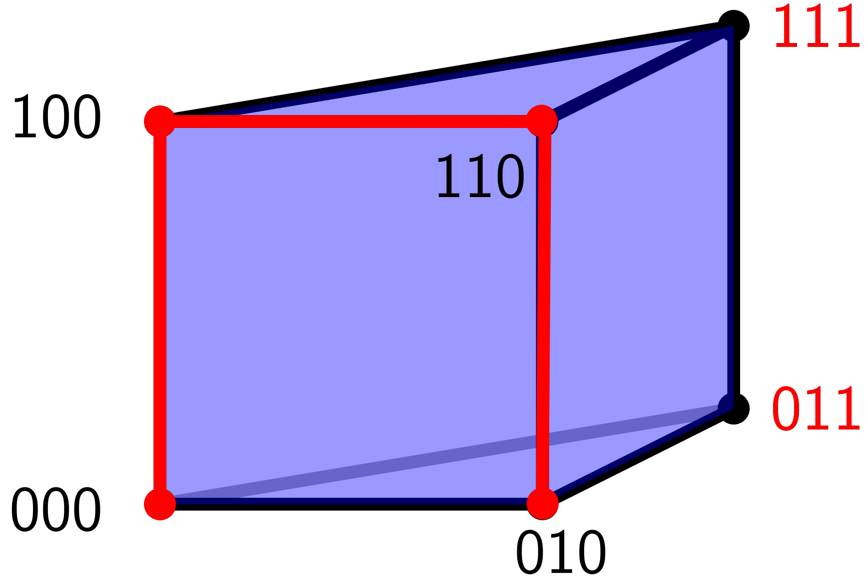
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

○ y is unvisited,

○ among those, y differs to x in the shortest possible prefix,

○ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



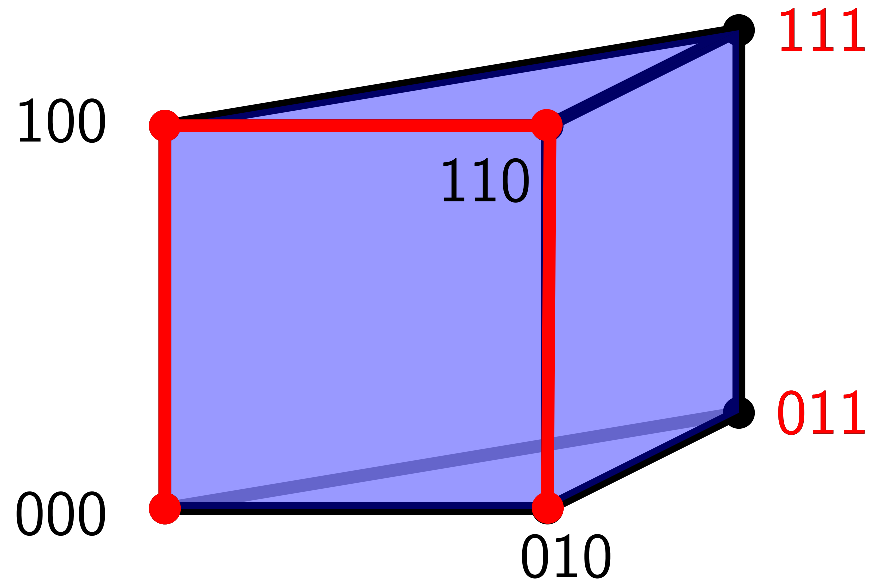
The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,
- ◦ among those, y differs to x in the shortest possible prefix,
- ◦ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



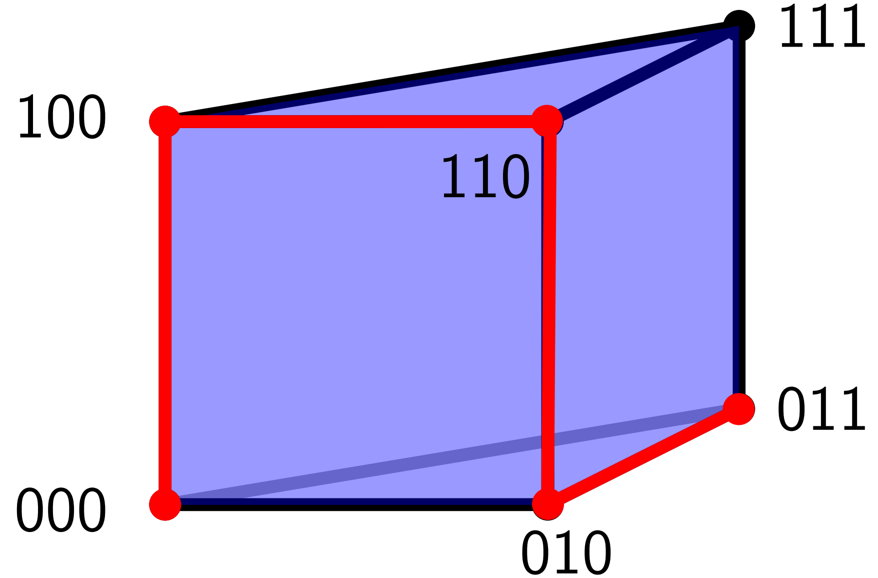
The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

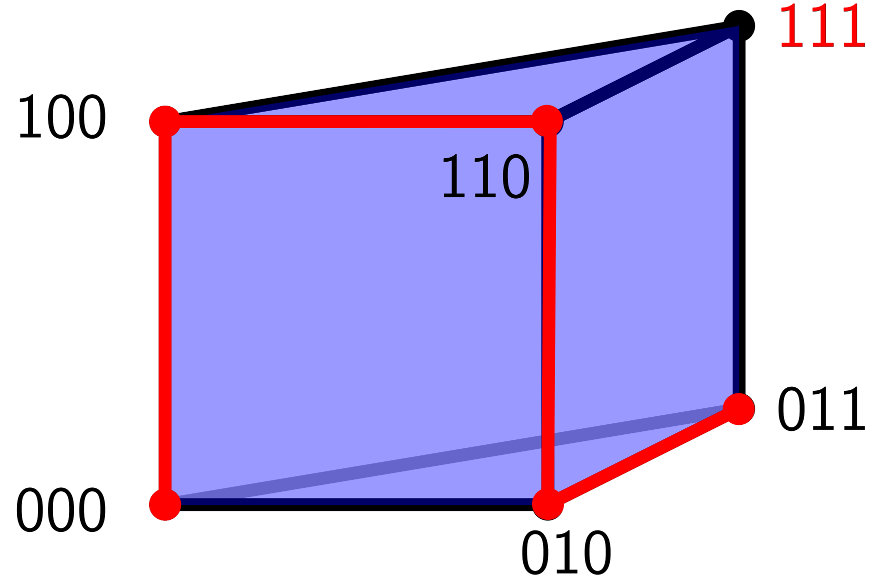
P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

∘ y is unvisited,

∘ among those, y differs to x in the shortest possible prefix,

∘ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

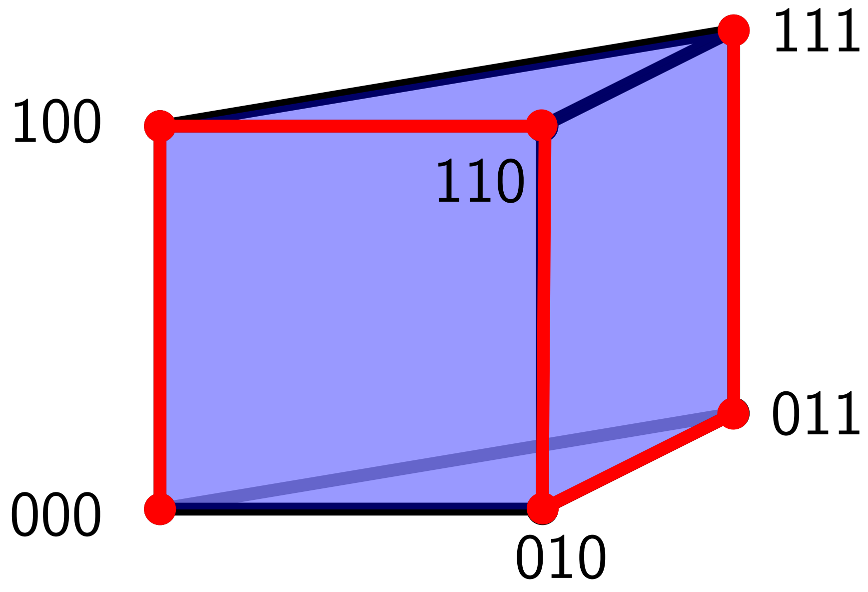
P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- y is unvisited,
- among those, y differs to x in the shortest possible prefix,
- among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

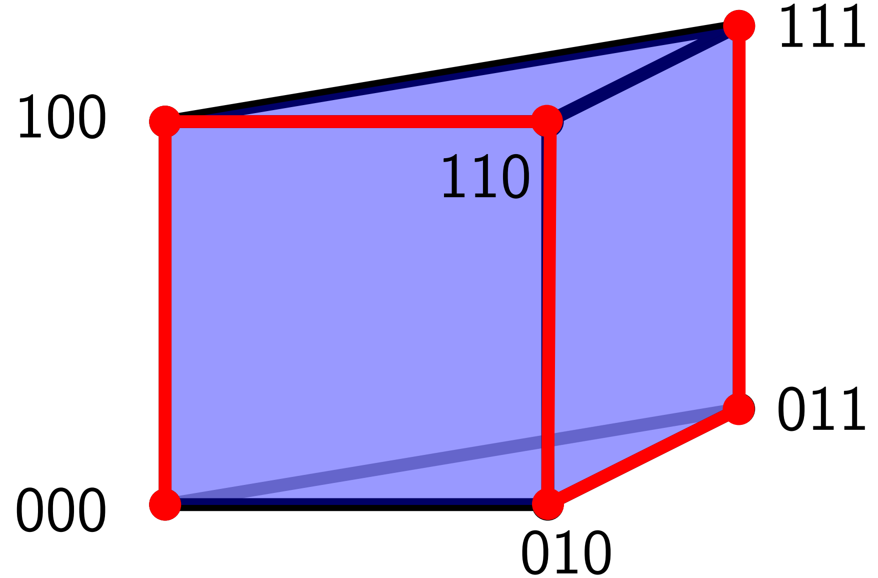
P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,
- ◦ among those, y differs to x in the shortest possible prefix,
- ◦ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

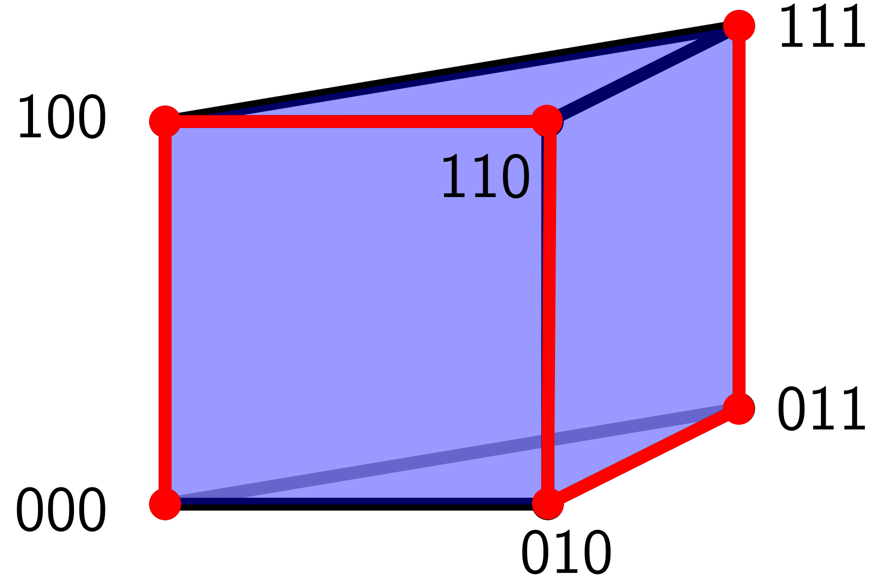
P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ y is unvisited,
- ○ among those, y differs to x in the shortest possible prefix,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- **Example:**



The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.

P2. Visit x.

P3. Choose a vertex $y \in \mathcal{X}$ such that

- y is unvisited,
- among those, y differs to x in the shortest possible prefix,
- among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

∘ y is unvisited,

∘ among those, y differs to x in the shortest possible prefix,

∘ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,
- ◦ among those, y differs to x in the shortest possible prefix,
- ◦ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.
- Just a *conceptual* description.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ◦ y is unvisited,
- ◦ among those, y differs to x in the shortest possible prefix,
- ◦ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Otherwise, set $x \leftarrow y$, and go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.
- Just a *conceptual* description.
- *History-free* implementation.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$ and $U \leftarrow [n]$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

◦ ~~y is unvisited,~~

◦ among those, y differs to x in the shortest possible prefix of length in U,

◦ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Else, set $x \leftarrow y$, update U, go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.
- Just a *conceptual* description.
- *History-free* implementation.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$ and $U \leftarrow [n]$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

◦ ~~y is unvisited,~~

◦ among those, y differs to x in the shortest possible prefix of length in U,

◦ among those, choose y such that $d_H(x, y)$ is as small as possible.
If no such y exists, then terminate. Else, set $x \leftarrow y$, update U, go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.
- Just a *conceptual* description.
- *History-free* implementation.

The algorithm

P1. Select an initial bitstring $x^1 \in \mathcal{X}$. Set $x \leftarrow x^1$ and $U \leftarrow [n]$.
P2. Visit x.
P3. Choose a vertex $y \in \mathcal{X}$ such that

- ○ ~~y is unvisited,~~
- ○ among those, y differs to x in the shortest possible prefix of length in U,
- ○ among those, choose y such that $d_H(x, y)$ is as small as possible.

If no such y exists, then terminate. Else, set $x \leftarrow y$, update U, go to **P2**.

- Always works, no matter $\mathcal{X}$, x^1, or choice of y.
- Just a *conceptual* description.
- *History-free* implementation.
- Reduce to optimization problems

Why greedy works?

- Why it computes Hamilton paths?

Why greedy works?

- Why it computes Hamilton paths?
- How to implement in a history-free way?

Why greedy works?

- Why it computes Hamilton paths?
- How to implement in a history-free way?
- Optimization? Where?

Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

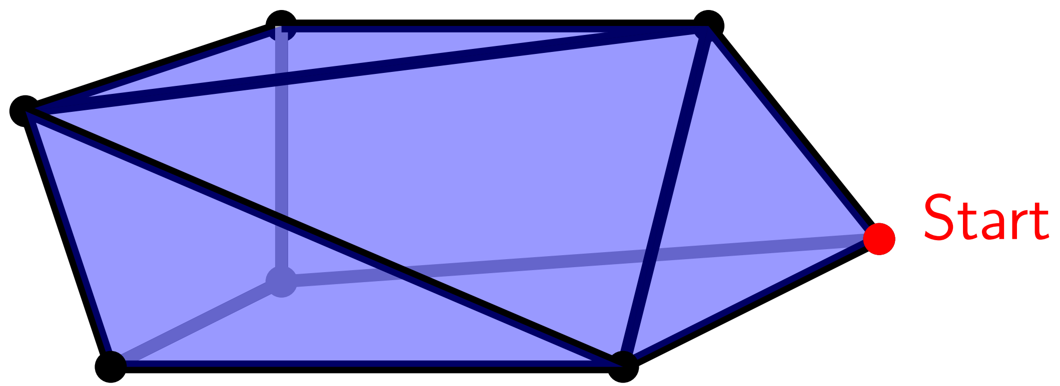
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



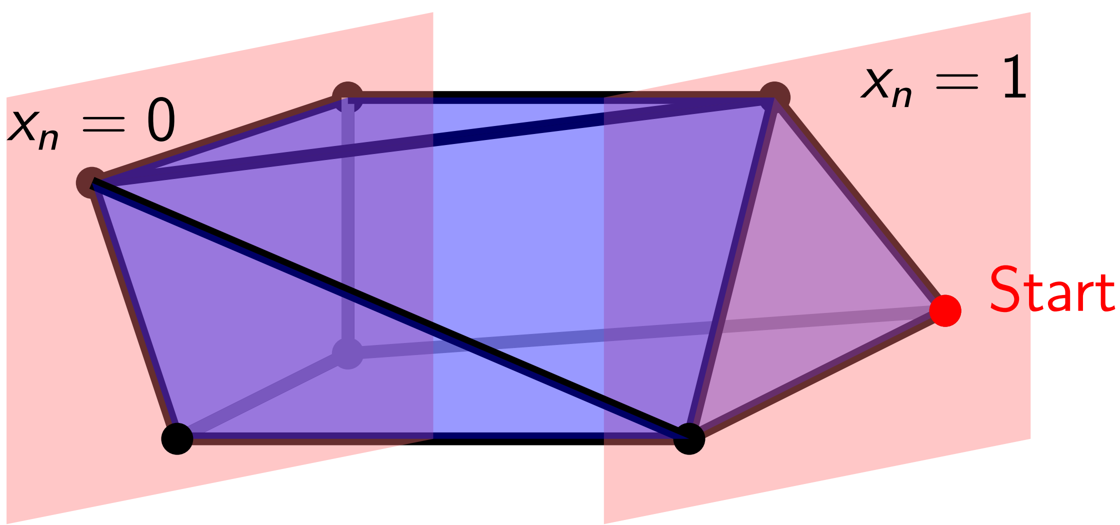
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



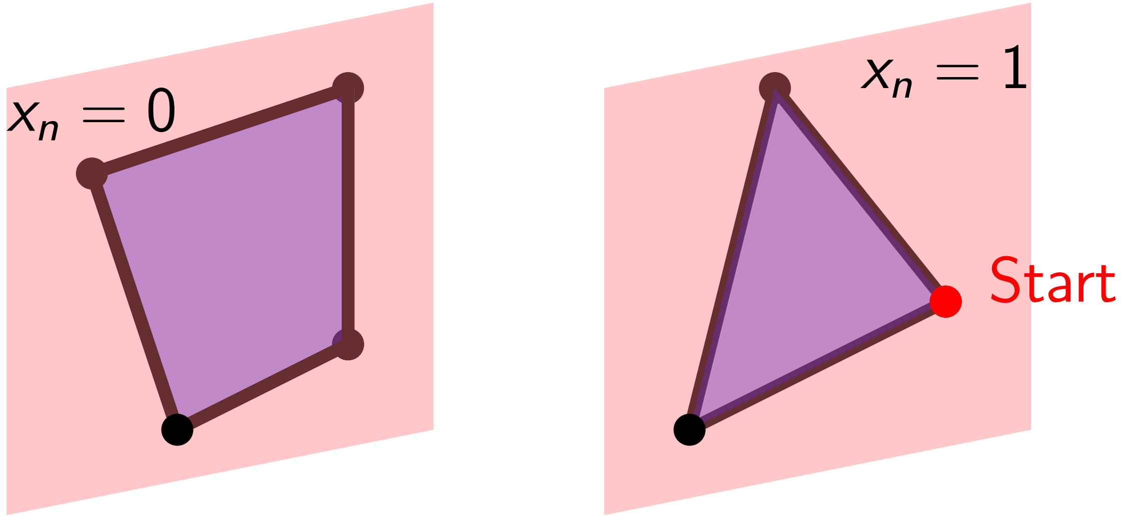
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



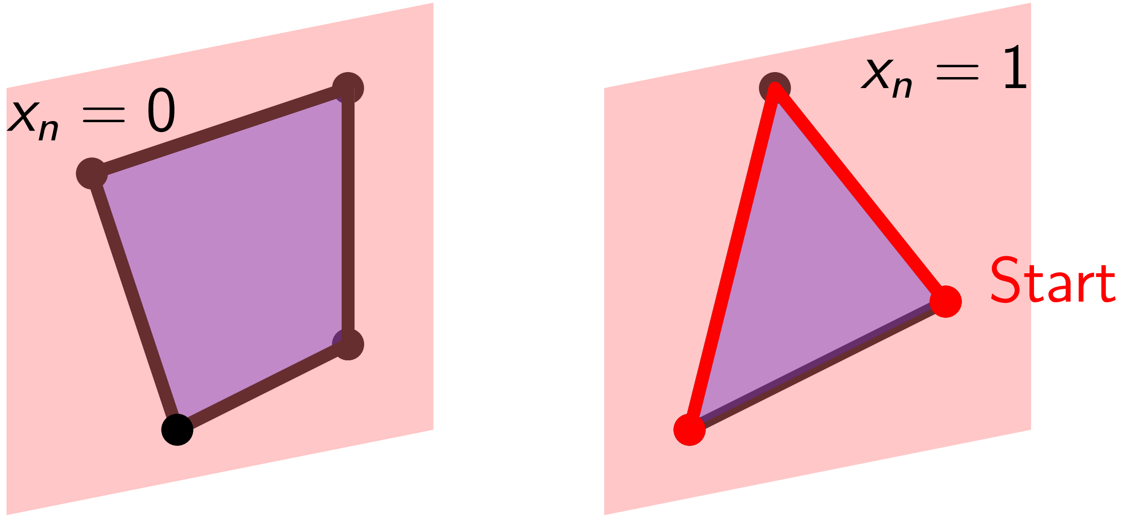
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



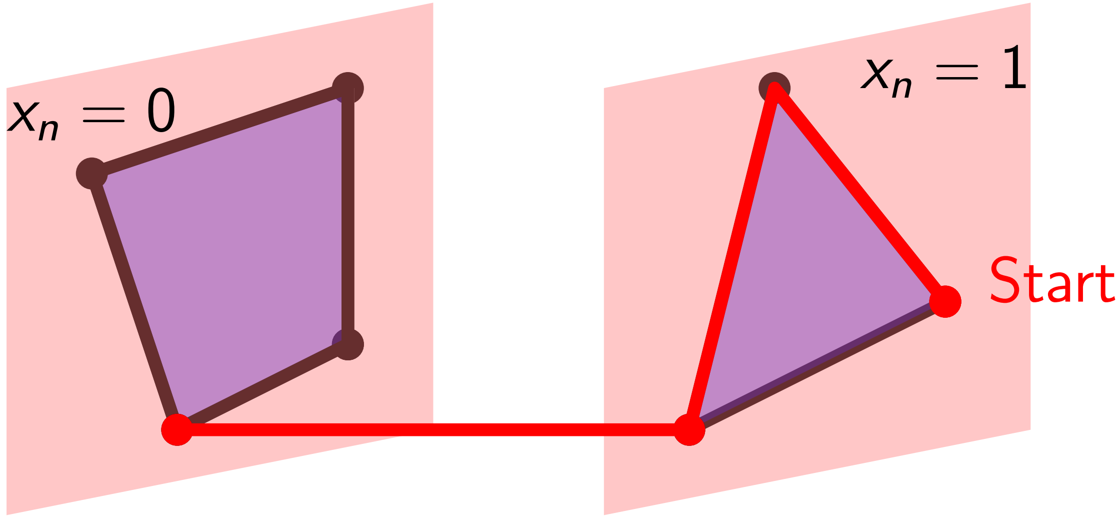
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



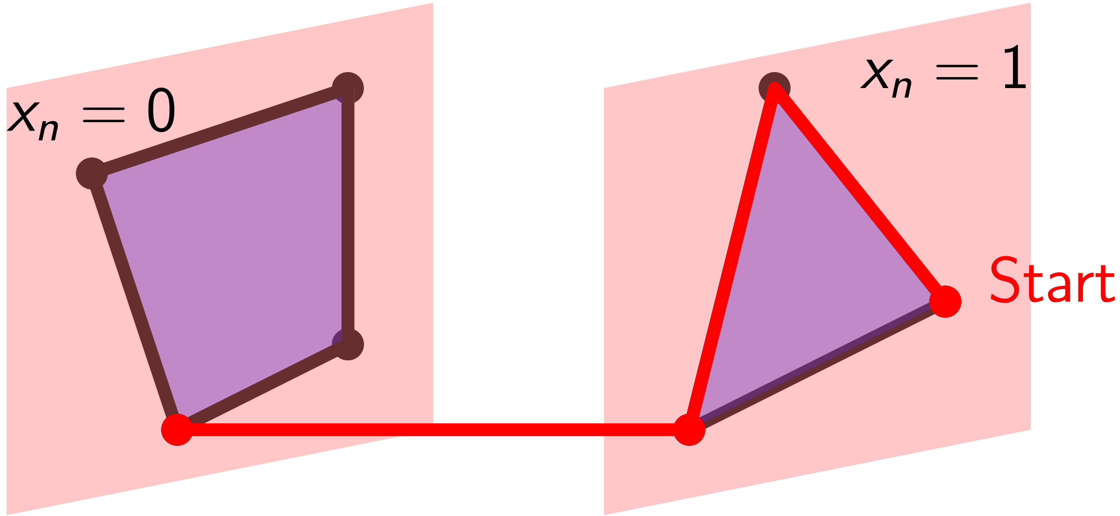
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



Lemma. If $x_n \neq y_n$ and $d_H(x, y)$ is minimized, among y with $y_n \neq x_n$, then xy is an edge of the polytope.

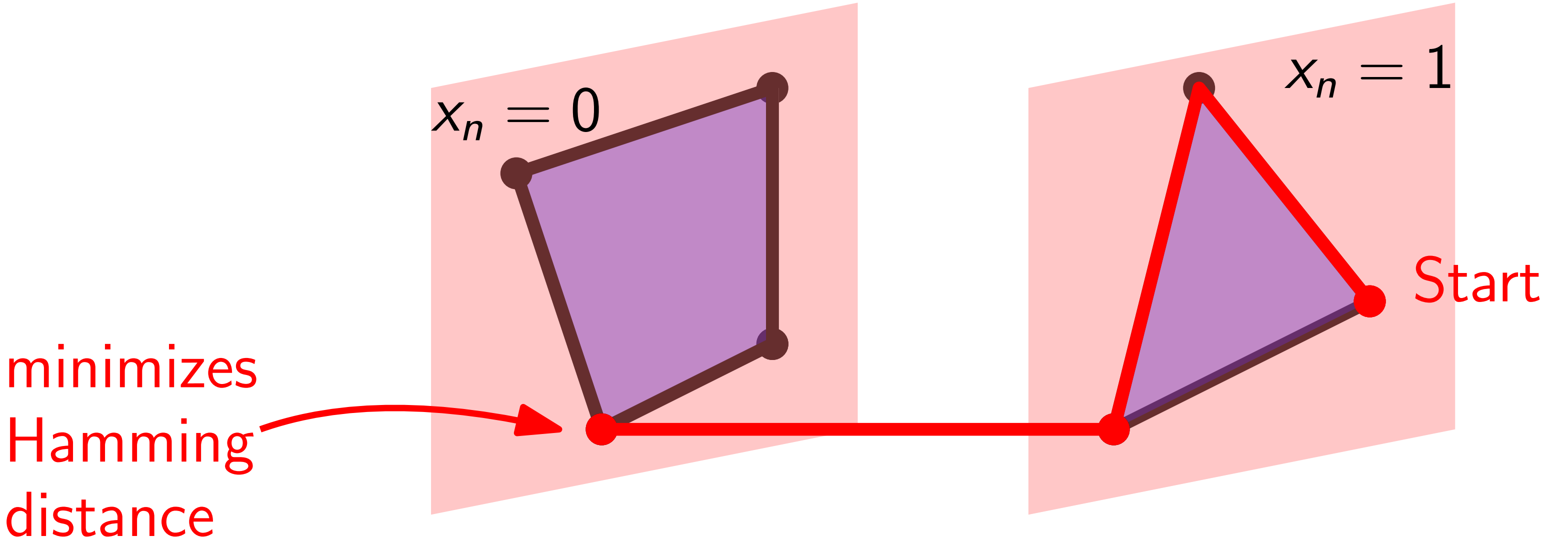
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



Lemma. If $x_n \neq y_n$ and $d_H(x, y)$ is minimized, among y with $y_n \neq x_n$, then xy is an edge of the polytope.

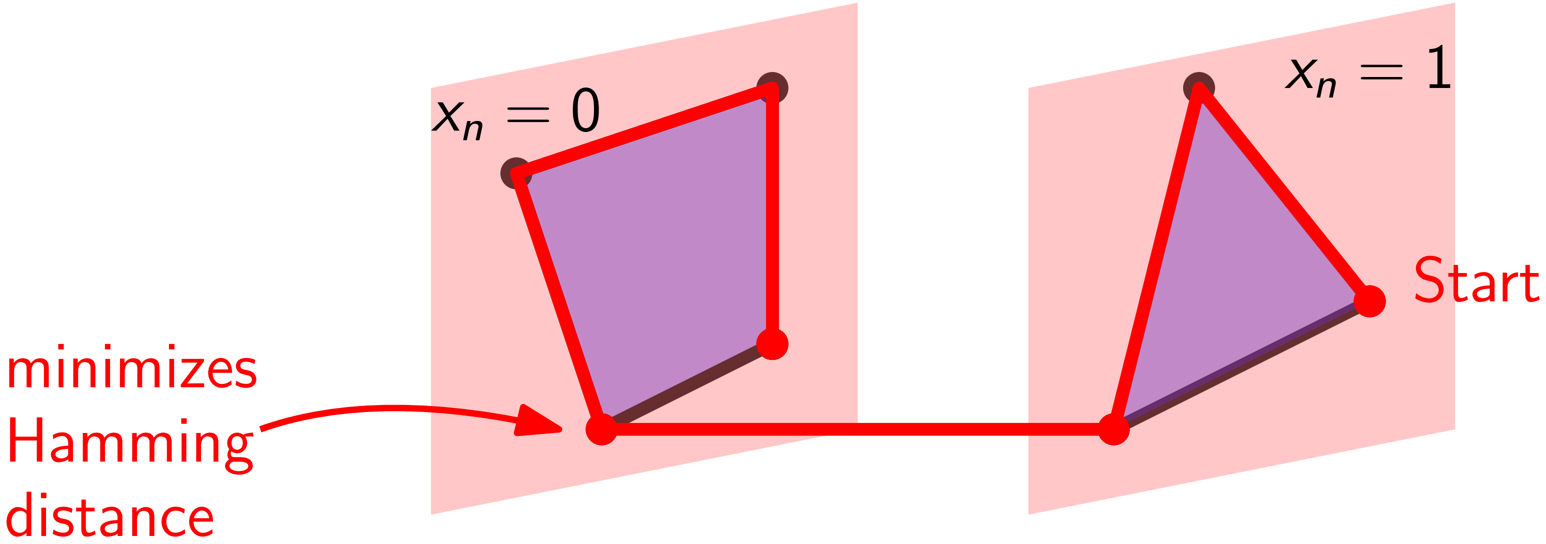
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



Lemma. If $x_n \neq y_n$ and $d_H(x, y)$ is minimized, among y with $y_n \neq x_n$, then xy is an edge of the polytope.

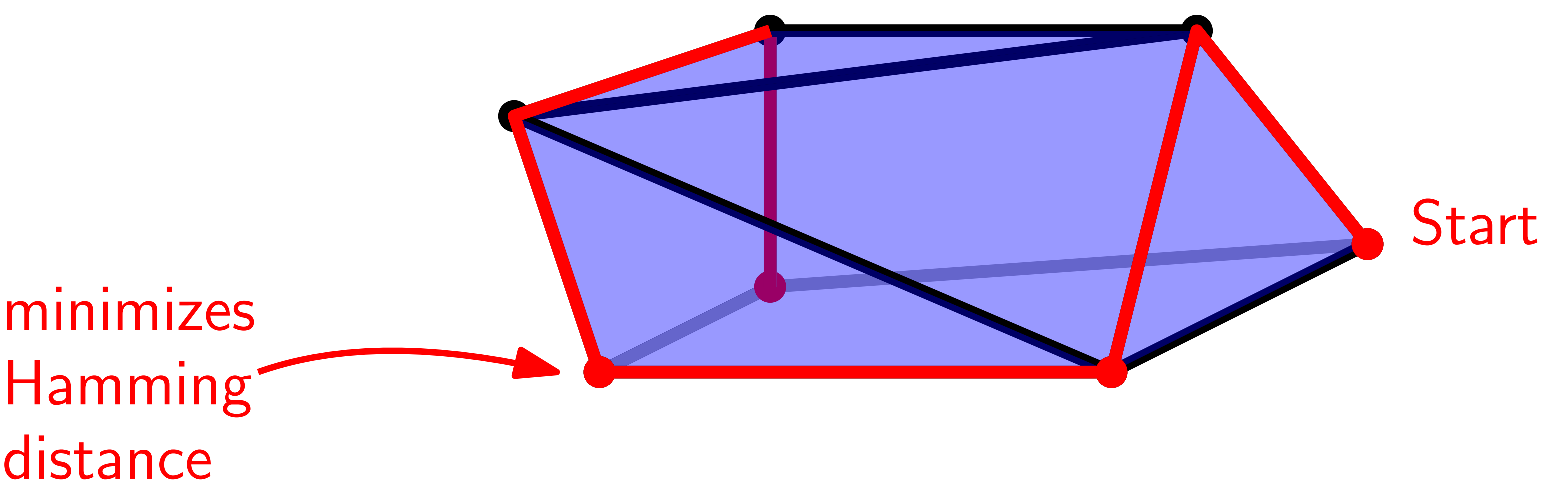
Why it works: Hamiltonicity of 0/1 polytopes

Spiritual Thm. [Naddef, Pulleyblank 84]

0/1 polytopes are *very* Hamiltonian.

Prop. For every $v \in \mathcal{X}$ there is a Hamilton path in $\mathcal{X}$ starting from v.

Pf.



Lemma. If $x_n \neq y_n$ and $d_H(x, y)$ is minimized, among y with $y_n \neq x_n$, then xy is an edge of the polytope.

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations

$$
\begin{array}{rccccc}
x_1 & 1 & 1 & 1 & 0 & 0 \\
x_2 & 1 & 0 & 1 & 1 & 0 \\
x_3 & 0 & 1 & 1 & 1 & 0 \\
x_4 & 1 & 1 & 0 & 1 & 0 \\
x_5 & 1 & 1 & 0 & 0 & 1 \\
x_6 & 0 & 1 & 1 & 0 & 1 \\
x_7 & 1 & 0 & 1 & 0 & 1 \\
x_8 & 1 & 0 & 0 & 1 & 1 \\
x_9 & 0 & 1 & 0 & 1 & 1 \\
x_{10} & 0 & 0 & 1 & 1 & 1
\end{array}
$$

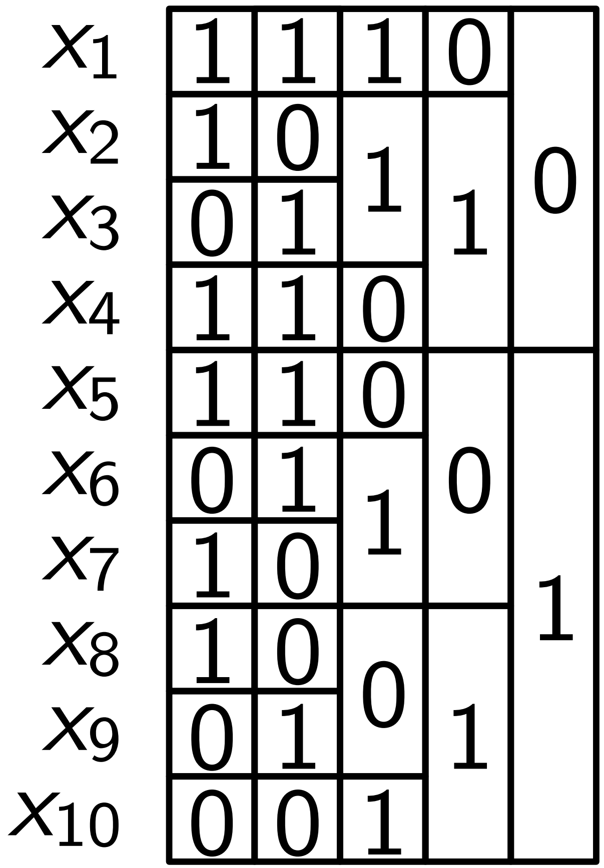
Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a “tree-like” structure.
- Example: (5,3)-combinations

x_1	1	1	1	0	0
x_2	1	0	1	1	0
x_3	0	1	1	1	0
x_4	1	1	0	1	0
x_5	1	1	0	0	1
x_6	0	1	1	0	1
x_7	1	0	1	0	1
x_8	1	0	0	1	1
x_9	0	1	0	1	1
x_{10}	0	0	1	1	1

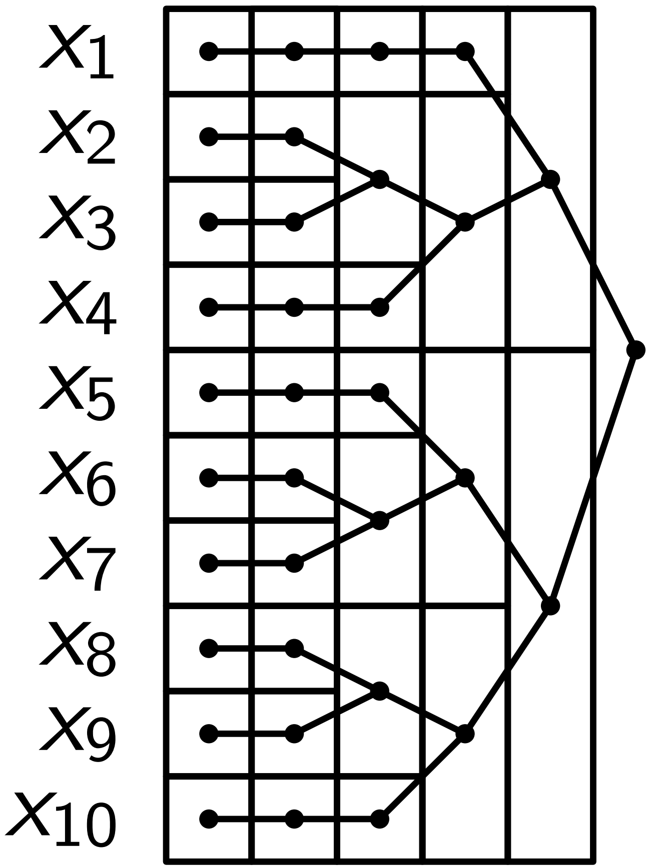
Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a “tree-like” structure.
- Example: (5,3)-combinations



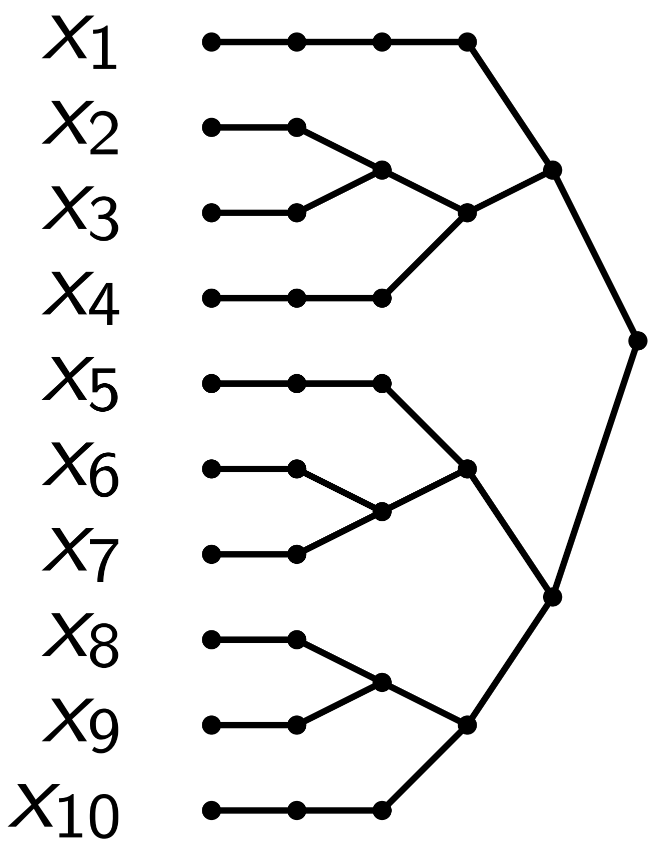
Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations



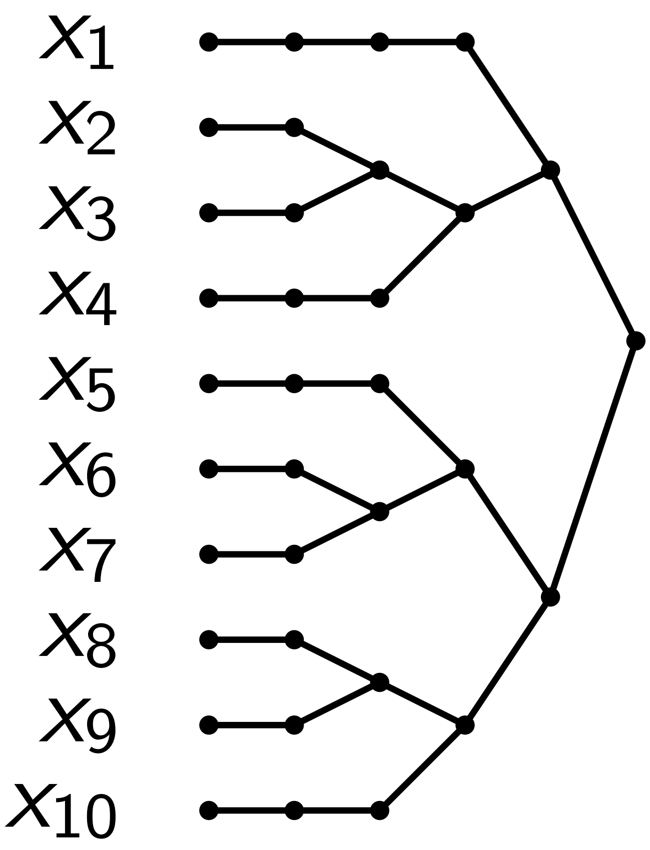
Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations



Why greedy?

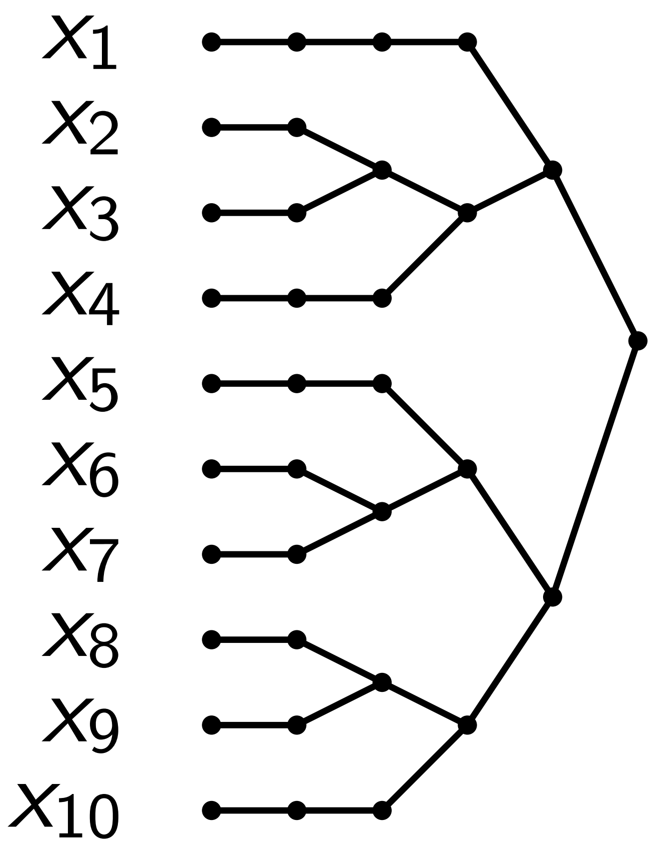
- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations



- ∘ $\mathcal{X}$ is the leaves of the tree.

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations

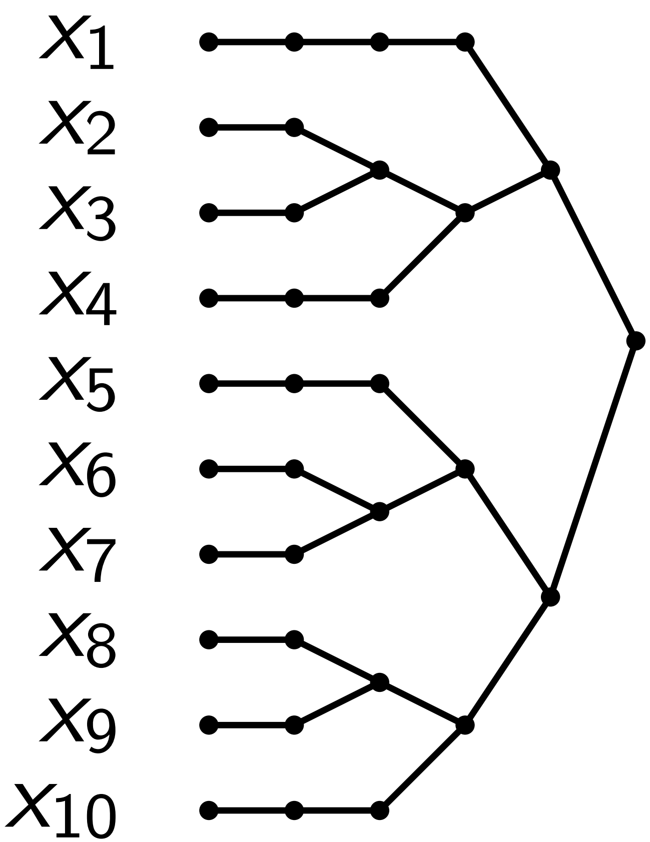


◦ $\mathcal{X}$ is the leaves of the tree.

◦ The tree "prioritizes" changes in the shortest prefix.

Why greedy?

- Hamilton paths where every suffix 0 and 1 appears consecutive.
- They have a "tree-like" structure.
- Example: (5,3)-combinations



◦ $\mathcal{X}$ is the leaves of the tree.

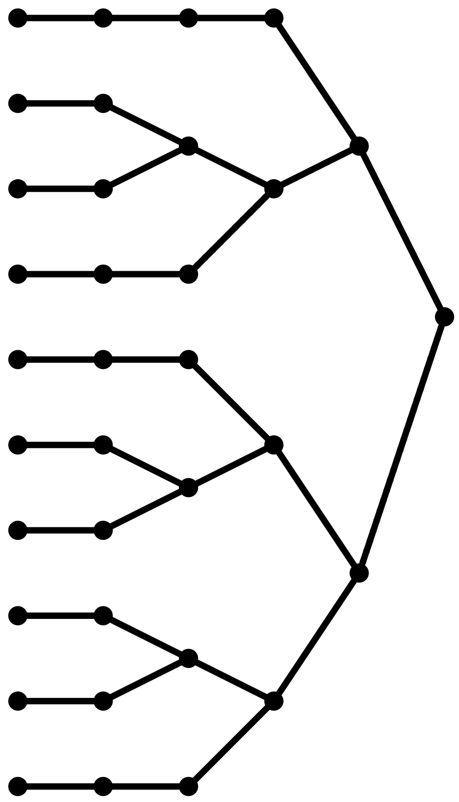
◦ The tree "prioritizes" changes in the shortest prefix.

◦ Our algorithm implictly traverses this tree

Why optimization?

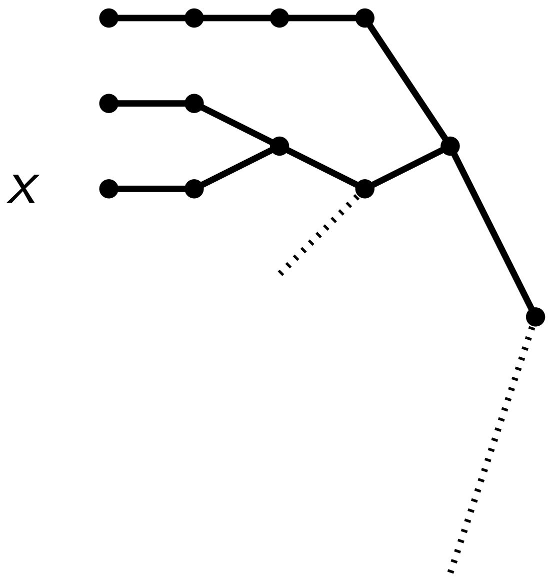
- What do we need to compute to proceed from x?

x



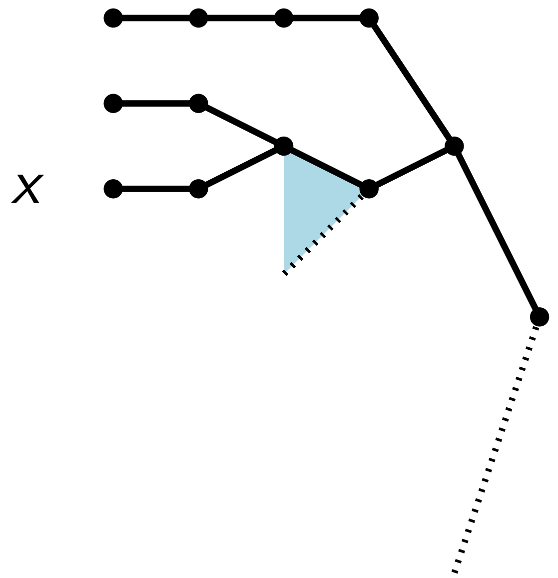
Why optimization?

- What do we need to compute to proceed from x?



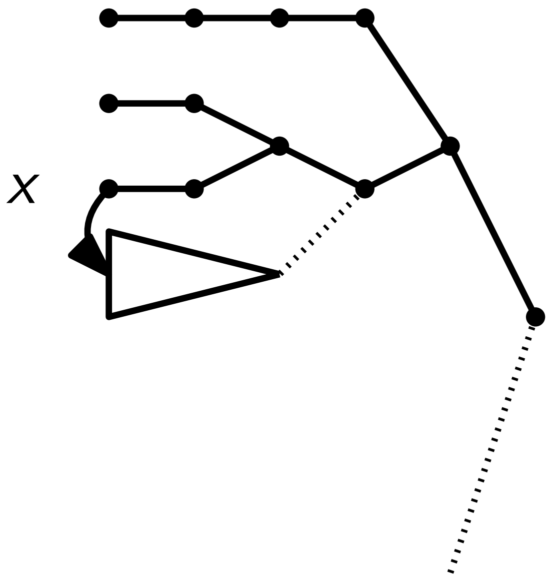
Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.



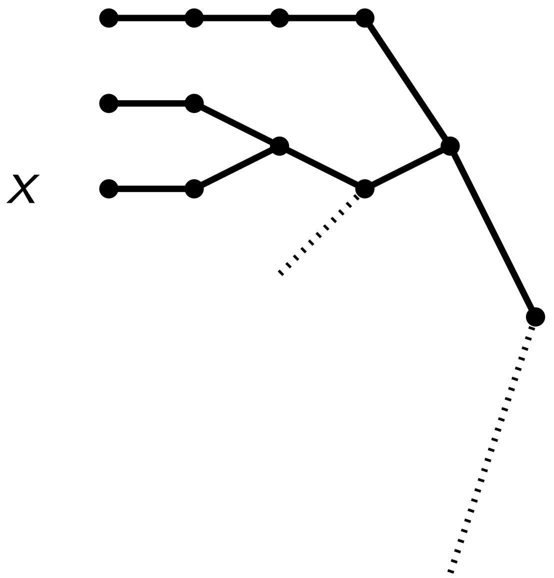
Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



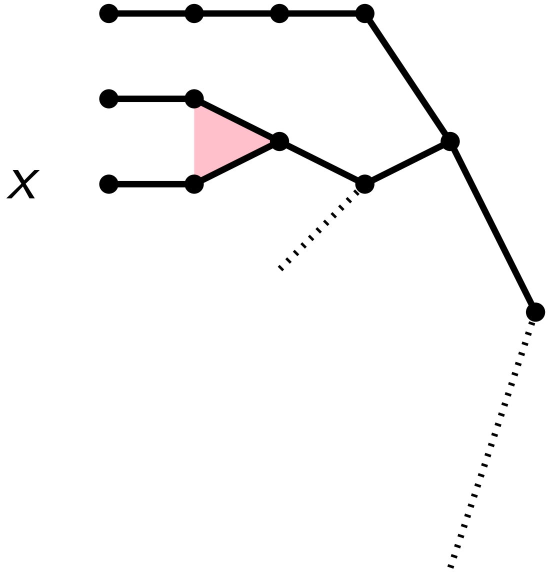
Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



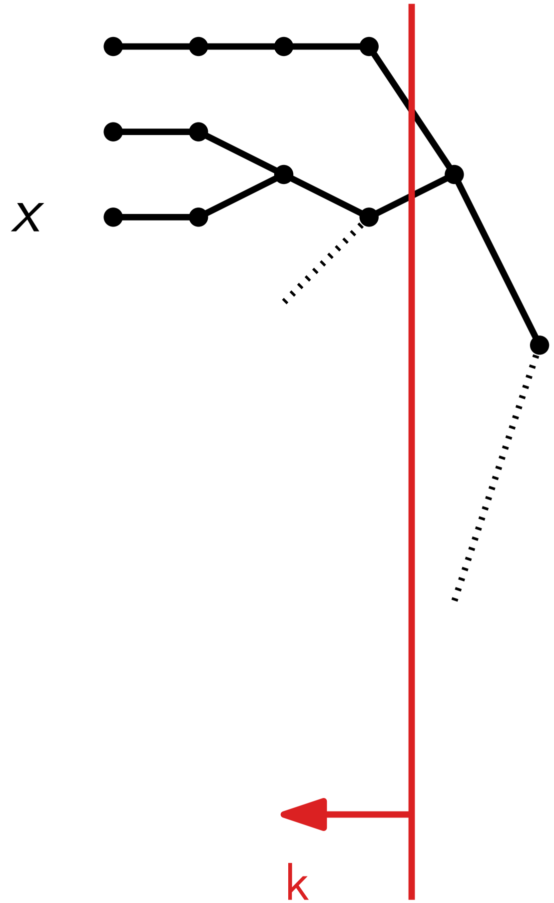
Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Why optimization?

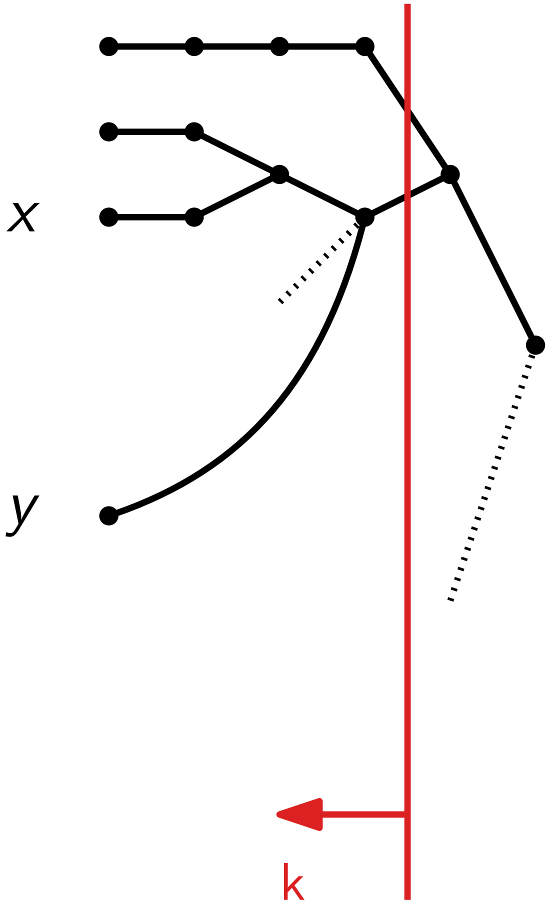
- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Branching at distance at least k from root

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



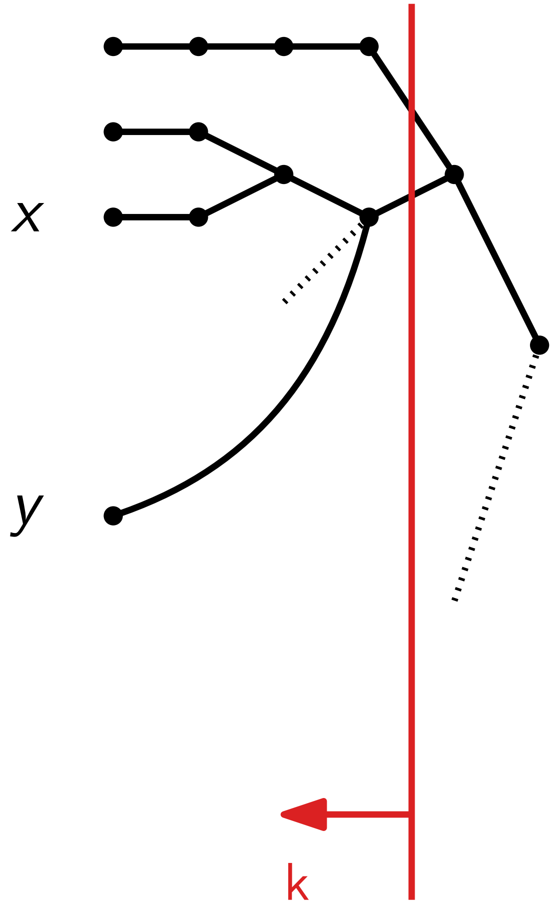
Branching at distance at least k from root

$$\iff$$

$\exists y \neq x$ with $x_{n-k} \dots x_n$ as suffix.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Branching at distance at least k from root

$$\Longleftrightarrow$$

$\exists y \neq x$ with $x_{n-k} \dots x_n$ as suffix.

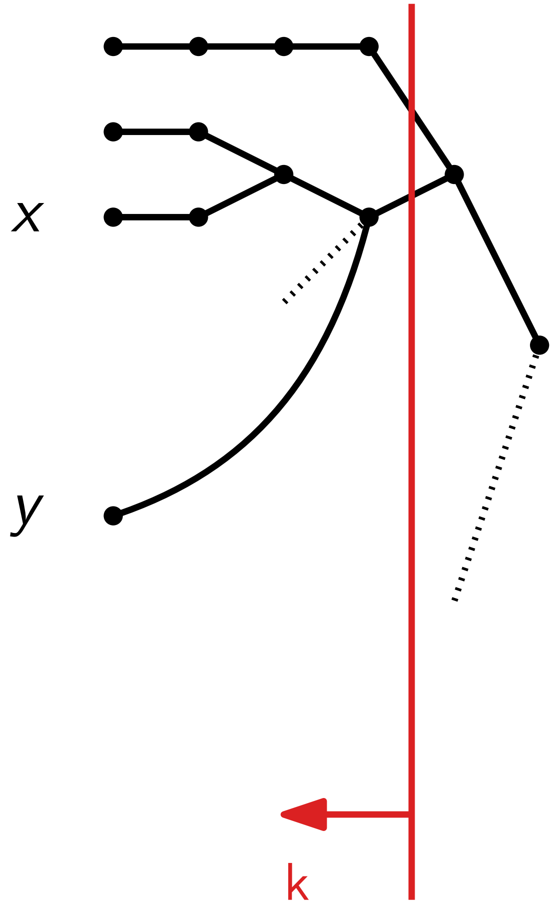
$$\Longleftrightarrow$$

x is not an optimal solution of

$$\max d_H(x, y)$$
$$s.t.\ y \text{ has } x_{n-k} \dots x_n \text{ as a suffix.}$$

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Branching at distance at least k from root

$$\Longleftrightarrow$$

$\exists y \neq x$ with $x_{n-k} \dots x_n$ as suffix.

$$\Longleftrightarrow$$

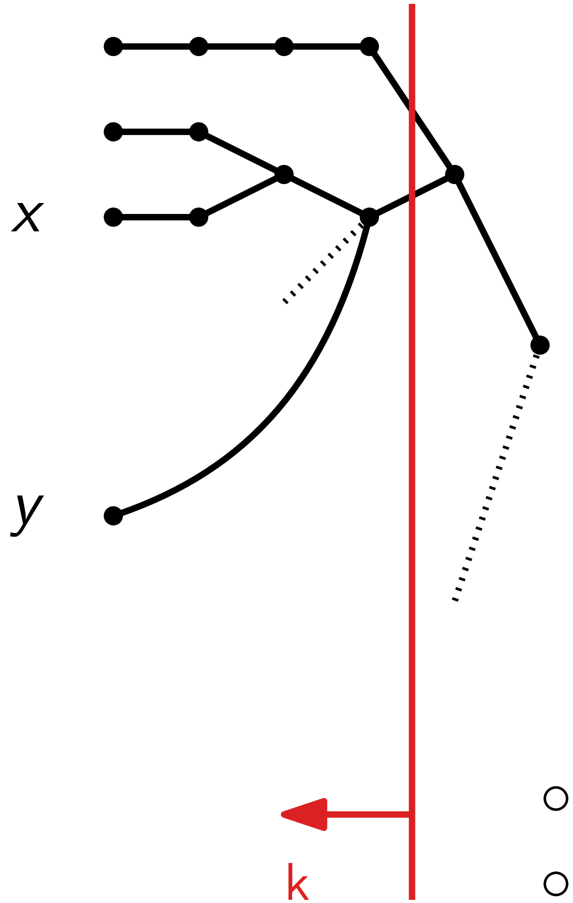
x is not an optimal solution of

$$\begin{aligned} &\max d_H(x, y) \\ &s.t.\ y \text{ has } x_{n-k} \dots x_n \text{ as a suffix.} \end{aligned}$$

- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Branching at distance at least k from root

$$\iff$$

$\exists y \neq x$ with $x_{n-k} \dots x_n$ as suffix.

$$\iff$$

x is not an optimal solution of

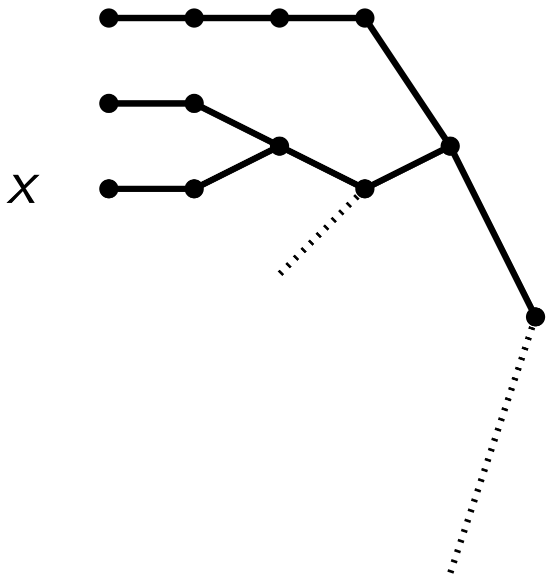
$$\max d_H(x, y)$$
$$s.t.\ y \text{ has } x_{n-k} \dots x_n \text{ as a suffix.}$$

  - Optimization problem!
  - Do binary search, we obtain $\mathcal{O}(T \operatorname{polylog} n)$ time.

- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

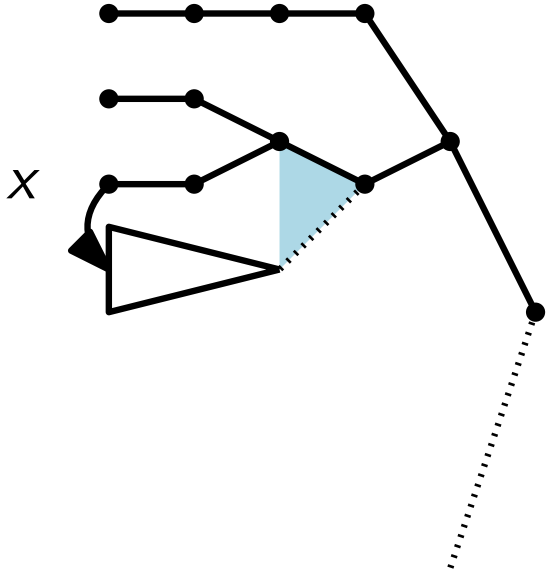
- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

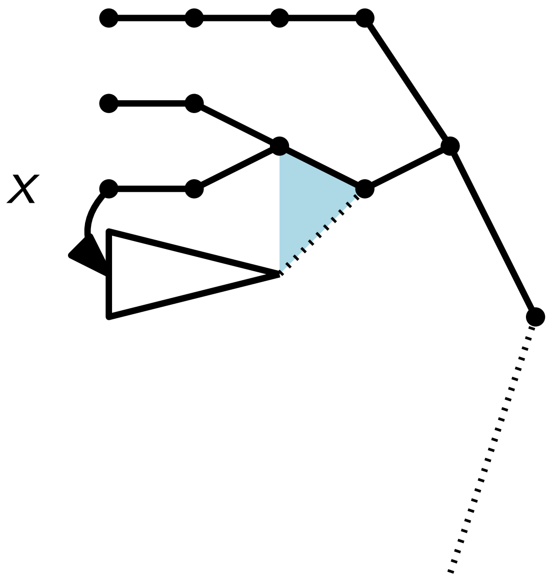
- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.

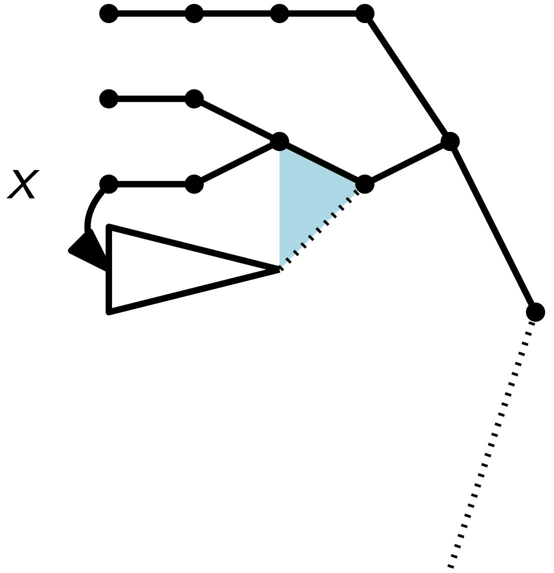


  - Being in a subtree prescribes a suffix.

- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.

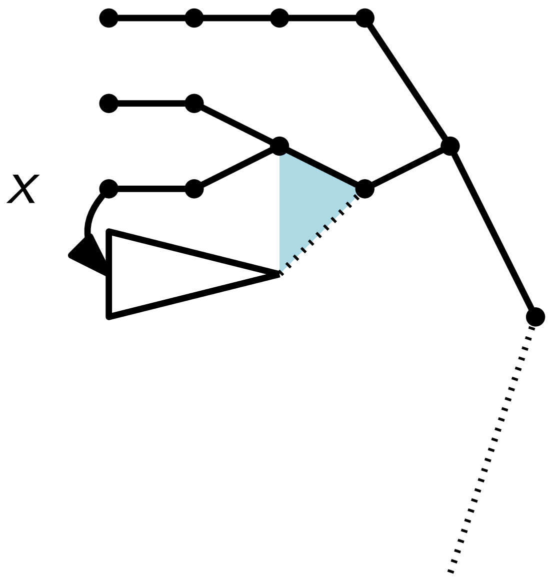


- Being in a subtree prescribes a suffix.
- Need to minimize $d_H(x, y)$ among them.

- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.

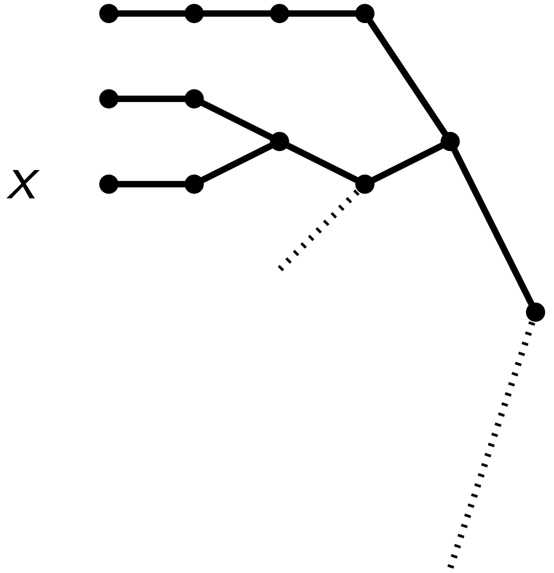


  - Being in a subtree prescribes a suffix.
  - Need to minimize $d_H(x, y)$ among them.
  - Just one more optimization step is needed.

- **Fact:** $d_H(x, y)$ is an affine function for *fixed* x.

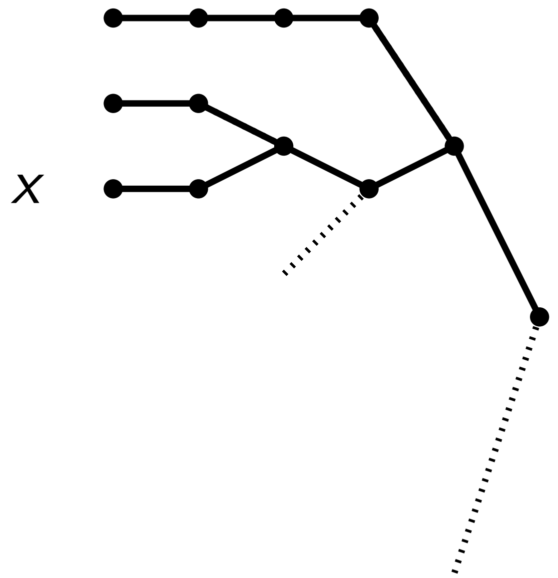
Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



Why optimization?

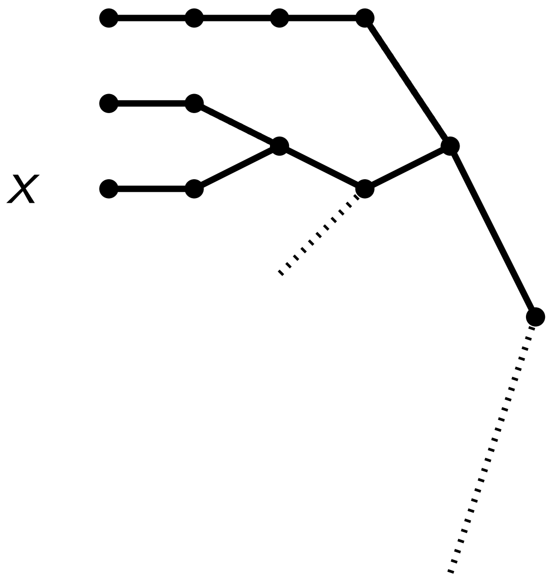
- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



- We obtain a runtime of $\mathcal{O}(T \operatorname{polylog} n)$.

Why optimization?

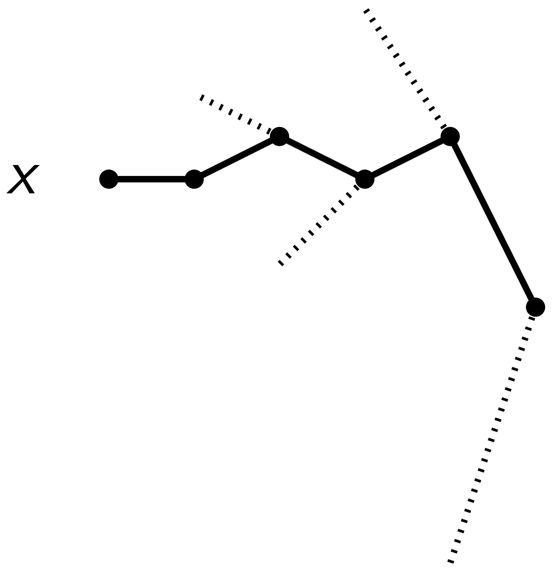
- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



- We obtain a runtime of $\mathcal{O}(T \operatorname{polylog} n)$.
- More tricky: guarantee that branching leads to something new + history-freeness.

Why optimization?

- What do we need to compute to proceed from x?
  - Leftmost branching that leads to something new.
  - A polytope edge towards its subtree.



- We obtain a runtime of $\mathcal{O}(T \operatorname{polylog} n)$.

- More tricky: guarantee that branching leads to something new + history-freeness.

Open questions

Open questions

- Better understanding of the black box.

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).

Open questions

- Better understanding of the black box.
    - Amortized analysis (cf. **[M, Mütze, Williams 22]**).

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** $\text{poly}(n, m)$.
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).

- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.

- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)

- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...
    - **This work:** Efficient optimization implies eff. Hamiltonicity.

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...
    - **This work:** Efficient optimization implies eff. Hamiltonicity.
    - Shortest path is sometimes very hard **[Cardinal, Steiner 23]**

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...
    - **This work:** Efficient optimization implies eff. Hamiltonicity.
    - Shortest path is sometimes very hard **[Cardinal, Steiner 23]**
  - Given all N vertices of P, sort them in Ham path order.

Open questions

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).
- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.
- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)
- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...
    - **This work:** Efficient optimization implies eff. Hamiltonicity.
    - Shortest path is sometimes very hard **[Cardinal, Steiner 23]**
  - Given all N vertices of P, sort them in Ham path order.
    - **This work:** $\mathcal{O}(nN^2)$ time algorithm. What about $\mathcal{O}(nN)$?

Open questions

Thanks for your attention!

- Better understanding of the black box.
  - Amortized analysis (cf. **[M, Mütze, Williams 22]**).

- $\mathcal{O}(T_{LP})$-delay vertex enumeration in 0/1 polytopes. **Crazy:** poly(n, m).
  - **This work:** $\mathcal{O}(\log n \cdot T_{LP})$-delay.

- Generalizations to non-binary $\mathcal{X}$.
  - Much more complicated case. (e.g., Hamiltonicity barrier)

- Additional Questions on 0/1 polytopes
  - Pancyclicity-type result?
  - Algorithmic questions for: diameter, shortest path, etc...
    - **This work:** Efficient optimization implies eff. Hamiltonicity.
    - Shortest path is sometimes very hard **[Cardinal, Steiner 23]**
  - Given all N vertices of P, sort them in Ham path order.
    - **This work:** $\mathcal{O}(nN^2)$ time algorithm. What about $\mathcal{O}(nN)$?