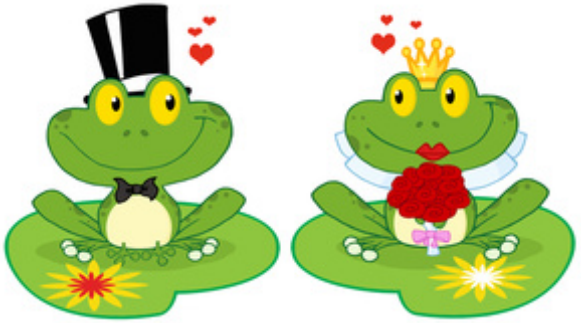
Algorithms for the discrete Fréchet distance under translation

by Omrit Filtser

Ben-Gurion University of the Negev

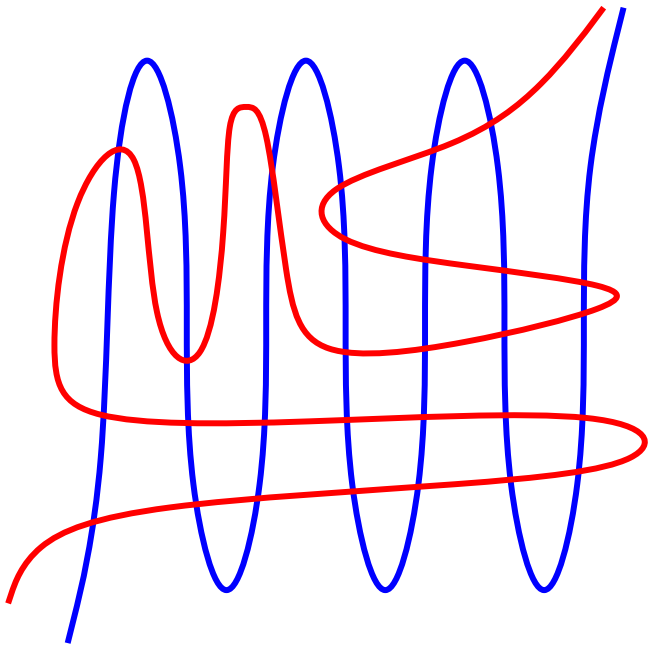


December 27, 2017

Based on joint work with Matthew J. Katz

Similarity of curevs

To what extent the two given curves resemble each other?



Similarity of curevs

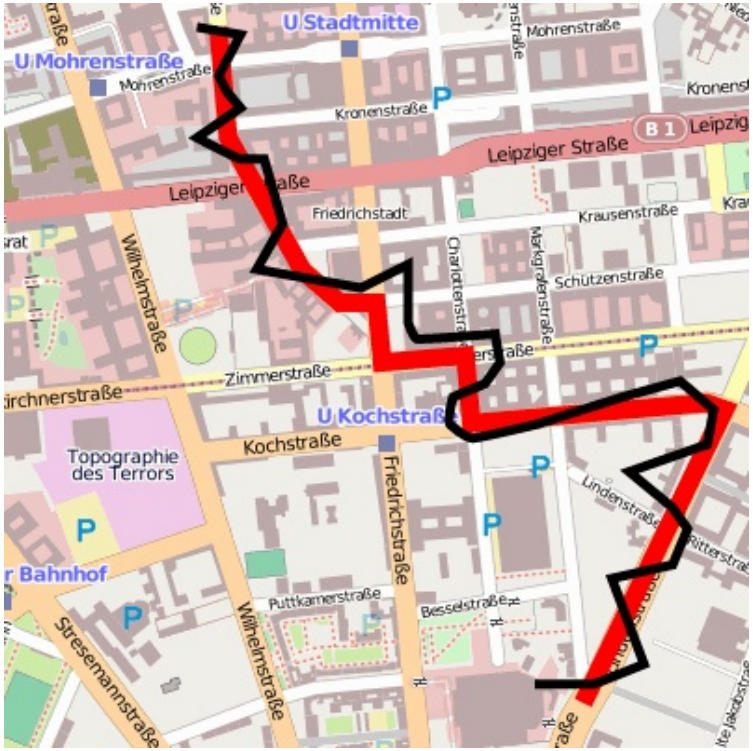
To what extent the two given curves resemble each other?

Applications: Signature verification

Similarity of curevs

To what extent the two given curves resemble each other?

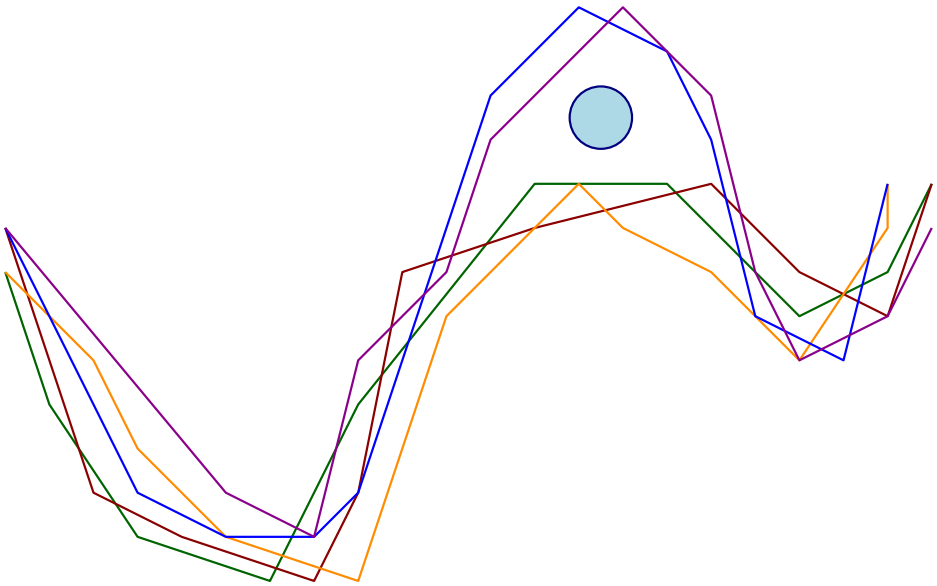
Applications: Map-matching of vehicle tracking data



Similarity of curevs

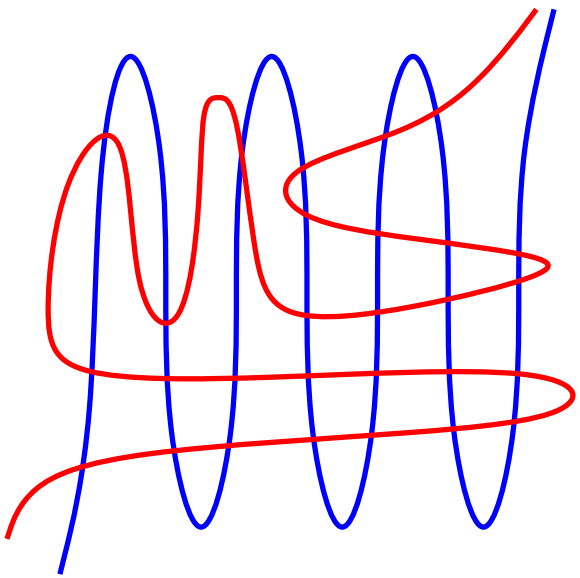
To what extent the two given curves resemble each other?

Applications: Analysis of moving objects



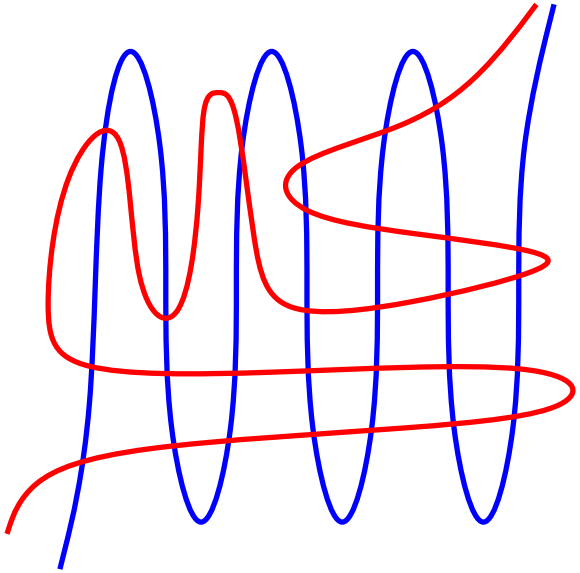
How to compare curves?

What distance measure between curves should be used?



How to compare curves?

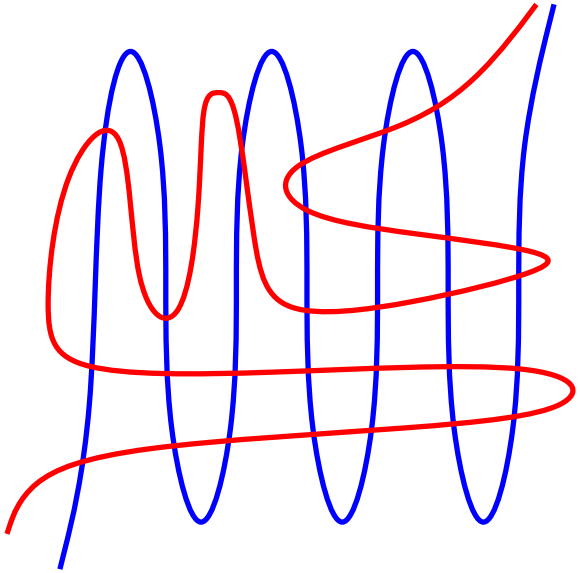
What distance measure between curves should be used?



- Hausdorff distance.
  $d_H = \max\{\sup_{a \in A} \inf_{b \in B} d(a,b), \sup_{b \in B} \inf_{a \in A} d(a,b)\}$

How to compare curves?

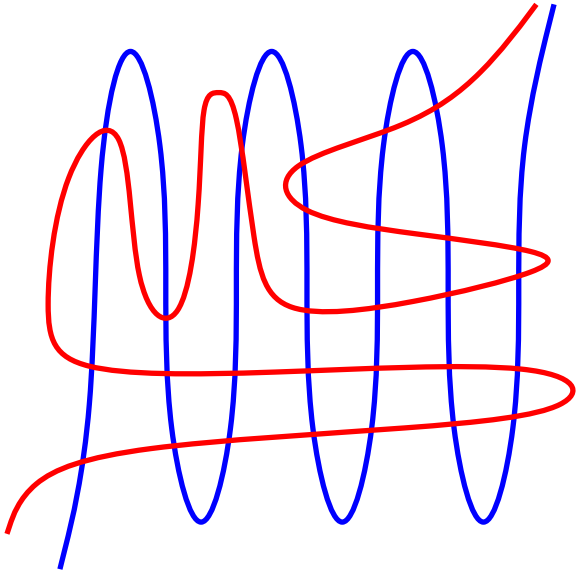
What distance measure between curves should be used?



- Hausdorff distance.
  $d_H = \max\{\sup_{a \in A} \inf_{b \in B} d(a,b), \sup_{b \in B} \inf_{a \in A} d(a,b)\}$
- × Only takes into account the sets of points but not the ordering.

How to compare curves?

What distance measure between curves should be used?



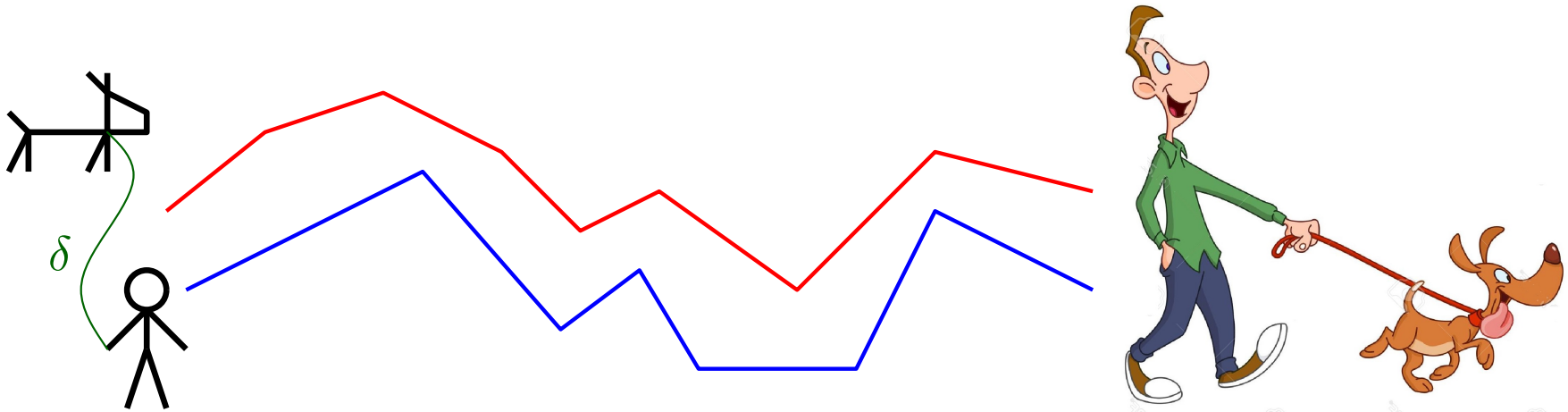
- Hausdorff distance.
  $d_H = \max\{\sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\}$

× Only takes into account the sets of points but not the ordering.

✓ Fréchet distance.

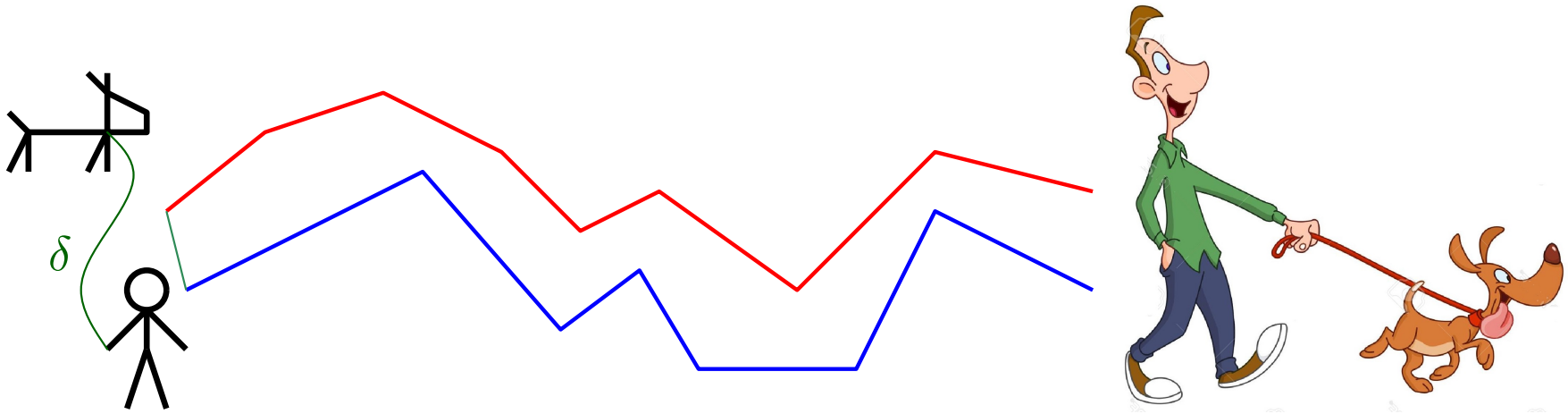
The (continuous) Fréchet distance

A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



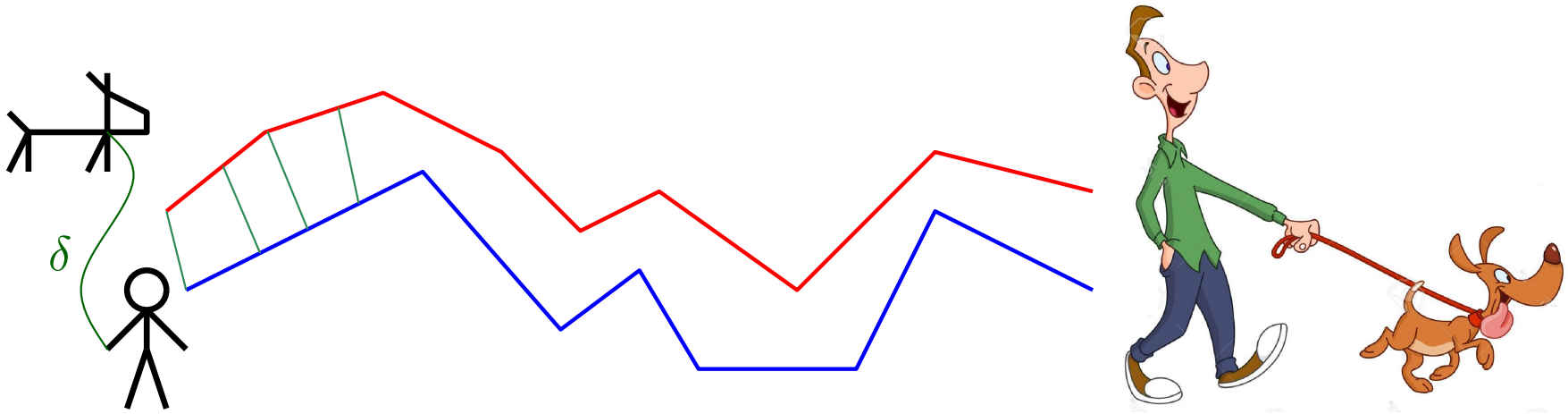
The (continuous) Fréchet distance

A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



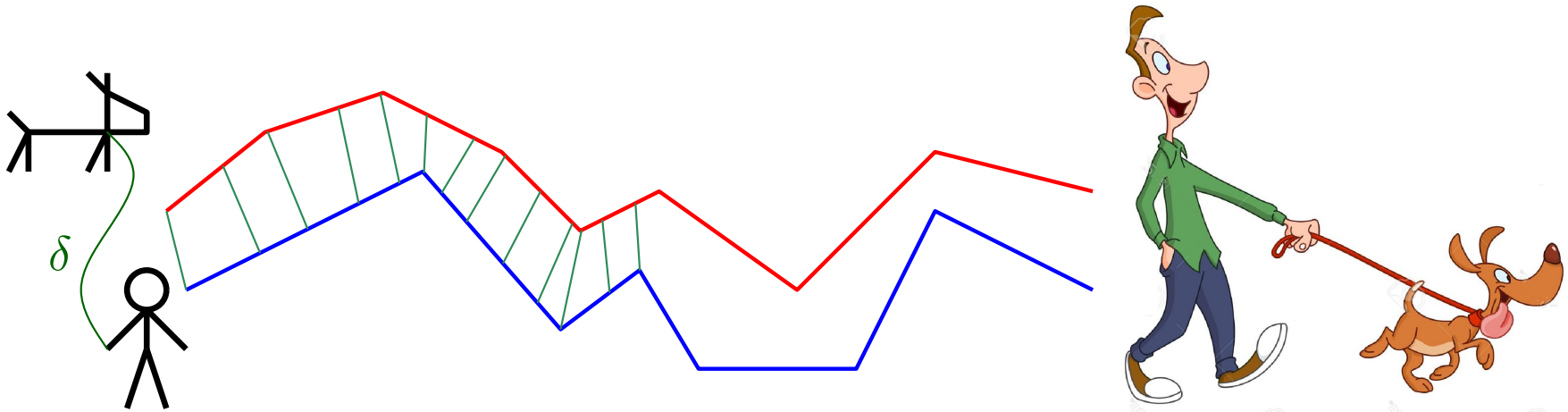
The (continuous) Fréchet distance

A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



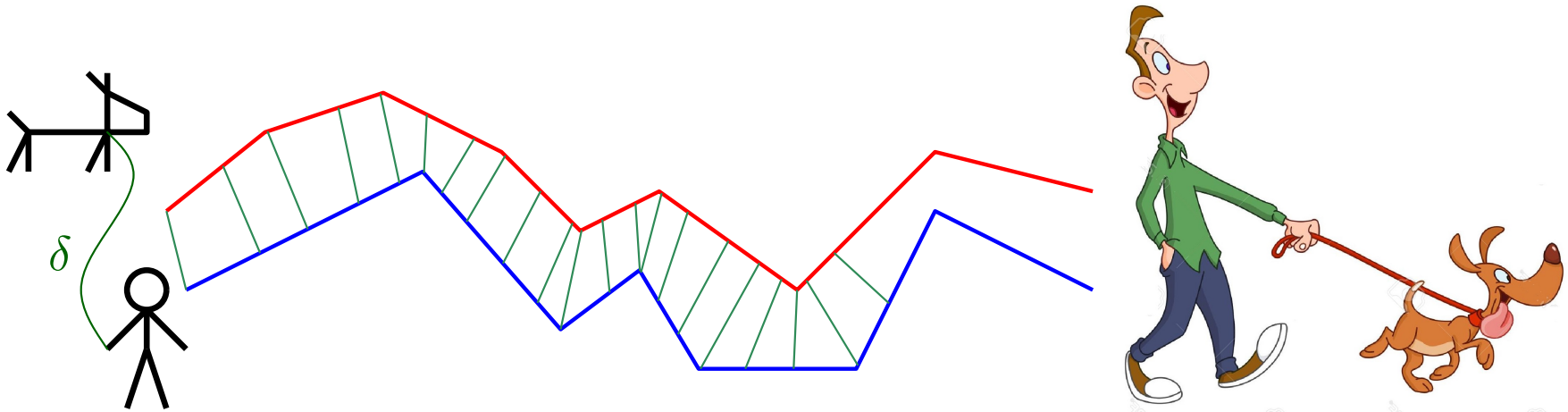
The (continuous) Fréchet distance

A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



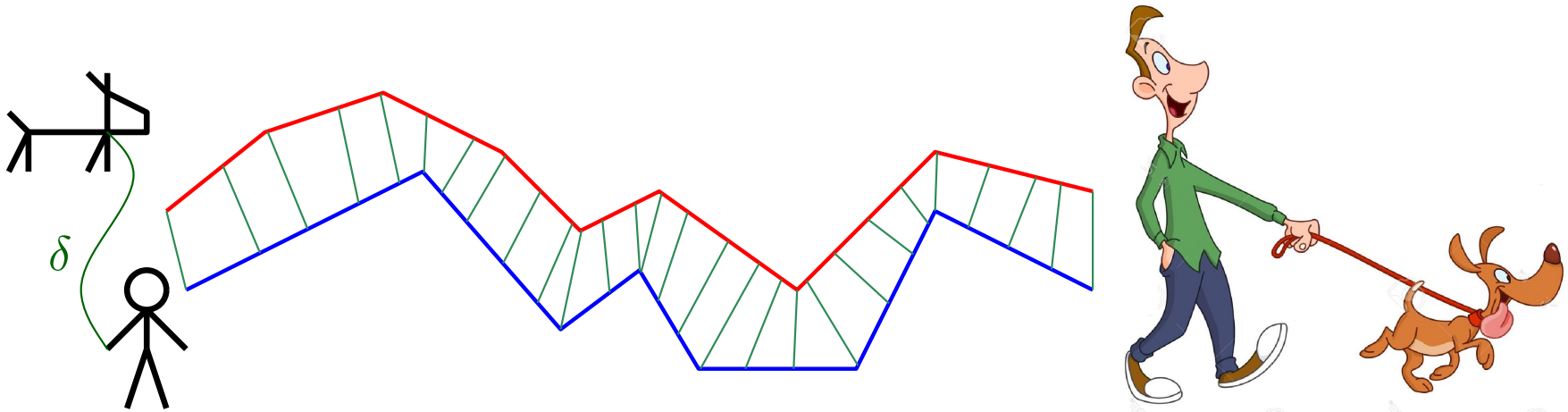
The (continuous) Fréchet distance

A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



The (continuous) Fréchet distance

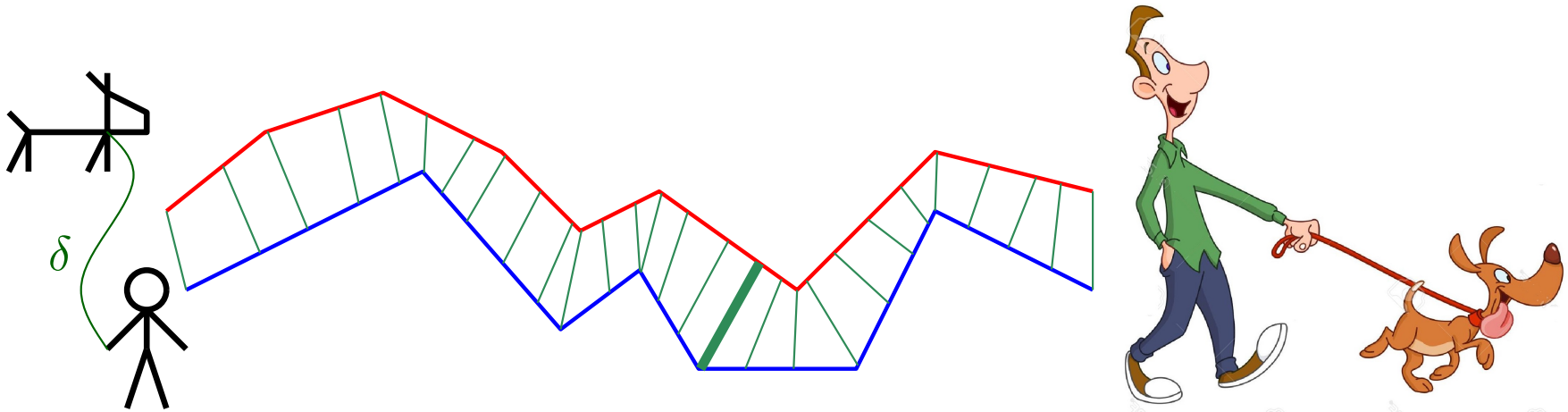
A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.



The (continuous) Fréchet distance

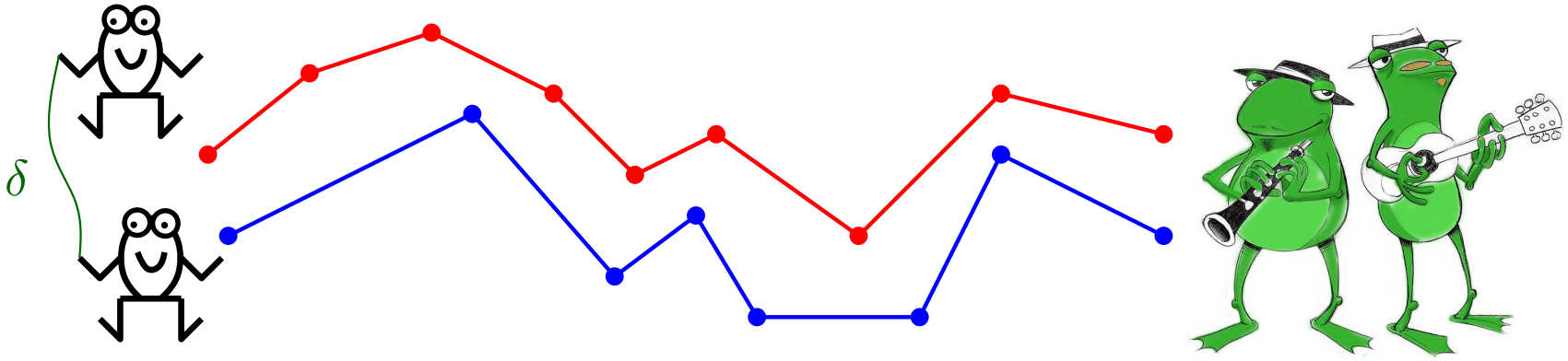
A person and a dog connected by a leash of length δ.
They walk along the curves A and B, respectively, no backtracking.

- The **Fréchet distance** ($d_F(A, B)$) is the **minimum** δ that is sufficient for traversing both curves in this manner.



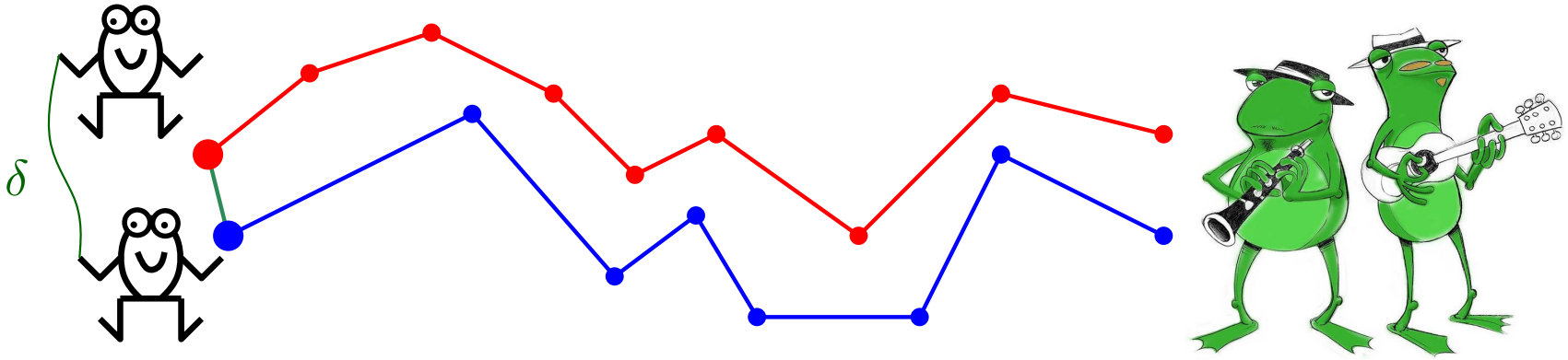
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



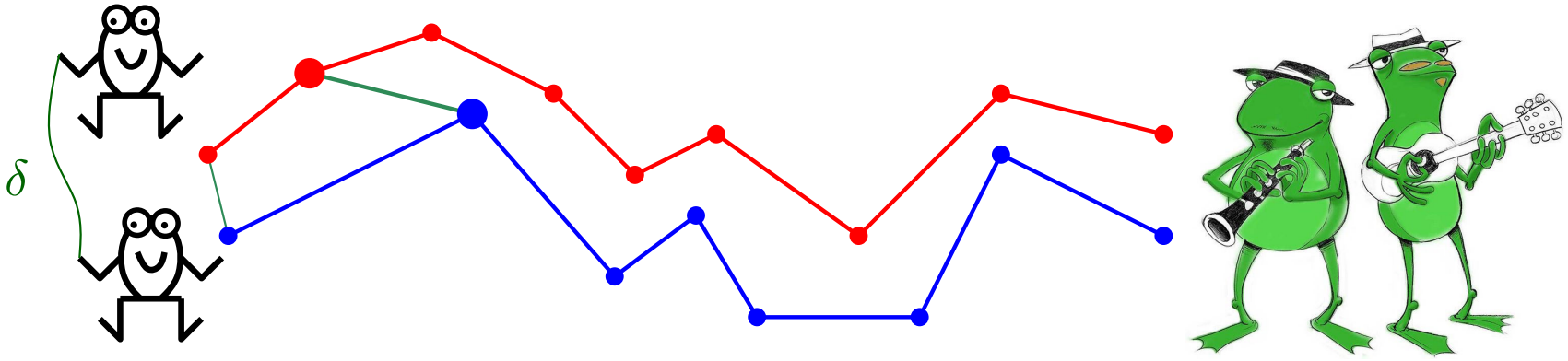
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



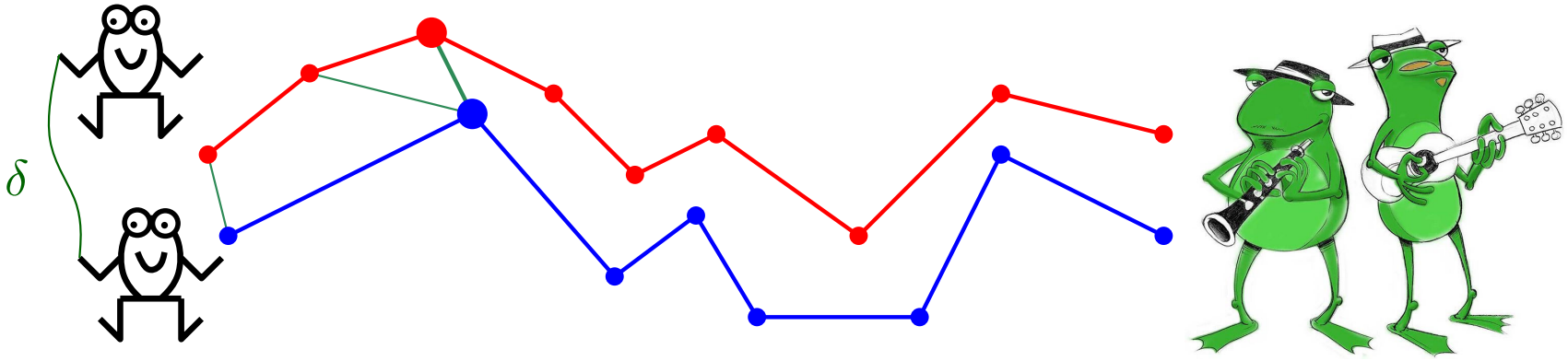
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



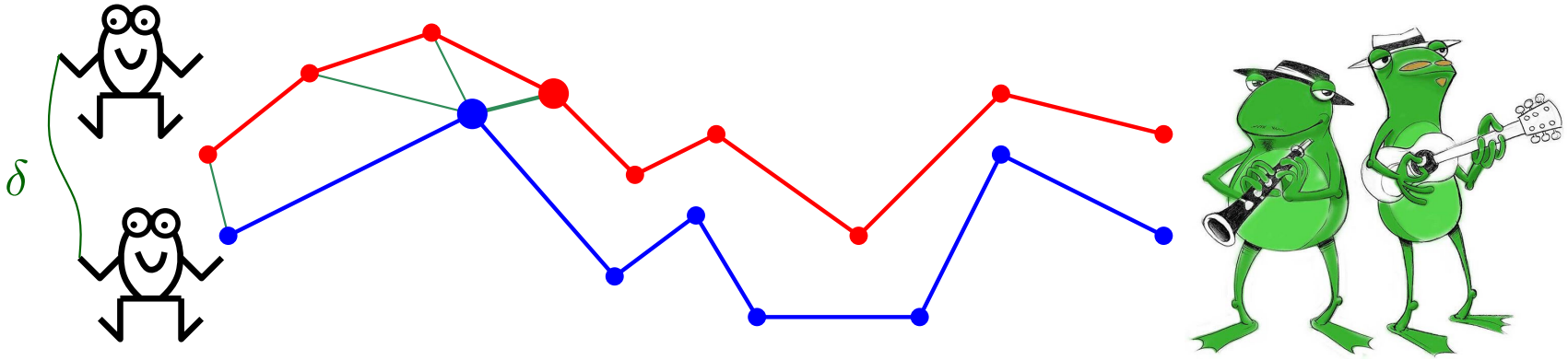
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



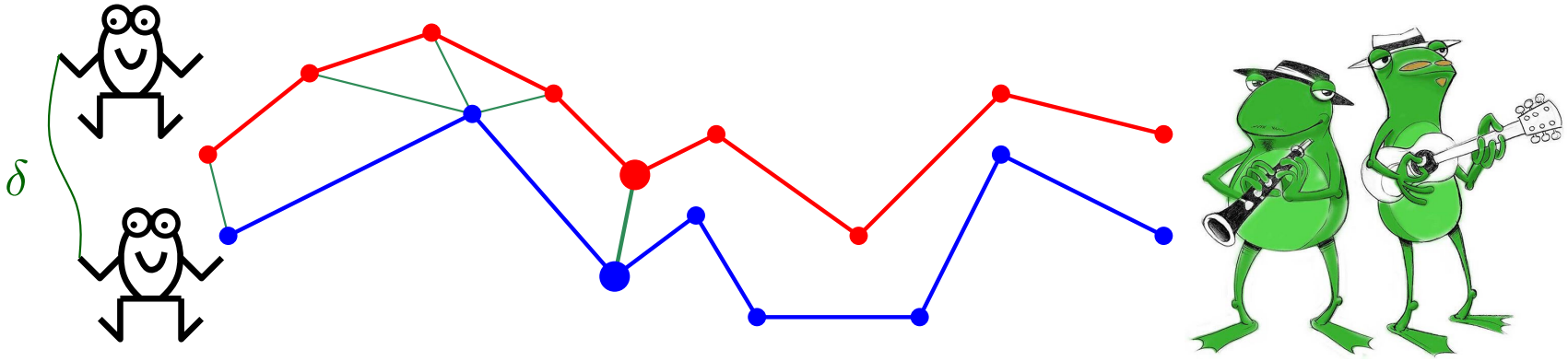
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



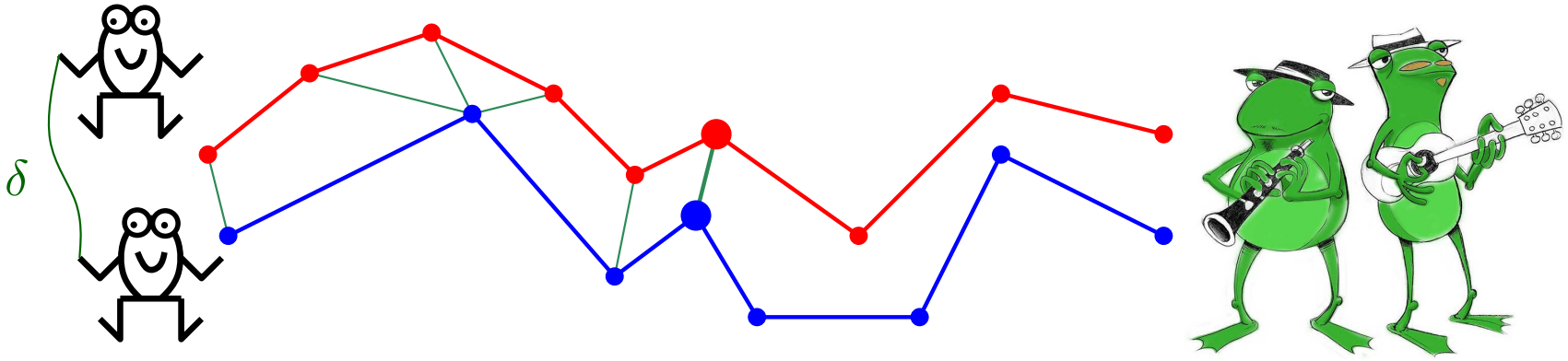
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



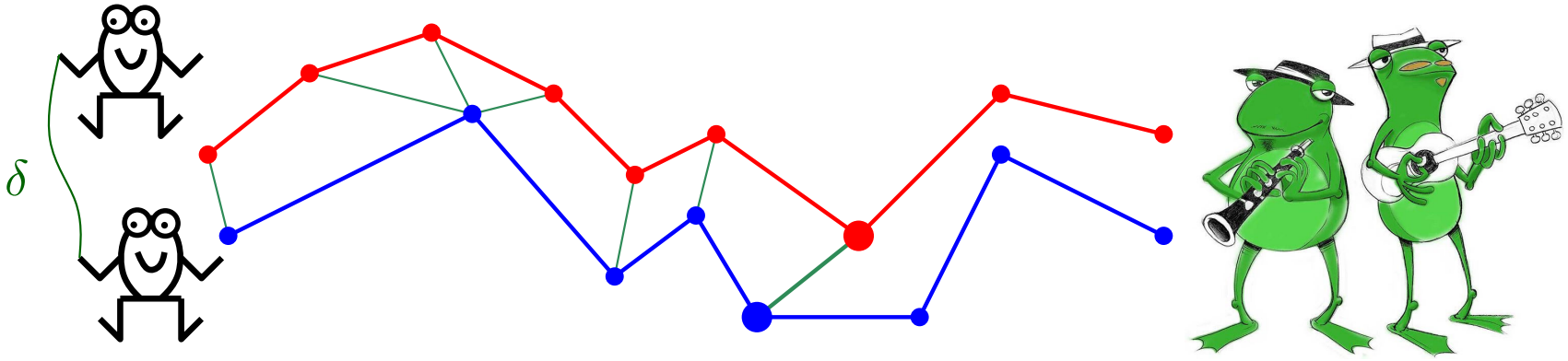
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



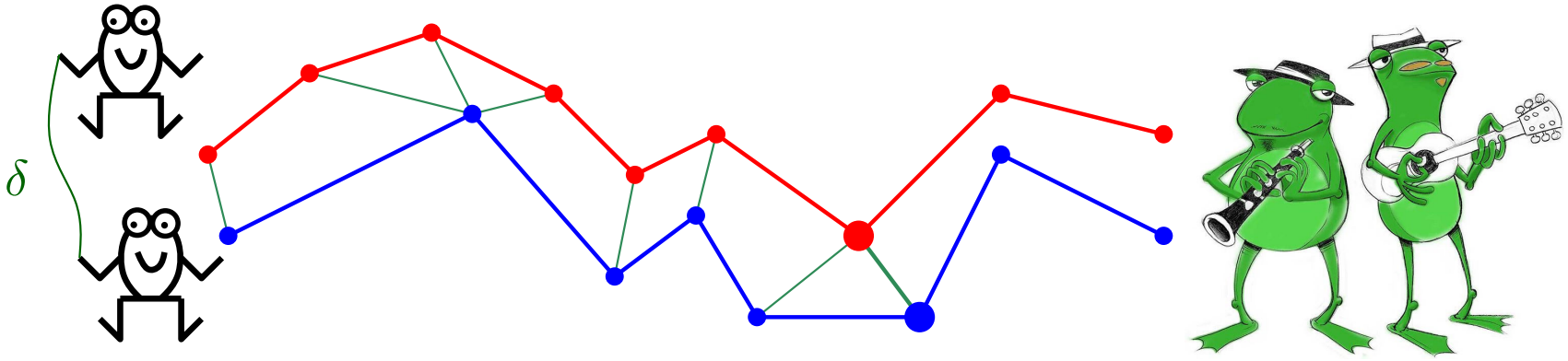
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



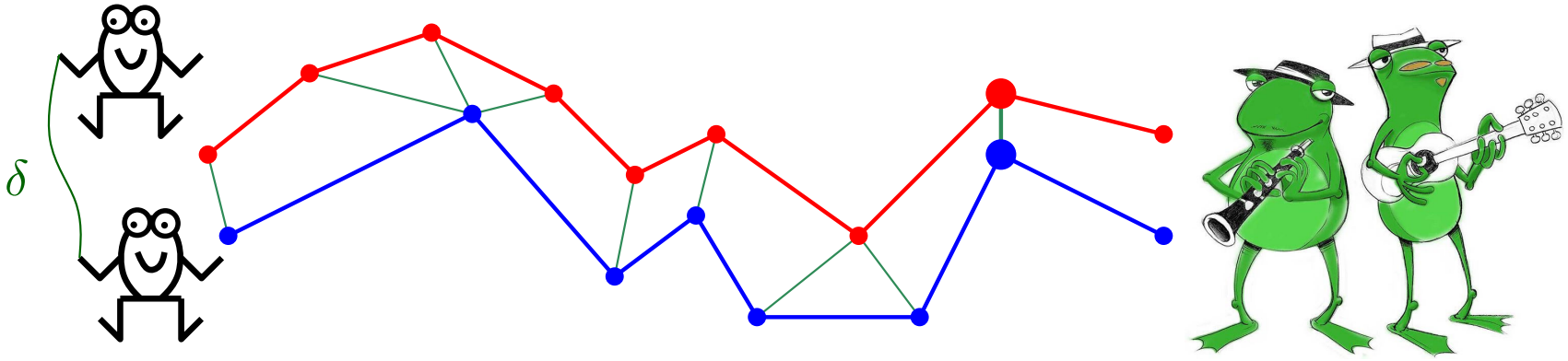
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



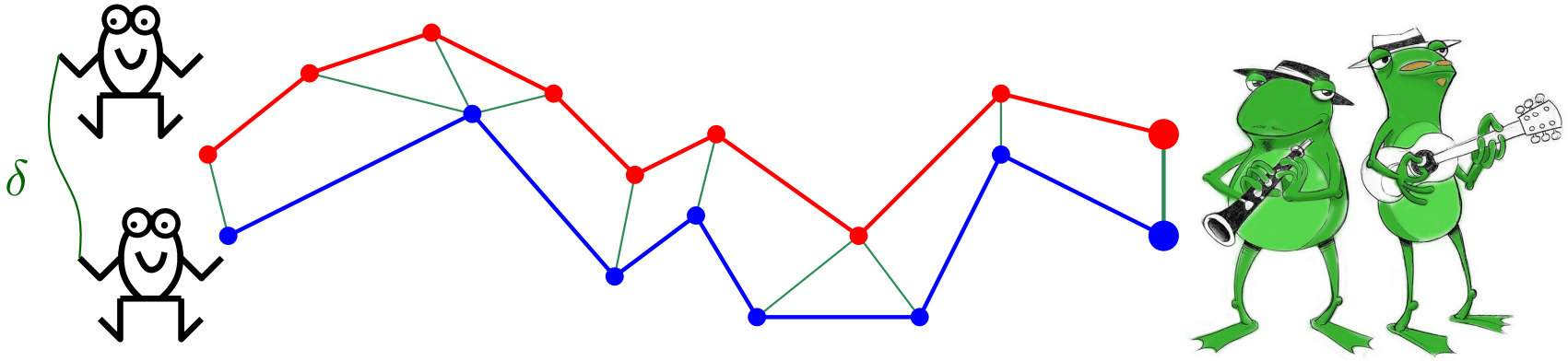
The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



The discrete Fréchet distance

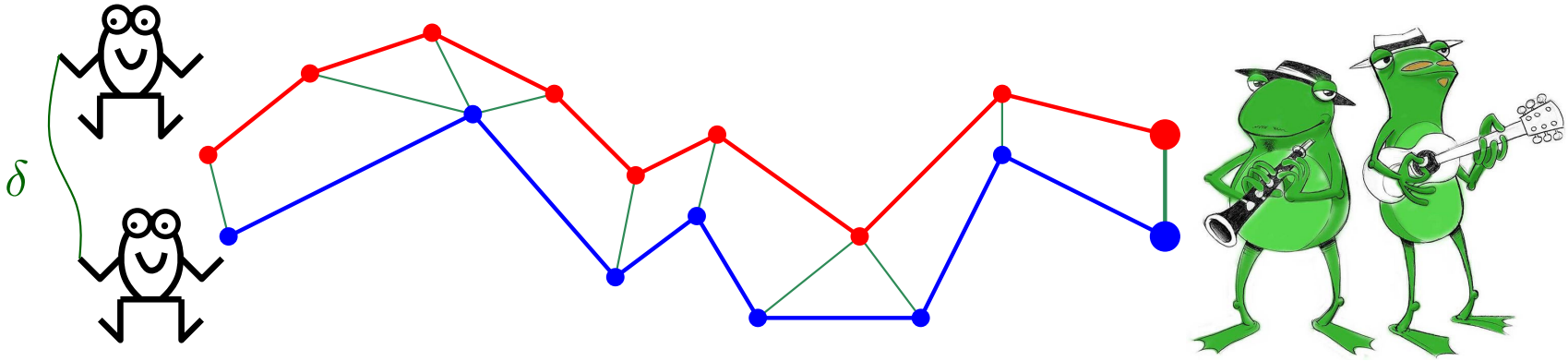
Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.



The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.

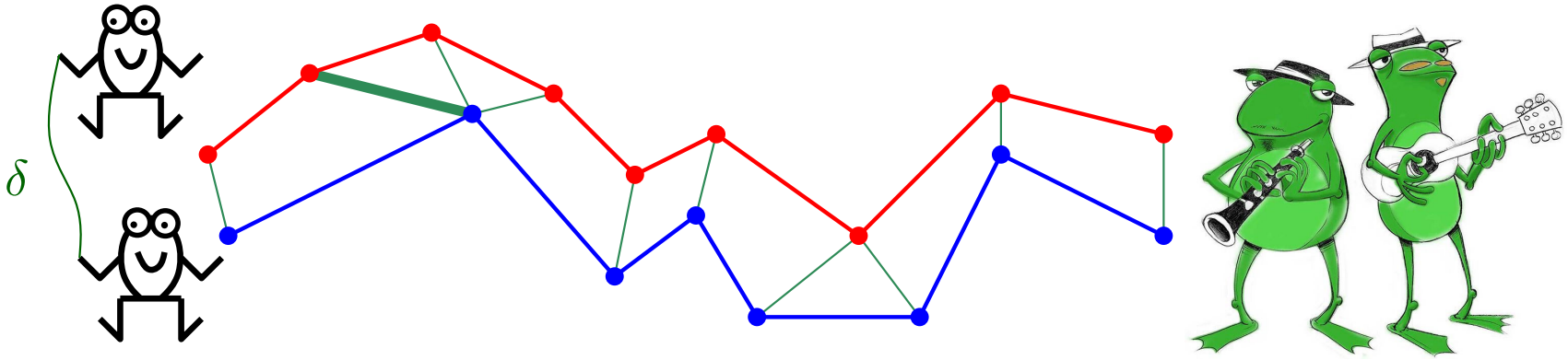
- The **discrete Fréchet distance** ($d_{dF}(A, B)$) is the **minimum** δ that allows the frogs to reach a_n and b_m.



The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.

- The **discrete Fréchet distance** ($d_{dF}(A, B)$) is the **minimum** δ that allows the frogs to reach a_n and b_m.

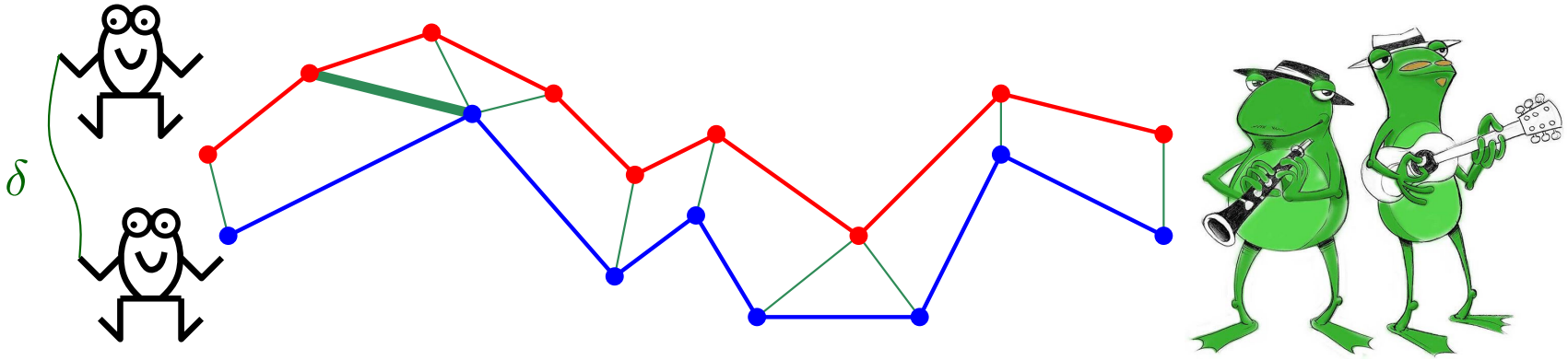


The discrete Fréchet distance

Two sequences of points, $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$.
Two frogs, the A-frog and the B-frog, connected by a leash of length δ, **hopping** along their respective sequences.

- The **discrete Fréchet distance** ($d_{dF}(A, B)$) is the **minimum** δ that allows the frogs to reach a_n and b_m.

- A good approximation of the continuous distance.
- Makes more sense in some situations (computational biology).

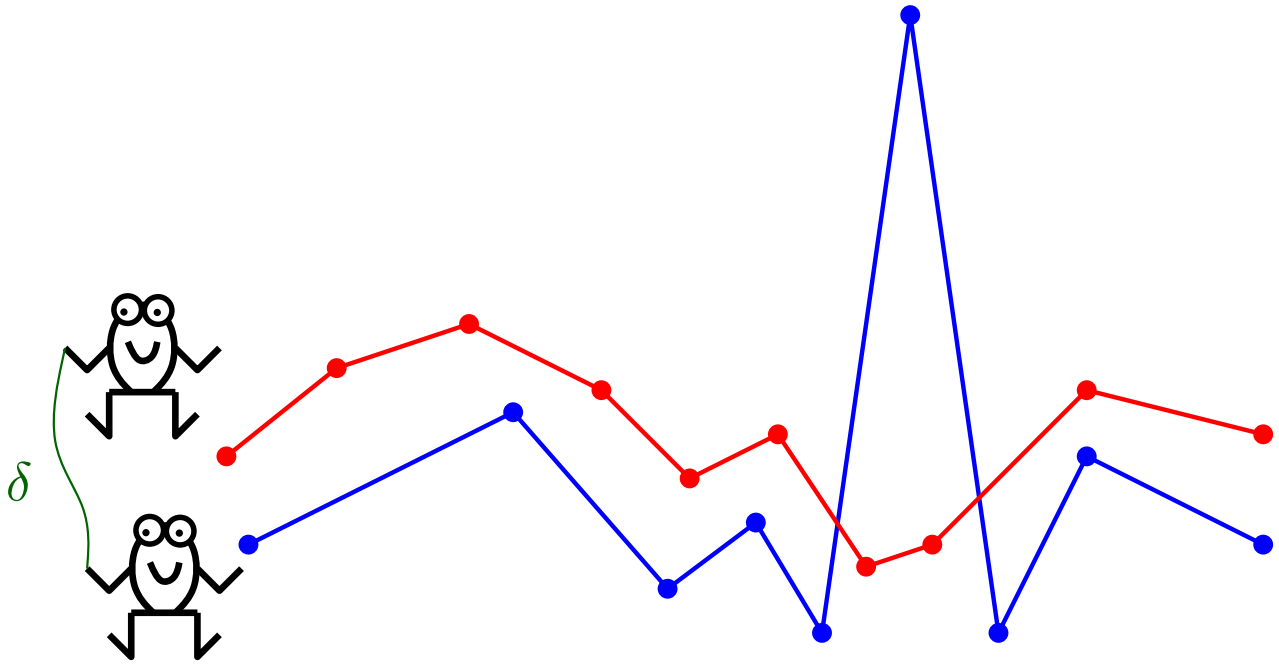


Related work

- **Eiter and Mannila ('94)** showed that $d_{dF}(A, B)$ can be computed in $O(n^2)$ time.
- **Agarwal et al. ('13)** showed how to compute $d_{dF}(A, B)$ in $O\left(\frac{n^2 \log \log n}{\log n}\right)$ time.
- **Bringmann and Mulzer ('15)** presented a conditional lower bound that **strongly subquadratic algorithms** for the discrete Fréchet distance **are unlikely to exist**, even in the one-dimensional case and even if the solution may be approximated up to a factor of 1.399.

Handling outliers

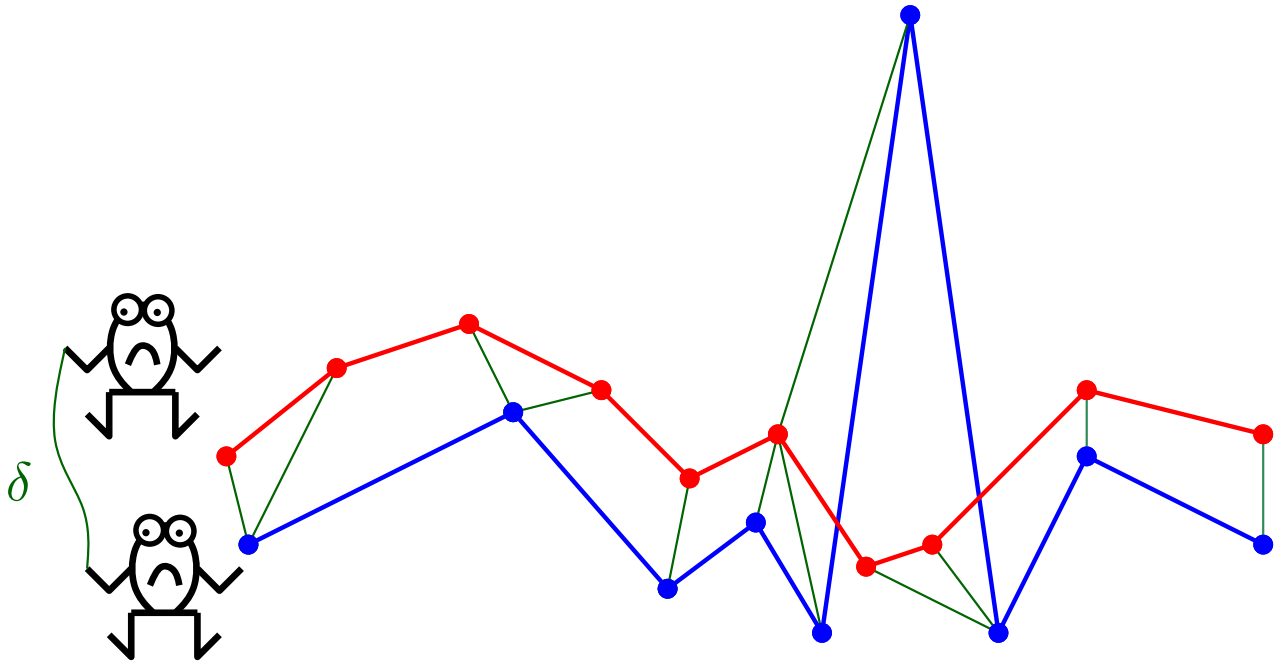
Physical sensors, such as GPS, may generate inaccurate measurements, which we refer to as **outliers**.



Handling outliers

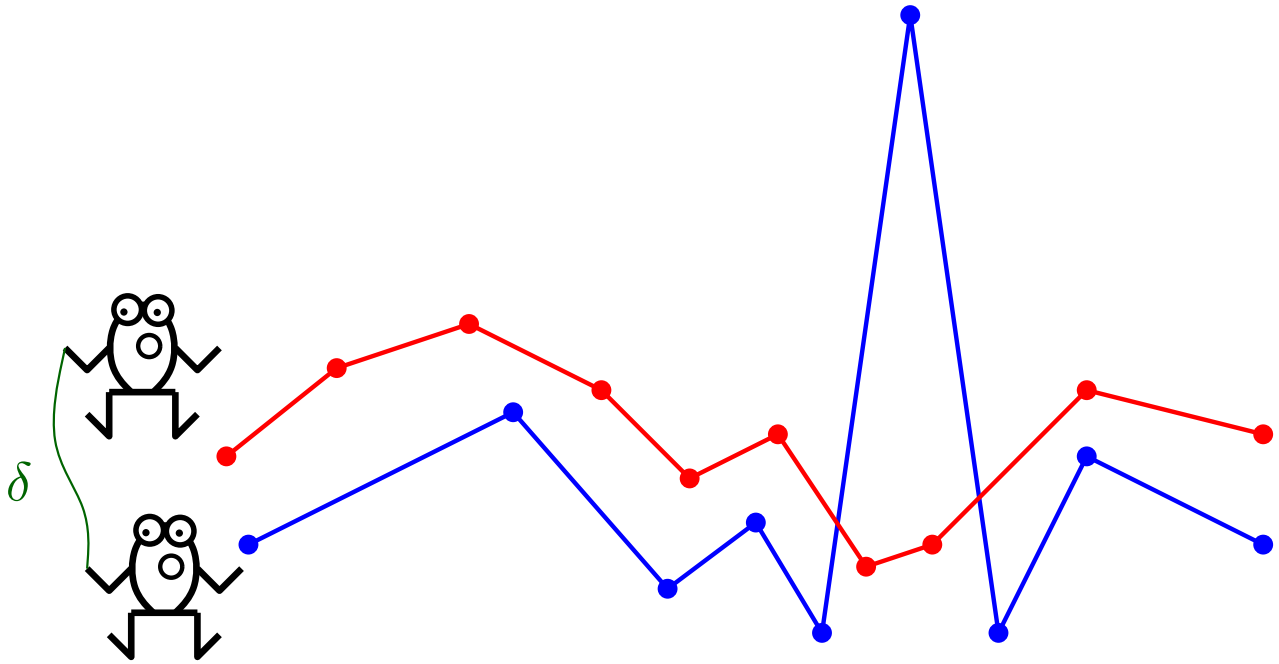
Physical sensors, such as GPS, may generate inaccurate measurements, which we refer to as **outliers**.

- The Fréchet distance and the discrete Fréchet distance are sensitive to outliers.



Handling outliers

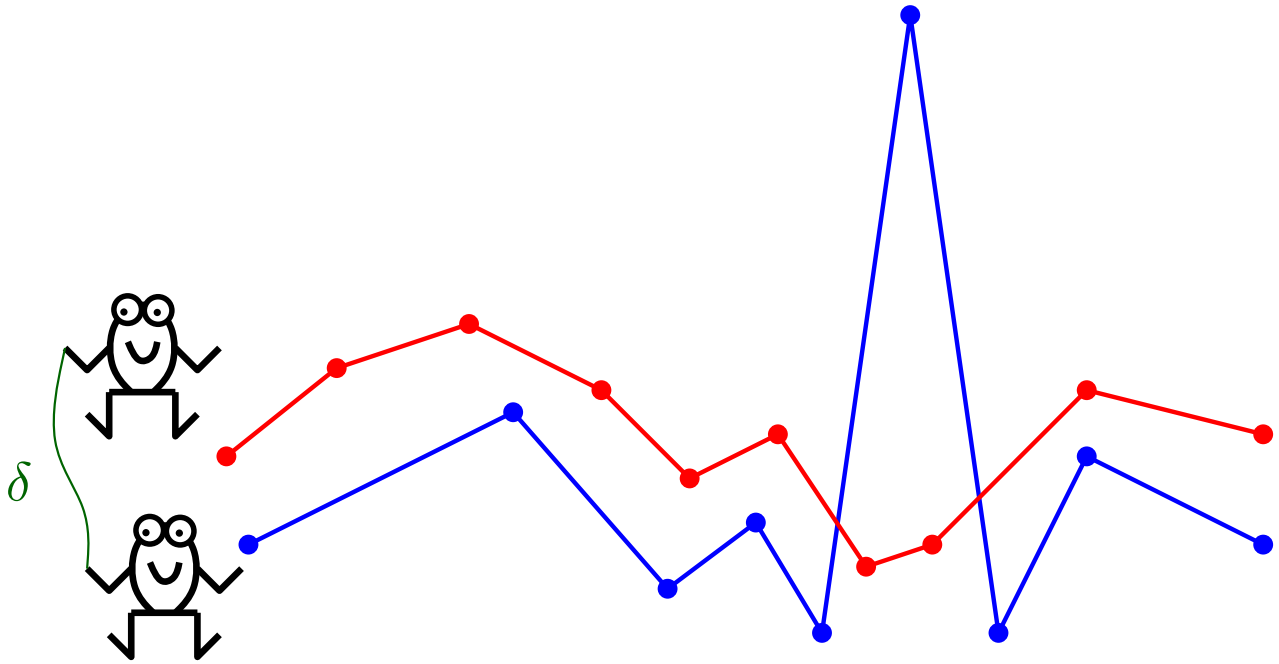
How to reduce sensitivity to outliers?



Handling outliers

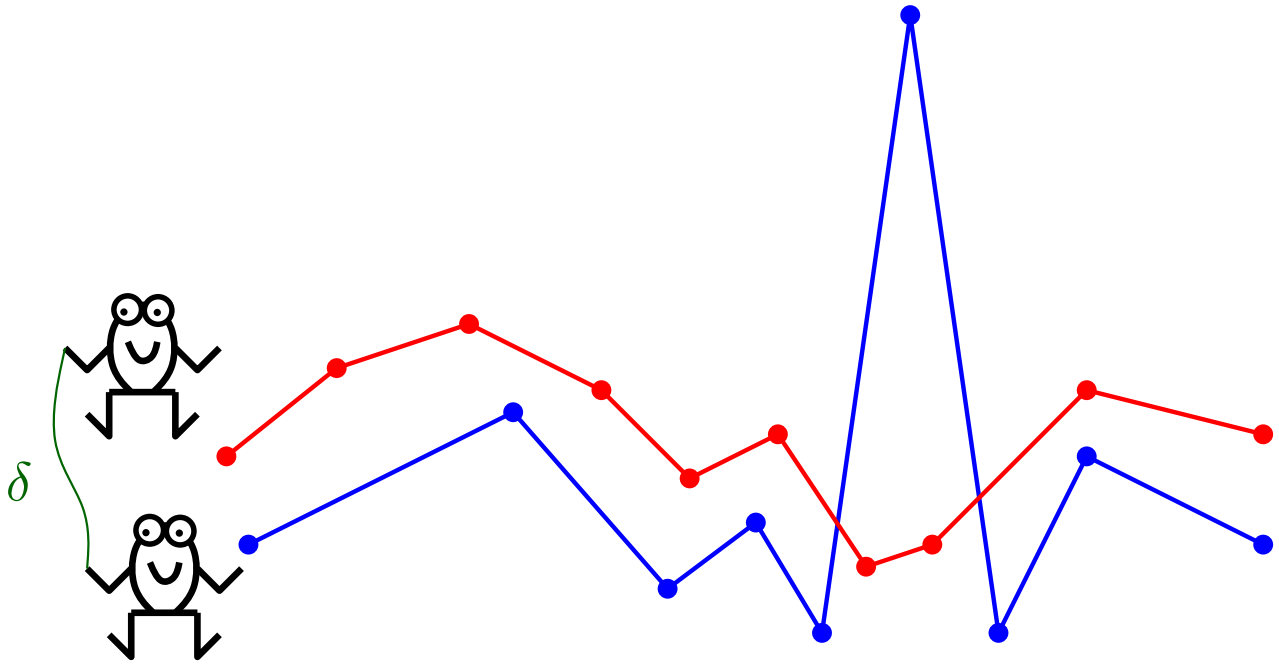
How to reduce sensitivity to outliers?
Take **shortcuts**!

- Allow the A-frog or the B-frog (or both) to "ignore" subcurves or points which might be considered as noise.



The one-sided discrete Fréchet distance with shortcuts

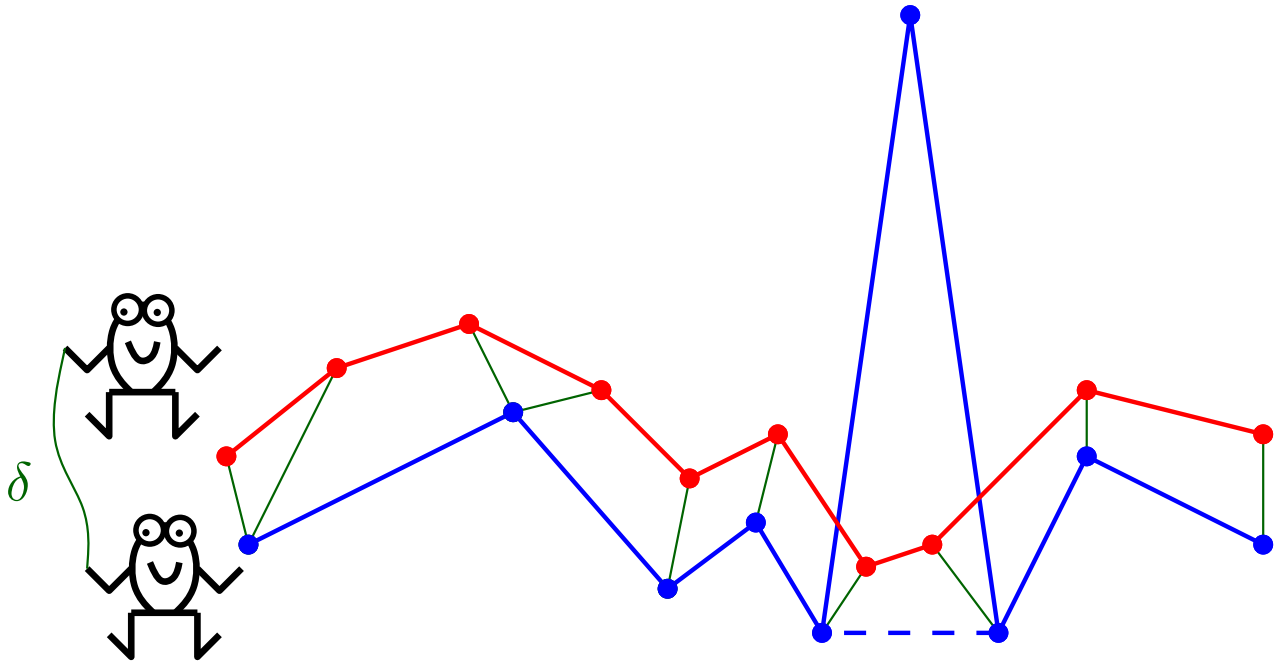
$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
Only the A-frog can skip points.



The one-sided discrete Fréchet distance with shortcuts

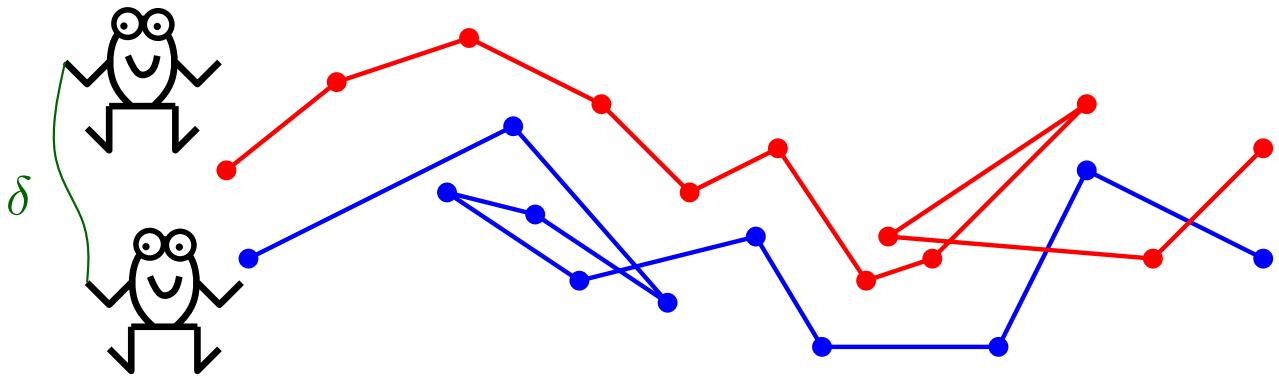
$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
Only the A-frog can skip points.

- The **one-sided discrete Fréchet distance with shortcuts** is the **minimum** such δ that allows the frogs to reach a_m and b_n.



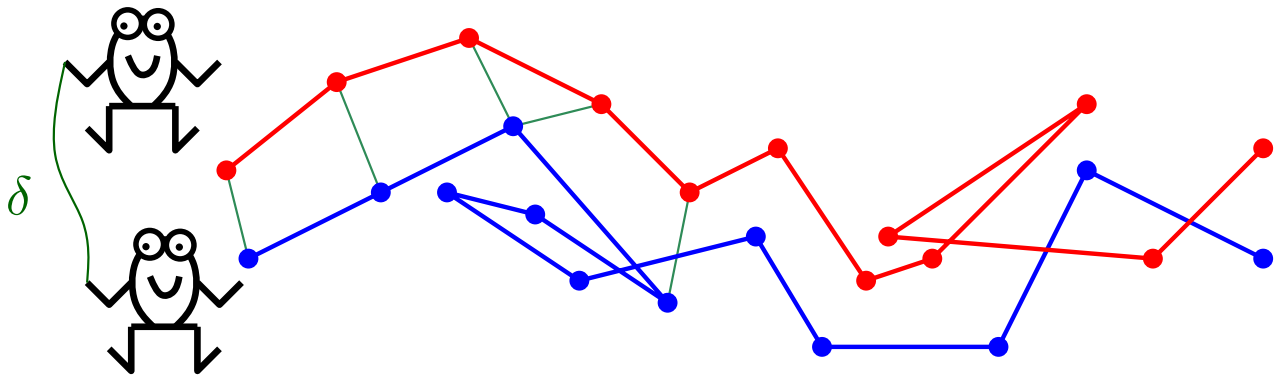
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



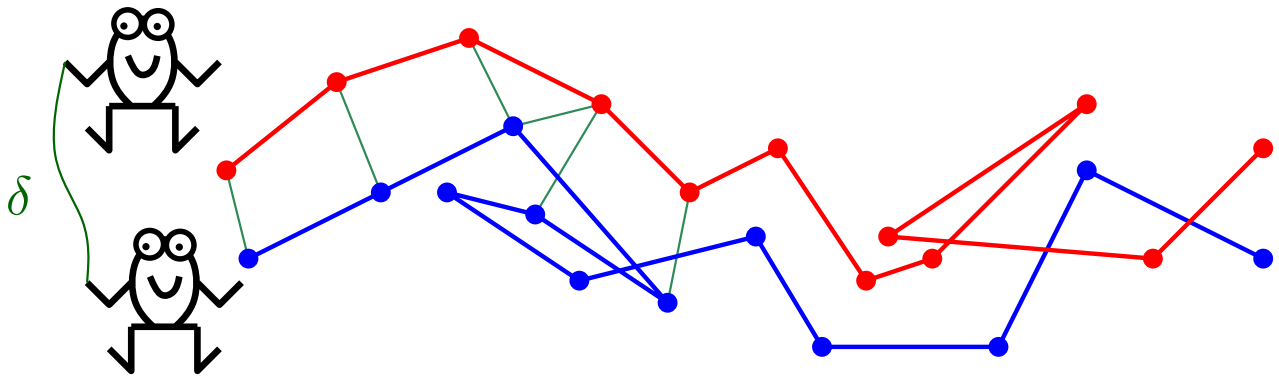
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



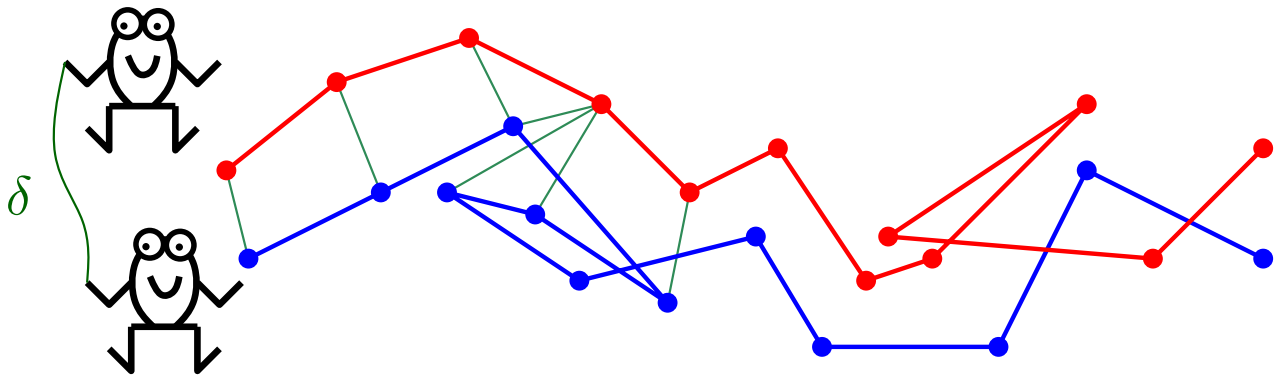
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



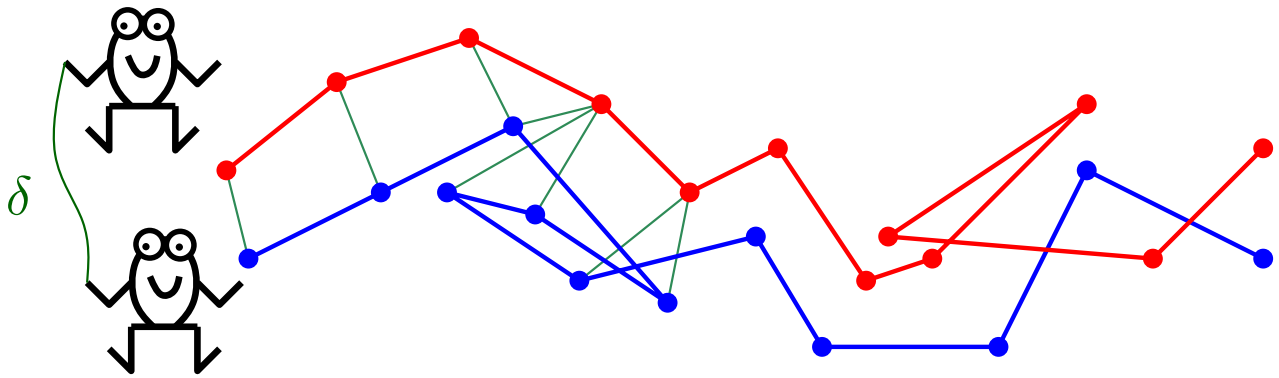
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



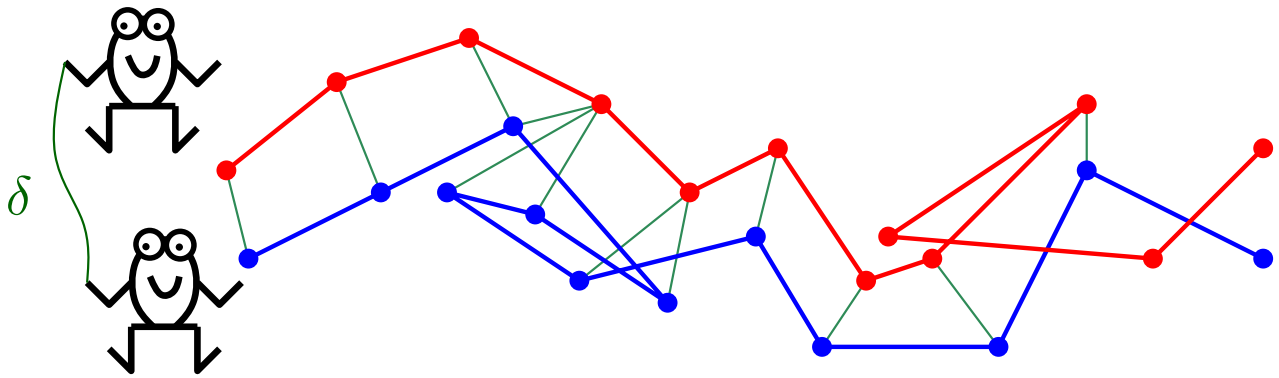
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



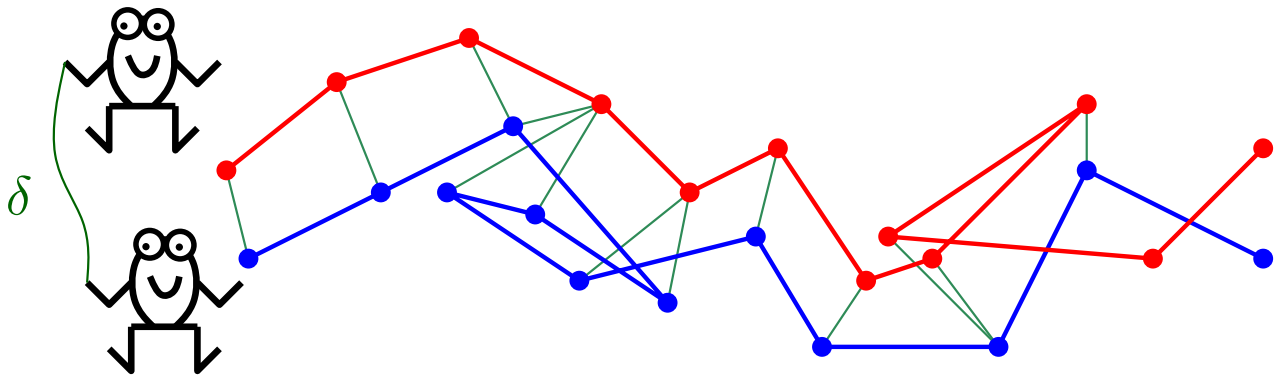
The weak discrete Fréchet distance

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



The weak discrete Fréchet distance

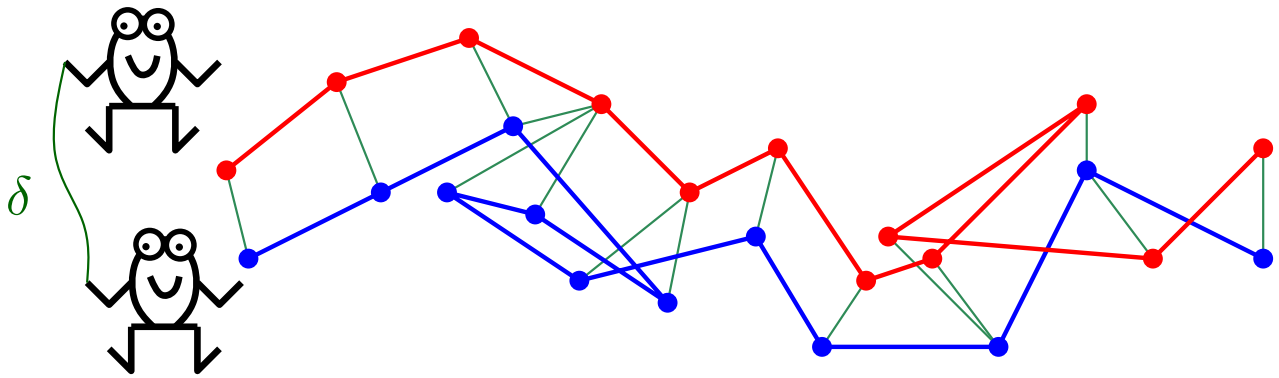
$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.



The weak discrete Fréchet distance

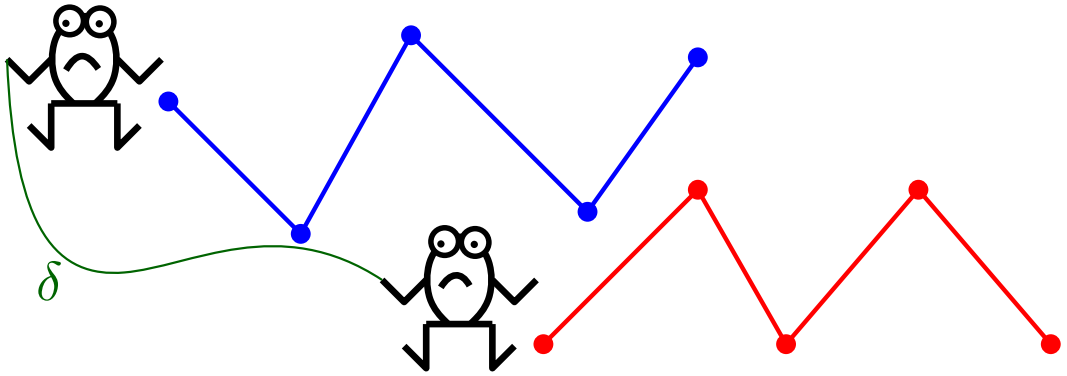
$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, a leash of length δ.
The frogs are also allowed to jump (one step) backwards.

- The **weak discrete Fréchet distance** is the **minimum** such δ that allows the frogs to reach a_m and b_n.



Fréchet under translation

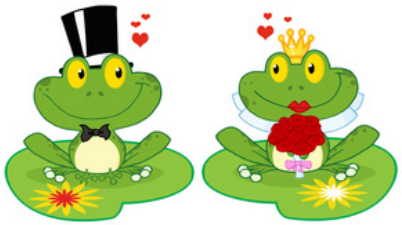
The input curves are not necessarily **aligned**, and one of them needs to be **adjusted**.

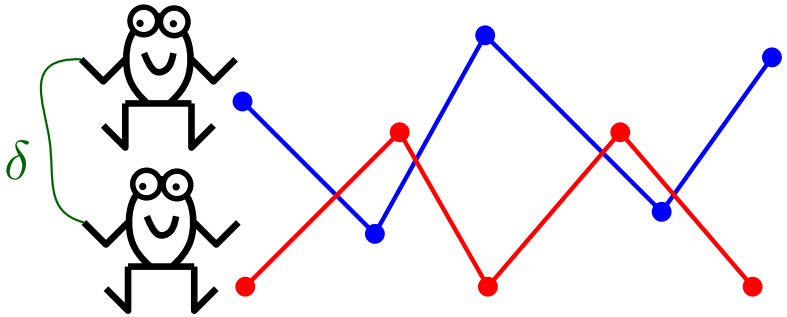


Fréchet under translation

The input curves are not necessarily **aligned**,
and one of them needs to be **adjusted**.

- Given $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$, find a translation t that minimizes the discrete Fréchet distance between A and $B + t$.





Related work

Continuous Fréchet distance under translation:

- **Alt et al. ('01)**: $O(m^3 n^3 (m+n)^2 \log(m+n))$-time algorithm for points in 2D, and an algorithm computing a $(1+\epsilon)$-approximation in $O(\epsilon^{-2} mn)$ time.
- **Wenk ('03)**: $O((m+n)^{11} \log(m+n))$-time algorithm for points in 3D (general results for d dimensions and other families of transformations).

Related work

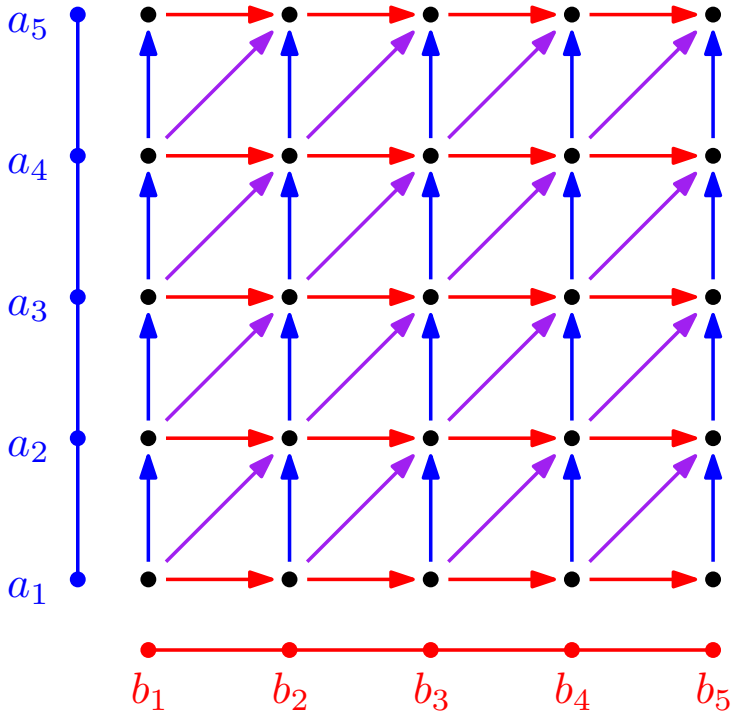
Discrete Fréchet distance under translation:

- **Mosig et al. ('05)**: $O(m^2n^2)$-time approximation algorithm for DFD under translation, rotation and scaling in 2D, with approximation factor close to 2.
- **Jiang et al. ('08)**: $O(m^3n^3 \log(m+n))$-time algorithm for DFD under translation, and an $O(m^4n^4 \log(m+n))$-time algorithm when both rotations and translations are allowed.
- **Ben Avraham et al. ('15)**: $O(m^3n^2(1+\log(\frac{n}{m}))\log(m+n))$-time algorithm, based on a dynamic data structure which supports updates and reachability queries in $O(m(1+\log(n/m))$ time (up next).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



$E_A = \{\langle (a_i, b_j), (a_{i+1}, b_j) \rangle\}$

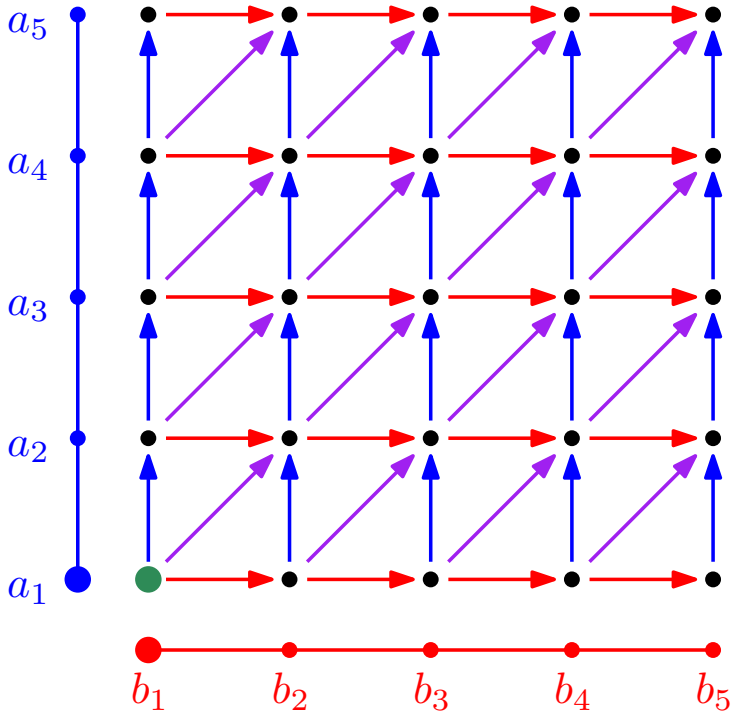
$E_B = \{\langle (a_i, b_j), (a_i, b_{j+1}) \rangle\}$

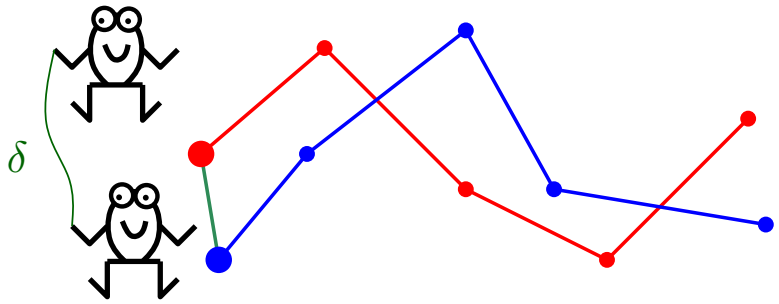
$E_{AB} = \{\langle (a_i, b_j), (a_{i+1}, b_{j+1}) \rangle\}$

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



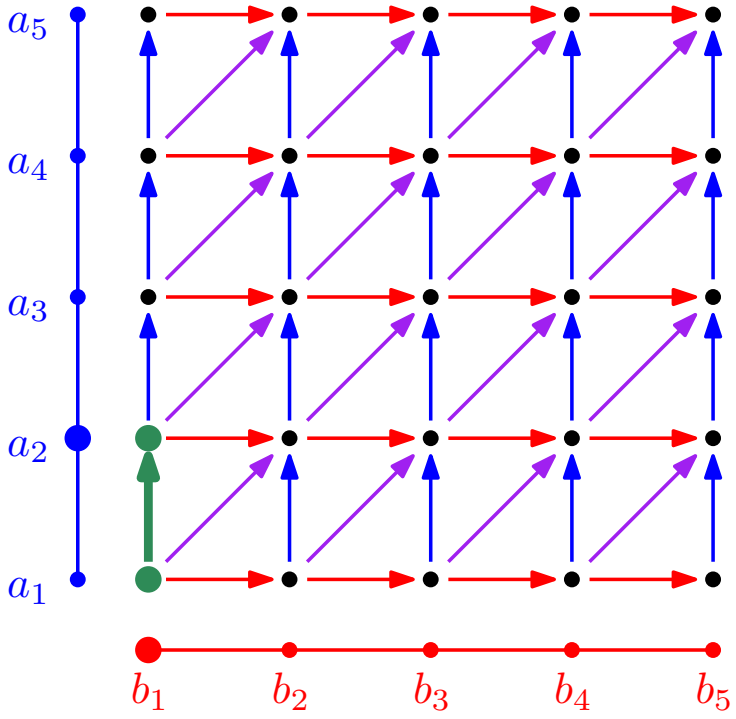


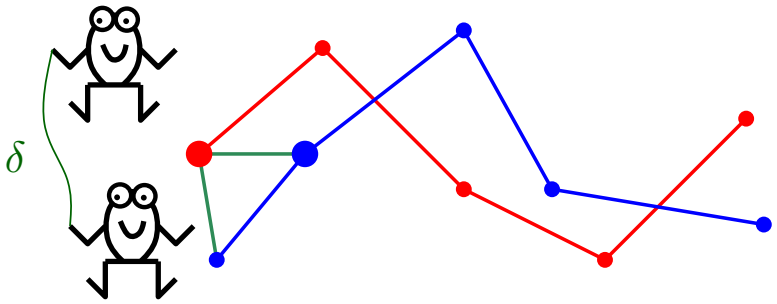
Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



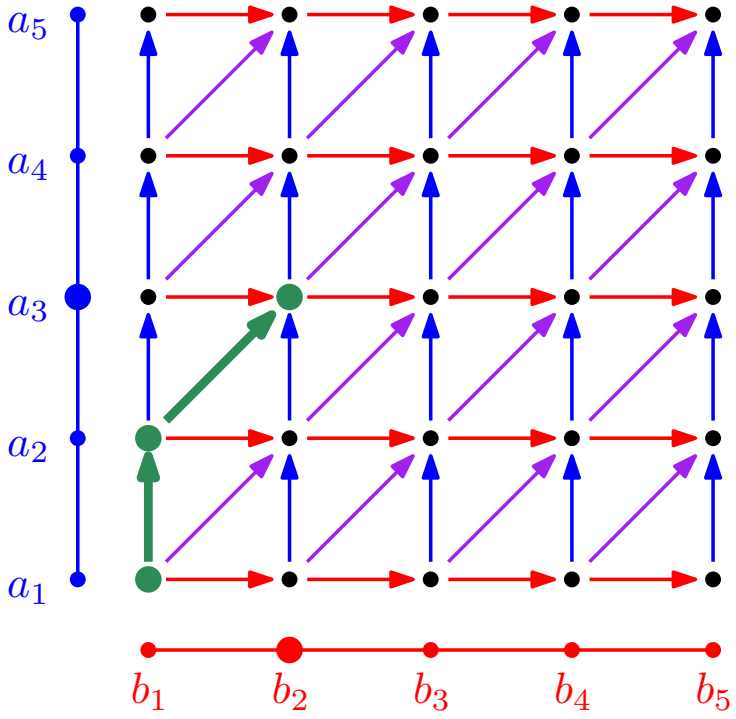


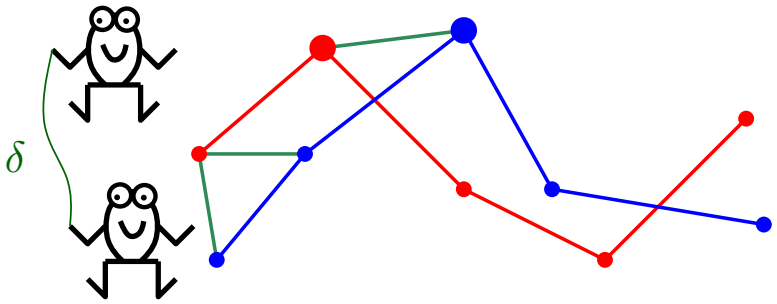
Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



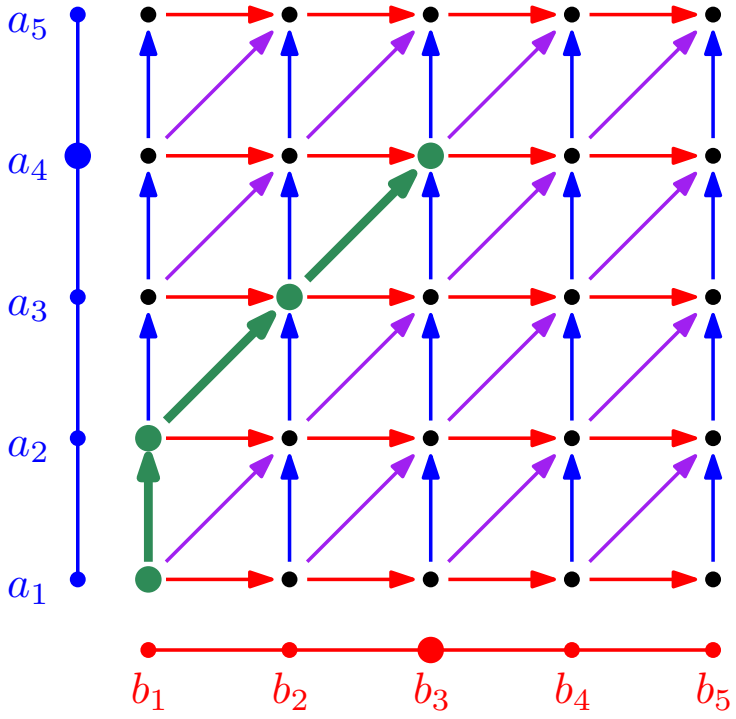


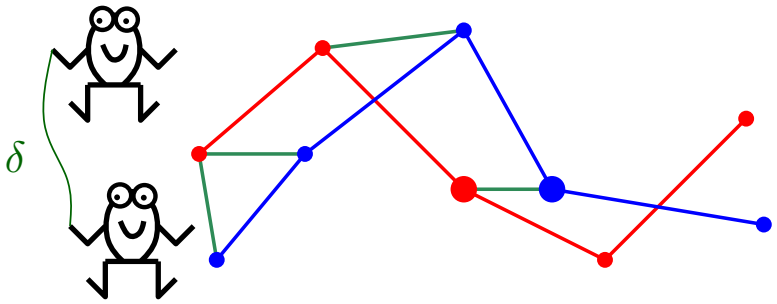
Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



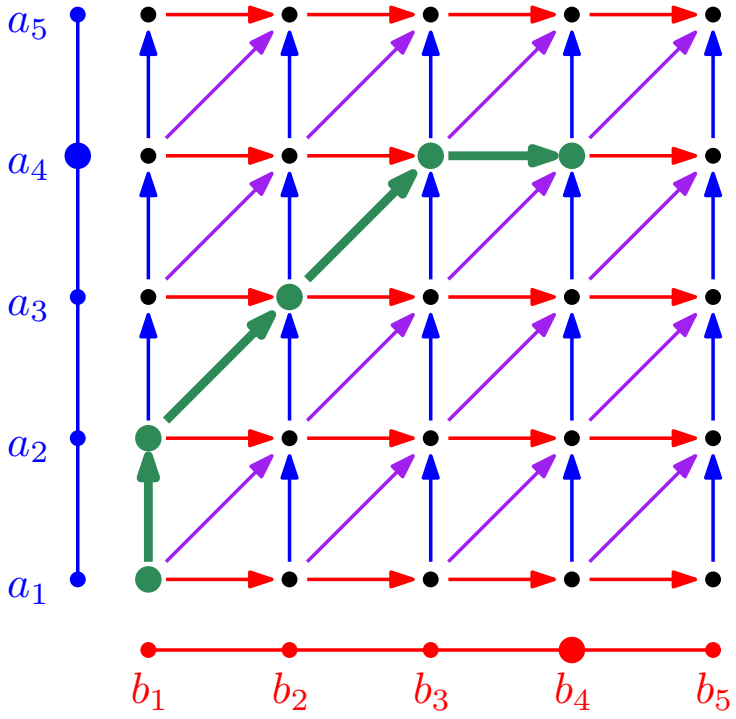


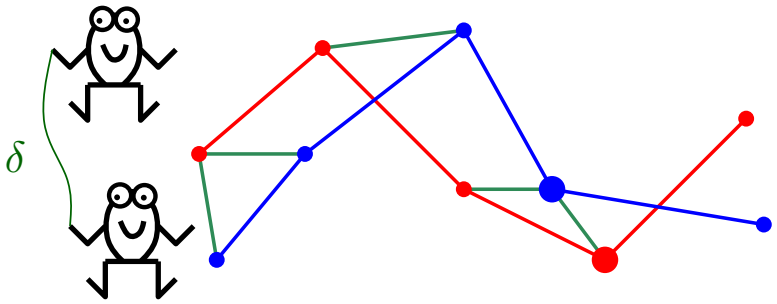
Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.



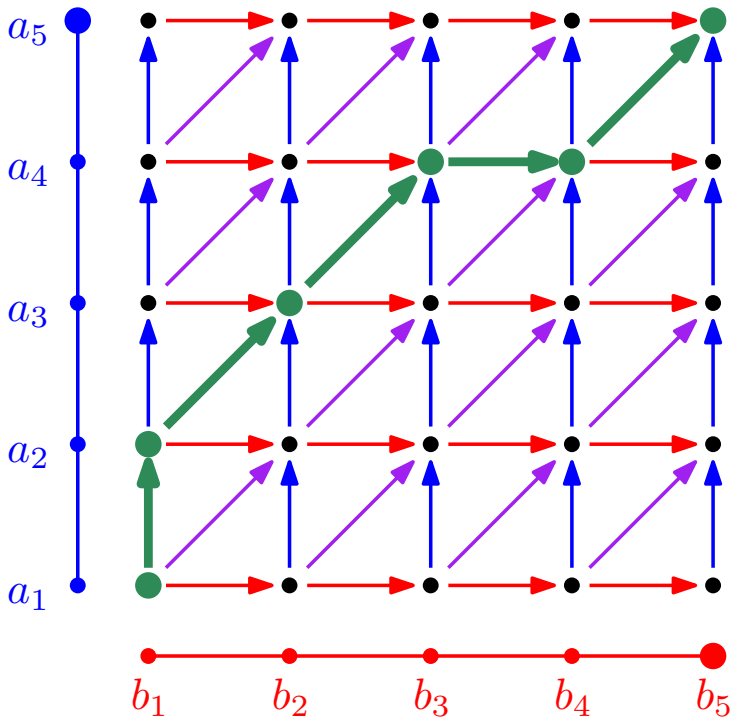


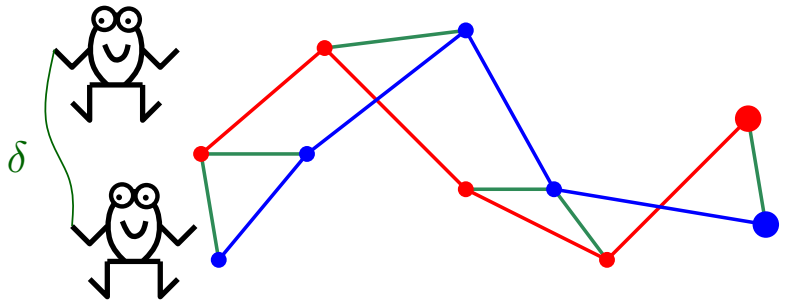
Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The positions graph

$G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$

- V: all possible **positions** of the frogs.
- E: all possible **moves** between positions.

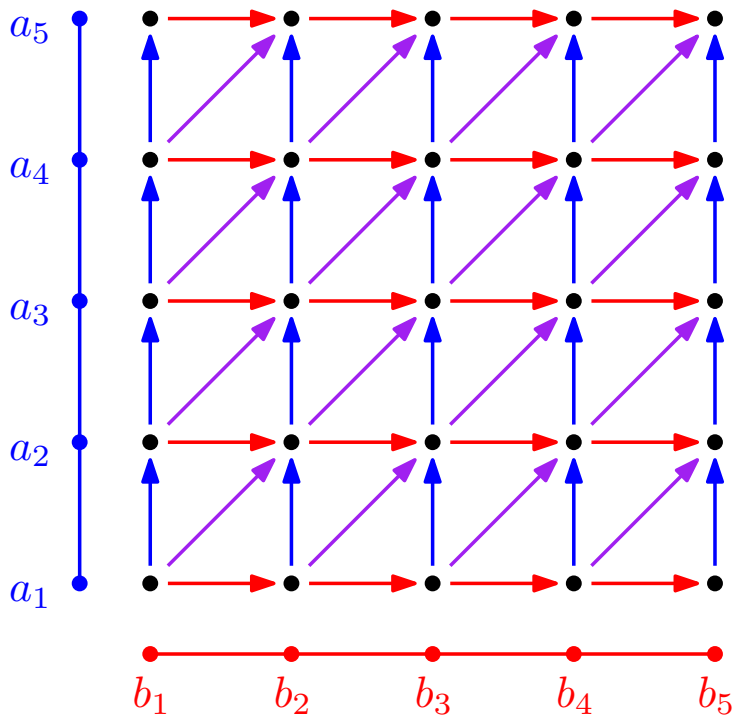




Sequence of moves (with unlimited leash length) $\Longleftrightarrow$
A path in G from (a_1, b_1) to (a_n, b_m).

The indicator function

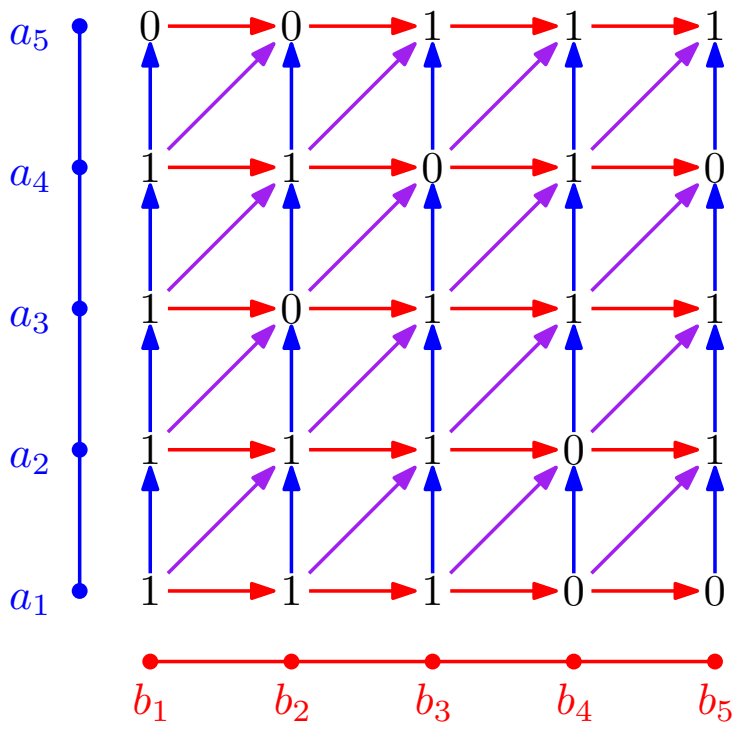
When the leash is short — not all positions in $A \times B$ are **valid**.



The indicator function

When the leash is short — not all positions in $A \times B$ are **valid**.

- **indicator** function $\sigma : A \times B \to \{0, 1\}$.

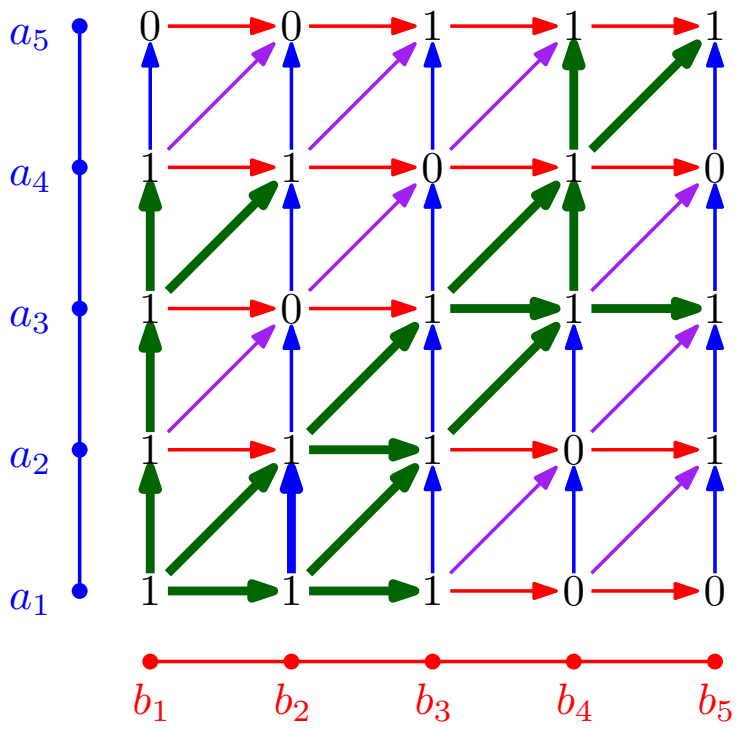


The indicator function

When the leash is short — not all positions in $A \times B$ are **valid**.

- **indicator** function $\sigma : A \times B \to \{0, 1\}$.

(a_i, b_j) is a **reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), consisting of only valid positions.



Discrete Fréchet distance

Let $d(a_i, b_j)$ denote the **Euclidean distance** between a_i and b_j.

Our indicator function: given a distance $\delta \geq 0$,

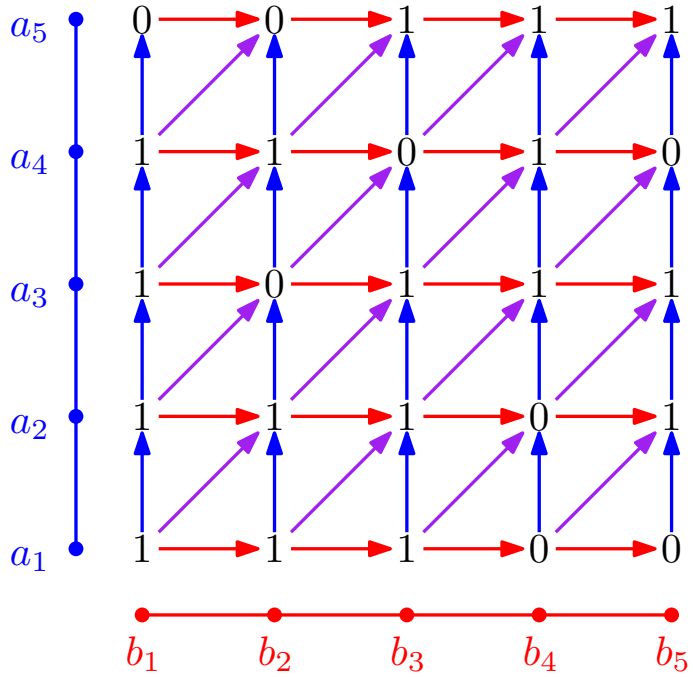
$$\sigma_\delta(a_i, b_j) = \begin{cases} 1, & d(a_i, b_j) \leq \delta \\ 0, & \text{otherwise} \end{cases}$$

The **discrete Fréchet distance** $d_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a reachable position w.r.t. σ_δ.

One-sided shortcuts

(a_i, b_j) is an **s-reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that:

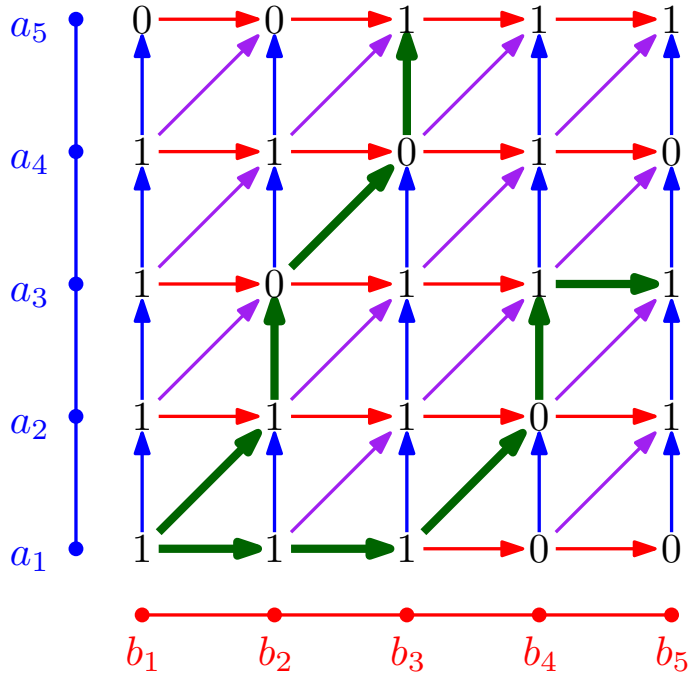
- $\sigma(a_1, b_1) = 1$ and $\sigma(a_i, b_j) = 1$.
- For each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$).



One-sided shortcuts

(a_i, b_j) is an **s-reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that:

- $\sigma(a_1, b_1) = 1$ and $\sigma(a_i, b_j) = 1$.
- For each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$).



One-sided shortcuts

(a_i, b_j) is an **s-reachable position** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that:

- $\sigma(a_1, b_1) = 1$ and $\sigma(a_i, b_j) = 1$.
- For each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$).

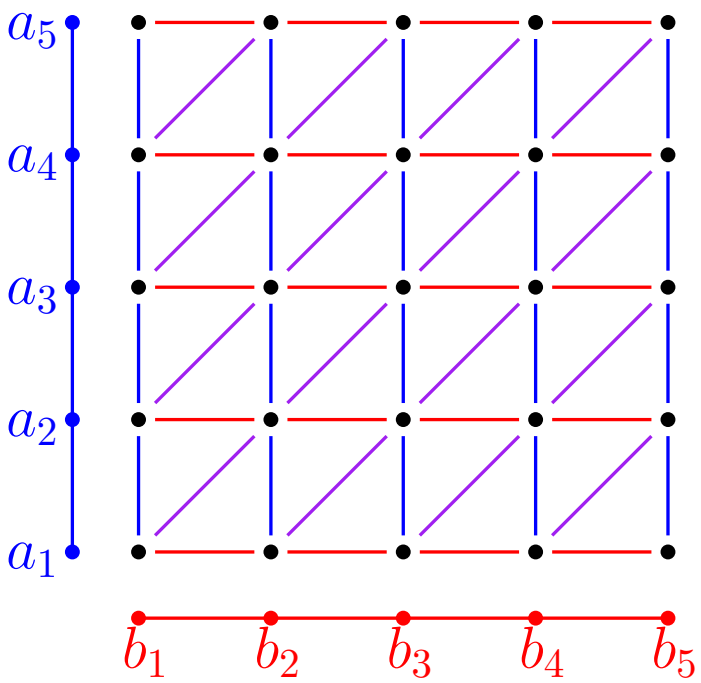
The **discrete Fréchet distance with shortcuts** $d^s_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is an s-reachable position w.r.t. σ_δ.

Weak Fréchet distance

Backtracking is allowed!

Remove the directions from G, a new graph: $G_w = G(V = A \times B, E_w)$

- $E_w = \{(u, v) | \langle u, v \rangle \in E_A \cup E_B \cup E_{AB}\}$



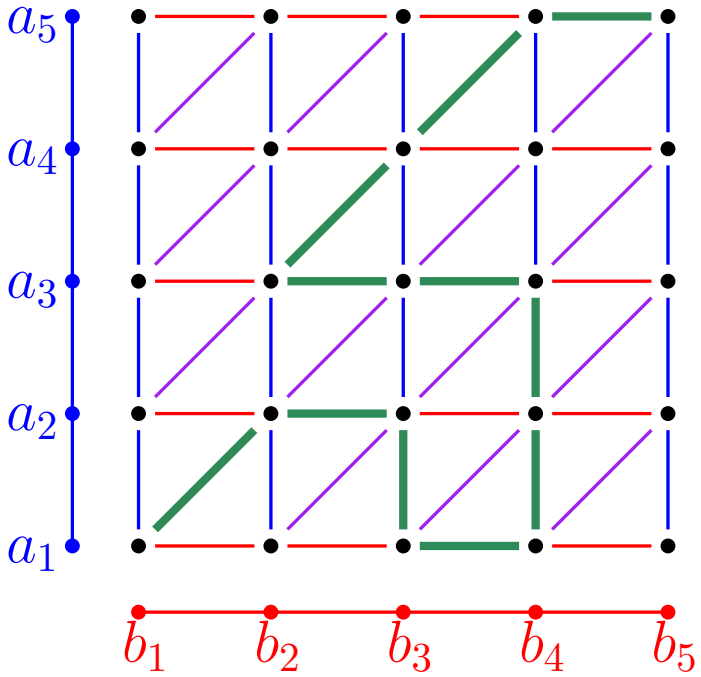
Weak Fréchet distance

Backtracking is allowed!

Remove the directions from G, a new graph: $G_w = G(V = A \times B, E_w)$

- $E_w = \{(u, v) | \langle u, v \rangle \in E_A \cup E_B \cup E_{AB}\}$

(a_i, b_j) is a **w-reachable position** (w.r.t. σ), if there exists a path P in G_w from (a_1, b_1) to (a_i, b_j) consisting of only valid positions.



Weak Fréchet distance

Backtracking is allowed!
Remove the directions from G, a new graph: $G_w = G(V = A \times B, E_w)$

- $E_w = \{(u, v) | \langle u, v \rangle \in E_A \cup E_B \cup E_{AB}\}$

(a_i, b_j) is a **w-reachable position** (w.r.t. σ), if there exists a path P in G_w from (a_1, b_1) to (a_i, b_j) consisting of only valid positions.

The **weak discrete Fréchet distance** $d_{dF}^{w}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a w-reachable position w.r.t. σ_δ.

The translation problem

Given two sequences of points $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_m)$, find a translation t^* that minimizes $d_{dF}(A, B + t)$ over all translations t.

Denote:
$\widehat{d_{dF}}(A, B) = \min_t\{d_{dF}(A, B + t)\}$
$\widehat{d^s_{dF}}(A, B) = \min_t\{d^s_{dF}(A, B + t)\}$
$\widehat{d^w_{dF}}(A, B) = \min_t\{d^w_{dF}(A, B + t)\}$

The algorithm of Ben Avaraham, Kaplan, and Sharir

- **Ben Avraham et al. ('15)**: $O(m^3 n^2 (1 + \log(\frac{n}{m})) \log(m + n))$-time algorithm for the discrete Fréchet distance under translation, based on a dynamic data structure which supports updates and reachability queries in $O(m(1 + \log(n/m))$ time.

The algorithm of Ben Avaraham, Kaplan, and Sharir

STEP 1: Build a **dynamic data structure** for (discrete Fréchet) reachability queries.

The algorithm of Ben Avaraham, Kaplan, and Sharir

STEP 1: Build a **dynamic data structure** for (discrete Fréchet) reachability queries.

Given sequences A and B and an indicator function σ:

The algorithm of Ben Avaraham, Kaplan, and Sharir

STEP 1: Build a **dynamic data structure** for (discrete Fréchet) reachability queries.

Given sequences A and B and an indicator function σ:

- The dynamic data structure is constructed in $O(mn)$ time.

The algorithm of Ben Avaraham, Kaplan, and Sharir

STEP 1: Build a **dynamic data structure** for (discrete Fréchet) reachability queries.

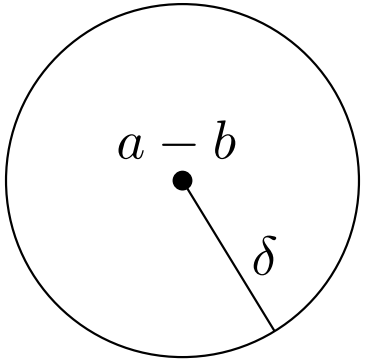
Given sequences A and B and an indicator function σ:

- The dynamic data structure is constructed in $O(mn)$ time.
- Allows the following operations in $O(m(1 + \log(n/m)))$ time:
  - **Reachability query:** return TRUE if and only if (a_n, b_m) is a reachable position w.r.t. σ.
  - **A single change in σ:** switch $\sigma(a_i, b_j)$ from 1 to 0 or vice versa.

STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

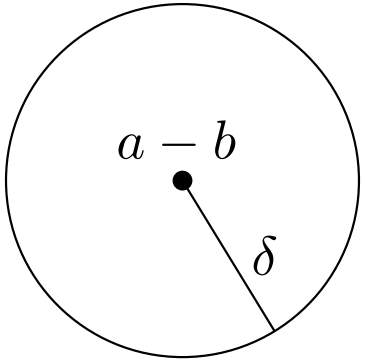
STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$:



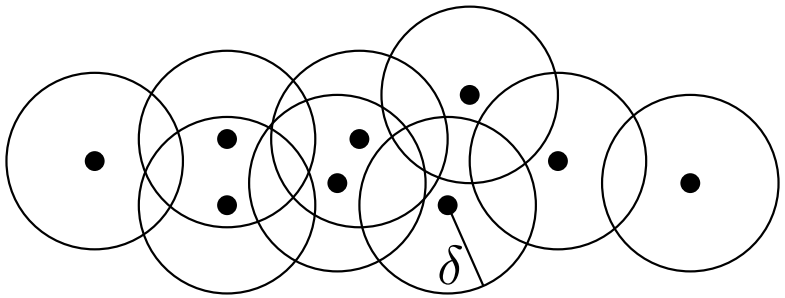
STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$: $\boxed{t \in D_\delta(a - b) \Leftrightarrow d(a - b, t) \leq \delta \Leftrightarrow d(a, b + t) \leq \delta.}$



STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$: $\boxed{t \in D_\delta(a - b) \Leftrightarrow d(a - b, t) \leq \delta \Leftrightarrow d(a, b + t) \leq \delta.}$
- Construct the **arrangement** A_δ of the disks in $\{D_\delta(a - b) \mid (a, b) \in A \times B\}$ — it has $O(m^2 n^2)$ cells.

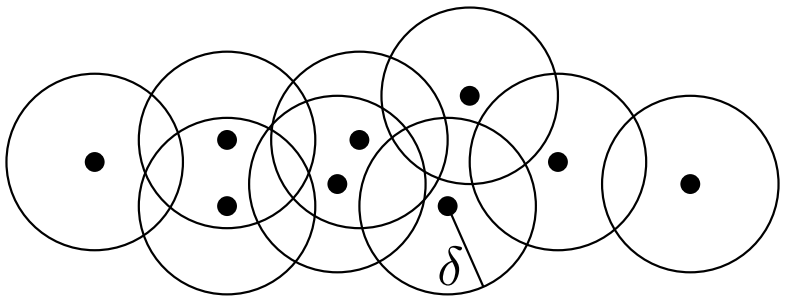


STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$: $\boxed{t \in D_\delta(a - b) \Leftrightarrow d(a - b, t) \leq \delta \Leftrightarrow d(a, b + t) \leq \delta.}$
- Construct the **arrangement** A_δ of the disks in $\{D_\delta(a - b) \mid (a, b) \in A \times B\}$ — it has $O(m^2 n^2)$ cells.
- Initialize a **dynamic data structure** for (discrete Fréchet) reachability queries.

STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$: $\boxed{t \in D_\delta(a - b) \Leftrightarrow d(a - b, t) \leq \delta \Leftrightarrow d(a, b + t) \leq \delta.}$
- Construct the **arrangement** A_δ of the disks in $\{D_\delta(a - b) \mid (a, b) \in A \times B\}$ — it has $O(m^2 n^2)$ cells.
- Initialize a **dynamic data structure** for (discrete Fréchet) reachability queries.
- **Traverse the cells of** A_δ: when moving between neighboring cells, the data structure is updated and queried in $O(m(1 + \log(n/m))$ time.



STEP 2: The **decision problem**: given a distance δ, is there a translation t such that $d_{dF}(A, B + t) \leq \delta$?

- Given $a \in A$, $b \in B$, consider the **disk** $D_\delta(a - b)$ of radius δ centered at $a - b$: $\boxed{t \in D_\delta(a - b) \Leftrightarrow d(a - b, t) \leq \delta \Leftrightarrow d(a, b + t) \leq \delta.}$
- Construct the **arrangement** A_δ of the disks in $\{D_\delta(a - b) \mid (a, b) \in A \times B\}$ — it has $O(m^2 n^2)$ cells.
- Initialize a **dynamic data structure** for (discrete Fréchet) reachability queries.
- **Traverse the cells of** A_δ: when moving between neighboring cells, the data structure is updated and queried in $O(m(1 + \log(n/m))$ time.

STEP 3: Use **parametric search** in order to find an optimal translation (adds only a $O(\log(m + n))$ factor to the running time).

Other variants of DFD

An efficient dynamic data structure for other variants of DFD?

Other variants of DFD

An efficient dynamic data structure for other variants of DFD?

- **WDFD**: dynamic reachability in an undirected planar graph.
  - **Eppstein et al. ('92)**: updates and reachability queries in an undirected planar graph in $O(\log |V|)$ time per operation.

Other variants of DFD

An efficient dynamic data structure for other variants of DFD?

- **WDFD**: dynamic reachability in an undirected planar graph.
  - **Eppstein et al. ('92)**: updates and reachability queries in an undirected planar graph in $O(\log |V|)$ time per operation.
- **DFDS**: reachability by **shortcut paths** — an s-path consists of both valid and non-valid positions, and not every path in G is an s-path.

Other variants of DFD

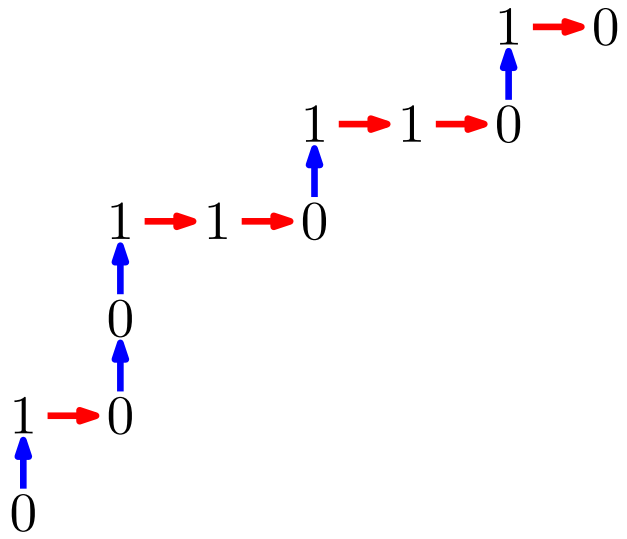
An efficient dynamic data structure for other variants of DFD?

- **WDFD**: dynamic reachability in an undirected planar graph.
  - **Eppstein et al. ('92)**: updates and reachability queries in an undirected planar graph in $O(\log |V|)$ time per operation.
- **DFDS**: reachability by **shortcut paths** — an s-path consists of both valid and non-valid positions, and not every path in G is an s-path.

Build a graph of s-paths!

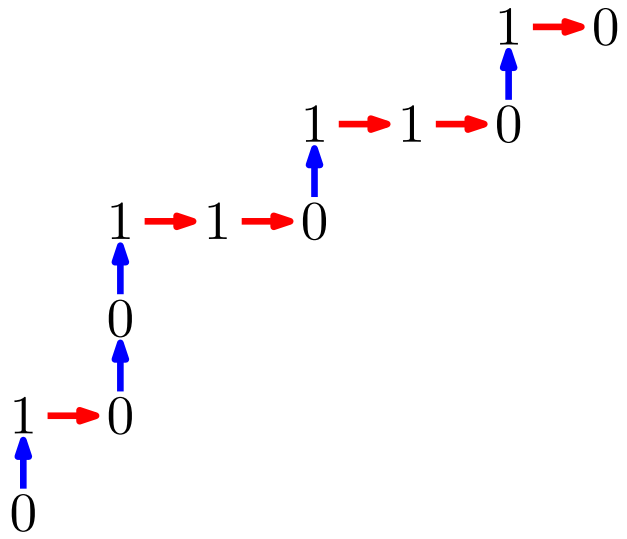
(partial) s-paths

A path P in G_δ from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \leq i'$, $j \leq j'$, is a **partial s-path**, if for each b_l, $j \leq l < j'$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma_\delta(a_k, b_l) = 1$).



(partial) s-paths

A path P in G_δ from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \leq i'$, $j \leq j'$, is a **partial s-path**, if for each b_l, $j \leq l < j'$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma_\delta(a_k, b_l) = 1$).



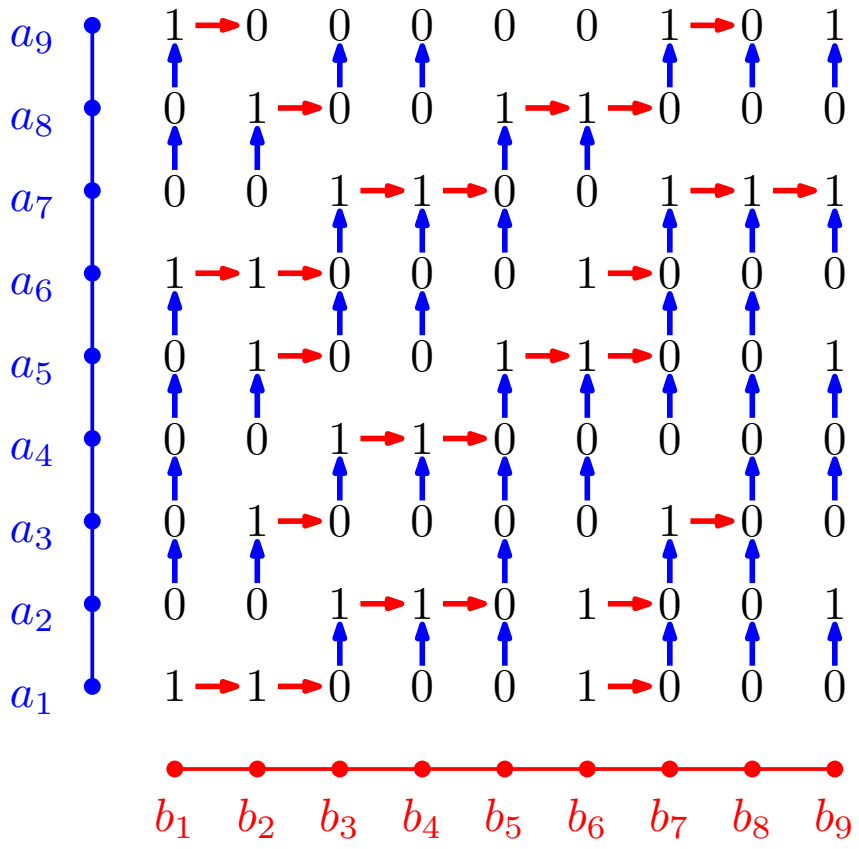
A path P in G_δ from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \leq i'$, $j \leq j'$, is an **s-path**, if it is a partial s-path and also $\sigma_\delta(a_i, b_j) = \sigma_\delta(a_{i'}, b_{j'}) = 1$.

The graph of (partial) s-paths

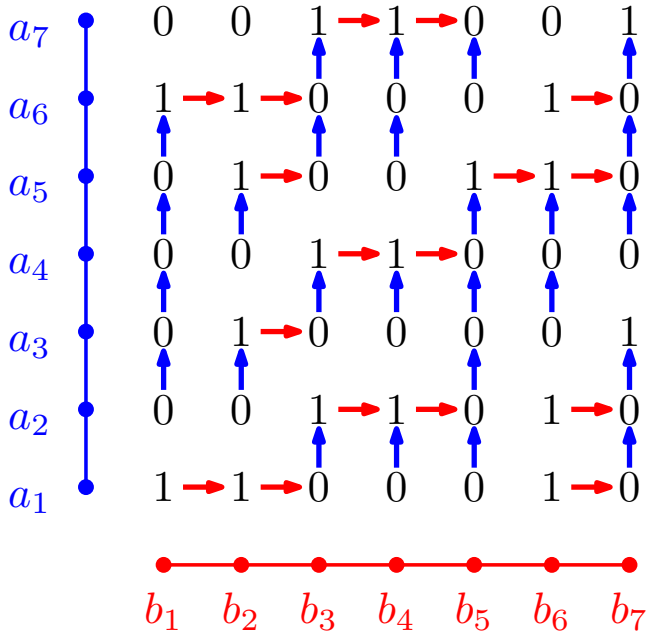
Consider $G_\delta = G(V = A \times B, E = E'_A \cup E'_B)$

$$E'_A = \{\langle (a_i, b_j), (a_{i+1}, b_j) \rangle \mid \sigma_\delta(a_i, b_j) = 0\}$$

$$E'_B = \{\langle (a_i, b_j), (a_i, b_{j+1}) \rangle \mid \sigma_\delta(a_i, b_j) = 1\}$$



Properties of G_δ



Property 1

All the paths in G_δ are partial s-paths.

Property 2

G_δ is a set of rooted binary trees
(where the root is a vertex of out-degree 0).

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of* (a_1, b_1) *in* G_δ *is* (a_i, b_m), *for some* $1 \leq i \leq n$.

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of* (a_1, b_1) *in* G_δ *is* (a_i, b_m), *for some* $1 \leq i \leq n$.

Proof.

$\Leftarrow$: By Property 1.

□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.

$\Leftarrow$: By Property 1.

$\Rightarrow$: Clearly, $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$,

Let P be an s-path in G from (a_1, b_1) to (a_n, b_m).

Let P' be the path in G_δ from (a_1, b_1) to its root.

P' is always not above P: if a position (a_i, b_j) is an s-reachable position in G, then there exists a position $(a_{i'}, b_j) \in P'$, $i' \leq i$, such that $\sigma_\delta(a_{i'}, b_j) = 1$.

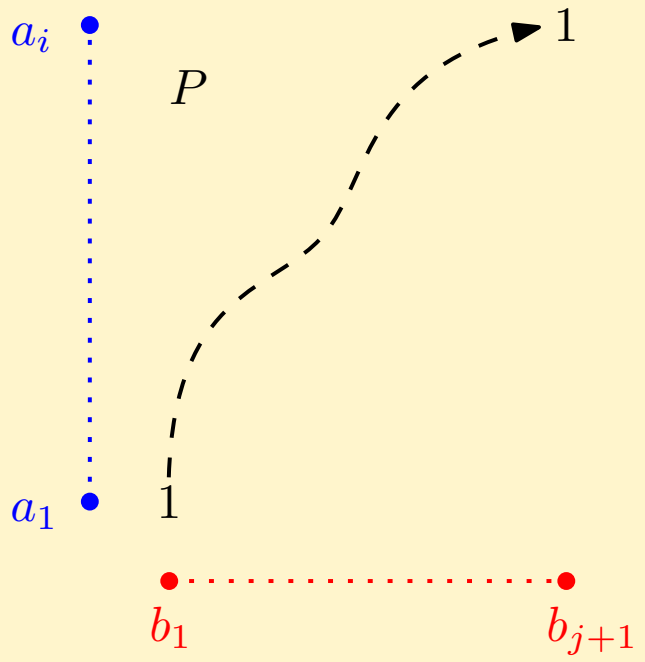
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



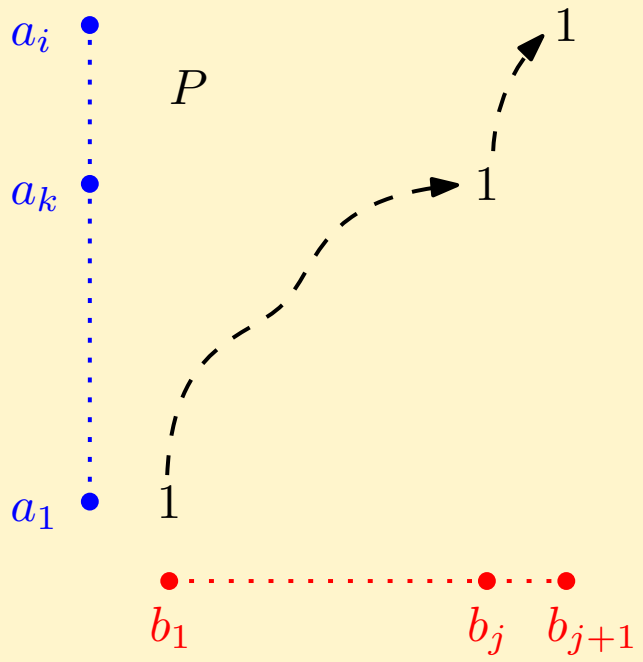
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



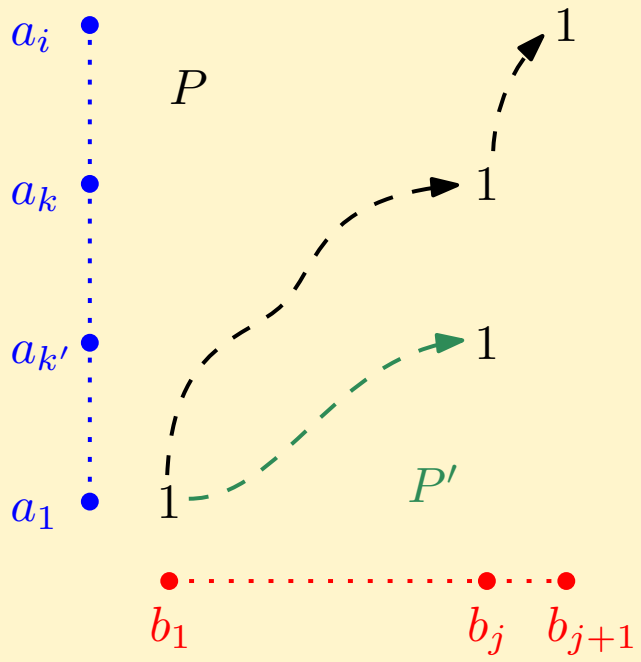
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



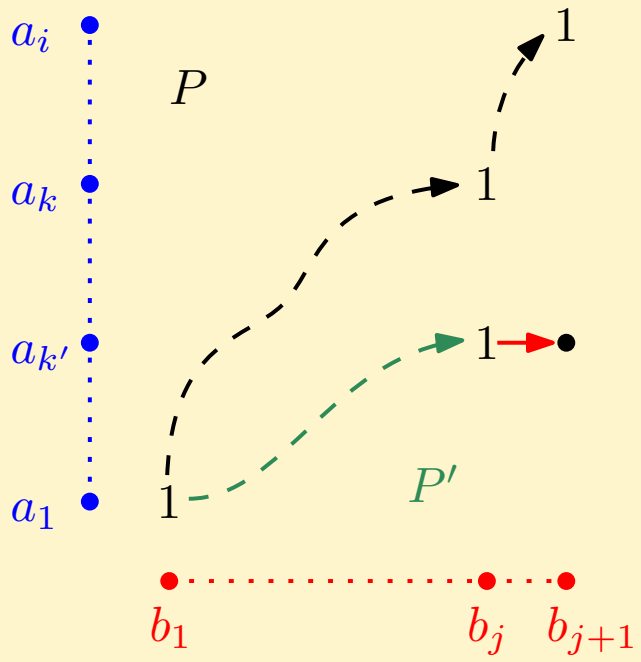
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



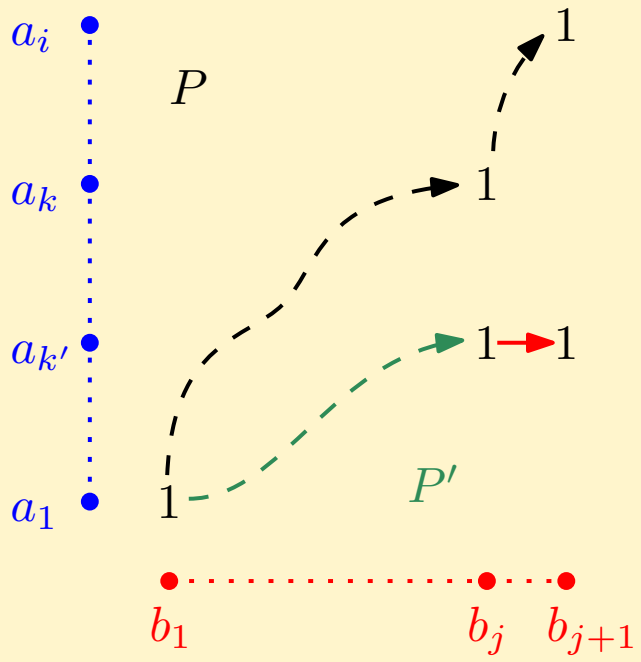
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



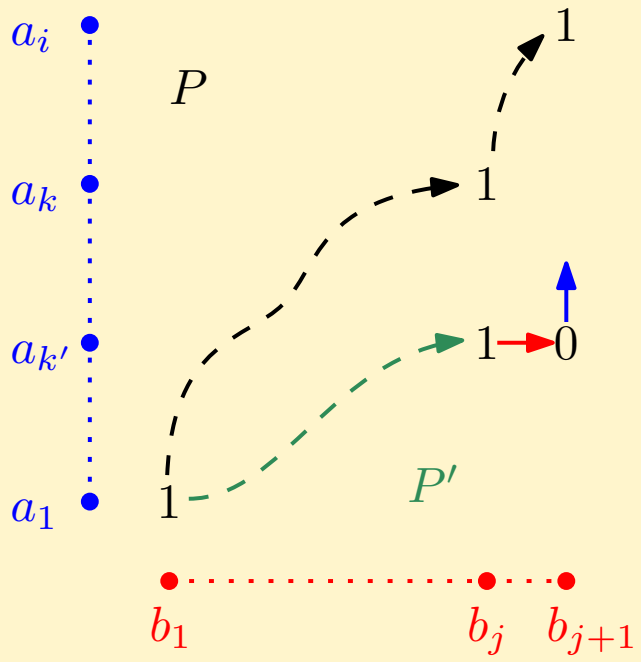
□

Lemma

(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if

- $\sigma_\delta(a_1, b_1) = 1$ *and* $\sigma_\delta(a_n, b_m) = 1$.
- *The* **root** *of (a_1, b_1) in G_δ is (a_i, b_m), for some $1 \leq i \leq n$.*

Proof.



A Link-Cut tree

We represent G_δ using the Link-Cut tree data structure:

Sleator and Tarjan ('83)

The Link-Cut tree data structure stores a set of rooted trees and supports the following operations in $O(\log n)$ amortized time:

- $Link(v, u)$ — connect a root v to another node u as its child.
- $Cut(v)$ — disconnect the subtree rooted at v from its tree.
- $FindRoot(v)$ — find the root of the tree to which v belongs.

The dynamic data structure for DFDS

- Switch $\sigma_\delta(a_i, b_j)$ from 1 to 0:
  - remove the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ ($Cut(a_i, b_j)$).
  - add the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ ($Link((a_i, b_j), (a_{i+1}, b_j))$).
- Switch $\sigma_\delta(a_i, b_j)$ from 0 to 1:
  - remove the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ ($Cut(a_i, b_j)$).
  - add the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ ($Link((a_i, b_j), (a_i, b_{j+1}))$).
- Reachability query: return TRUE if and only if
  (i) $\sigma_\delta(a_1, b_1) = \sigma_\delta(a_n, b_m) = 1$, and
  (ii) $FindRoot(a_1, b_1)$ is (a_i, b_m) for some $1 \leq i \leq n$.

The dynamic data structure for DFDS

- Switch $\sigma_\delta(a_i, b_j)$ from 1 to 0:
  - remove the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ ($Cut(a_i, b_j)$).
  - add the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ ($Link((a_i, b_j), (a_{i+1}, b_j))$).
- Switch $\sigma_\delta(a_i, b_j)$ from 0 to 1:
  - remove the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ ($Cut(a_i, b_j)$).
  - add the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ ($Link((a_i, b_j), (a_i, b_{j+1}))$).
- Reachability query: return TRUE if and only if
  (i) $\sigma_\delta(a_1, b_1) = \sigma_\delta(a_n, b_m) = 1$, and
  (ii) $FindRoot(a_1, b_1)$ is (a_i, b_m) for some $1 \leq i \leq n$.

Theorem

Given sequences A and B with n and m points respectively in the plane, $\widehat{d^s_{dF}}(A, B)$ can be computed in $O(m^2 n^2 \log^2(m+n))$-time.

Back to the algorithm of BKS

Assume the points of A and B are in $\mathbb{R}^d$:

Back to the algorithm of BKS

Assume the points of A and B are in $\mathbb{R}^d$:

- The size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$.

Back to the algorithm of BKS

Assume the points of A and B are in $\mathbb{R}^d$:

- The size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$.
- DFD: $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$ (BKS data structure)

Back to the algorithm of BKS

Assume the points of A and B are in $\mathbb{R}^d$:

- The size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$.
- DFD: $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$ (BKS data structure)
- DFDS: $O(m^d n^d \log^2(m+n))$ (our data structure)

Back to the algorithm of BKS

Assume the points of A and B are in $\mathbb{R}^d$:

- The size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$.
- DFD: $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$ (BKS data structure)
- DFDS: $O(m^d n^d \log^2(m+n))$ (our data structure)
- WDFD: $O(m^d n^d \log^2(m+n))$ (the data structure of Eppstein et al.)

A more direct approach for translation in $\mathbb{R}^d$

$A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_m)$ — points in $\mathbb{R}^d$.
$S(o, r)$ - the sphere with center o and radius r.

A more direct approach for translation in $\mathbb{R}^d$

$A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_m)$ — points in $\mathbb{R}^d$.
$S(o, r)$ - the sphere with center o and radius r.

New indicator function:

$$\sigma_{S(o,r)}(a_i, b_j) = \begin{cases} 1, & d(a_i - b_j, o) \leq r \\ 0, & \text{otherwise} \end{cases}$$

A more direct approach for translation in $\mathbb{R}^d$

$A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_m)$ — points in $\mathbb{R}^d$.
$S(o, r)$ - the sphere with center o and radius r.

New indicator function:

$$\sigma_{S(o,r)}(a_i, b_j) = \begin{cases} 1, & d(a_i - b_j, o) \leq r \\ 0, & \text{otherwise} \end{cases}$$

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.

$\square$

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.
- $d(a - b, t^*) = d(a, b + t^*) \ \Rightarrow \boxed{d_{dF}(A, B + t^*) \leq \delta}$.

$\square$

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.
- $d(a - b, t^*) = d(a, b + t^*) \ \Rightarrow \boxed{d_{dF}(A, B + t^*) \leq \delta}$.
- Let t be a translation such that $d_{dF}(A, B + t) = \delta'$.

□

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.
- $d(a - b, t^*) = d(a, b + t^*) \ \Rightarrow \boxed{d_{dF}(A, B + t^*) \leq \delta}$.
- Let t be a translation such that $d_{dF}(A, B + t) = \delta'$.
- There exist a path P from (a_1, b_1) to (a_n, b_m) in G such that: $\forall (a, b) \in P,\ d(a - b, t) = d(a, b + t) \leq \delta'$.

□

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.
- $d(a - b, t^*) = d(a, b + t^*) \ \Rightarrow \boxed{d_{dF}(A, B + t^*) \leq \delta}$.
- Let t be a translation such that $d_{dF}(A, B + t) = \delta'$.
- There exist a path P from (a_1, b_1) to (a_n, b_m) in G such that: $\forall (a, b) \in P,\ d(a - b, t) = d(a, b + t) \leq \delta'$.
- (a_n, b_m) is a reachable position w.r.t. $\sigma_{S'}$, where $S' = S(t, \delta')$.

□

Lemma

Let $S = S(t^*, \delta)$ be a smallest sphere for which (a_n, b_m) is a reachable position w.r.t. σ_S. Then, $\widehat{d_{dF}}(A, B) = d_{dF}(A, B + t^*) = \delta$.

Proof.

- There exists a path P from (a_1, b_1) to (a_n, b_m) in G, such that: $\forall (a, b) \in P,\ d(a - b, t^*) \leq \delta$.
- $d(a - b, t^*) = d(a, b + t^*) \ \Rightarrow \boxed{d_{dF}(A, B + t^*) \leq \delta}$.
- Let t be a translation such that $d_{dF}(A, B + t) = \delta'$.
- There exist a path P from (a_1, b_1) to (a_n, b_m) in G such that: $\forall (a, b) \in P,\ d(a - b, t) = d(a, b + t) \leq \delta'$.
- (a_n, b_m) is a reachable position w.r.t. $\sigma_{S'}$, where $S' = S(t, \delta')$.
- S is the smallest sphere for which (a_n, b_m) is a reachable position w.r.t. $\sigma_S \Rightarrow \boxed{\delta' \geq \delta}$.

□

New goal: find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S.

New goal: find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S.

- For $d > 1$ dimensions: exhaustive search? no better running time...

New goal: find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S.

- For $d > 1$ dimensions: exhaustive search? no better running time...
- For $d = 1$, this leads to an improved result:
  reduce a $\log(m + n)$ **factor** and **avoid the parametric search**.

New goal: find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S.

- For $d > 1$ dimensions: exhaustive search? no better running time...
- For $d = 1$, this leads to an improved result:
  reduce a $\log(m + n)$ **factor** and **avoid the parametric search**.
  - DFD: $O(m^2 n(1 + \log(n/m)))$

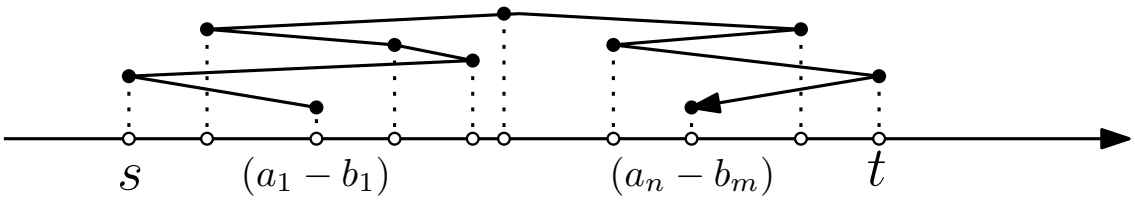
New goal: find the smallest sphere S for which (a_n, b_m) is a reachable position w.r.t. σ_S.

- For $d > 1$ dimensions: exhaustive search? no better running time...
- For $d = 1$, this leads to an improved result:
  reduce a $\log(m+n)$ **factor** and **avoid the parametric search**.
  - DFD: $O(m^2 n(1 + \log(n/m)))$
  - DFDS, WDFD: $O(mn \log(m+n))$

Translation in 1D

New indicator function: given a range $[s, t]$,

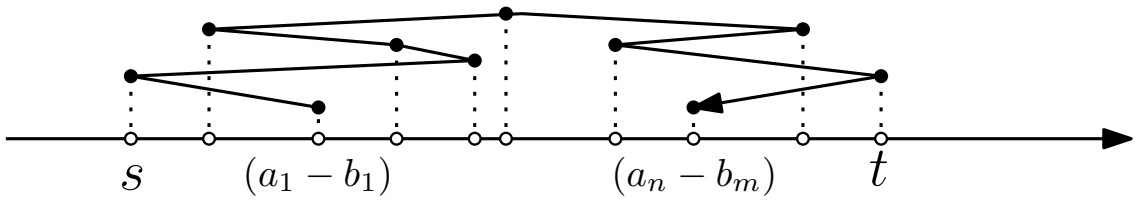
$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq a - b \leq t \\ 0, & \text{otherwise} \end{cases}$$



Translation in 1D

New indicator function: given a range $[s, t]$,

$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq a - b \leq t \\ 0, & \text{otherwise} \end{cases}$$

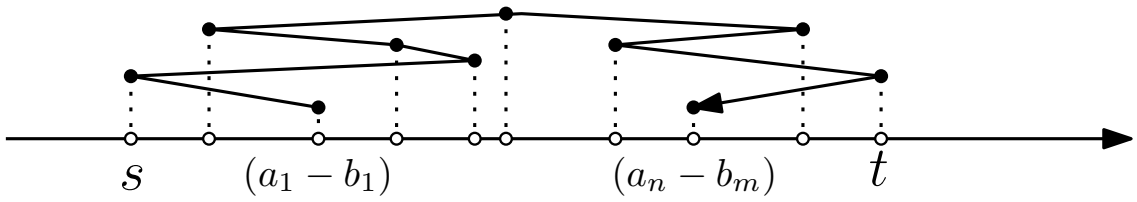


A range $[s, t]$ is a **feasible range** if (a_n, b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$.

Translation in 1D

New indicator function: given a range $[s, t]$,

$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq a - b \leq t \\ 0, & \text{otherwise} \end{cases}$$

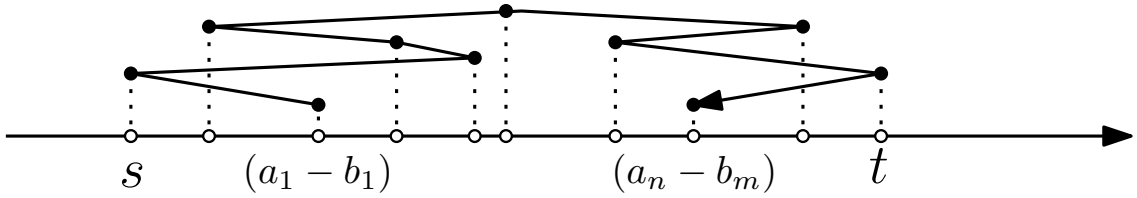


A range $[s, t]$ is a **feasible range** if (a_n, b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$ (there exists a path P from (a_1, b_1) to (a_n, b_m), such that $s \leq \max\{a - b \mid (a, b) \in P\} \leq \min\{a - b \mid (a, b) \in P\} \leq t$).

Translation in 1D

New indicator function: given a range $[s, t]$,

$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq a - b \leq t \\ 0, & \text{otherwise} \end{cases}$$



A range $[s, t]$ is a **feasible range** if (a_n, b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$ (there exists a path P from (a_1, b_1) to (a_n, b_m), such that $s \leq \max\{a - b \mid (a, b) \in P\} \leq \min\{a - b \mid (a, b) \in P\} \leq t$).

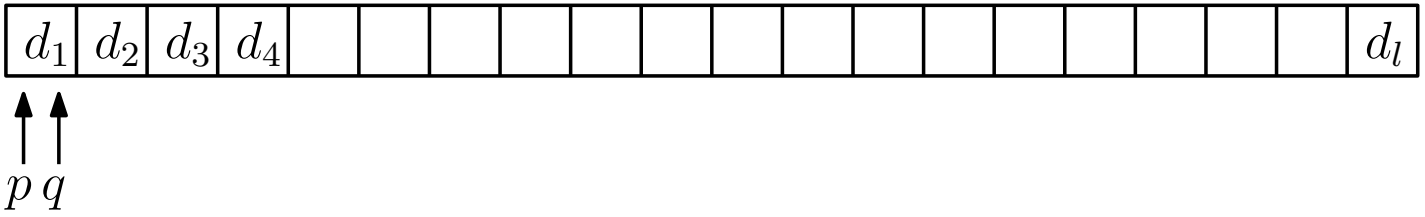
Let $\mathcal{D} = \{a_i - b_j \mid a_i \in A, b_j \in B\}$.

New goal: Find the smallest feasible range delimited by two points of $\mathcal{D}$.

Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.

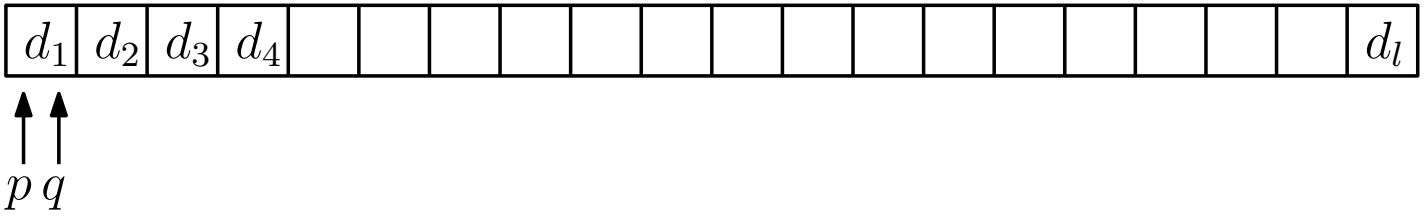
$[d_p, d_q]$ is not a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.

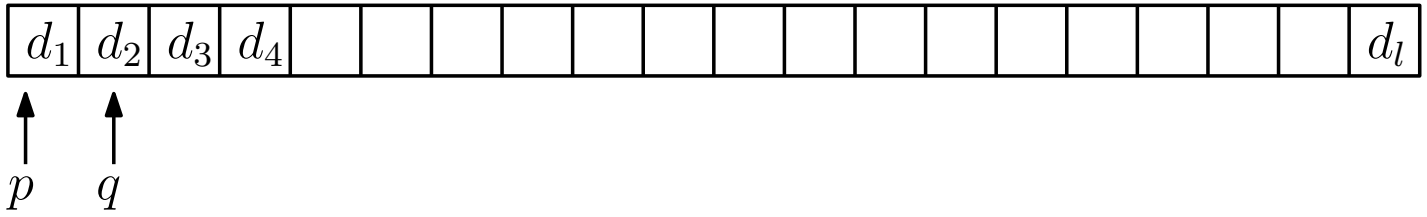
$[d_p, d_q]$ is not a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, set $p \leftarrow p + 1$, **else set** $q \leftarrow q + 1$.

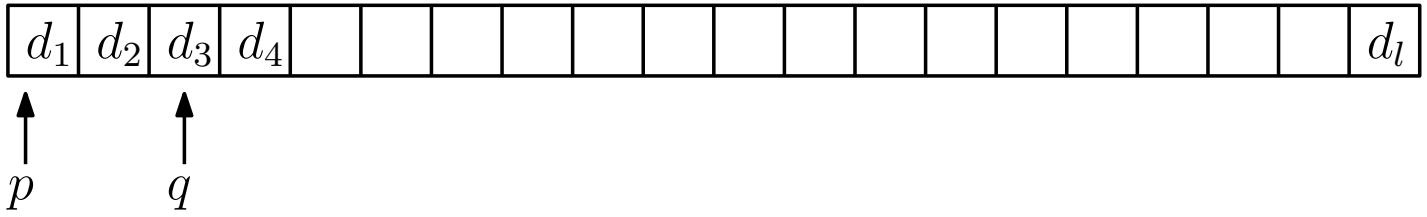
$[d_p, d_q]$ is not a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, set $p \leftarrow p + 1$, **else set** $q \leftarrow q + 1$.

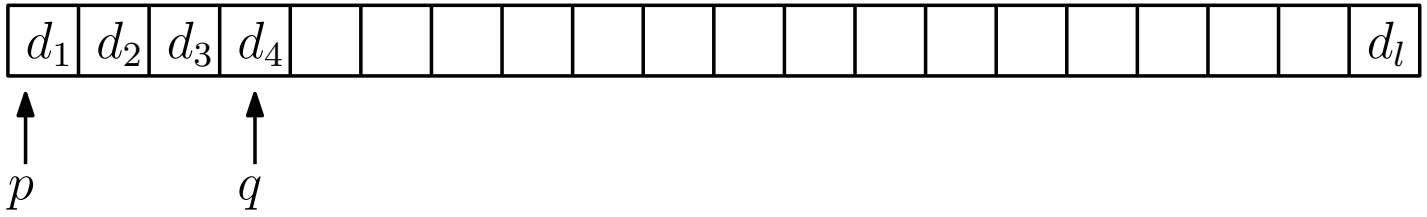
$[d_p, d_q]$ is not a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, set $p \leftarrow p + 1$, **else set** $q \leftarrow q + 1$.

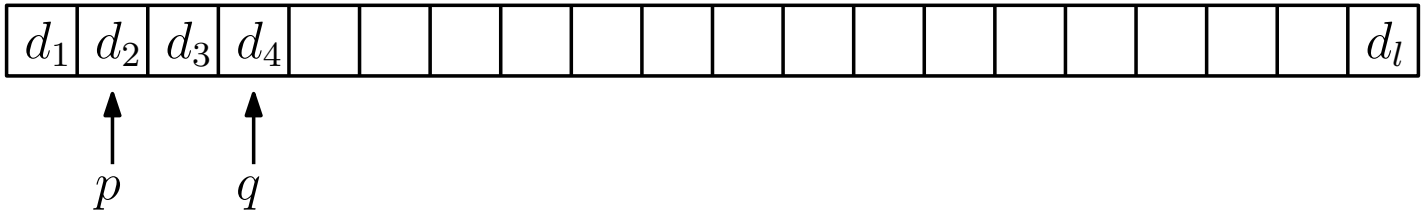
$[d_p, d_q]$ is a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, **set** $p \leftarrow p + 1$, else set $q \leftarrow q + 1$.

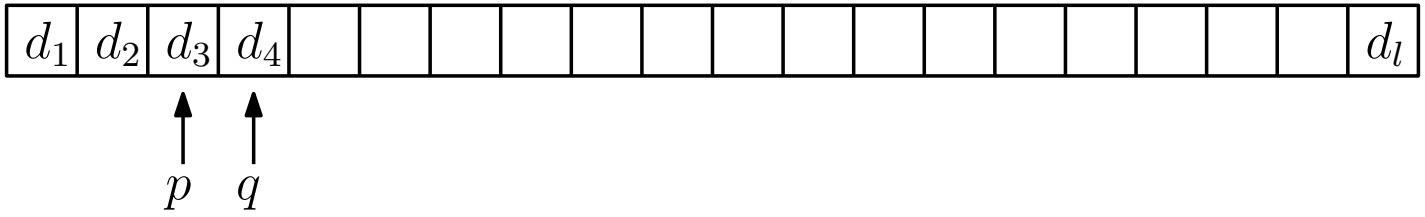
$[d_p, d_q]$ is a feasible range



Algorithm:

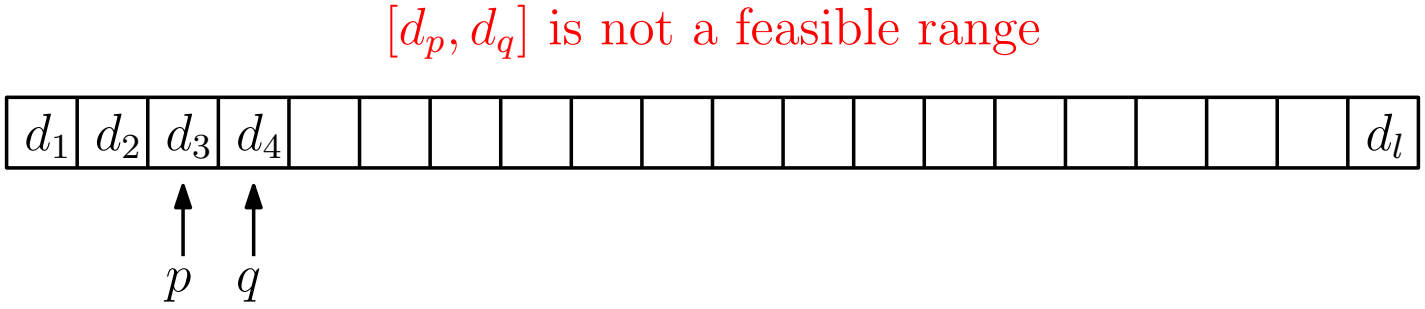
- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, **set** $p \leftarrow p + 1$, else set $q \leftarrow q + 1$.

$[d_p, d_q]$ is not a feasible range



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, set $p \leftarrow p + 1$, else set $q \leftarrow q + 1$.
- Return the translation corresponding to the smallest feasible range $[d_p, d_q]$ that was found during the while loop.



Algorithm:

- Sort the values in $\mathcal{D} = \{d_1, \ldots, d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$.
- Set $p \leftarrow 1, q \leftarrow 1$.
- While $q \leq l$, if (a_n, b_m) is a reachable position in G w.r.t. $\sigma_{[d_p, d_q]}$, set $p \leftarrow p + 1$, else set $q \leftarrow q + 1$.
- Return the translation corresponding to the smallest feasible range $[d_p, d_q]$ that was found during the while loop.

Theorem

Let A and B be two sequences of n and m points ($m \leq n$), respectively, on the line. Then, $\widehat{d_{dF}}(A, B)$ can be computed in $O(m^2 n(1 + \log(n/m))$ time, and $\widehat{d^s_{dF}}(A, B)$ and $\widehat{d^w_{dF}}(A, B)$ can be computed in $O(mn \log(m + n))$ time.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

- $\mathcal{F}$ is a family of well-defined structures: matchings, paths, spanning trees, cut-sets, edge covers, etc.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

- $\mathcal{F}$ is a family of well-defined structures: matchings, paths, spanning trees, cut-sets, edge covers, etc.

For $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

- $\mathcal{F}$ is a family of well-defined structures: matchings, paths, spanning trees, cut-sets, edge covers, etc.

For $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$.

Definition: BOP

Find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

- $\mathcal{F}$ is a family of well-defined structures: matchings, paths, spanning trees, cut-sets, edge covers, etc.

For $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$.

Definition: BOP

Find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$.

A range $[l, u]$ is a **feasible range** if there exists a feasible subset $S \in \mathcal{F}$ such that $l \leq S_{min} \leq S_{max} \leq u$.

Balanced Optimization Problem (on Graphs)

$G = (V, E, w)$ – a **weighted graph** with n vertices and m edges.
$\mathcal{F}$ – a set of **feasible subsets** of E.

- $\mathcal{F}$ is a family of well-defined structures: matchings, paths, spanning trees, cut-sets, edge covers, etc.

For $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$.

Definition: BOP

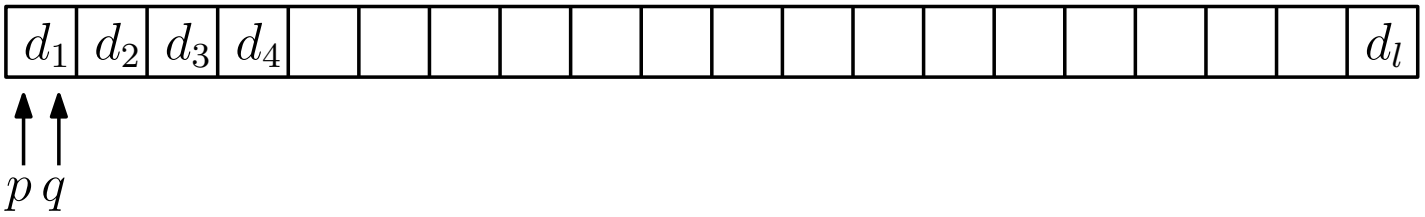
Find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$.

A range $[l, u]$ is a **feasible range** if there exists a feasible subset $S \in \mathcal{F}$ such that $l \leq S_{min} \leq S_{max} \leq u$.

Goal: find the smallest feasible range.

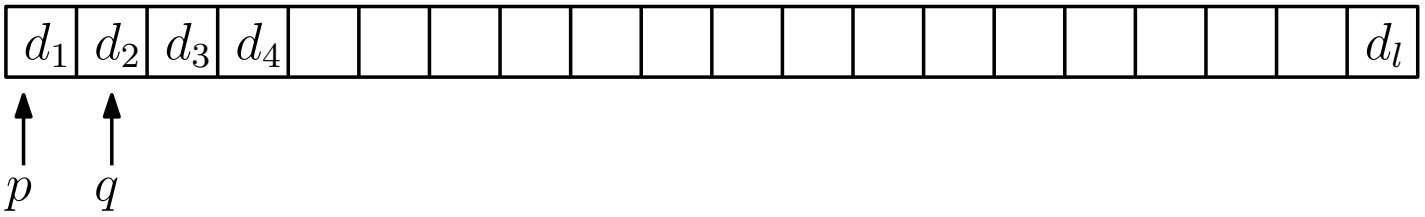
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is not a feasible range



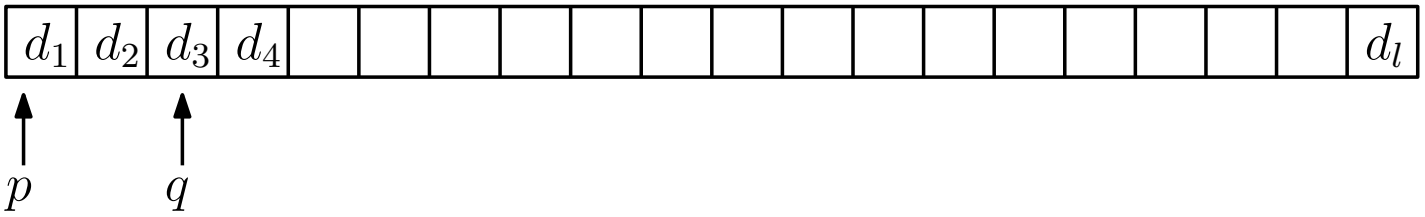
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is not a feasible range



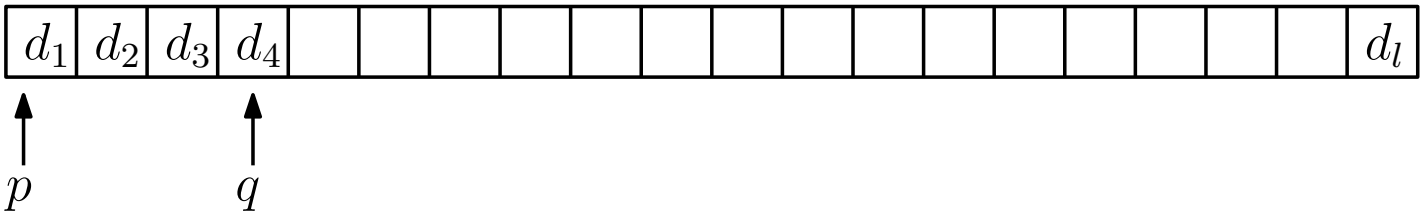
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is not a feasible range



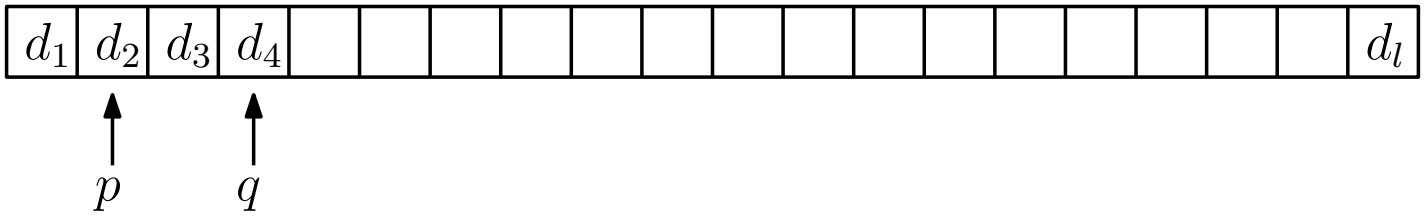
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is a feasible range



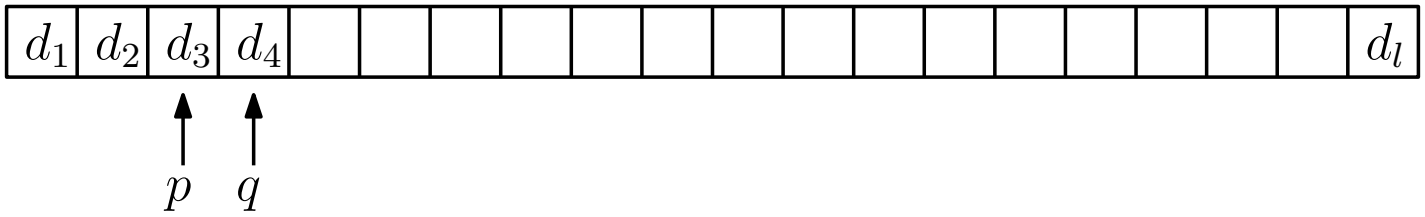
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is a feasible range



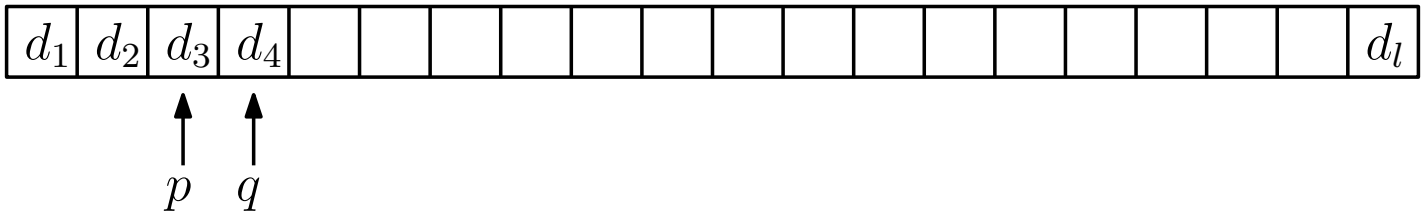
- **Martello et al. ('84)**: a general optimization algorithm BOP:

$[d_p, d_q]$ is not a feasible range

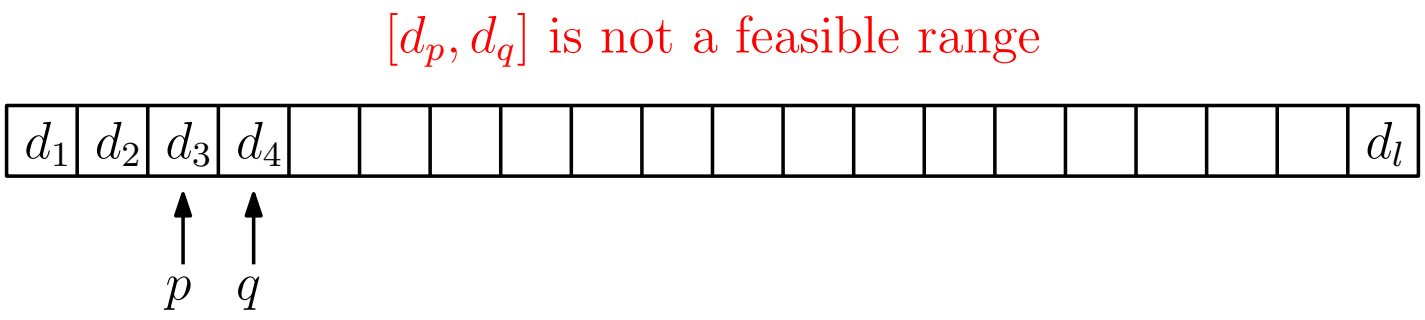


- **Martello et al. ('84)**: a general optimization algorithm BOP:
  - Given a **feasibility decider** that decides whether a subset is feasible or not in $f(l)$ time, their algorithm finds an optimal range in $O(lf(l) + l \log l)$-time.

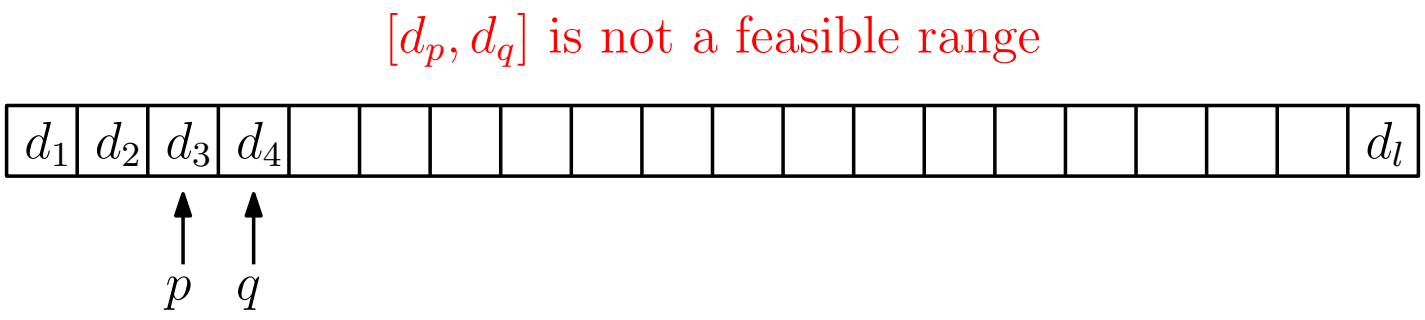
$[d_p, d_q]$ is not a feasible range



- **Martello et al. ('84)**: a general optimization algorithm BOP:
  - Given a **feasibility decider** that decides whether a subset is feasible or not in $f(l)$ time, their algorithm finds an optimal range in $O(lf(l) + l \log l)$-time.
  - Especially useful when an efficient **dynamic version** of the feasibility decider is available.

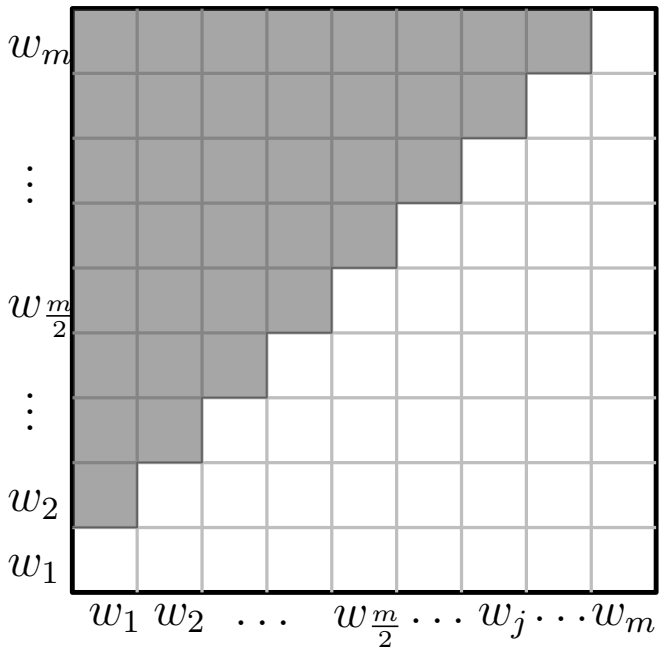


- **Martello et al. ('84)**: a general optimization algorithm BOP:
  - Given a **feasibility decider** that decides whether a subset is feasible or not in $f(l)$ time, their algorithm finds an optimal range in $O(lf(l) + l \log l)$-time.
  - Especially useful when an efficient **dynamic version** of the feasibility decider is available.
- We present an **alternative scheme for BOP** - does not require a dynamic version of the feasibility decider.

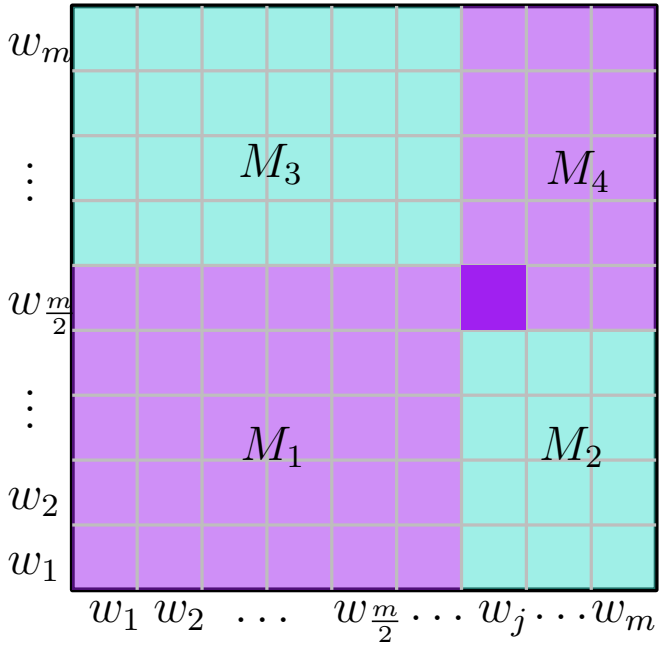


The matrix of ranges

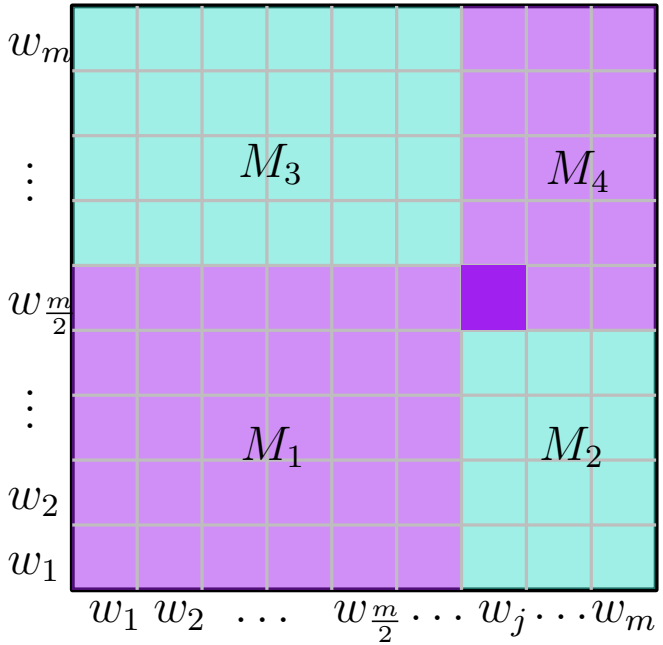
- Let $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- Let M be the matrix whose rows and columns correspond to $w_1, w_2, \ldots, w_m$.
- Cell $M_{i,j} \equiv$ Range $[w_i, w_j]$.



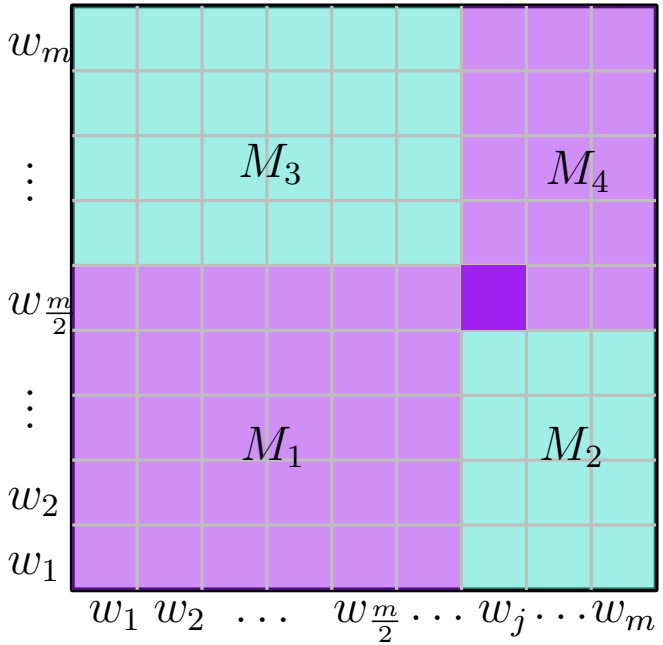
- $M_{i,j}$ is contained in all the ranges $M_{i',j'}$ with $i' \leq i \leq j \leq j'$.



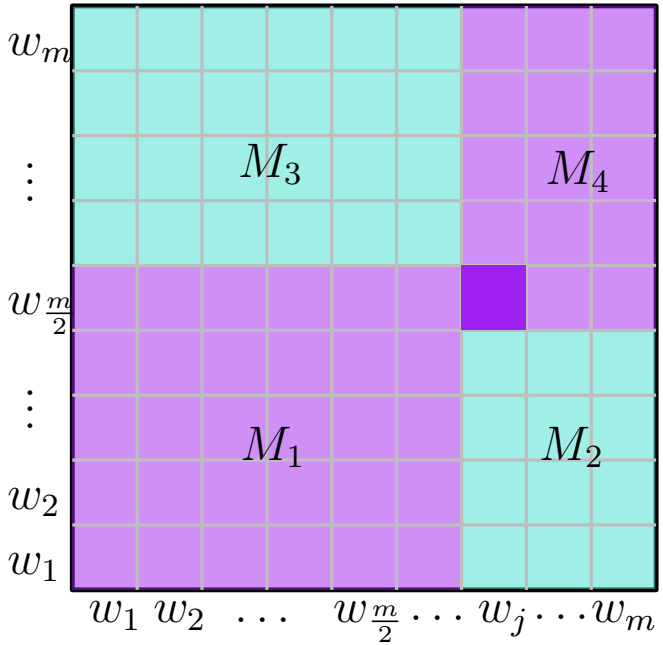
- $M_{i,j}$ is contained in all the ranges $M_{i',j'}$ with $i' \leq i \leq j \leq j'$.
- Perform a binary search in row $\frac{m}{2}$ to find the smallest feasible range $M_{\frac{m}{2},j} = [w_{\frac{m}{2}}, w_j]$ in this row.



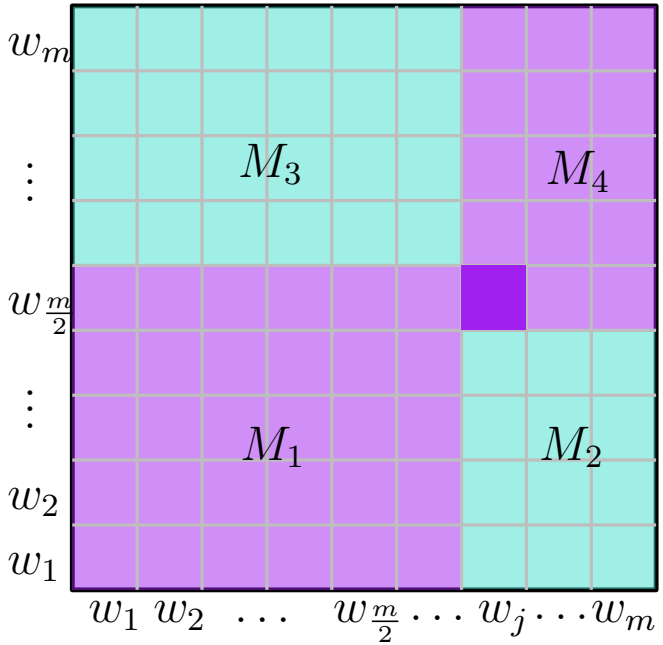
- $M_{i,j}$ is contained in all the ranges $M_{i',j'}$ with $i' \leq i \leq j \leq j'$.
- Perform a binary search in row $\frac{m}{2}$ to find the smallest feasible range $M_{\frac{m}{2},j} = [w_{\frac{m}{2}}, w_j]$ in this row.
- $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4



- None of the ranges in M_1 is a feasible range.

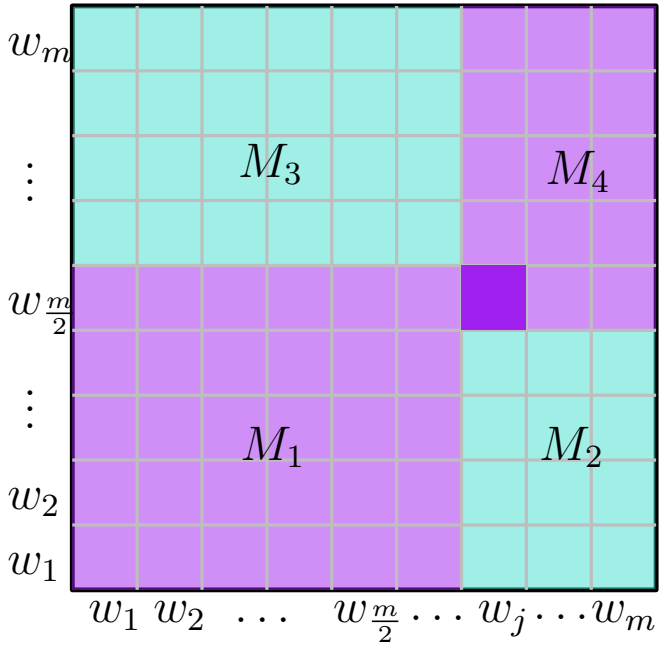


- None of the ranges in M_1 is a feasible range.
- Each of the ranges in M_4 is at least as large as $M_{\frac{m}{2},j}$.



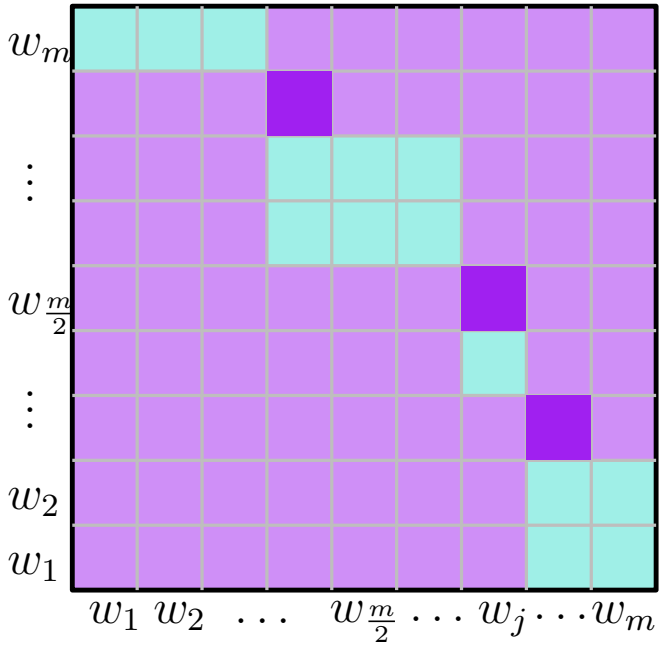
- None of the ranges in M_1 is a feasible range.
- Each of the ranges in M_4 is at least as large as $M_{\frac{m}{2},j}$.

$\Rightarrow$ We may ignore M_1 and M_4 and continue recursively with the submatrices M_2 and M_3.



- None of the ranges in M_1 is a feasible range.
- Each of the ranges in M_4 is at least as large as $M_{\frac{m}{2},j}$.

$\Rightarrow$ We may ignore M_1 and M_4 and continue recursively with the submatrices M_2 and M_3.



The algorithm

Each recursive call is associated with:

- a submatrix $M' = M([p, p'] \times [q', q])$ of M.
- a corresponding graph $G' = G([p, p'] \times [q', q])$.

A recursive call has 2 steps:

1. **Perform a binary search in the middle row** of M' to find the smallest feasible range in this row, using the corresponding graph G'.
2. **Construct two new graphs** for the two submatrices of M' in which we still need to search in the next level of the recursion.

The algorithm

Each recursive call is associated with:

- a submatrix $M' = M([p, p'] \times [q', q])$ of M.
- a corresponding graph $G' = G([p, p'] \times [q', q])$.

A recursive call has 2 steps:

1. **Perform a binary search in the middle row** of M' to find the smallest feasible range in this row, using the corresponding graph G'.
2. **Construct two new graphs** for the two submatrices of M' in which we still need to search in the next level of the recursion.

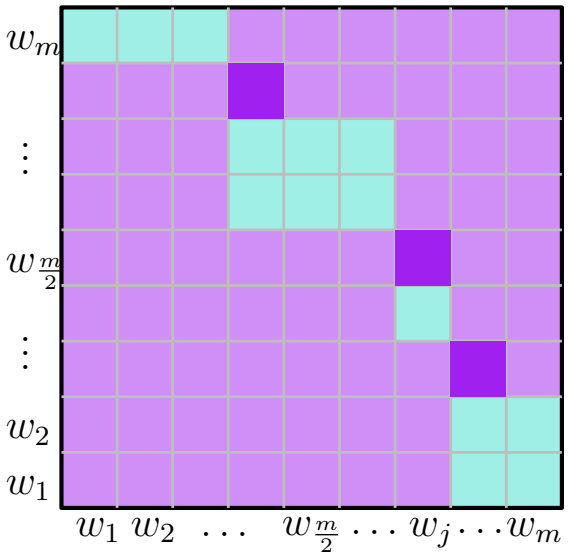
The scheme requires the followings properties of G':

1. The size of G' is $O(|M'|)$.
2. Given G', the feasibility decider can answer a feasibility query for any range in M', in $O(f(|G'|))$ time.
3. Constructing the graphs for the next level takes $O(|G'|)$ time.

Running time

The recursion tree consists of $O(\log m)$ levels.

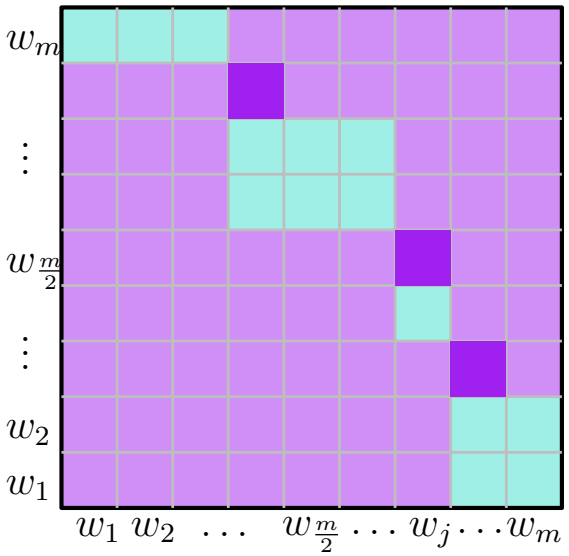
- The i'th level is associated with 2^i disjoint submatrices of M.
- In the i'th level we apply the recursive algorithm to each of the 2^i submatrices associated with this level.



Running time

The recursion tree consists of $O(\log m)$ levels.

- The i'th level is associated with 2^i disjoint submatrices of M.
- In the i'th level we apply the recursive algorithm to each of the 2^i submatrices associated with this level.



- Let $\{M_k^i\}_{k=1}^{2^i}$ be the submatrices associated with the i'th level.
- Let G_k^i be the graph corresponding to M_k^i.

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

- The total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|))$

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

Lemma

The total size of the matrices in each level is at most $2m$.

- The total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|))$

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

Lemma

The total size of the matrices in each level is at most $2m$.

- The total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|)) = O(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))$.

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

Lemma

The total size of the matrices in each level is at most $2m$.

- The total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|)) = O(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))$.
- The running time of the entire algorithm is $O(m\log m + \log m \sum_{i=1}^{\log m}\sum_{k=1}^{2^i} f(|M_k^i|))$.

In each recursive call:

1. The size of G_k^i is $O(M_k^i)$.
2. The feasibility decider runs in $O(f(|G_k^i|))$ time $\Rightarrow$ the binary search in M_k^i runs in $O(f(|M_k^i|)\log|M_k^i|)$ time.
3. Constructing the graphs for the next level takes $O(|M_k^i|)$ time.

Lemma

The total size of the matrices in each level is at most $2m$.

- The total time spent on the i'th level is $O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|)) = O(m + \log m \sum_{k=1}^{2^i} f(|M_k^i|))$.
- The running time of the entire algorithm is $O(m\log m + \log m \sum_{i=1}^{\log m}\sum_{k=1}^{2^i} f(|M_k^i|))$.

if $f(|M_k^i|) = O(|M_k^i|)$, then we get $O(m\log^2 n)$.

The Most Uniform Path problem (MUPP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUPP

Given two vertices $s, t \in V$, find a path P^* in G between s and t, which minimizes $P_{max} - P_{min}$, over all paths P between s and t.

The Most Uniform Path problem (MUPP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUPP

Given two vertices $s, t \in V$, find a path P^* in G between s and t, which minimizes $P_{max} - P_{min}$, over all paths P between s and t.

- Introduced by **Hansen et al. ('97)**: $O(m^2)$-time algorithm.

The Most Uniform Path problem (MUPP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUPP

Given two vertices $s, t \in V$, find a path P^* in G between s and t, which minimizes $P_{max} - P_{min}$, over all paths P between s and t.

- Introduced by **Hansen et al. ('97)**: $O(m^2)$-time algorithm.
- Observation: using a dynamic connectivity data structure for general graphs of **Holm et al.('01)** we can get $O(m \log^2 n)$-time algorithm.

The Most Uniform Path problem (MUPP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUPP

Given two vertices $s, t \in V$, find a path P^* in G between s and t, which minimizes $P_{max} - P_{min}$, over all paths P between s and t.

- Introduced by **Hansen et al. ('97)**: $O(m^2)$-time algorithm.
- Observation: using a dynamic connectivity data structure for general graphs of **Holm et al.('01)** we can get $O(m \log^2 n)$-time algorithm.
- Our scheme: simpler algorithm with the same running time.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. Maintain the following properties of G':

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. Maintain the following properties of G':
  1. The size of G' is at most $O(|M'|)$.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. Maintain the following properties of G':
  1. The size of G' is at most $O(|M'|)$.
  2. Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. Maintain the following properties of G':
  1. The size of G' is at most $O(|M'|)$.
  2. Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.
  3. Constructing the graphs for the next level takes $O(|G'|)$ time.

Applying our BOP scheme:

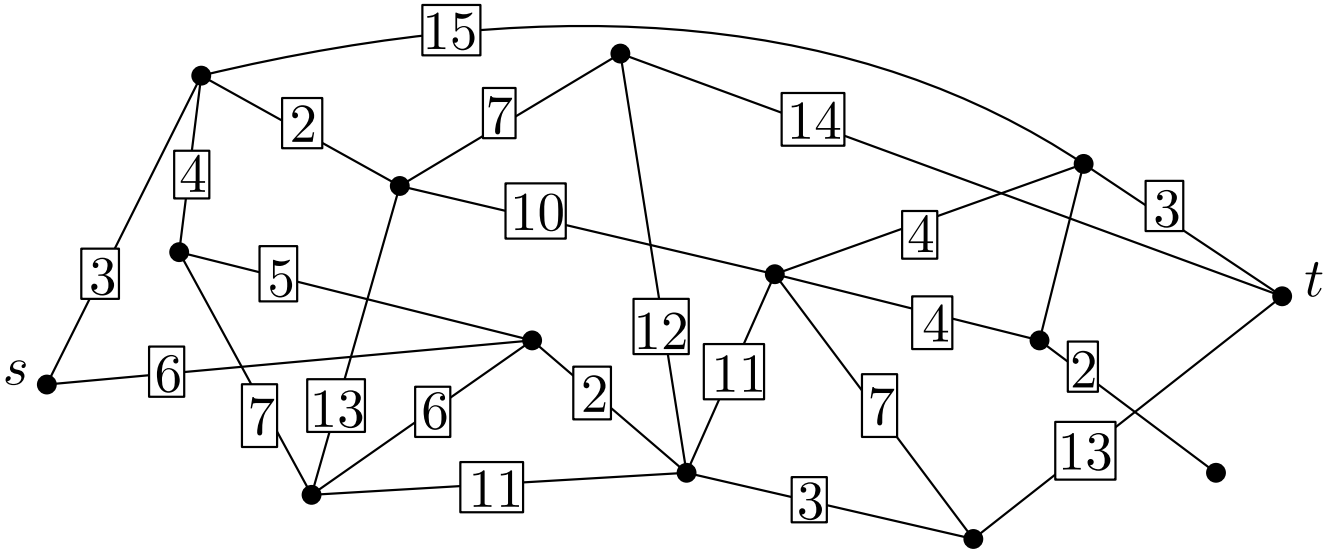
- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. Maintain the following properties of G':
  1. The size of G' is at most $O(|M'|)$.
  2. Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.
  3. Constructing the graphs for the next level takes $O(|G'|)$ time.
- The **feasibility decider**: a BFS algorithm (which ignores edges outside the given range) runs in $O(|G'|)$ time.

Applying our BOP scheme:

- A **feasible subset** $S \in \mathcal{F}$ is a **path** in G between s and t.
- Let $E = \{e_1, \ldots, e_m\}$,
  $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.
- The matrix for the **initial call** is M, and G is its associated graph.
- In a **recursive call**: Let M' be the submatrix and G' the graph associated with it. **Maintain the following properties of G'**:
  1. The size of G' is at most $O(|M'|)$.
  2. Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.
  3. Constructing the graphs for the next level takes $O(|G'|)$ time.
- The **feasibility decider**: a BFS algorithm (which ignores edges outside the given range) runs in $O(|G'|)$ time.

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

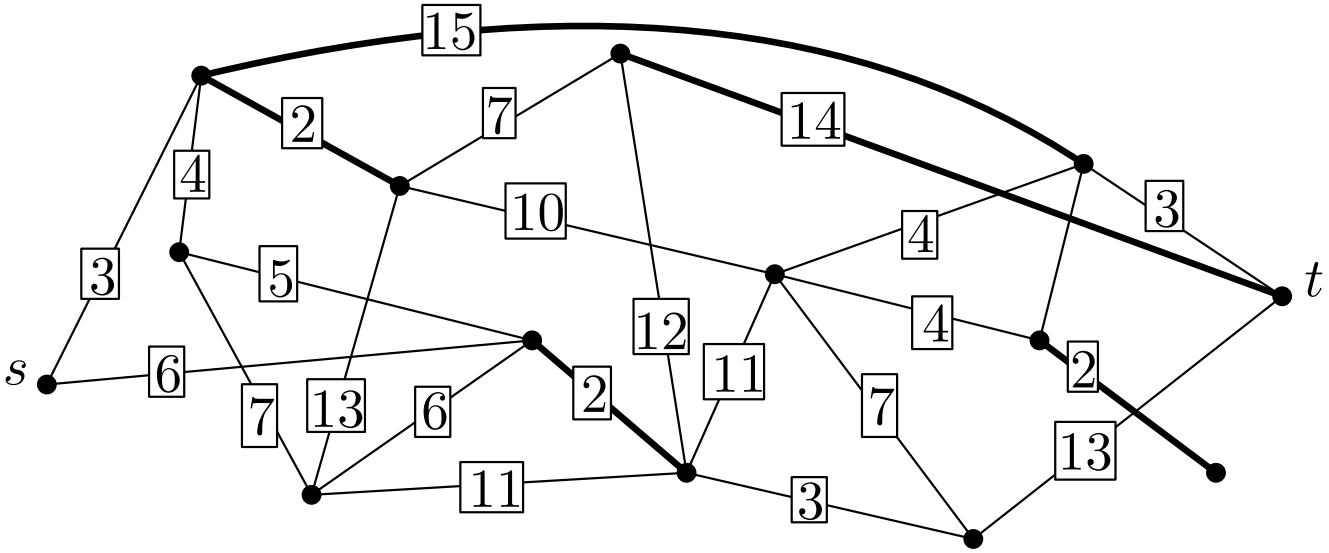
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. Remove all the isolated vertices.



$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

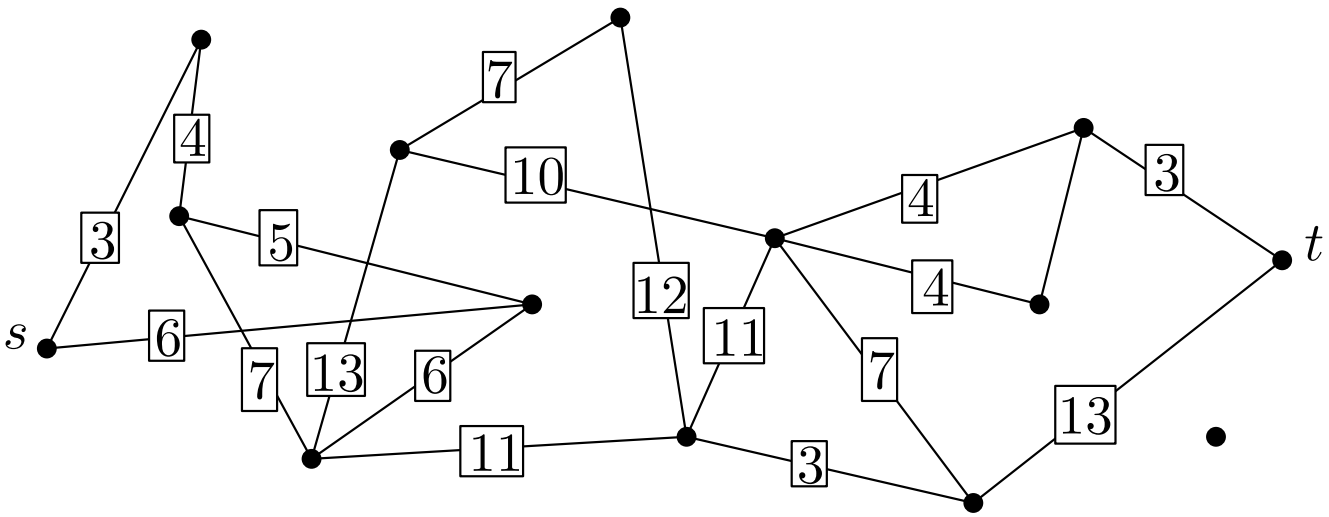
1. **Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.**
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. Remove all the isolated vertices.



$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

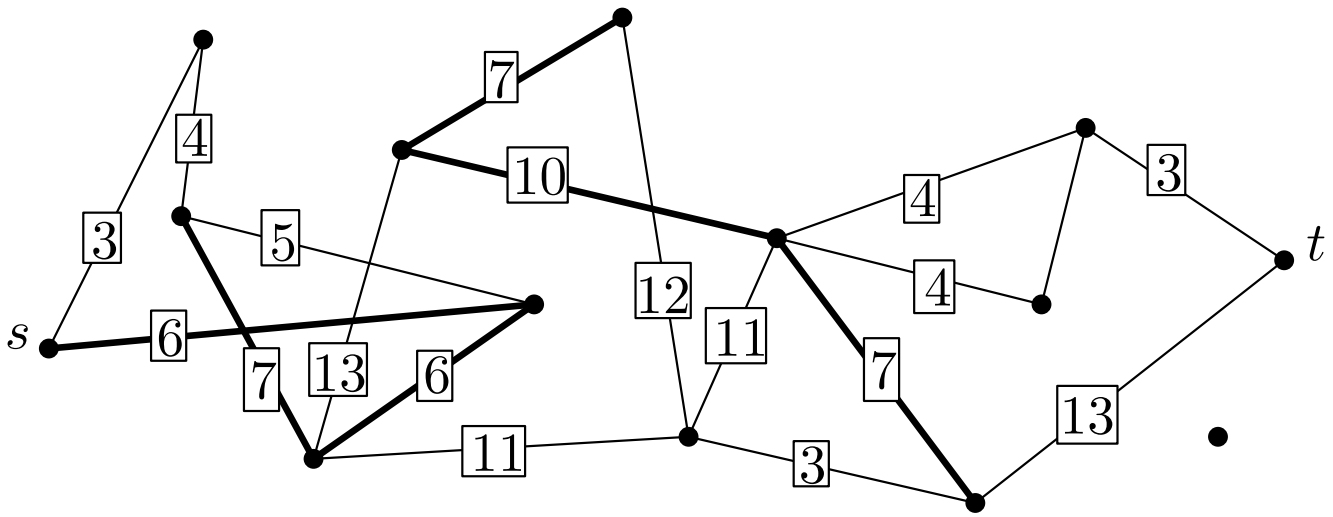
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. Remove all the isolated vertices.



$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

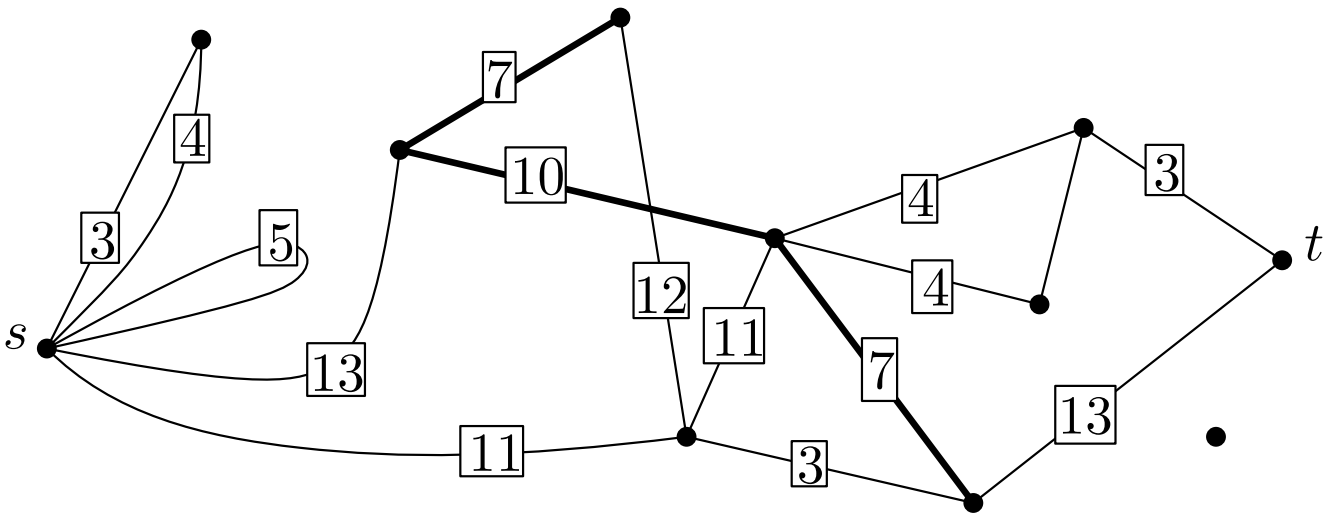
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. **Contract edges with weights in the range $(w_{p'}, w_{q'})$.**
3. Remove all the isolated vertices.



$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

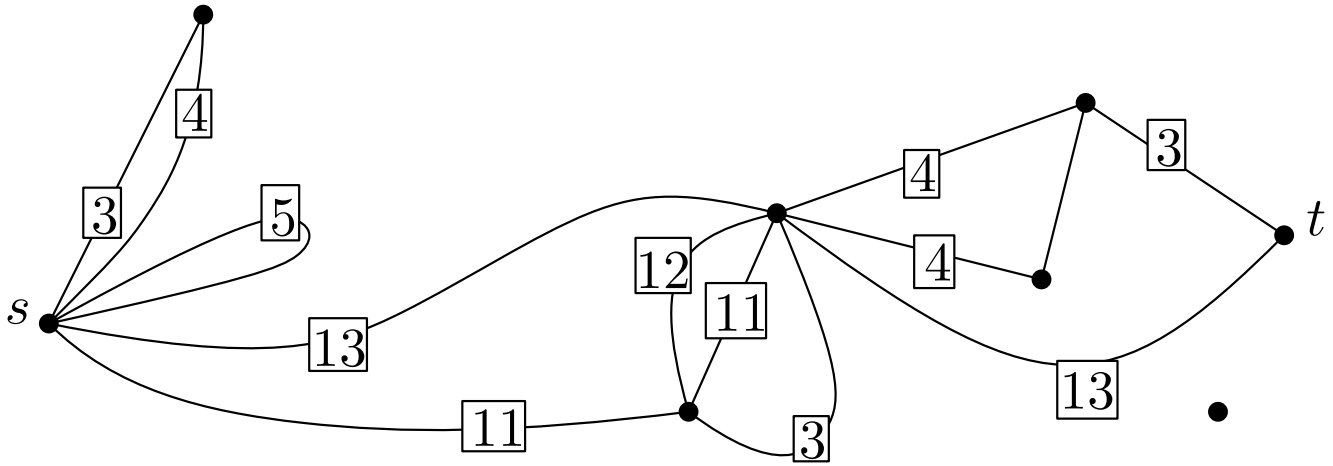
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. **Contract edges with weights in the range $(w_{p'}, w_{q'})$.**
3. Remove all the isolated vertices.



$$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

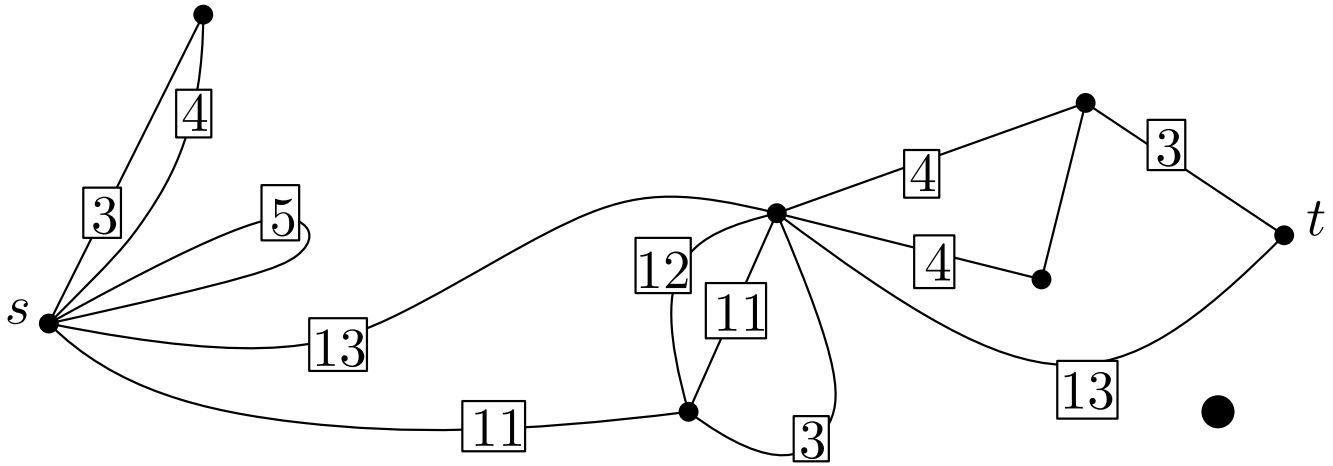
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. **Contract edges with weights in the range $(w_{p'}, w_{q'})$.**
3. Remove all the isolated vertices.



$$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

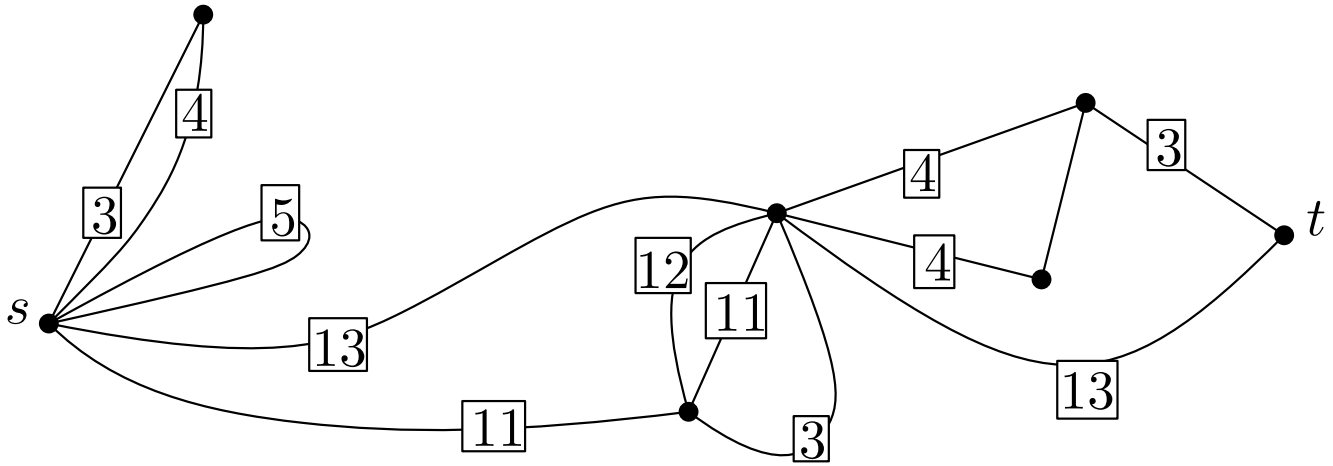
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. **Remove all the isolated vertices.**



$$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

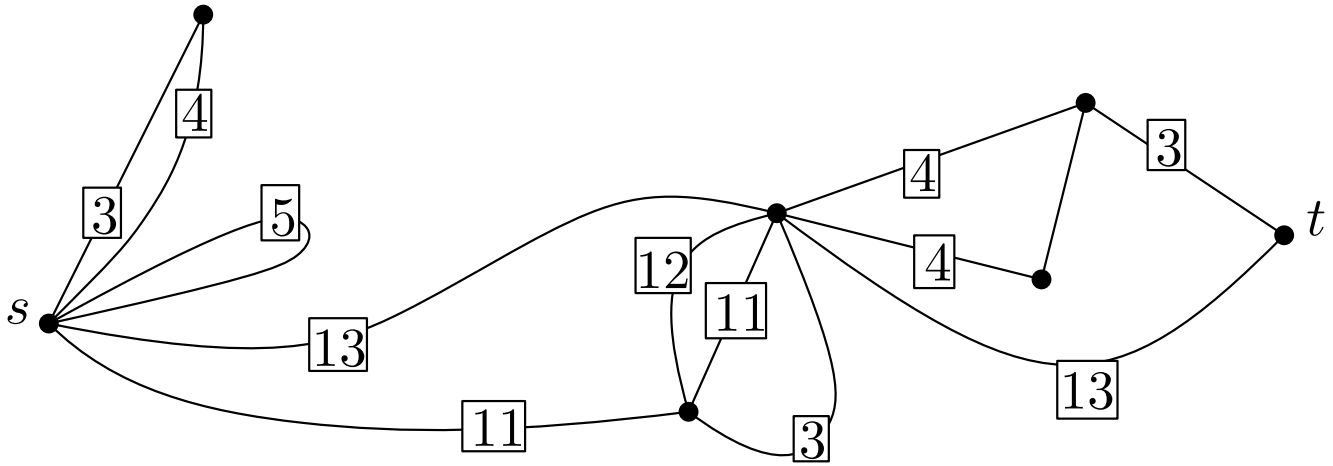
1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. **Remove all the isolated vertices.**



$$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$$

Construction of the graph $G'' = G([p, p'] \times [q', q])$, given G':

1. Remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$.
2. Contract edges with weights in the range $(w_{p'}, w_{q'})$.
3. Remove all the isolated vertices.



$$G' = G([2, 7] \times [10, 15]) \Rightarrow G'' = G([3, 5] \times [11, 13])$$

Running time: $O(|G'|)$.

WDFD under translation in 1D

WDFD under translation in 1D can be viewed as BOP:
$[s, t]$ is a **feasible range** $\Leftrightarrow$
(a_n, b_m) is a **w-reachable position** in G_w w.r.t. $\sigma_{[s,t]}$.

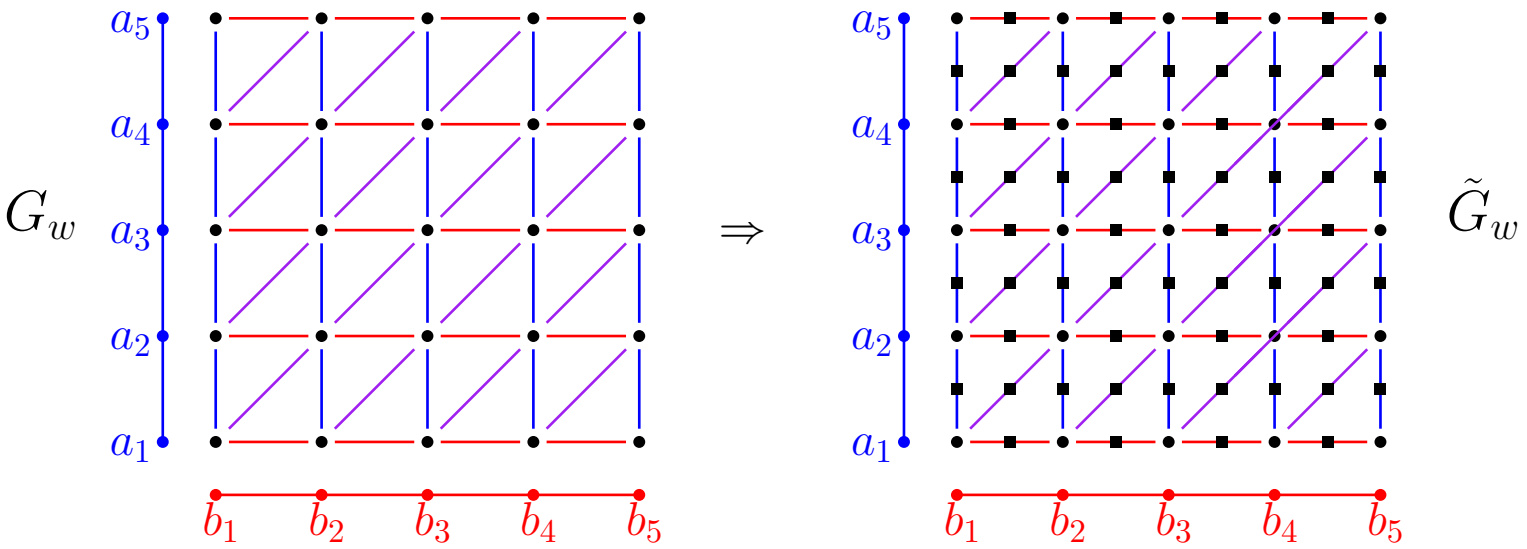
WDFD under translation in 1D

WDFD under translation in 1D can be viewed as BOP:
$[s, t]$ is a **feasible range** $\Leftrightarrow$
(a_n, b_m) is a **w-reachable position** in G_w w.r.t. $\sigma_{[s,t]}$.

WDFD under translation in 1D is a special case of MUPP!

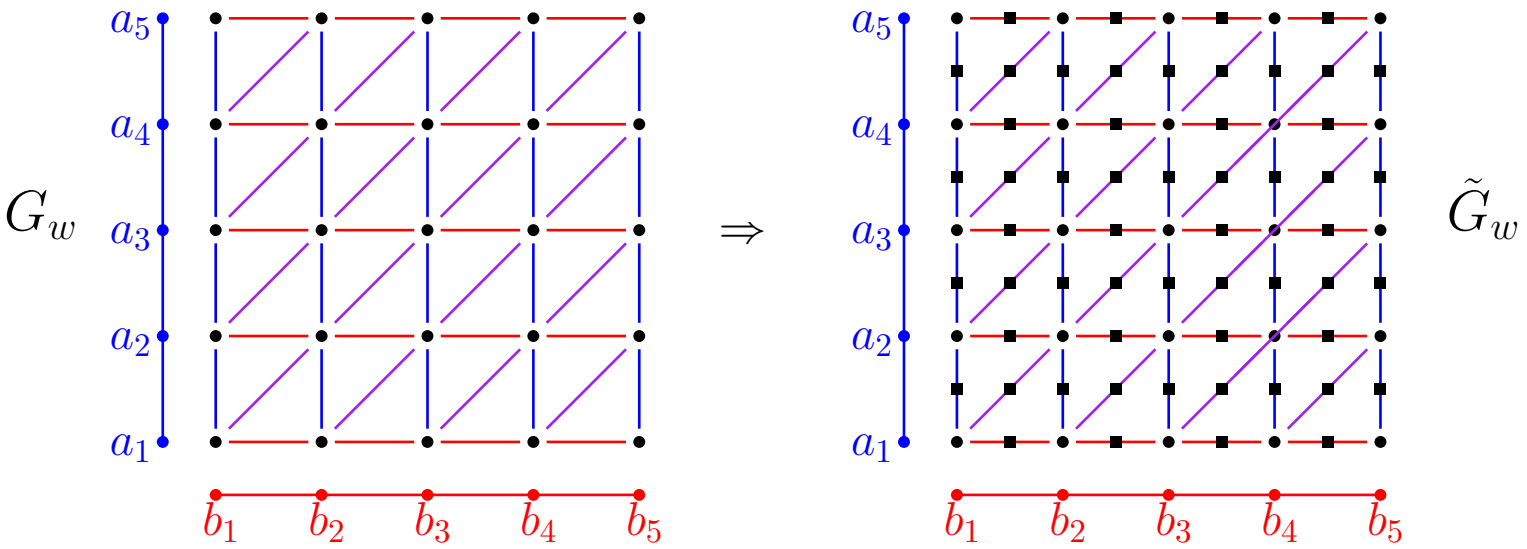
MUPP $\Rightarrow$ WDFD under translation in 1D

For MUPP we need a **weighted graph** $\tilde{G}_w = (\tilde{V}_w, \tilde{E}_w, \omega)$
$\tilde{V}_w = (A \times B) \cup \{v_e \mid e \in E_w\}$, $\tilde{E}_w = \{(u, v_e), (v_e, v) \mid e = (u, v) \in E_w\}$,
and $\omega(((a_i, b_j), v_e)) = a_i - b_j$.



MUPP $\Rightarrow$ WDFD under translation in 1D

For MUPP we need a **weighted graph** $\tilde{G}_w = (\tilde{V}_w, \tilde{E}_w, \omega)$
$\tilde{V}_w = (A \times B) \cup \{v_e \mid e \in E_w\}$, $\tilde{E}_w = \{(u, v_e), (v_e, v) \mid e = (u, v) \in E_w\}$,
and $\omega(((a_i, b_j), v_e)) = a_i - b_j$.



(a_n, b_m) is a **w-reachable position** in G_w w.r.t. $\sigma_{[s,t]} \Leftrightarrow$
there exists a path $\tilde{P}$ between (a_1, b_1) and (a_n, b_m) in $\tilde{G}_w$ such that for each edge $e \in \tilde{P}$, $\omega(e) \in [s, t]$.

MUPP $\Rightarrow$ WDFD under translation in 1D

MUPP between (a_1, b_1) and (a_n, b_m) in $\tilde{G}_w \Rightarrow$
WDFD under translation in 1D.

Theorem

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points in 1D. Then, the weak discrete Fréchet distance under translation, $\widehat{d^w_{dF}}(A, B)$, can be computed in $O(mn \log^2(m+n))$ time.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.
  - A **feasible subset** $S \in \mathcal{F}$ is a **spanning tree** of G.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.
  - A **feasible subset** $S \in \mathcal{F}$ is a **spanning tree** of G.
  - The **construction** of G'': similar to the construction in MUPP.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.
  - A **feasible subset** $S \in \mathcal{F}$ is a **spanning tree** of G.
  - The **construction** of G'': similar to the construction in MUPP.
  - The **feasibility decider**: check that G' has a connected spanning subgraph with edges in the given range.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.
  - A **feasible subset** $S \in \mathcal{F}$ is a **spanning tree** of G.
  - The **construction** of G'': similar to the construction in MUPP.
  - The **feasibility decider**: check that G' has a connected spanning subgraph with edges in the given range.

The Most Uniform Spanning Tree problem (MUTP)

$G = (V, E, w)$ is a weighted graph with n vertices and m edges.

Definition: MUTP

Find a spanning tree T^* of G, which minimizes $T_{max} - T_{min}$, over all spanning trees T of G.

- **Galil and Schieber ('88)**: $O(m \log n)$-time algorithm for the problem, using an involved dynamic data structure.
- Using our optimization scheme: $O(m \log^2 n)$ time algorithm.
  - A **feasible subset** $S \in \mathcal{F}$ is a **spanning tree** of G.
  - The **construction** of G'': similar to the construction in MUPP.
  - The **feasibility decider**: check that G' has a connected spanning subgraph with edges in the given range.

Our algorithm: slower by a factor of $\log n$, BUT does not require any special data structures, and has easy and shorter description.

A new variant: The discrete Fréchet gap

Two frogs, two curves: $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$.
Same rules: traverse all the points in order, no backtracking.

New indicator function: given a range $[s, t]$,

$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq d(a, b) \leq t \\ 0, & \text{otherwise} \end{cases}$$

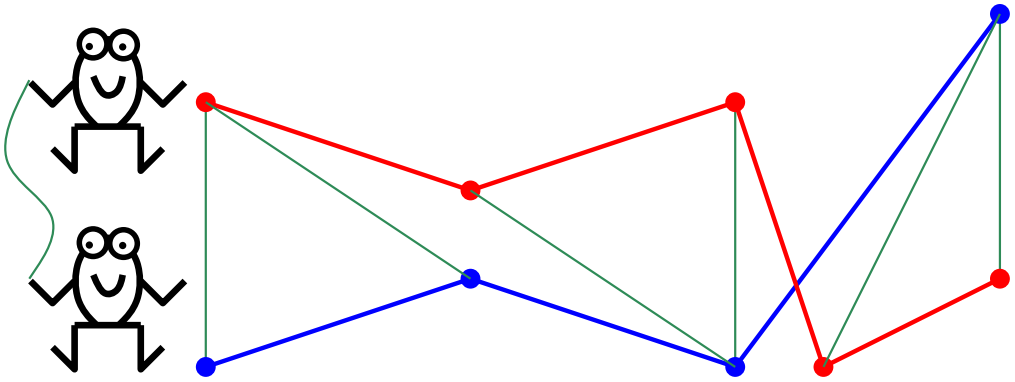
A new variant: The discrete Fréchet gap

Two frogs, two curves: $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$.
Same rules: traverse all the points in order, no backtracking.

New indicator function: given a range $[s, t]$,

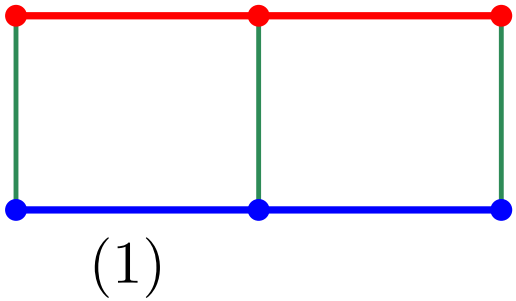
$$\sigma_{[s,t]}(a, b) = \begin{cases} 1, & s \leq d(a, b) \leq t \\ 0, & \text{otherwise} \end{cases}$$

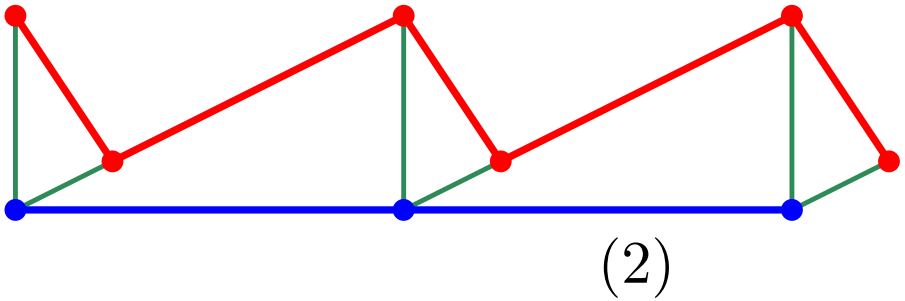
- The **discrete Fréchet gap** ($d^g_{dF}(A, B)$) is the smallest range $[s, t]$, $s \geq t \geq 0$, for which (a_n, b_m) is a reachable position w.r.t. $\sigma_{[s,t]}$.

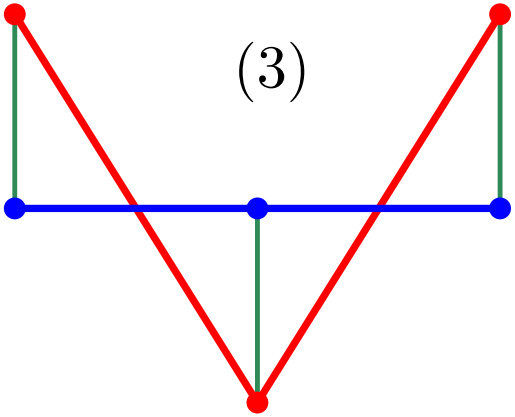


Is this a better variant?

- Gives a better reflection of reality?

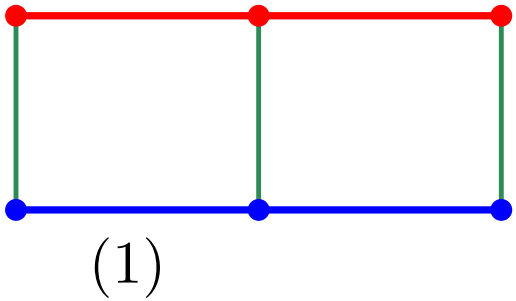


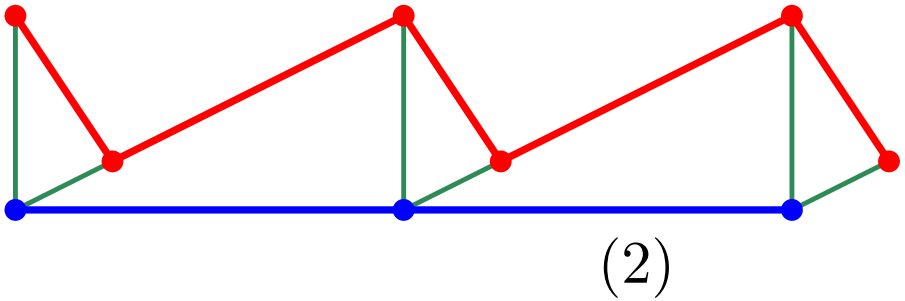


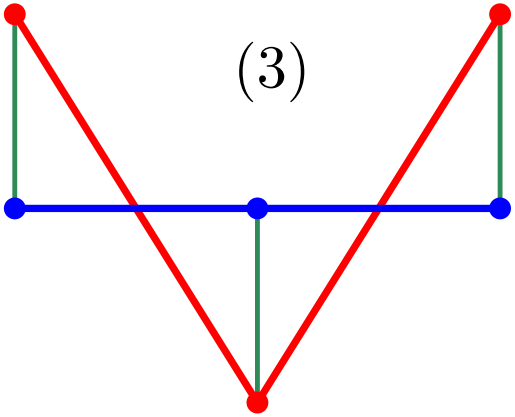


Is this a better variant?

- Gives a better reflection of reality? sometimes, but not always.

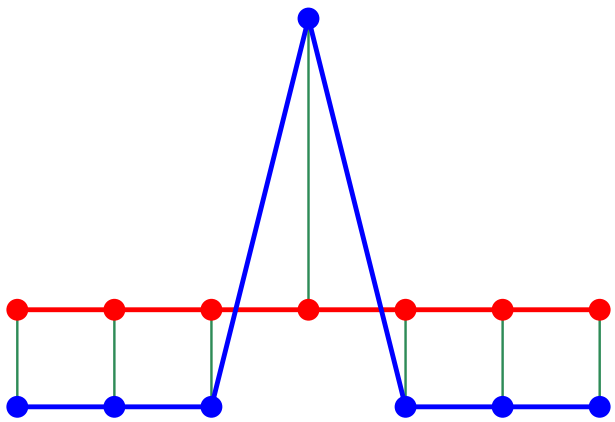






Is this a better variant?

- Gives a better reflection of reality? sometimes, but not always.
- Handling **outliers**? no.



Let us combine gap and shortcuts...

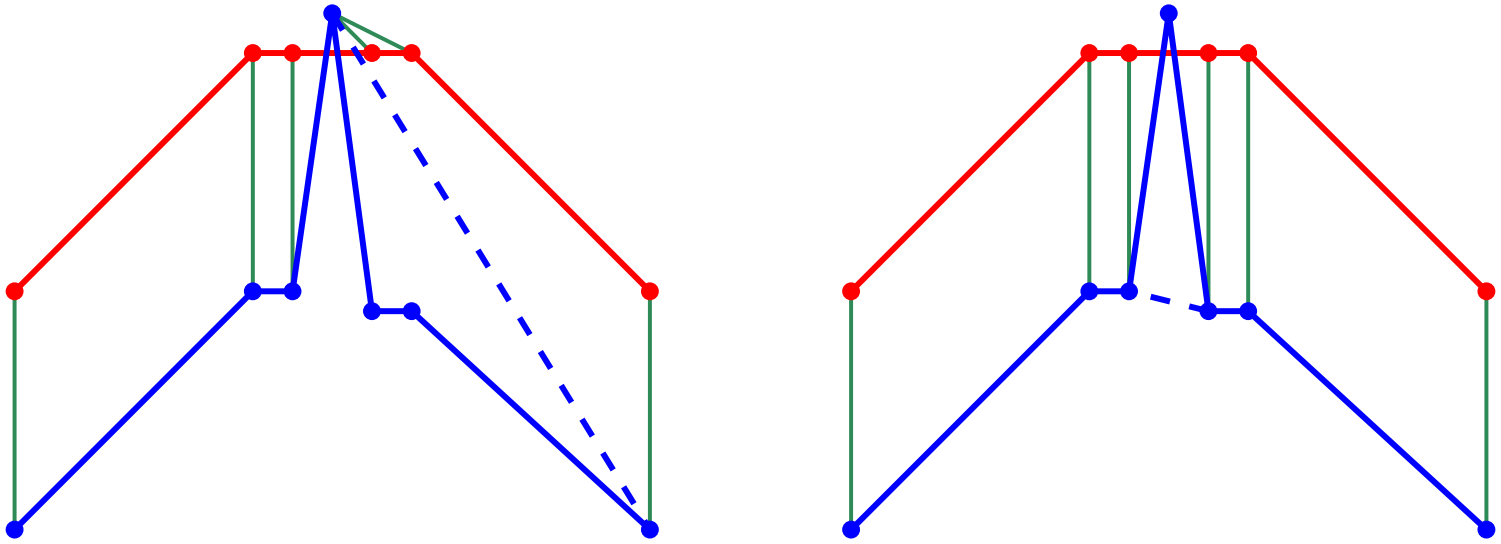
$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$, a leash of length δ.
Only the A-frog can skip points.

- The **(one-sided) discrete Fréchet gap with shortcuts** is the smallest range $[s, t]$, $s \geq t \geq 0$, for which (a_n, b_m) is an s-reachable position w.r.t. $\sigma_{[s,t]}$.

Let us combine gap and shortcuts...

$A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$, a leash of length δ.
Only the A-frog can skip points.

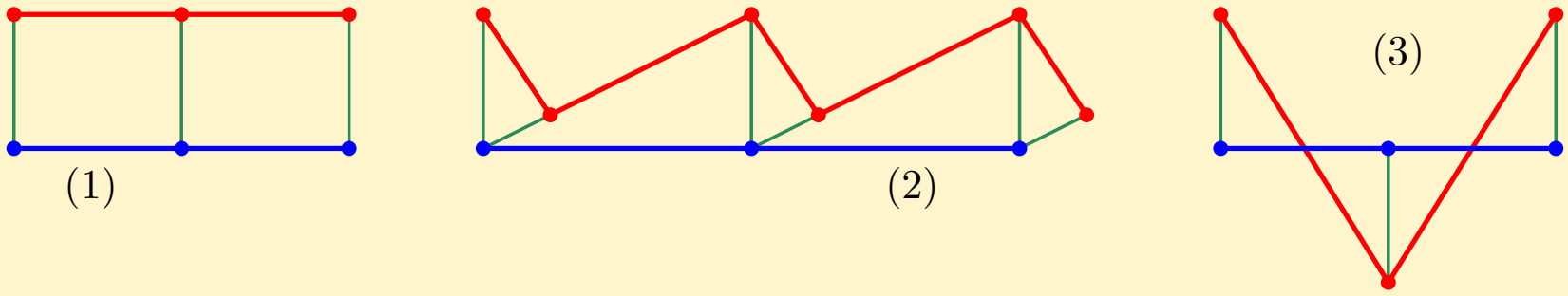
- The **(one-sided) discrete Fréchet gap with shortcuts** is the smallest range $[s, t]$, $s \geq t \geq 0$, for which (a_n, b_m) is an s-reachable position w.r.t. $\sigma_{[s,t]}$.



That seems to give better results!

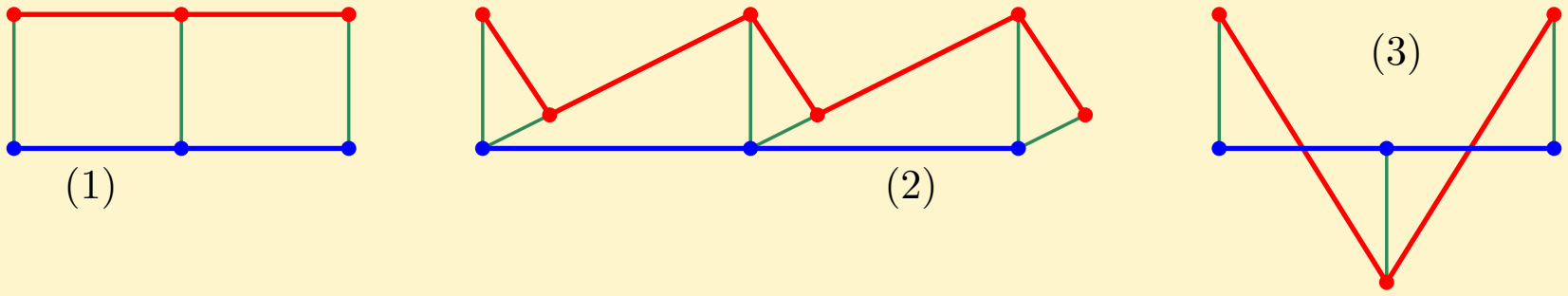
What is between Gap and Translation?

Is there some connection between the discrete Fréchet gap and the discrete Fréchet distance under translation?



What is between Gap and Translation?

Is there some connection between the discrete Fréchet gap and the discrete Fréchet distance under translation?



DFDS and WDFD, both in 1D under translation, are in some sense analogous to their respective gap variants (in d dimensions and no translation):

- We can use **similar algorithms** to compute them, but with **different indicator functions**!

Thank You!