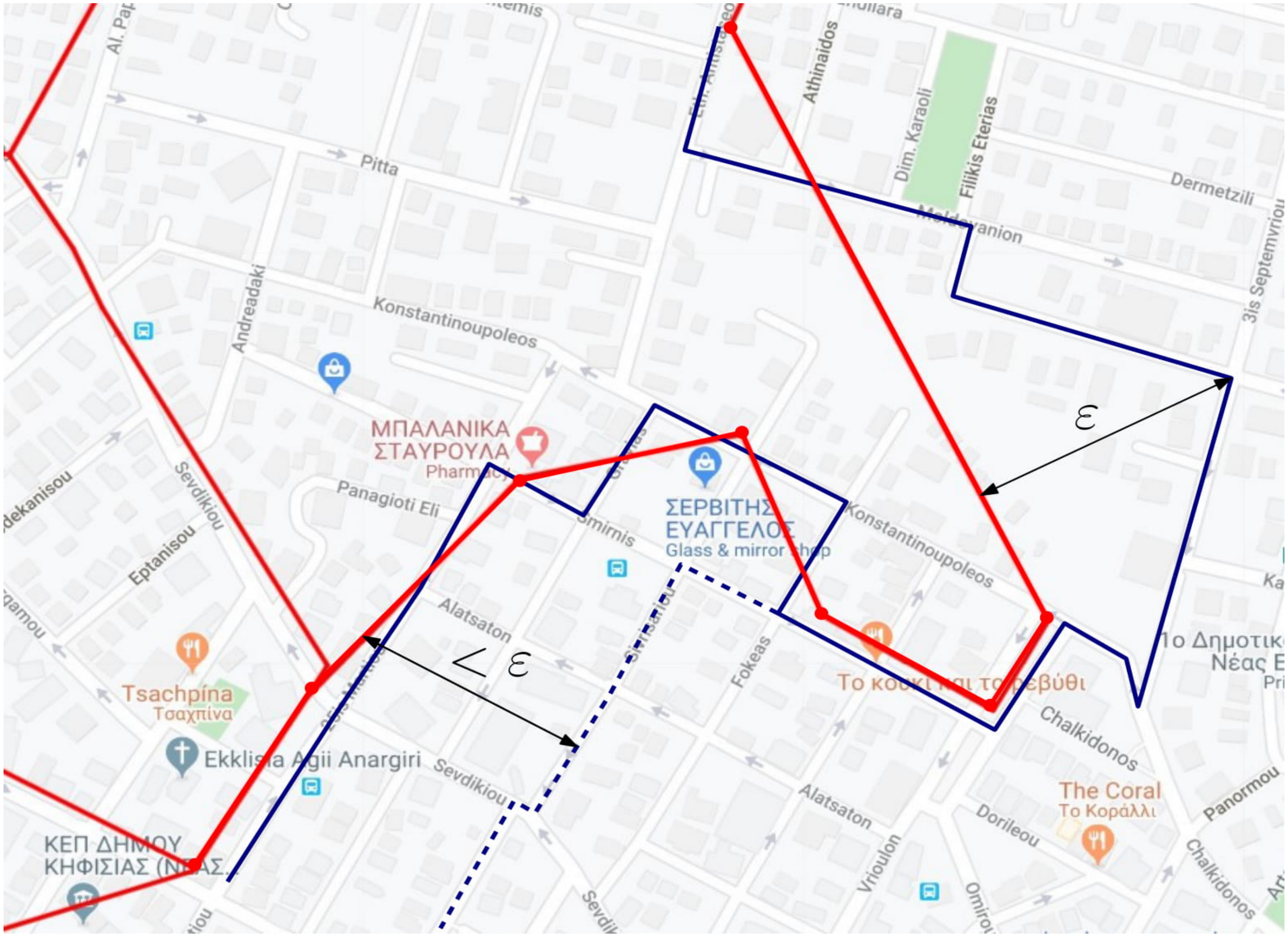
Graph Distance - Optimal Mappings

Bernhard Kilgus
joint work with Maike Buchin

RUHR-UNIVERSITÄT BOCHUM

Comparing Embedded Graphs

Motivation: road networks reconstructed from trajectory data



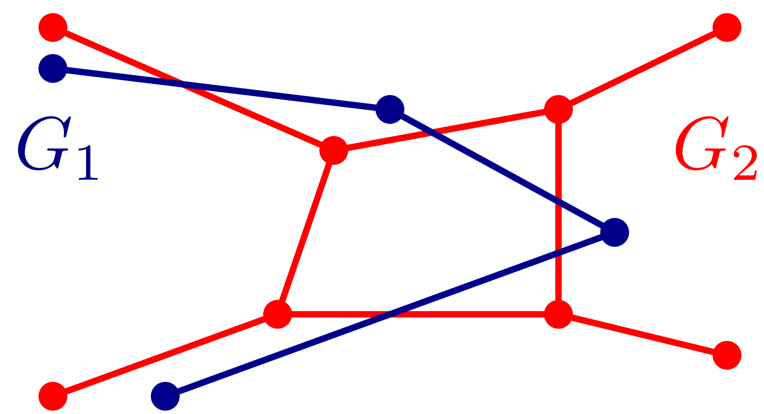
Our approach

RUB

Map one graph on part of the other and compare geometrically

A map $s\colon G_1 \to G_2$ is a *graph mapping* if it maps

- each vertex $v \in V_1$ to a point $s(v)$ on an edge of G_2, and
- each edge $\{u, v\} \in E_1$ to a simple path from $s(u)$ to $s(v)$ in G_2



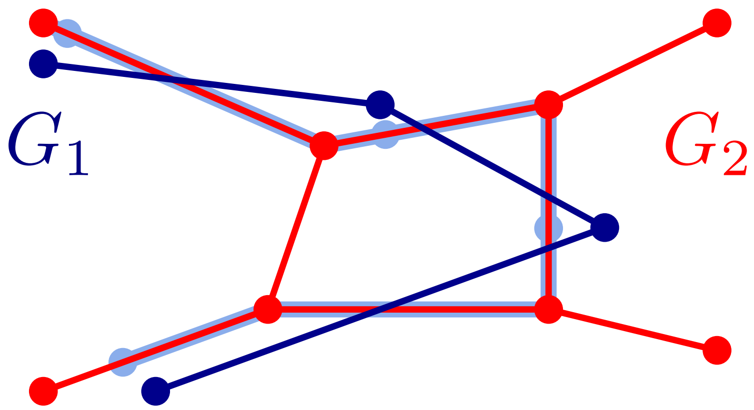
Our approach

RUB

Map one graph on part of the other and compare geometrically

A map $s\colon G_1 \to G_2$ is a *graph mapping* if it maps

- each vertex $v \in V_1$ to a point $s(v)$ on an edge of G_2, and
- each edge $\{u, v\} \in E_1$ to a simple path from $s(u)$ to $s(v)$ in G_2



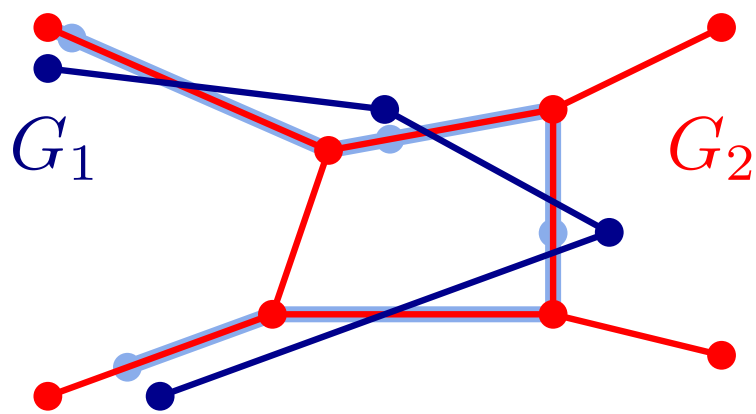
Our approach

RUB

Map one graph on part of the other and compare geometrically

A map $s\colon G_1 \to G_2$ is a *graph mapping* if it maps

- each vertex $v \in V_1$ to a point $s(v)$ on an edge of G_2, and
- each edge $\{u, v\} \in E_1$ to a simple path from $s(u)$ to $s(v)$ in G_2



We define the *(weak) directed graph distance* as

$$\vec{\delta}_{(w)G}(G_1, G_2) := \inf_{s:G_1 \to G_2} \max_{e \in E_1} \delta_{(w)F}(e, s(e)),$$

where s ranges over all graph mappings from G_1 to G_2 and $\delta_{(w)F}$ denotes the (weak) Fréchet distance.

Fréchet Distance

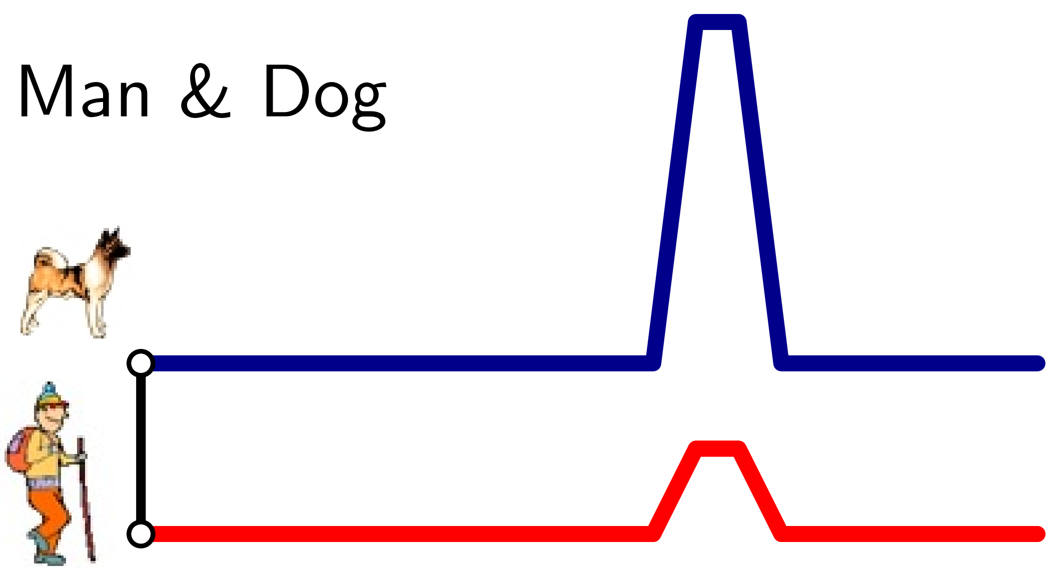
Definition:

$P, Q\colon [0,n] \to \mathbb{R}^d$ parameterised curves

$$\delta_F(P,Q) := \inf_{\sigma:[0,n]\to[0,n]} \max_{x\in[0,n]} d(P(x), Q(\sigma(x)))$$

homeomorphism

Illustration: Man & Dog



Fréchet distance equals shortest leash length

Fréchet Distance

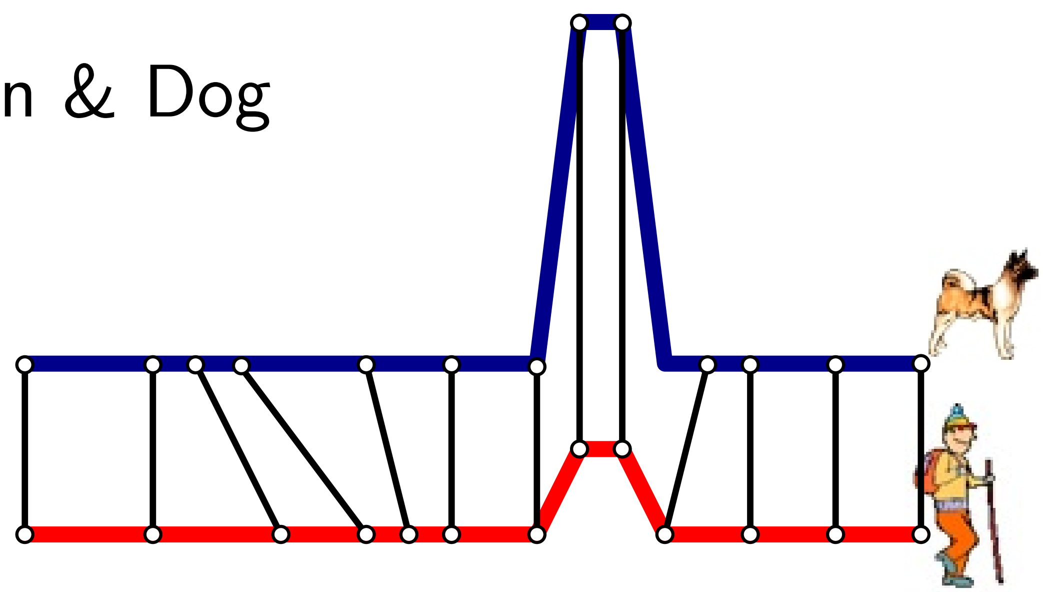
Definition:

$P, Q \colon [0, n] \to \mathbb{R}^d$ parameterised curves

$$\delta_F(P,Q) := \inf_{\sigma:[0,n]\to[0,n]} \max_{x\in[0,n]} d(P(x), Q(\sigma(x)))$$

homeomorphism

Illustration: Man & Dog



Fréchet distance equals shortest leash length

Previous Results

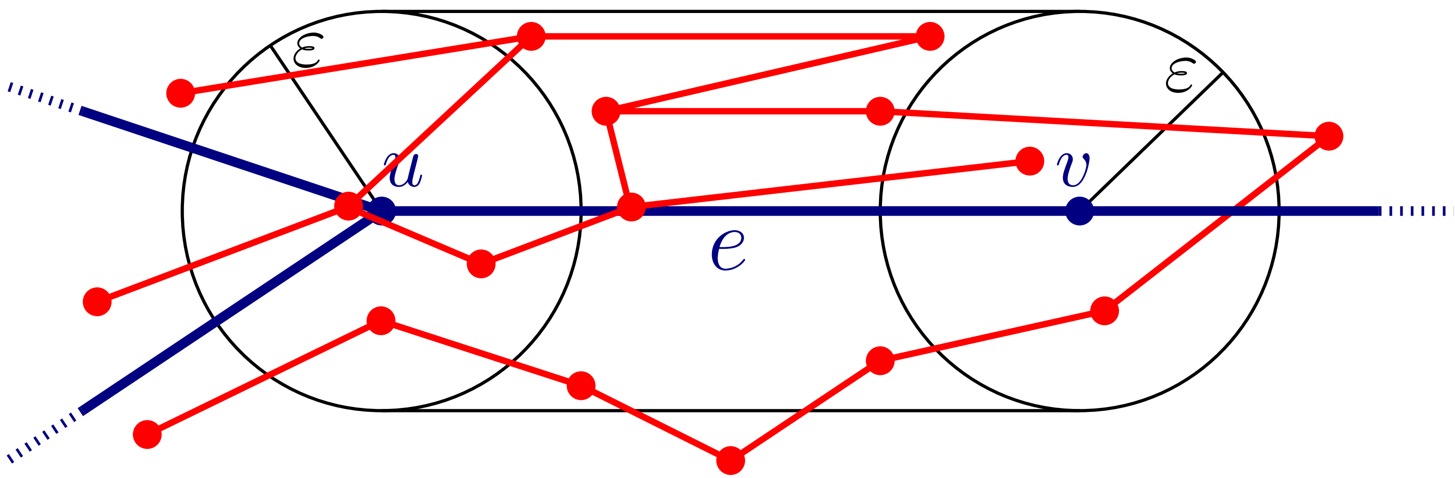
	Directed Weak Graph Distance	Directed Graph Distance
General Graphs	NP-Hard	NP-Hard
Plane Graphs	P	NP-Hard
G_1: Tree	P	P

Algorithmic Approach

RUB

Steps to to decide the (weak) graph distance:

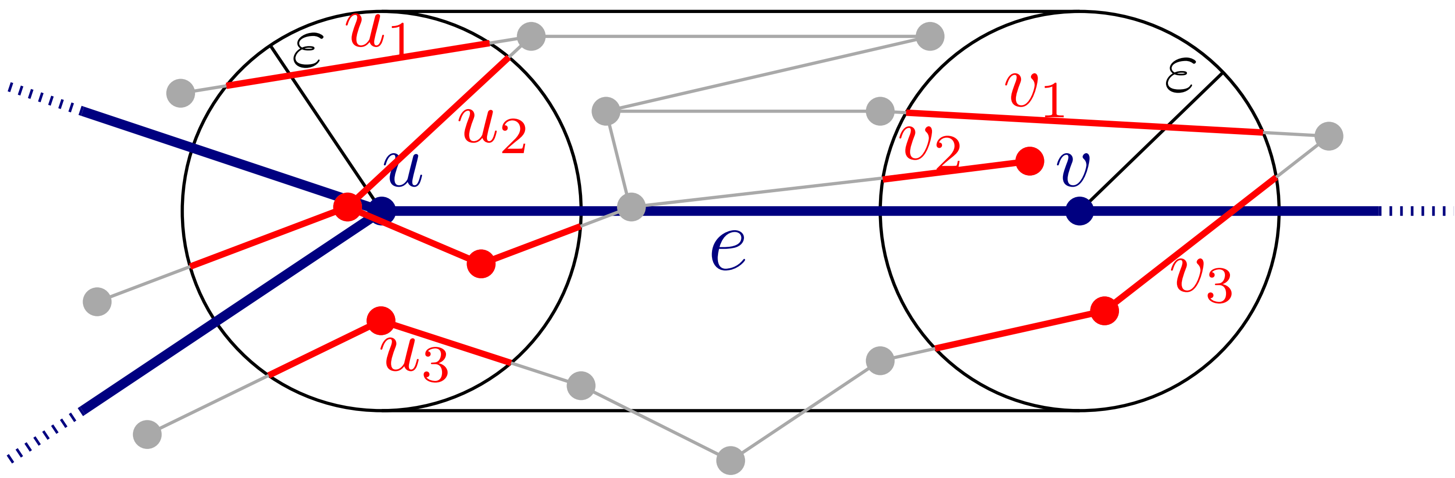
- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements



Algorithmic Approach

Steps to to decide the (weak) graph distance:

- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements

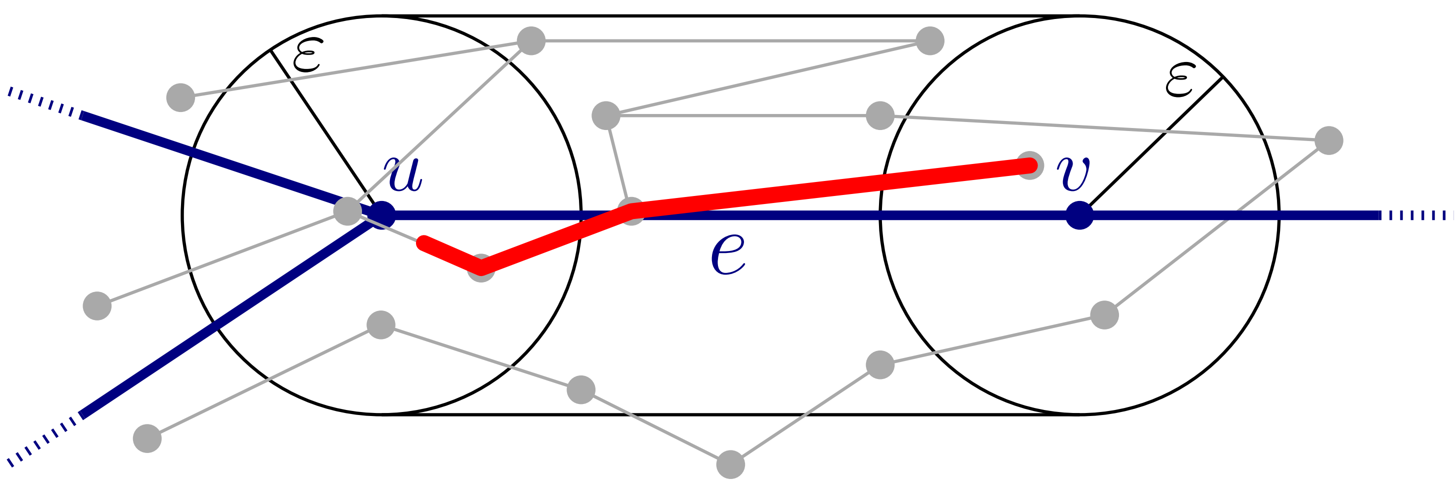


Algorithmic Approach

RUB

Steps to to decide the (weak) graph distance:

- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements

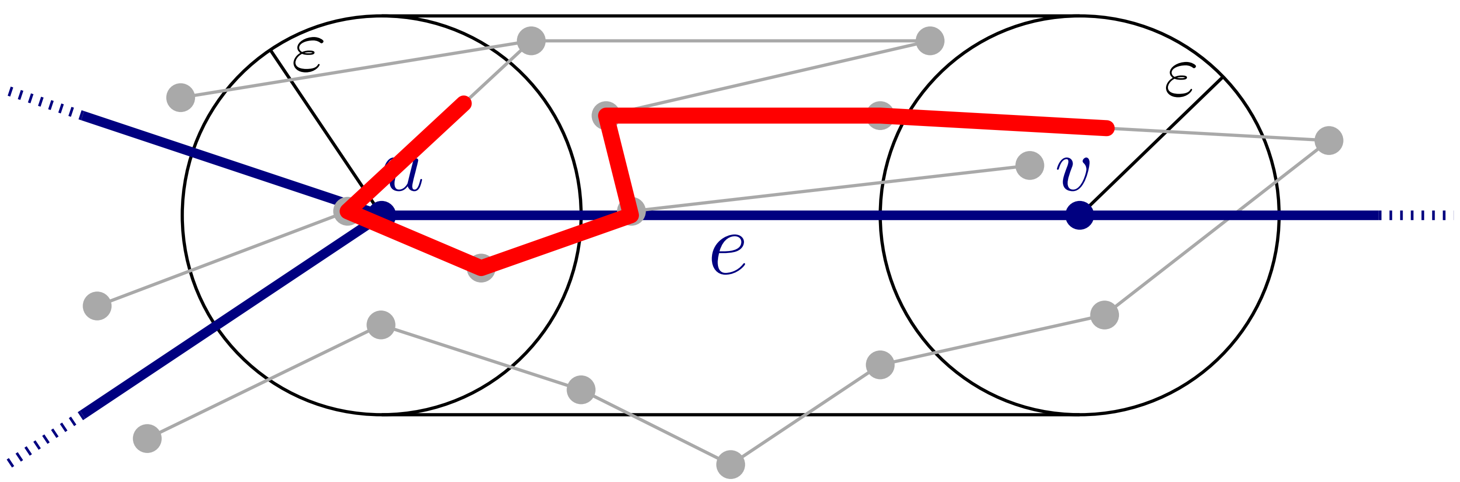


Algorithmic Approach

RUB

Steps to to decide the (weak) graph distance:

- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements

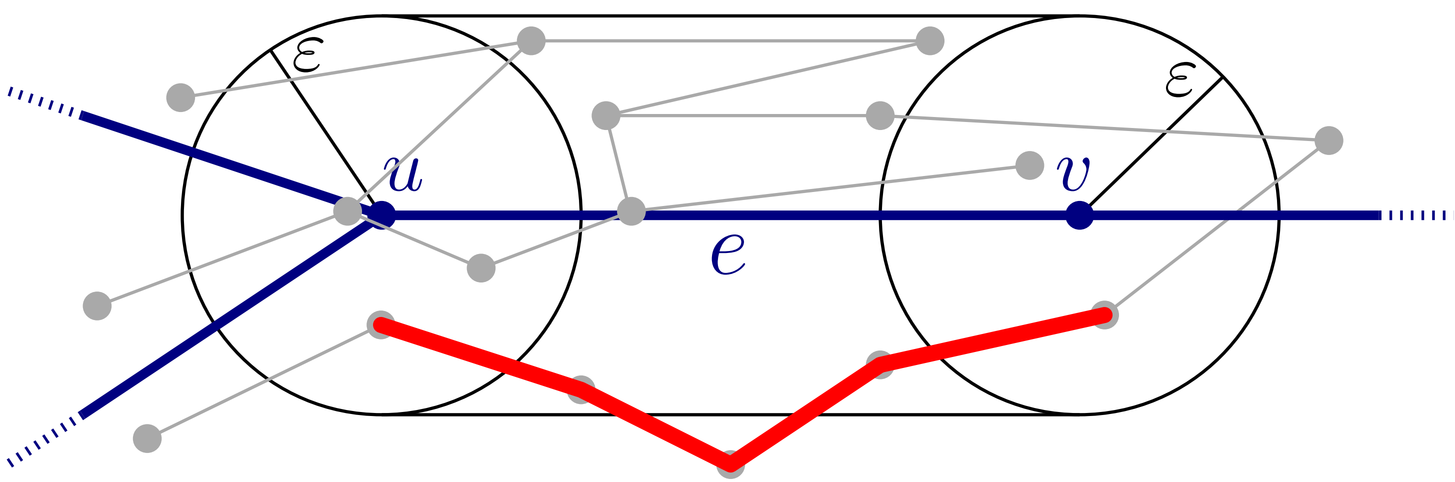


Algorithmic Approach

RUB

Steps to to decide the (weak) graph distance:

- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements

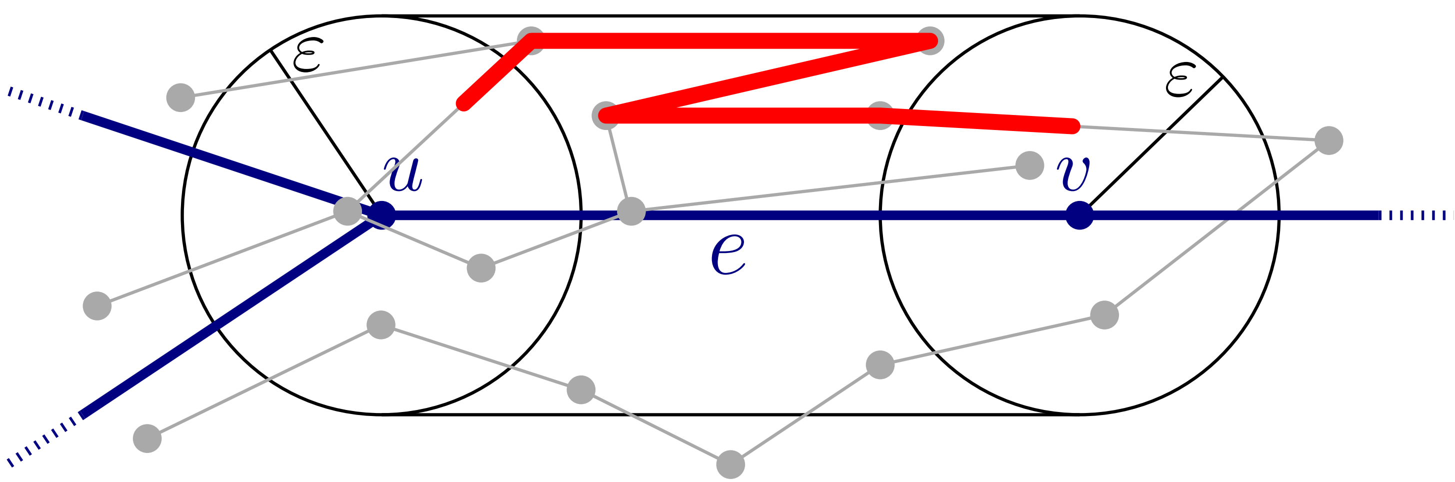


Algorithmic Approach

RUB

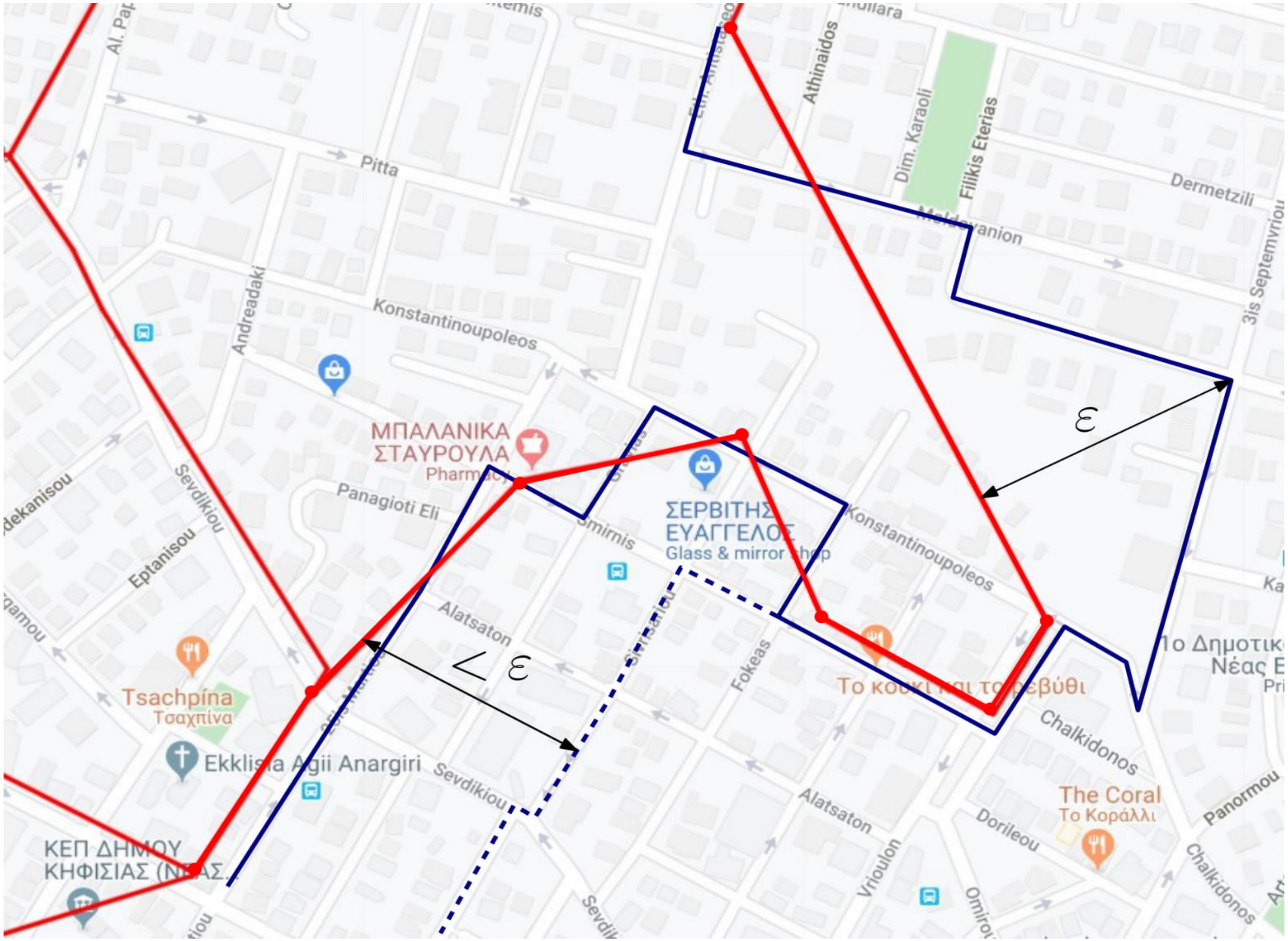
Steps to to decide the (weak) graph distance:

- compute placements of vertices
- compute reachability between placements
- delete all dead-end placements
- construct mapping based on remaining placements

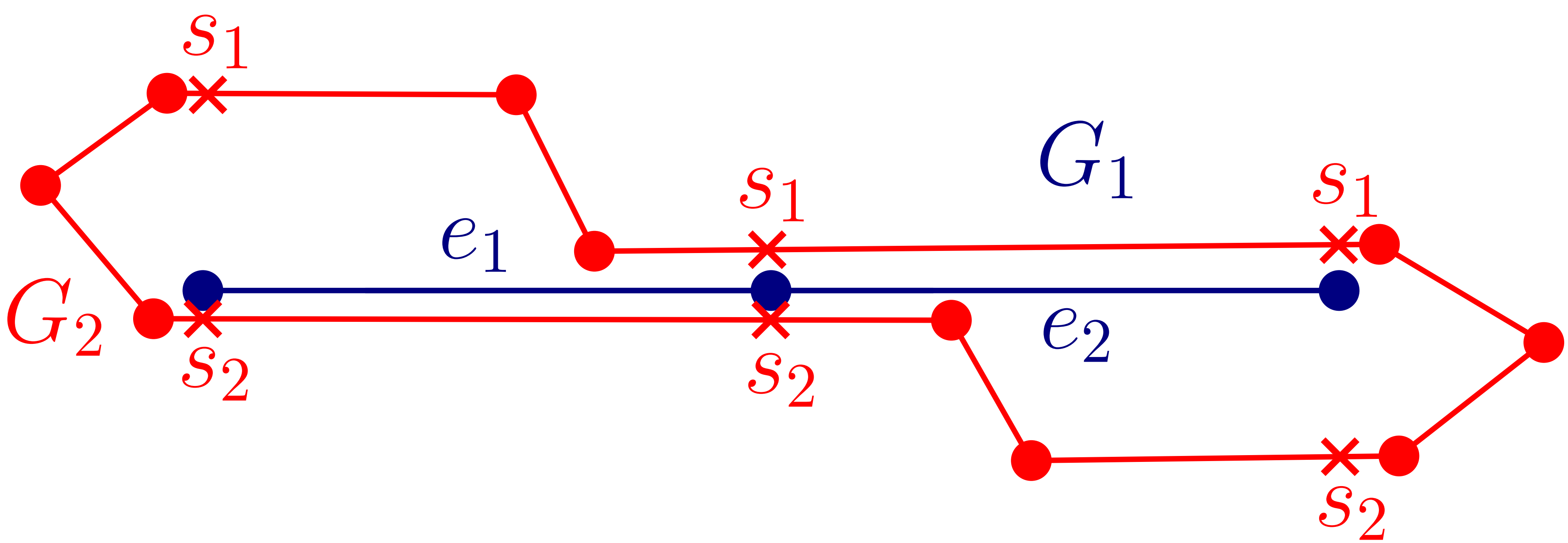


Locally Optimal Mappings

RUB



Locally Optimal Mappings



Min-Sum Graph Distance for Trees

RUB

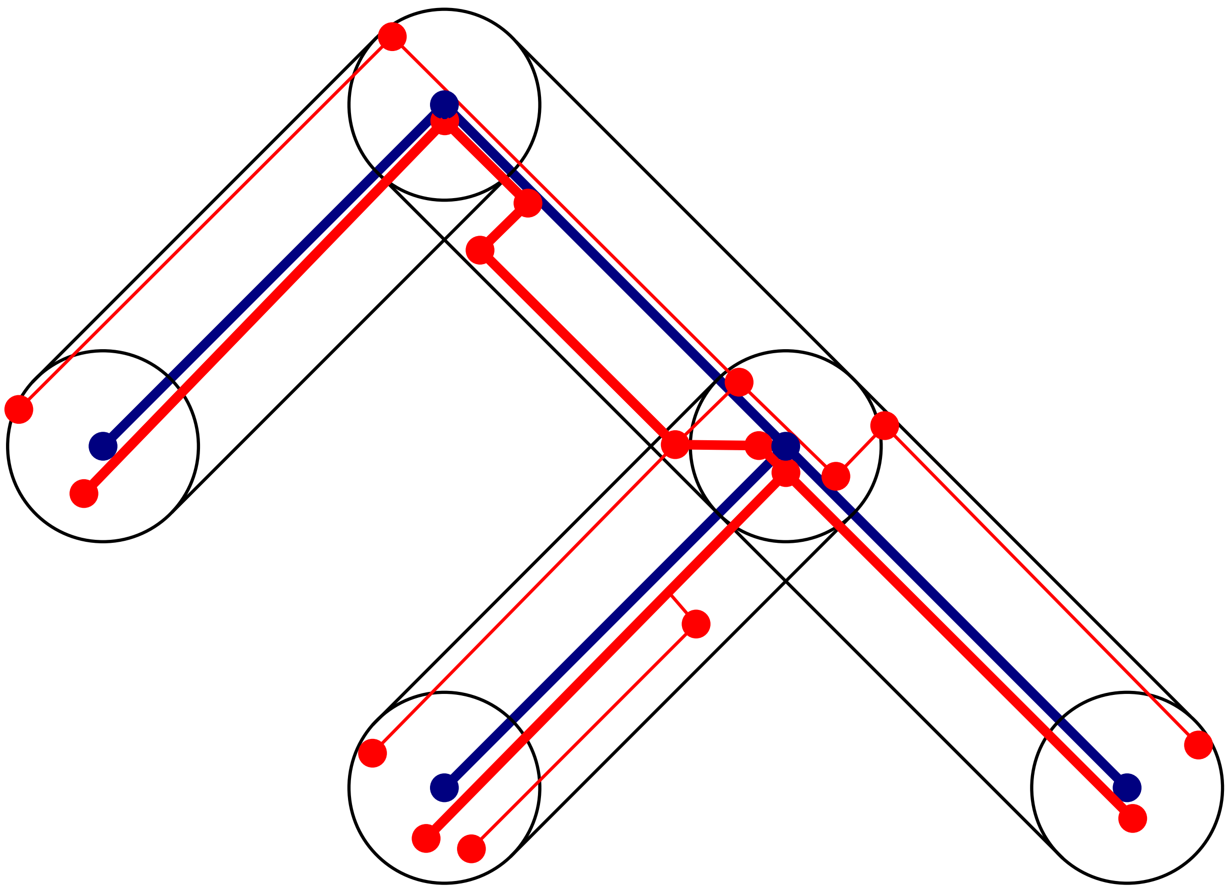
Definition:

A valid (w.r.t. an initial value $\varepsilon > 0$) mapping $s\colon G_1 \to G_2$ is a *mapping realizing the min-sum graph distance* if for any other valid mapping $\hat{s}\colon G_1 \to G_2$:

$$\sum_{e \in E_1} \delta_{(w)F}(e, \hat{s}(e)) \geq \sum_{e \in E_1} \delta_{(w)F}(e, s(e))$$

Min-Sum Graph Distance for Trees

Example:



Min-Sum Graph Distance for Trees

Computation:

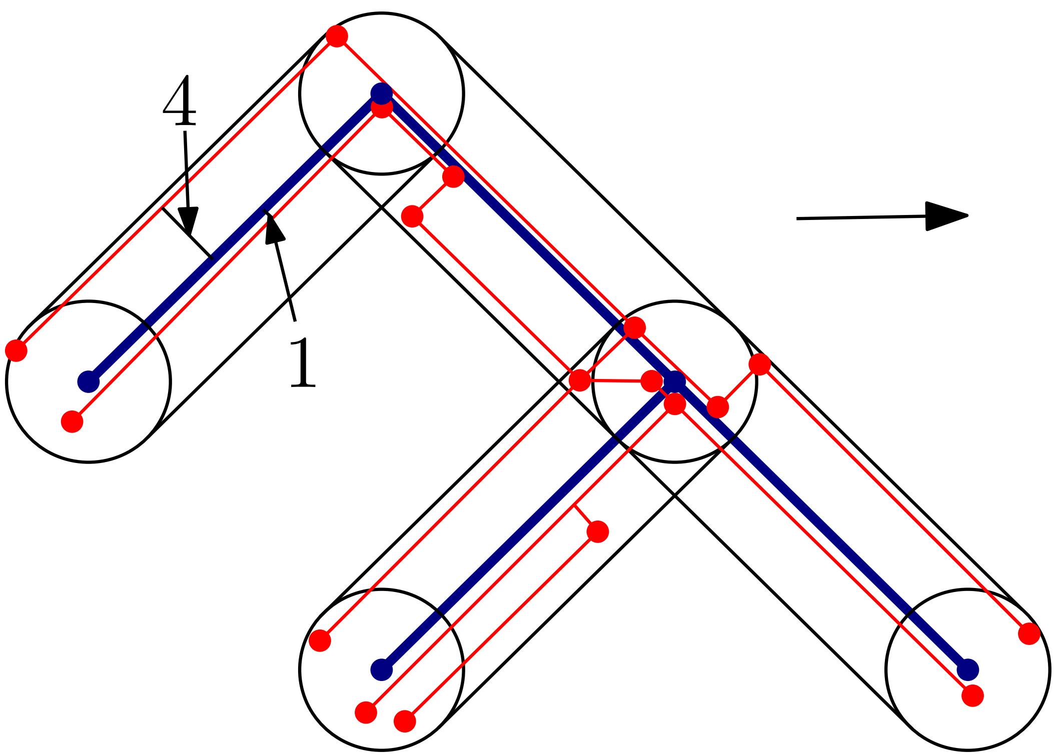
- Compute min-sum graph distance *bottom-up*
- Invariant: Subgraphs are mapped optimally w.r.t. a root-placement

Min-Sum Graph Distance for Trees

RUB

Computation:

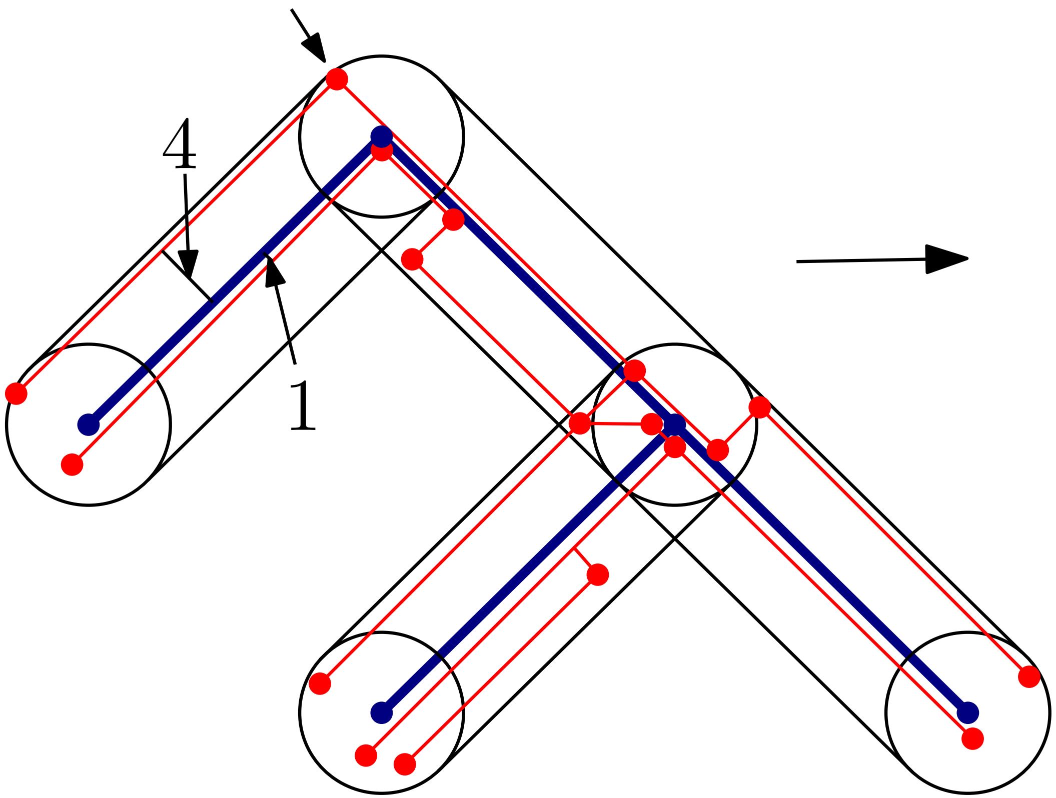
- Compute *reachability graph* H of the vertex placements. Edges weighted by the (weak) Fréchet distance.



Min-Sum Graph Distance for Trees

Computation:

- Compute *reachability graph* H of the vertex placements. Edges weighted by the (weak) Fréchet distance.



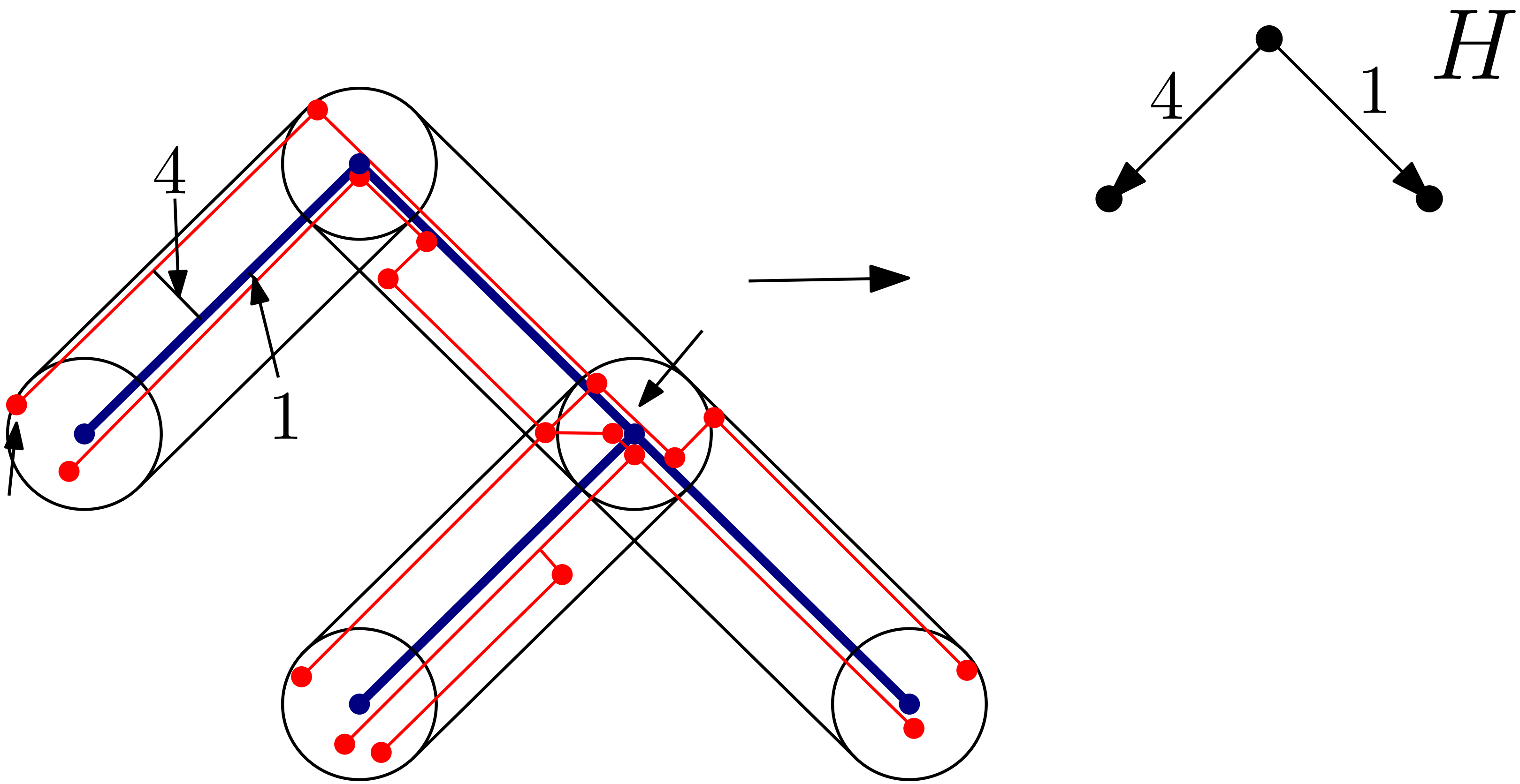
- H

Min-Sum Graph Distance for Trees

RUB

Computation:

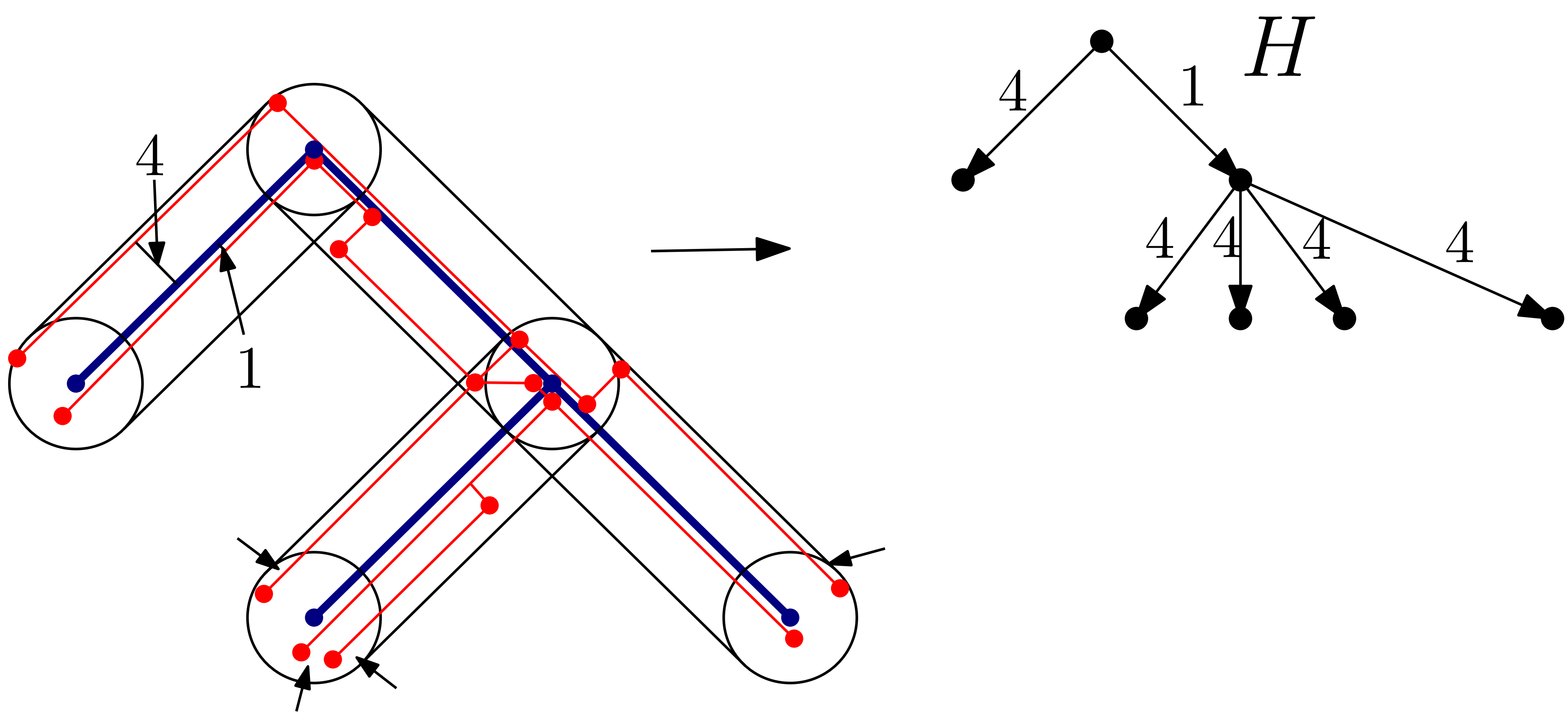
- Compute *reachability graph* H of the vertex placements. Edges weighted by the (weak) Fréchet distance.



Min-Sum Graph Distance for Trees

Computation:

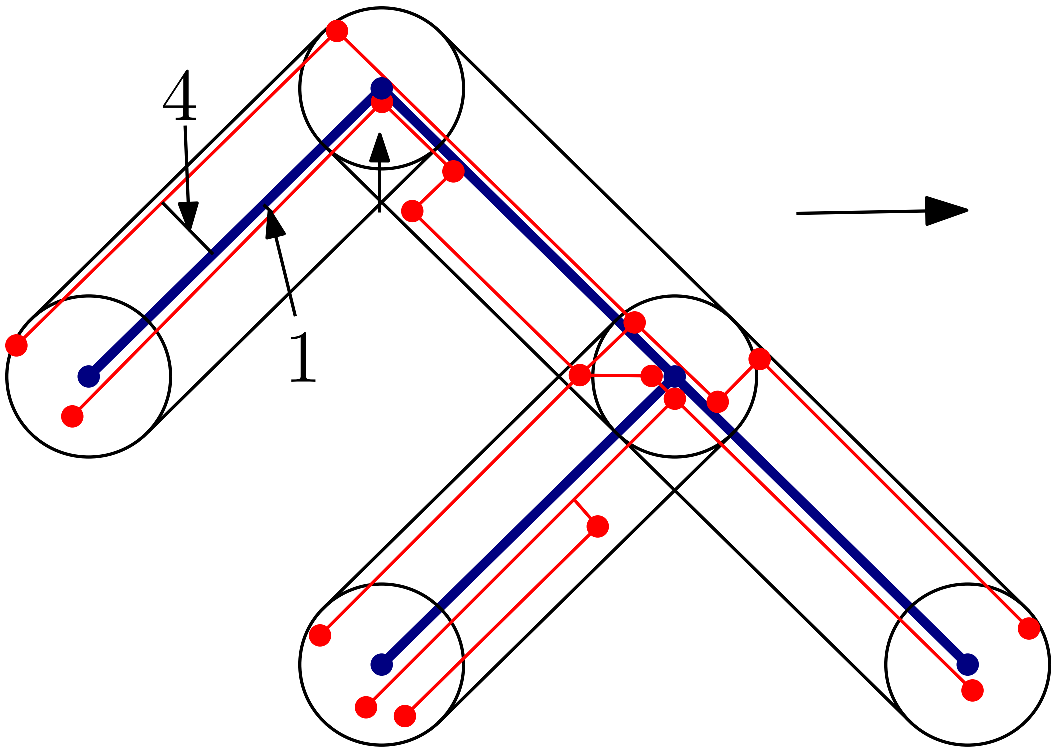
- Compute *reachability graph* H of the vertex placements. Edges weighted by the (weak) Fréchet distance.

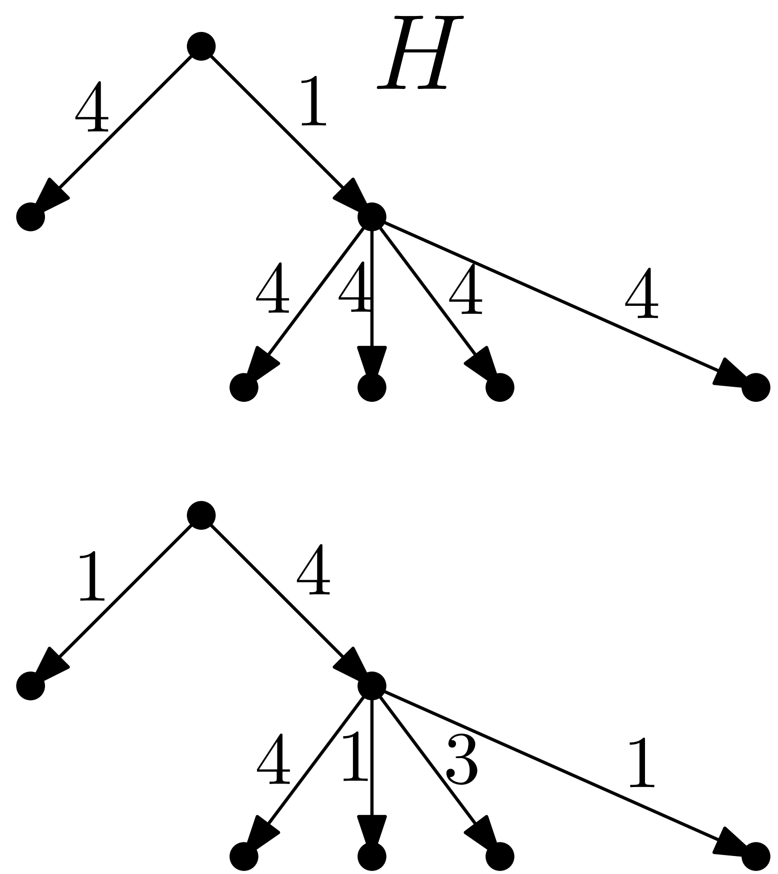


Min-Sum Graph Distance for Trees

Computation:

- Compute *reachability graph* H of the vertex placements. Edges weighted by the (weak) Fréchet distance.





Min-Sum Graph Distance for Trees

RUB

Computation:

- Set $w(p) = 0$ for all placements p of vertices of G_1

Min-Sum Graph Distance for Trees

RUB

Computation:

- Set $w(p) = 0$ for all placements p of vertices of G_1
- Let u be a vertex of G_1 with leaf-children only

$$w(C_u) = \sum_{u' : u' \text{is child of } u} \min_{C_{u'} \in P(u')} (w(C_{u'}) + w_H(C_u, C_{u'})),$$

where $w_H(C_u, C_{u'})$ is the weight of a minimum weight shortest path P between C_u and $C_{u'}$ in H

Min-Sum Graph Distance for Trees

RUB

Computation:

- Set $w(p) = 0$ for all placements p of vertices of G_1
- Let u be a vertex of G_1 with leaf-children only

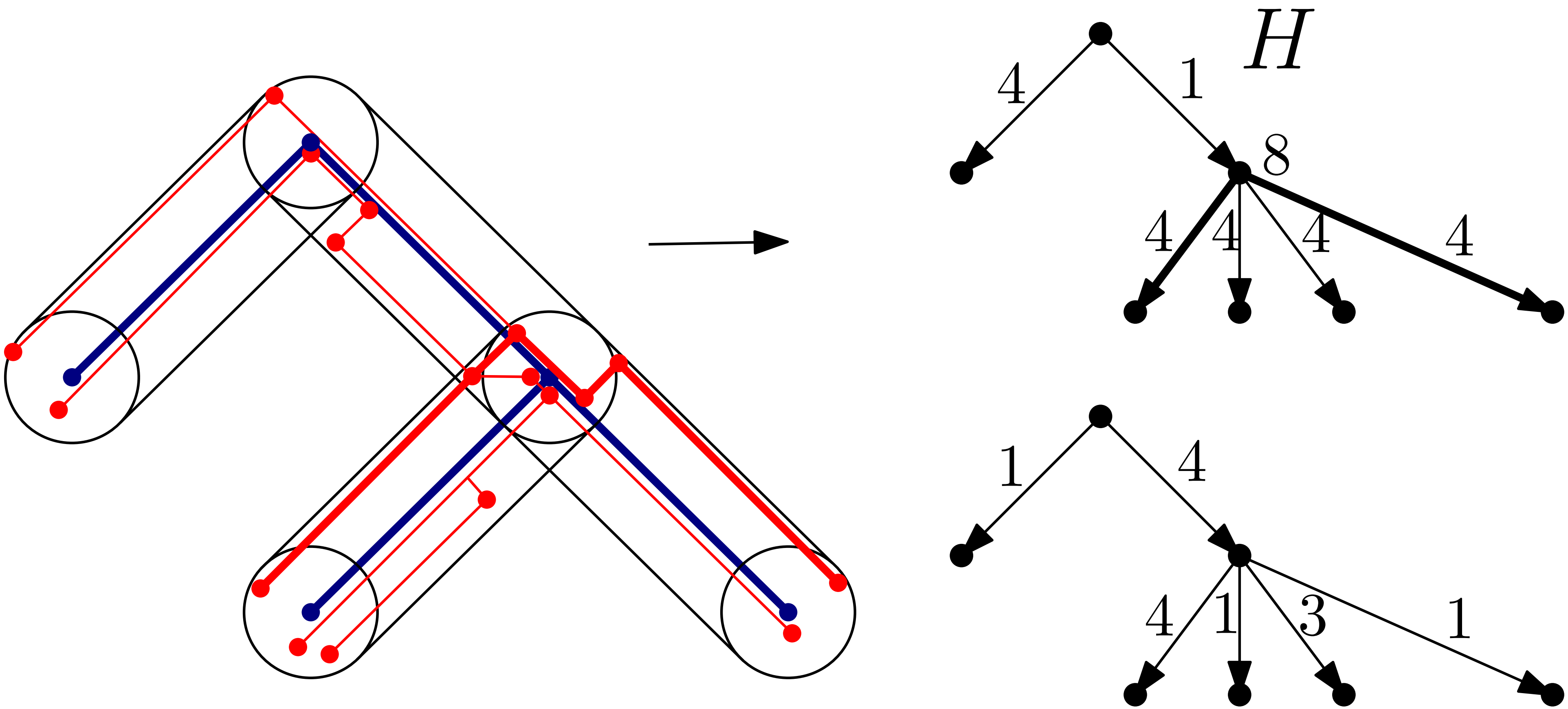
$$w(C_u) = \sum_{u' : u' \text{is child of } u} \min_{C_{u'} \in P(u')} (w(C_{u'}) + w_H(C_u, C_{u'})),$$

where $w_H(C_u, C_{u'})$ is the weight of a minimum weight shortest path P between C_u and $C_{u'}$ in H

- Store mapping realizing $w(C_u)$, delete subtree of G_1 rooted in u and iterate

Min-Sum Graph Distance for Trees

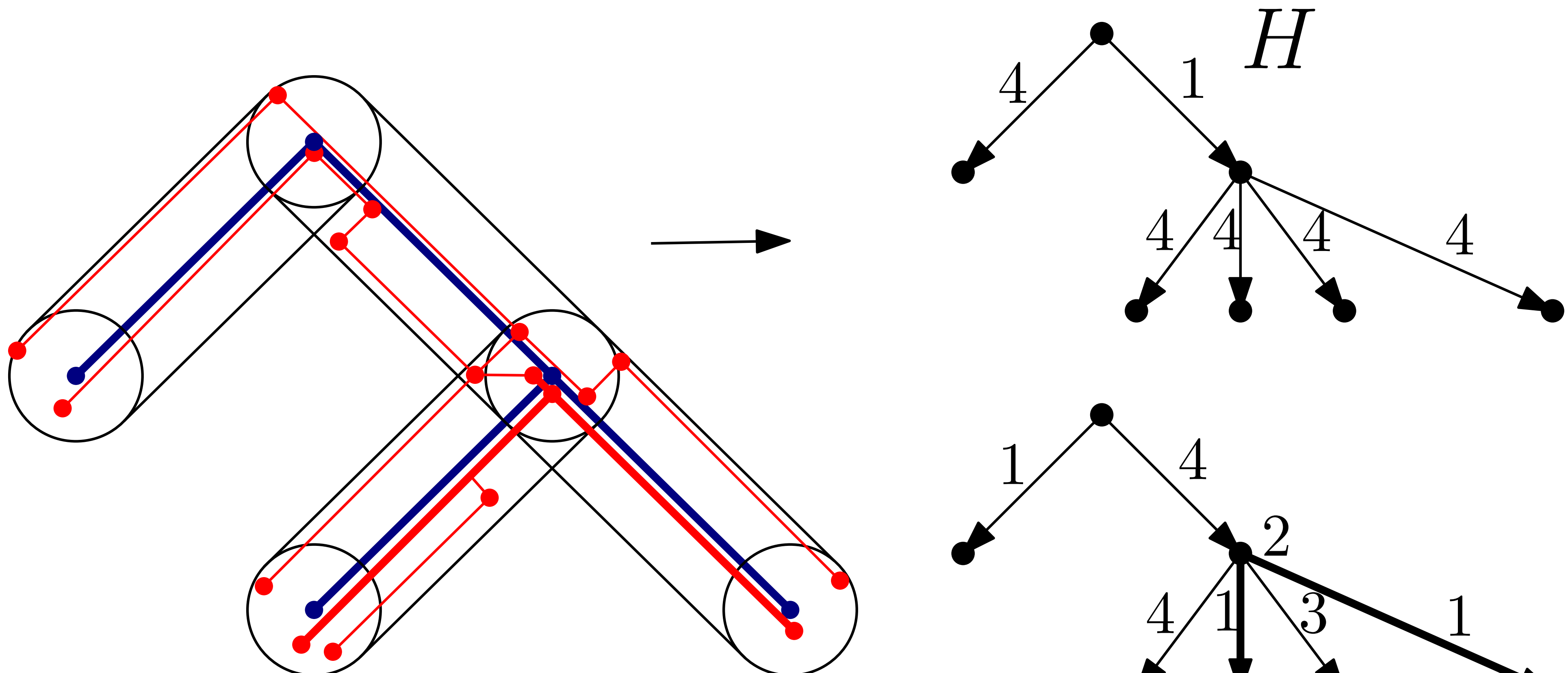
Computation:



Min-Sum Graph Distance for Trees

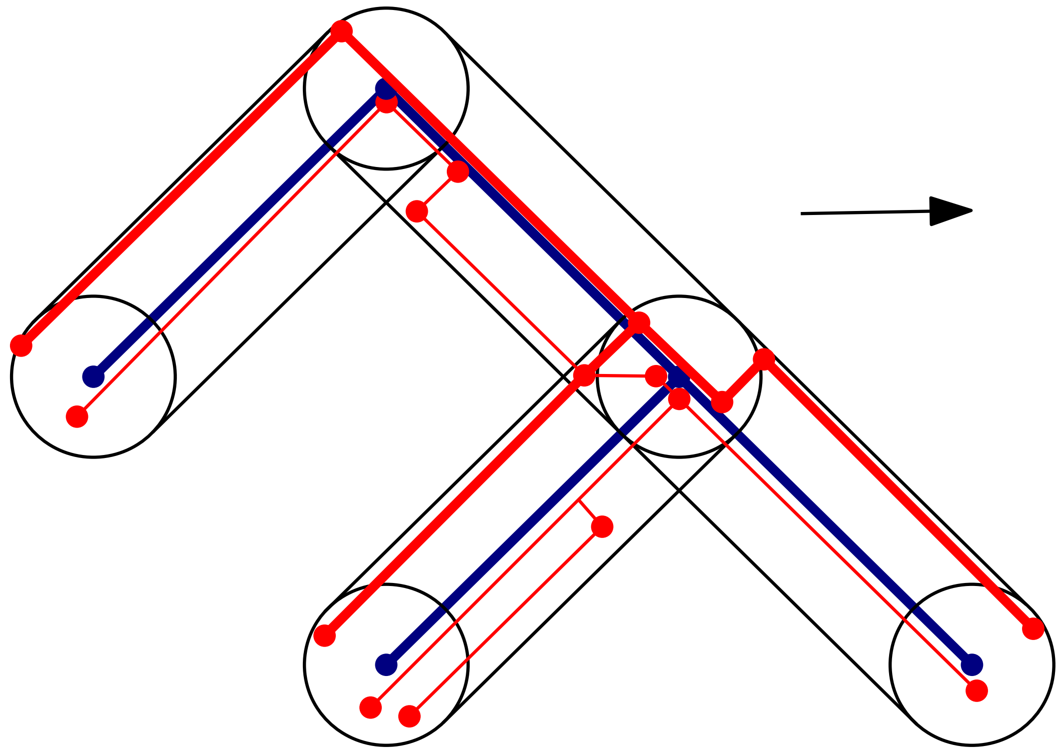
RUB

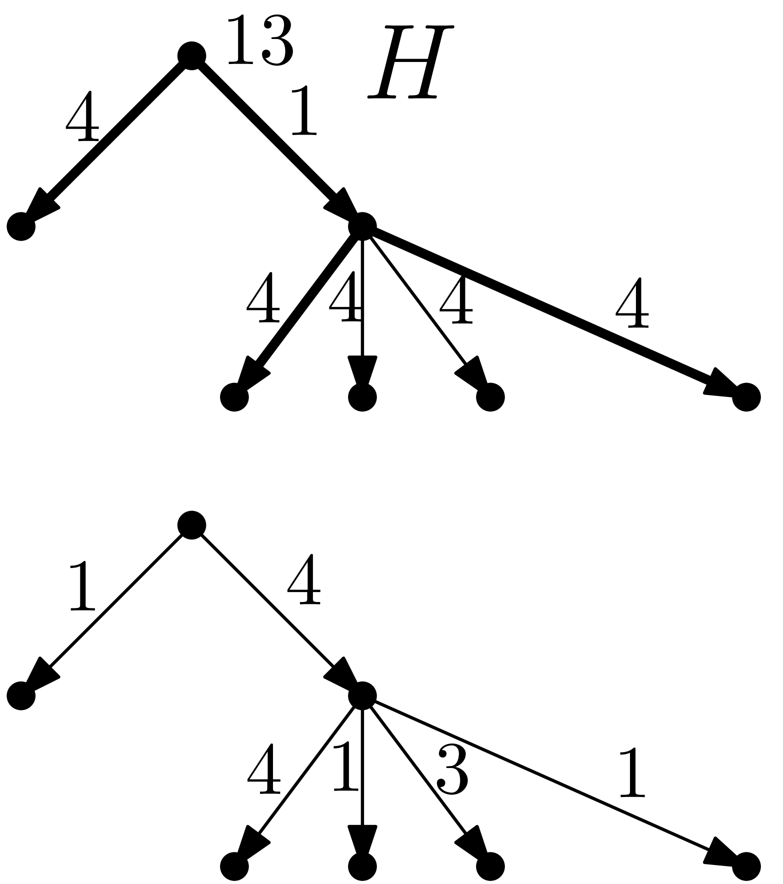
Computation:



Min-Sum Graph Distance for Trees

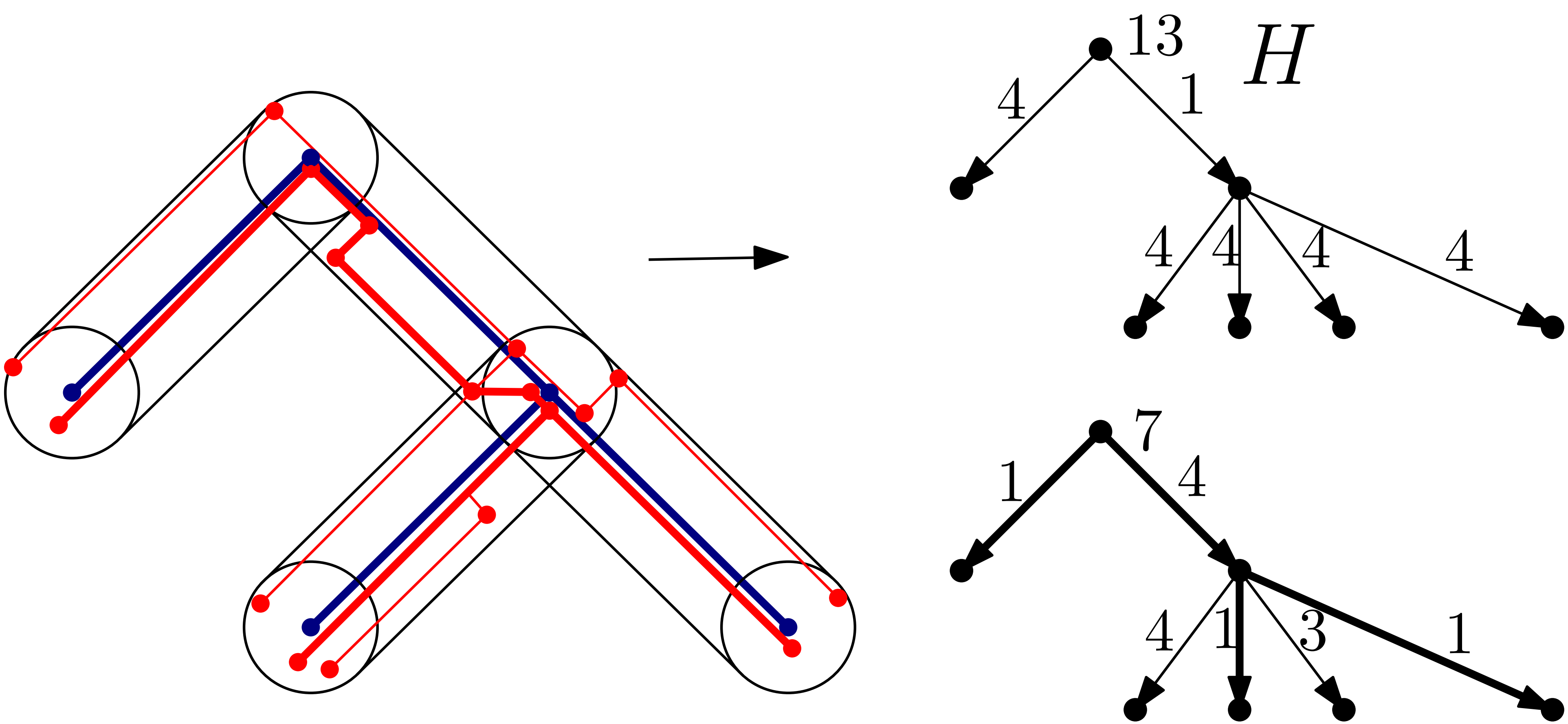
Computation:





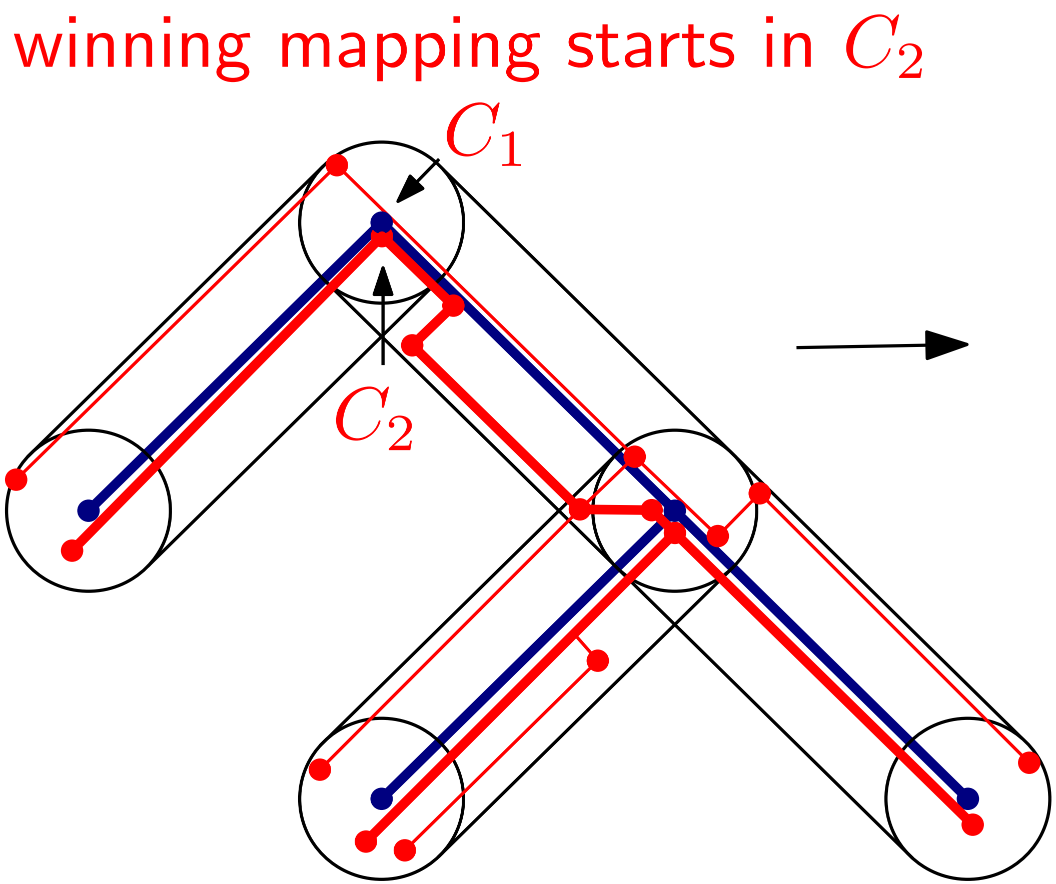
Min-Sum Graph Distance for Trees

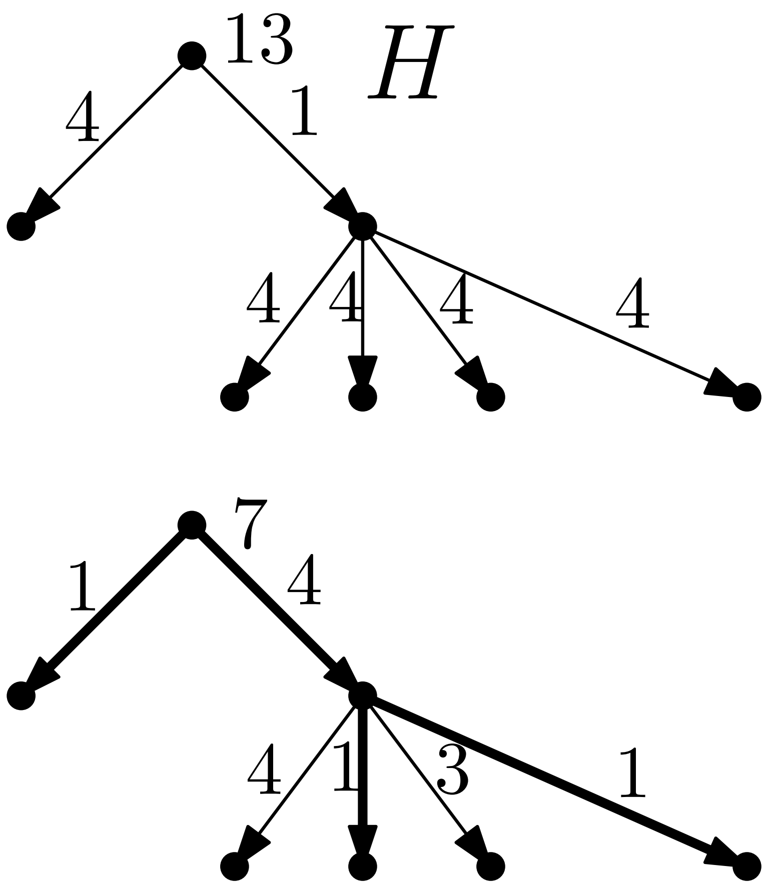
Computation:



Min-Sum Graph Distance for Trees

Computation:





Min-Sum Graph Distance for Trees

RUB

Result:

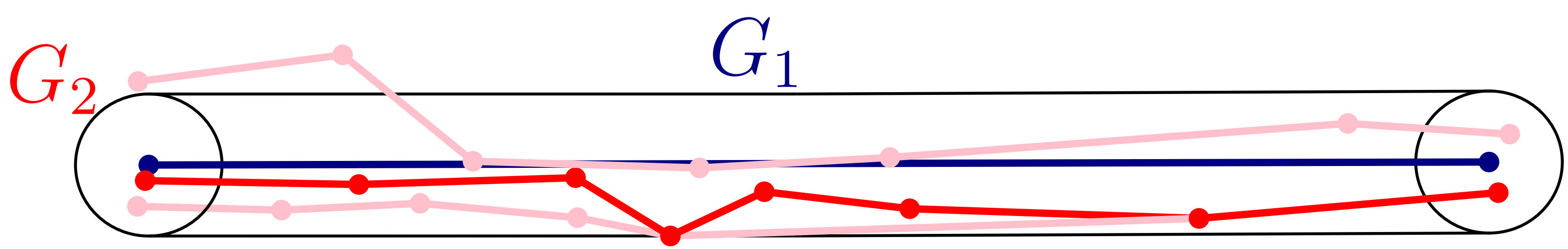
Theorem: If G_1 is a tree, we can compute a mapping s realizing the min-sum graph distance in $O(n_1 m_2^3)$ time and $O(n_1 m_2^2)$ space.

Lexicographic Graph Distance

RUB

Definition and Example:

A mapping $s\colon G_1 \to G_2$ is a *mapping realizing the lexicographic graph distance* if any local optimization induces a larger bottleneck distance compared to s.



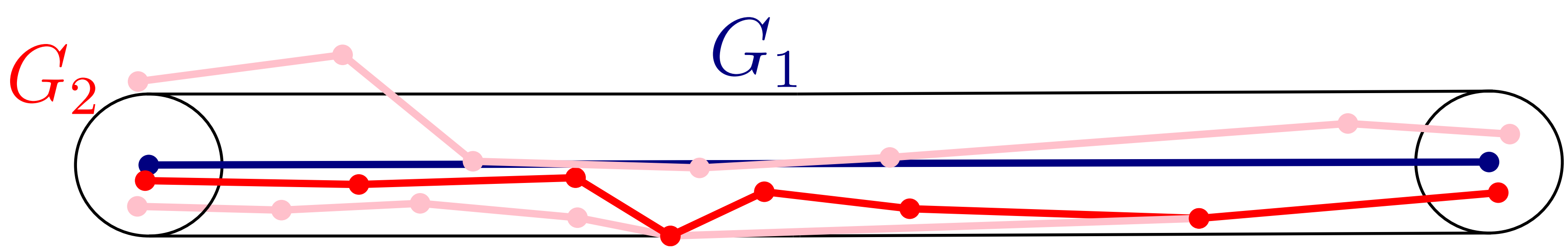
Lexicographic Graph Distance

RUB

Definition and Example:

A mapping $s\colon G_1 \to G_2$ is a *mapping realizing the lexicographic graph distance* if any local optimization induces a larger bottleneck distance compared to s.

Lexicographic: Optimally ordering the bottleneck distances between G_1 and a mapping in G_2.



Lexicographic Graph Distance

Computation:

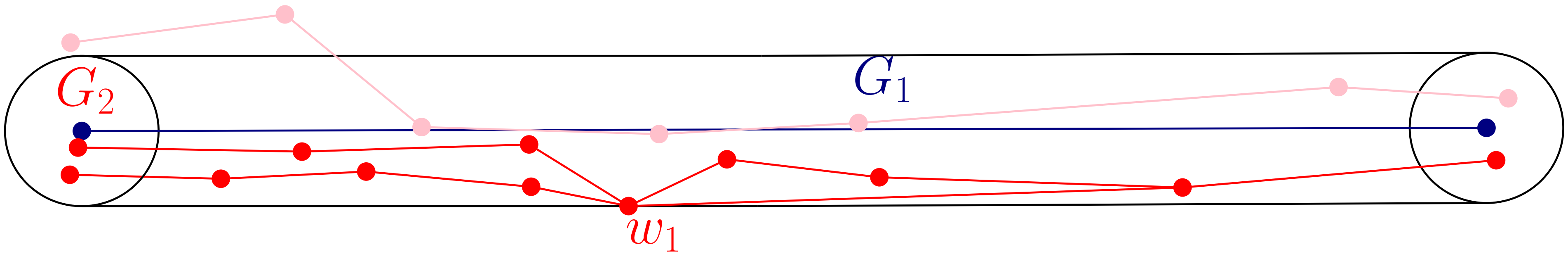
- Iteratively compute graph distance and update graph

Lexicographic Graph Distance

RUB

Computation:

- Iteratively compute graph distance and update graph

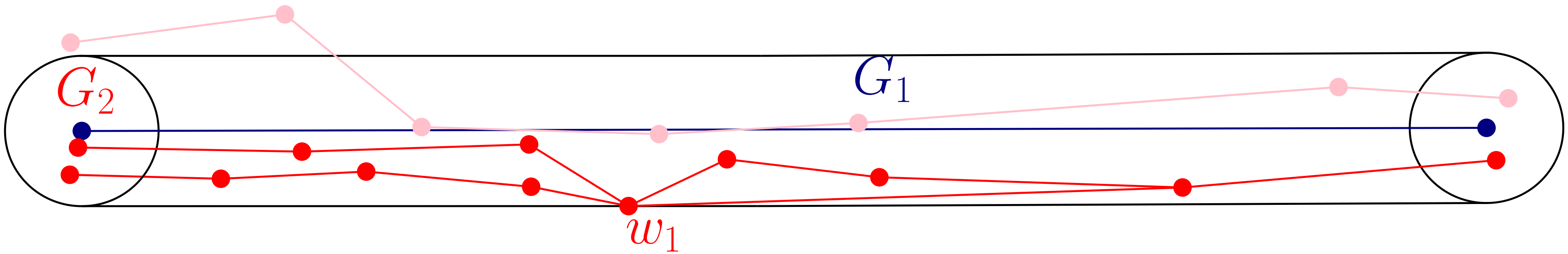


Lexicographic Graph Distance

RUB

Computation:

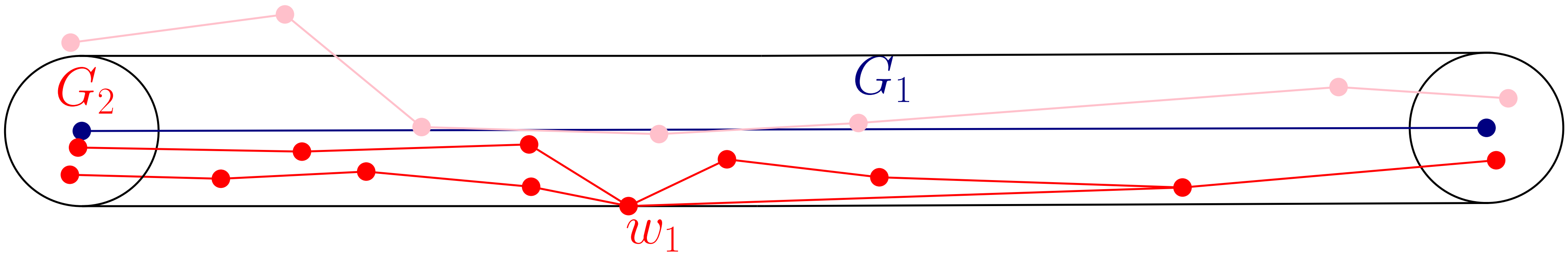
- Iteratively compute graph distance and update graph
- Update: Snap point of G_2 onto edge of G_1
- bottleneck $\rightarrow$ distance zero



Lexicographic Graph Distance

Computation:

- Iteratively compute graph distance and update graph
- Update: Snap point of G_2 onto edge of G_1
- bottleneck $\rightarrow$ distance zero

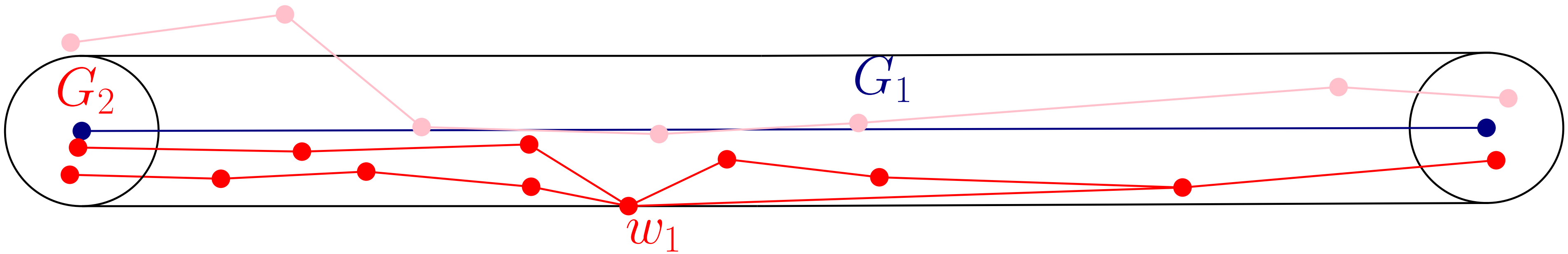


- Core observation: Any valid mapping must pass through w_1.

Lexicographic Graph Distance

Computation:

- Iteratively compute graph distance and update graph
- Update: Snap point of G_2 onto edge of G_1
- bottleneck $\rightarrow$ distance zero

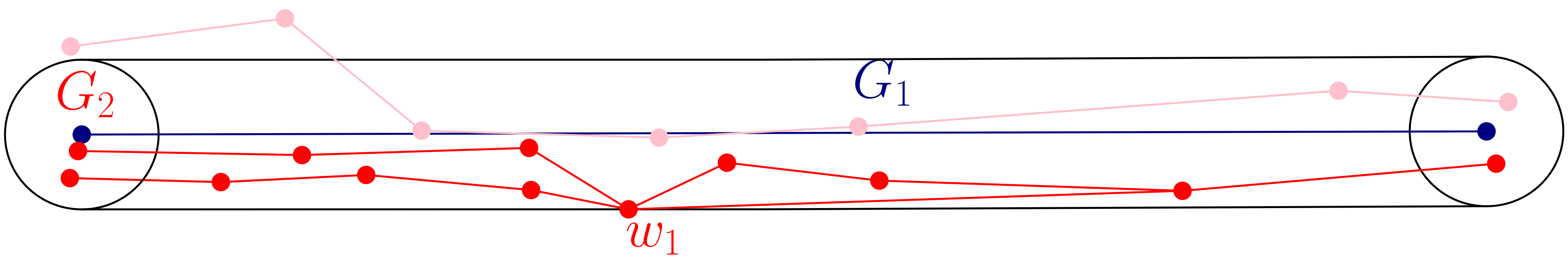


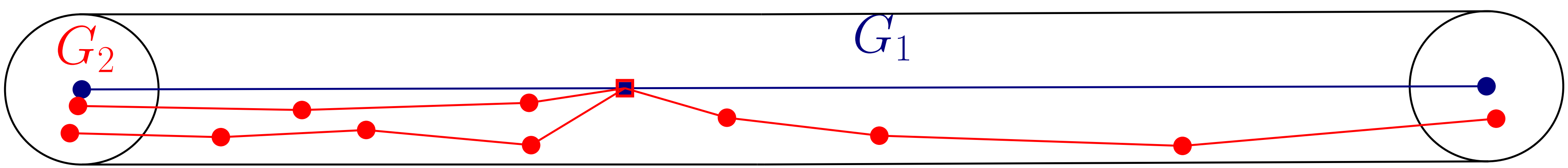
- Core observation: Any valid mapping must pass through w_1.
- Manipulating G_2 does not change the reachabiliy information between placements.

Lexicographic Graph Distance

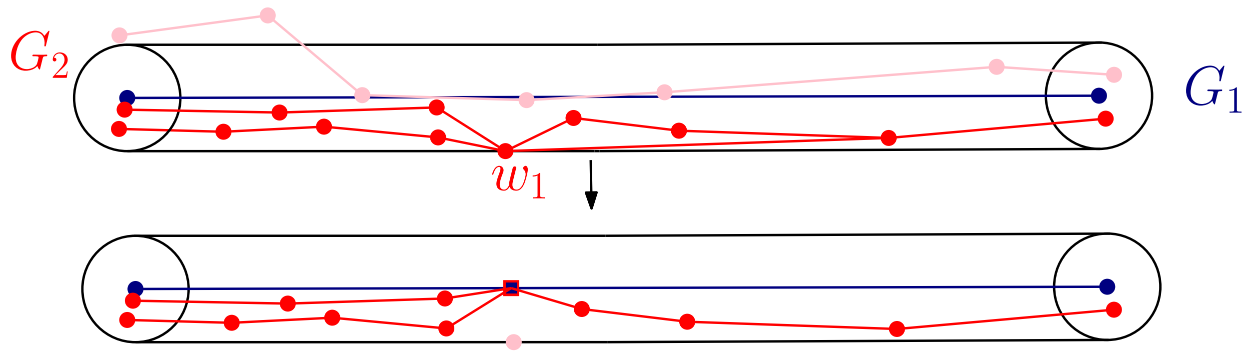
Computation:

- Iteratively compute graph distance and update graph
- Update: Snap point of G_2 onto edge of G_1
- bottleneck $\rightarrow$ distance zero

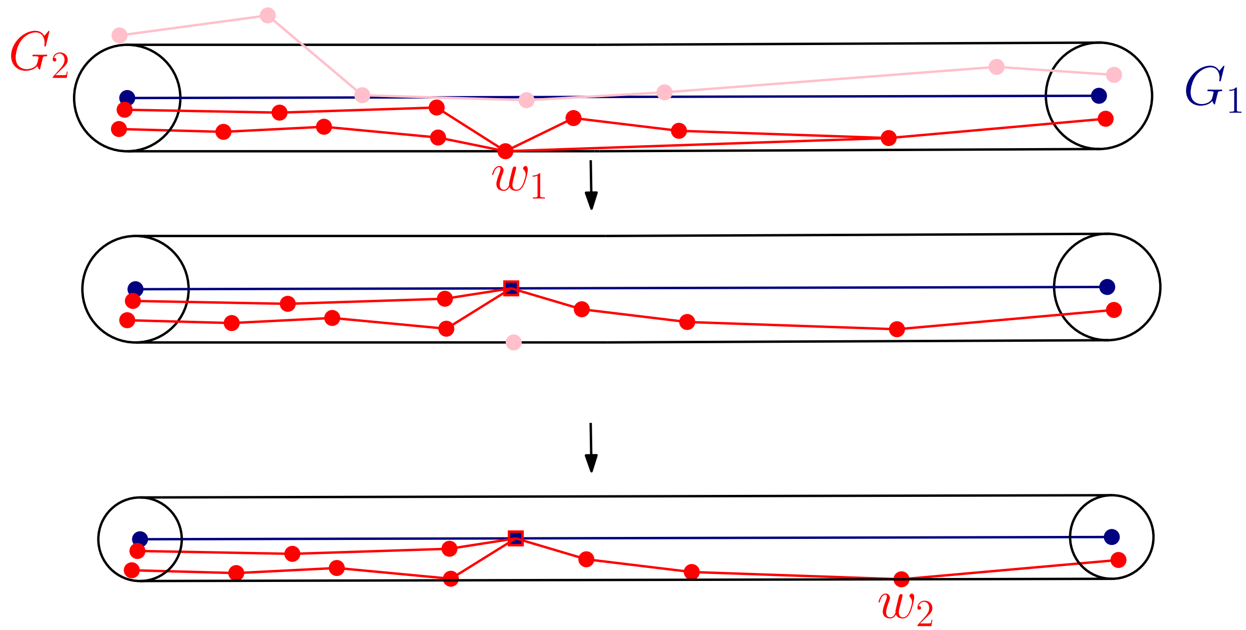




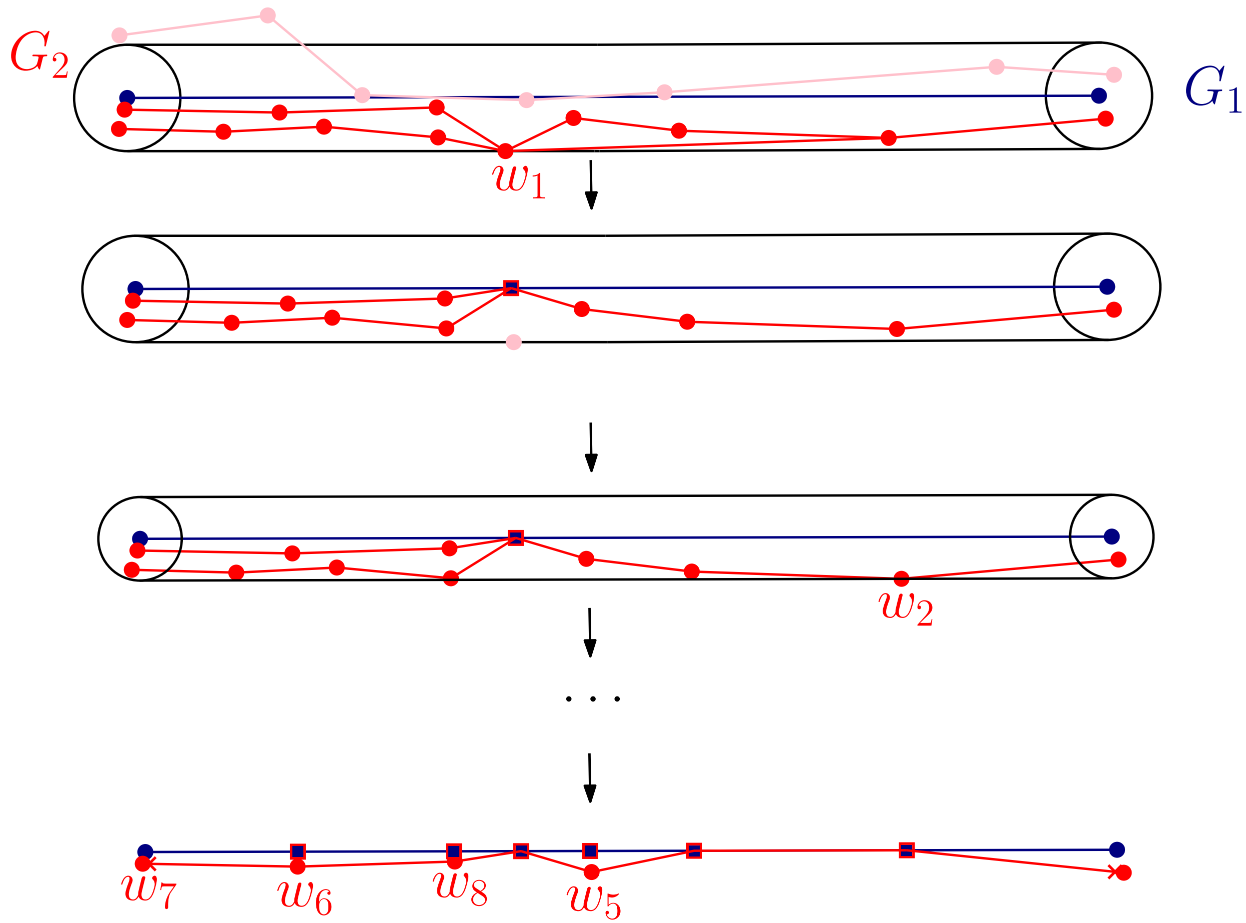
Lexicographic Graph Distance



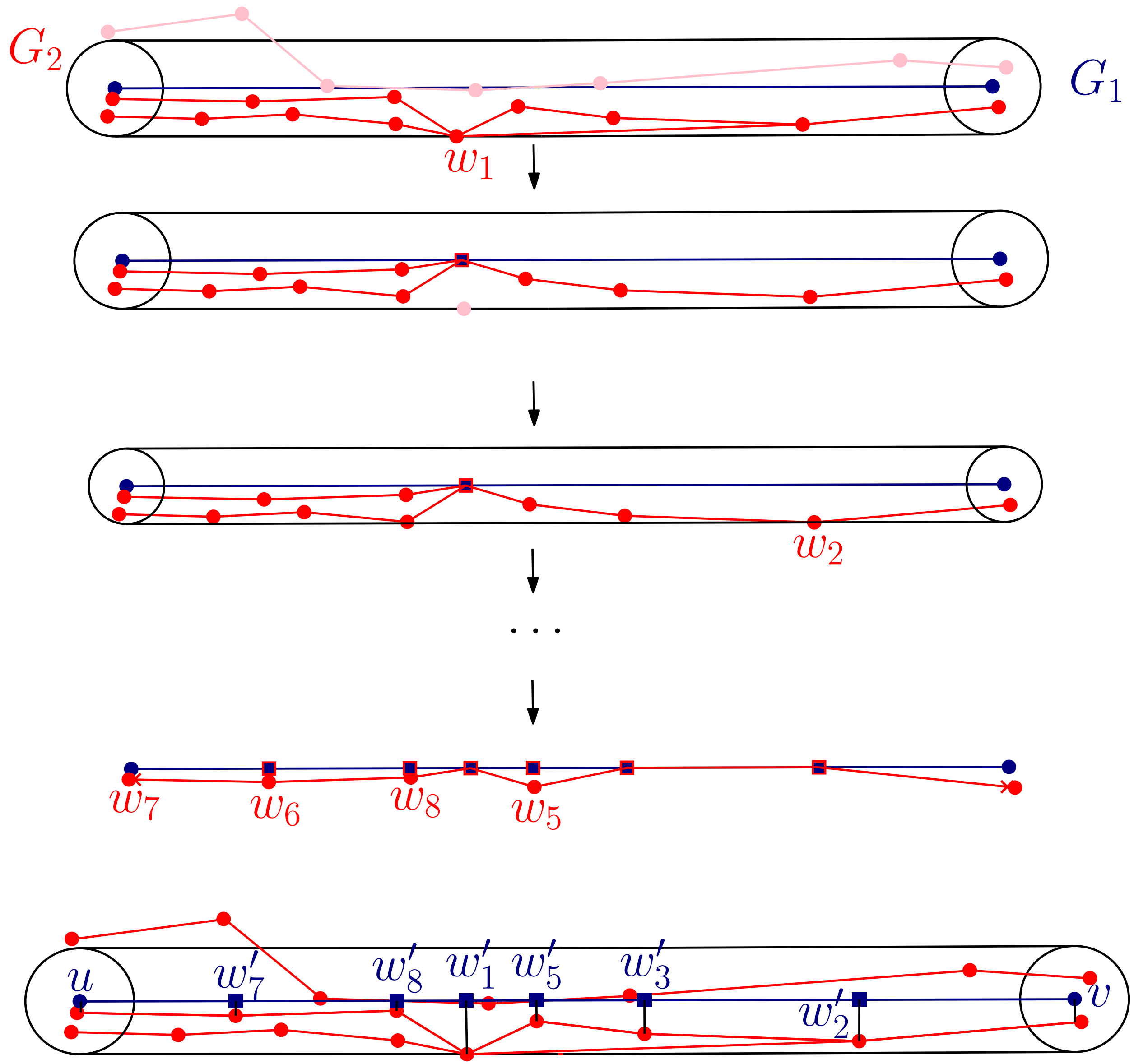
Lexicographic Graph Distance



Lexicographic Graph Distance

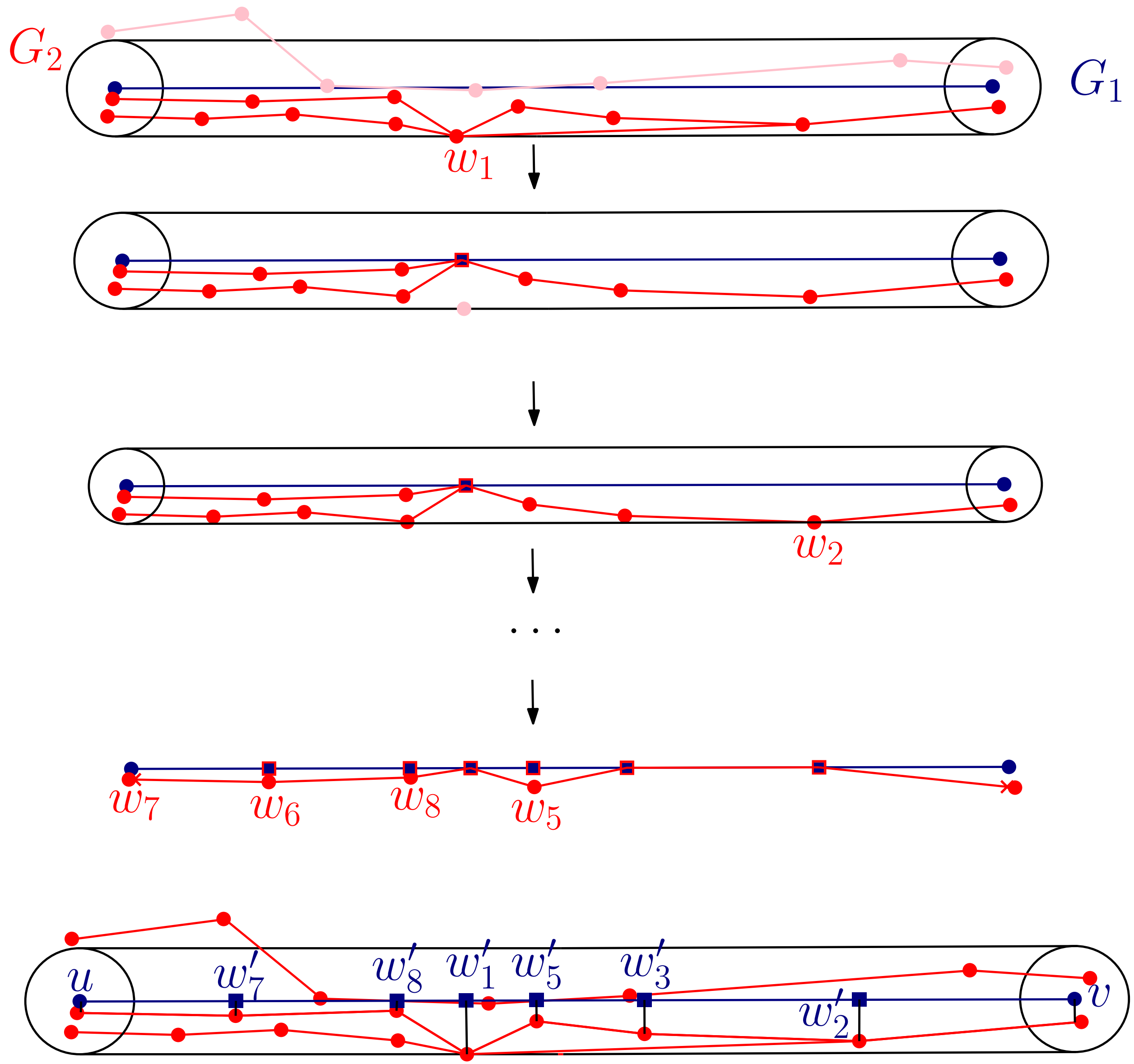


Lexicographic Graph Distance



Lexicographic Graph Distance

RUB



Snapped points define a unique mapping.

Lexicographic Graph Distance

RUB

Result:

Theorem: Given plane graphs G_1, G_2 one can compute a lexicographic graph mapping $s\colon G_1 \to G_2$ in $O(n_1^2 n_2^2 \log(n_1 + n_2))$ time using $O(n_1 n_2)$ space.

Summary

RUB

- No locally optimal mapping for graphs

Summary

- No locally optimal mapping for graphs
- Additional optimality criteria improve the mappings locally

Summary

RUB

- No locally optimal mapping for graphs
- Additional optimality criteria improve the mappings locally
- Min-sum graph distance computable in polynomial time, if G_1 is a tree. For plane graphs probably NP-hard to compute.

Summary

- No locally optimal mapping for graphs
- Additional optimality criteria improve the mappings locally
- Min-sum graph distance computable in polynomial time, if G_1 is a tree. For plane graphs probably NP-hard to compute.
- Lexicographic graph distance based on the weak graph distance computable in polynomial time if both graphs are planar embedded