All Pairs Shortest Paths in $\mathcal{O}(n^3 / \log(n))$ Time

Peter Oehme

10 January 2023

Overview

1. Introduction to APSP
2. APSP Algorithms with Subcubic Runtime
3. Connecting Computational Geometry and Matrix Multiplication
4. Computing the Runtime
5. Recovery of Shortest Paths and Setting Product Matrix Entries

Introduction to APSP

Shortest Paths Problems[1]

Consider a possibly directed graph $G = (V, E)$ with weights $w \in E'$. We say that a path $\tilde{p} = (e_1, e_2, \dots, e_n), e_1 = (a, \cdot), e_n = (\cdot, b)$ is a *shortest path* from a vertex a to a vertex b, iff

$$\tilde{p} = \operatorname*{arg\,min}_{p \text{ path from } a \text{ to } b} w(p),$$

where $w(p) = \sum_{i=1}^{n} w(e_i)$.

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd ed. McGraw-Hill, 2001, Section 24

Shortest Paths Problems

Variants

- Single Pair Shortest Paths Problems (Fix both origin and target vertices a and b)

Shortest Paths Problems

Variants

- Single Pair Shortest Paths Problems (Fix both origin and target vertices a and b)
- Single Destination Shortest Paths Problems (Fix target vertex b)

Shortest Paths Problems

Variants

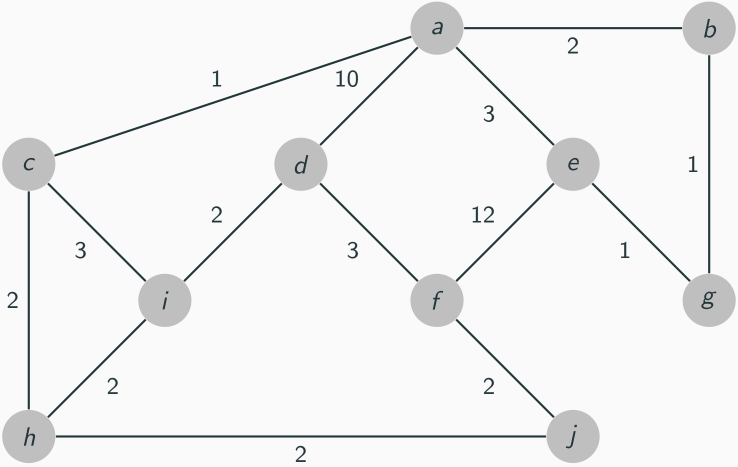
- Single Pair Shortest Paths Problems (Fix both origin and target vertices a and b)
- Single Destination Shortest Paths Problems (Fix target vertex b)
- Single Source Shortest Paths Problems (Fix origin vertex a)

Shortest Paths Problems

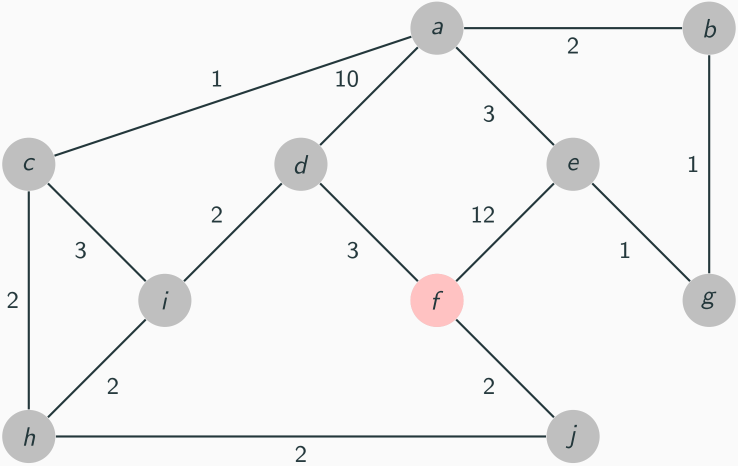
Variants

- Single Pair Shortest Paths Problems (Fix both origin and target vertices a and b)
- Single Destination Shortest Paths Problems (Fix target vertex b)
- Single Source Shortest Paths Problems (Fix origin vertex a)
- All Pairs Shortest Paths Problems

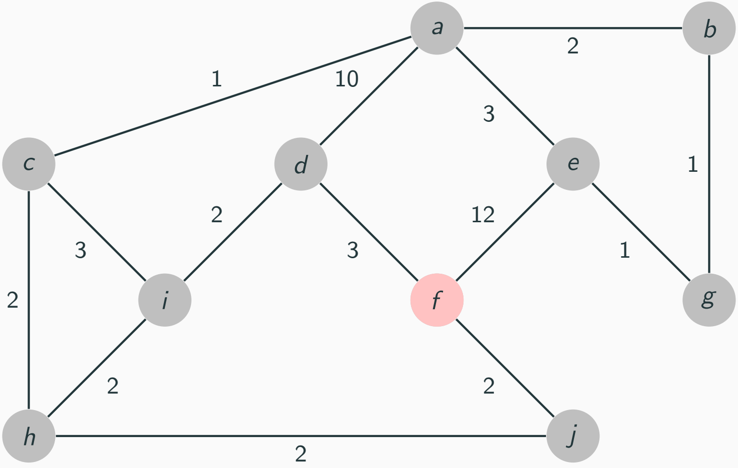
Single Source Shortest Paths (SSSP)



Single Source Shortest Paths (SSSP)

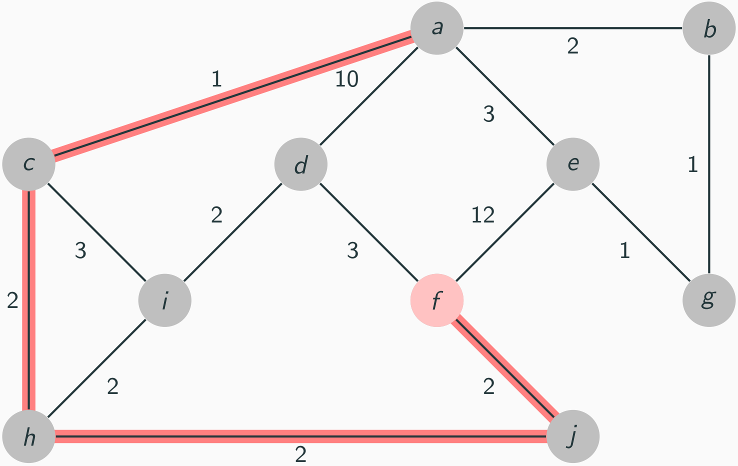


Single Source Shortest Paths (SSSP)



$$\Longrightarrow \begin{pmatrix} 7 \\ 9 \\ 6 \\ 3 \\ 10 \\ 0 \\ 10 \\ 4 \\ 5 \\ 2 \end{pmatrix} \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \\ j \end{matrix}$$

Single Source Shortest Paths (SSSP)



$$\Longrightarrow \begin{array}{cc} \begin{pmatrix} 7 \\ 9 \\ 6 \\ 3 \\ 10 \\ 0 \\ 10 \\ 4 \\ 5 \\ 2 \end{pmatrix} & \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \\ j \end{array} \end{array}$$

Solving SSSP

Assumption
We assume that the graph G contains no negative weight cycles.

Here a *negative weight cycle* refers to a path $p = (e_1, \ldots, e_n)$ such that $e_1 = (a, \cdot), e_n = (\cdot, a)$ and $w(p) < 0$.

Solving SSSP

Assumption
We assume that the graph G contains no negative weight cycles.

Here a *negative weight cycle* refers to a path $p = (e_1, \ldots, e_n)$ such that $e_1 = (a, \cdot), e_n = (\cdot, a)$ and $w(p) < 0$.

If we are looking for a shortest path containing from a vertex u to a vertex v, including a negative weight cycle always reduces the weight of the total path. Especially, the $\arg\min$ in the definition of a shortest path no longer exists, making itself invalid. $\lightning$

Solving SSSP

Assumption
We assume that the graph G contains no negative weight cycles.

Here a *negative weight cycle* refers to a path $p = (e_1, \ldots, e_n)$ such that $e_1 = (a, \cdot), e_n = (\cdot, a)$ and $w(p) < 0$.

If we are looking for a shortest path containing from a vertex u to a vertex v, including a negative weight cycle always reduces the weight of the total path. Especially, the $\arg\min$ in the definition of a shortest path no longer exists, making itself invalid. $\lightning$

Negative weight edges are of course allowed!

Solving SSSP

Among others, the *Bellman-Ford* algorithm solves the SSSP problem:

Data: Starting vertex a
Initialize $d(a) = 0, d(v) = \infty \, \forall v \in V \setminus \{a\}$;
foreach $v \in V \setminus \{a\}$ **do**
    foreach $e = (u, v) \in E$ **do**
        if $d(v) > d(u) + w(e)$ **then**
            $d(v) = d(u) + w(e)$

Among others, the *Bellman-Ford* algorithm solves the SSSP problem:

Data: Starting vertex a

Initialize $d(a) = 0, d(v) = \infty \, \forall v \in V \setminus \{a\}$;

foreach $v \in V \setminus \{a\}$ **do**

    foreach $e = (u, v) \in E$ **do**

        if $d(v) > d(u) + w(e)$ **then**

            $d(v) = d(u) + w(e)$

Note

It is easy to keep track of each vertex' predecessor to later recover the shortest path to any given vertex by setting the predecessor $p(v) = u$.

Solving SSSP

Among others, the *Bellman-Ford* algorithm solves the SSSP problem:

Data: Starting vertex a
Initialize $d(a) = 0, d(v) = \infty \, \forall v \in V \setminus \{a\}$;
foreach $v \in V \setminus \{a\}$ **do**
    foreach $e = (u, v) \in E$ **do**
        if $d(v) > d(u) + w(e)$ **then**
            $d(v) = d(u) + w(e)$

Note
It is easy to keep track of each vertex' predecessor to later recover the shortest path to any given vertex by setting the predecessor $p(v) = u$.

Bellman-Ford can also check for the existence of negative weight cycles.[1]

[1] Cormen et al., *Introduction to Algorithms*, Section 24.1

Solving SSSP

Among others, the *Bellman-Ford* algorithm solves the SSSP problem:

Data: Starting vertex a
Initialize $d(a) = 0, d(v) = \infty \, \forall v \in V \setminus \{a\}$;
foreach $v \in V \setminus \{a\}$ **do**
    foreach $e = (u, v) \in E$ **do**
        if $d(v) > d(u) + w(e)$ **then**
            $d(v) = d(u) + w(e)$

Bellman-Ford requires $\mathcal{O}(|V|\,|E|)$ time, which trivially amounts to $\mathcal{O}(n^3)$.

All Pairs Shortest Paths (APSP)[1]

The *All Pairs Shortest Paths* problem can be naively solved by applying Bellman-Ford to every vertex in G. However faster alternatives are available, such as making use of *repeated squaring*.

[1] Cormen et al., *Introduction to Algorithms*, Section 25.1

All Pairs Shortest Paths (APSP)

The *All Pairs Shortest Paths* problem can be naively solved by applying Bellman-Ford to every vertex in G. However faster alternatives are available, such as making use of *repeated squaring*.

To illustrate:

$$\begin{aligned}
W_1 &= W \\
W_2 &= W_1 \cdot W = W^2 \\
W_3 &= W_2 \cdot W \\
&\vdots \\
W_n &= W_{n-1} \cdot W
\end{aligned}$$

All Pairs Shortest Paths (APSP)

The *All Pairs Shortest Paths* problem can be naively solved by applying Bellman-Ford to every vertex in G. However faster alternatives are available, such as making use of *repeated squaring*.

To illustrate:

$$\begin{aligned} W_1 &= W \\ W_2 &= W_1 \cdot W = W^2 \\ W_3 &= W_2 \cdot W \\ &\vdots \\ W_n &= W_{n-1} \cdot W \end{aligned}$$

$$\begin{aligned} W_1 &= W \\ W_2 &= W_1 \cdot W_1 \\ W_4 &= W_2 \cdot W_2 \\ &\vdots \\ W_{2^{\lceil \log(n-1) \rceil}} &= W_{2^{\lceil \log(n-1) \rceil - 1}} \cdot W_{2^{\lceil \log(n-1) \rceil - 1}} \end{aligned}$$

All Pairs Shortest Paths (APSP)

The *All Pairs Shortest Paths* problem can be naively solved by applying Bellman-Ford to every vertex in G. However faster alternatives are available, such as making use of *repeated squaring*.

To illustrate:

$$\begin{aligned} W_1 &= W \\ W_2 &= W_1 \cdot W = W^2 \\ W_3 &= W_2 \cdot W \\ &\vdots \\ W_n &= W_{n-1} \cdot W \end{aligned}$$

$$\begin{aligned} W_1 &= W \\ W_2 &= W_1 \cdot W_1 \\ W_4 &= W_2 \cdot W_2 \\ &\vdots \\ W_{2^{\lceil \log(n-1) \rceil}} &= W_{2^{\lceil \log(n-1) \rceil - 1}} \cdot W_{2^{\lceil \log(n-1) \rceil - 1}} \end{aligned}$$

Note
The only relevant property of the matrix product $\cdot$ here is the fact that it preserves the quadratic matrix structure!

All Pairs Shortest Paths (APSP)

Data: Initialized weight matrix W
$L_1 = W, m = 1;$
while $m < n - 1$ **do**
  Consider L_m to have entries $l_{i,j}$;
  Consider L_{2m} to be a new $n \times n$ matrix with entries $l'_{i,j}$;
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do**
      $l'_{i,j} = \infty;$
      for $k = 1, \ldots, n$ **do**
        $l'_{i,j} = \min\{l_{i,j}, l_{i,k} + l_{k,j}\}$
  $m = 2m;$

Data: Initialized weight matrix W

$L_1 = W, m = 1$;

while $m < n - 1$ **do**

  Consider L_m to have entries $l_{i,j}$;

  Consider L_{2m} to be a new $n \times n$ matrix with entries $l'_{i,j}$;

  for $i = 1, \ldots, n$ **do**

    for $j = 1, \ldots, n$ **do**

      $l'_{i,j} = \infty$;

      for $k = 1, \ldots, n$ **do**

        $l'_{i,j} = \min\{l_{i,j}, l_{i,k} + l_{k,j}\}$

  $m = 2m$;

This takes a time of $\mathcal{O}\left(n^3 \log(n)\right)$.

All Pairs Shortest Paths (APSP)

Omitting the squared matrix approach and directly calculating the updates on the matrix, we obtain the *Floyd-Warshall* algorithm. This only takes $\mathcal{O}\left(n^3\right)$.

All Pairs Shortest Paths (APSP)

Omitting the squared matrix approach and directly calculating the updates on the matrix, we obtain the *Floyd-Warshall* algorithm. This only takes $\mathcal{O}\left(n^3\right)$.

Data: Initialized weight matrix W

$D_0 = W$ with entries $d_{i,j}$;

for $k = 1, \dots, n$ **do**

  Consider D_{k-1} to have entries $d_{i,j}$;

  Consider D_k to be a new $n \times n$ matrix with entries $d'_{i,j}$;

  for $i = 1, \dots, n$ **do**

    for $j = 1, \dots, n$ **do**

      $d'_{i,j} = \min\{d_{i,j}, d_{i,k} + d_{k,j}\}$;

Omitting the squared matrix approach and directly calculating the updates on the matrix, we obtain the *Floyd-Warshall* algorithm. This only takes $\mathcal{O}\left(n^3\right)$.

Data: Initialized weight matrix W

$D_0 = W$ with entries $d_{i,j}$;

for $k = 1, \ldots, n$ **do**

  Consider D_{k-1} to have entries $d_{i,j}$;

  Consider D_k to be a new $n \times n$ matrix with entries $d'_{i,j}$;

  for $i = 1, \ldots, n$ **do**

    for $j = 1, \ldots, n$ **do**

      $d'_{i,j} = \min\{d_{i,j}, d_{i,k} + d_{k,j}\}$;

Note

It is relatively easy to keep track of a vertex' predecessor when updating the weights, making it possible to retrieve the shortest path after runtime.

APSP Algorithms with Subcubic Runtime

Subcubic Runtimes[1]

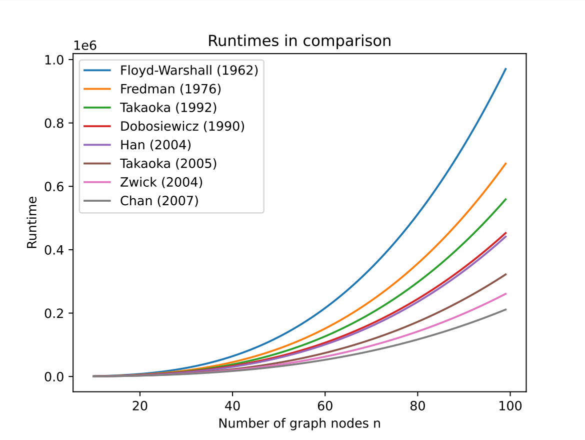


Figure 1: Representatives of complexity classes

[1] Timothy M. Chan. "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time". In: *Algorithmica 2007 50:2* 50 (2 Oct. 2007), pp. 236–243, Section 1

Subcubic Runtimes

Floyd-Warshall (1962)

$$\mathcal{O}\left(n^3\right)$$

Fredman (1976)

$$\mathcal{O}\left(n^3\left[\frac{\log(\log(n))}{\log(n)}\right]^{\frac{1}{3}}\right)$$

Takaoka (1992)

$$\mathcal{O}\left(n^3\sqrt{\frac{\log(\log(n))}{\log(n)}}\right)$$

Dobosiewizc (1990)

$$\mathcal{O}\left(\frac{n^3}{\sqrt{\log(n)}}\right)$$

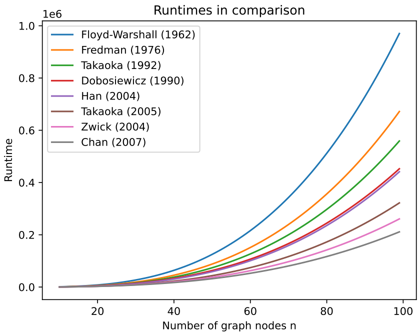


Figure 1: Representatives of complexity classes

Han (2004)

$$\mathcal{O}\left(n^3\left[\frac{\log(\log(n))}{\log(n)}\right]^{\frac{5}{7}}\right)$$

Takaoka (2005)

$$\mathcal{O}\left(n^3\frac{\log(\log(n))^2}{\log(n)}\right)$$

Zwick (2004)

$$\mathcal{O}\left(n^3\frac{\sqrt{\log(\log(n))}}{\log(n)}\right)$$

Chan (2007)

$$\mathcal{O}\left(\frac{n^3}{\log(n)}\right)$$

Connecting Computational Geometry and Matrix Multiplication

Basic Ideas

The proof is structured as follows:

1. A problem in computational geometry $\implies$ *Finding dominating pairs*

CG

Basic Ideas

The proof is structured as follows:

1. A problem in computational geometry $\implies$ *Finding dominating pairs*
2. Computing products of matrices $\implies$ *Sorting indices in sets*

$$\text{CG} \longrightarrow \text{MM}$$

Basic Ideas

The proof is structured as follows:

1. A problem in computational geometry $\implies$ *Finding dominating pairs*
2. Computing products of matrices $\implies$ *Sorting indices in sets*
3. Combining the two ideas $\implies$ *Multiplying matrices efficiently*

$$\begin{tikzcd} \text{CG} \arrow[r] & \text{MM} \arrow[d] \\ & \text{APSP} \end{tikzcd}$$

Basic Ideas

The proof is structured as follows:

1. A problem in computational geometry $\implies$ *Finding dominating pairs*
2. Computing products of matrices $\implies$ *Sorting indices in sets*
3. Combining the two ideas $\implies$ *Multiplying matrices efficiently*

$$\begin{tikzcd}
\mathrm{CG} \arrow[r] \arrow[dr] & \mathrm{MM} \arrow[d] \\
& \mathrm{APSP}
\end{tikzcd}$$

Dominating Pairs[1]

Dominating Pairs

Let $\mathbb{P}_{red}, \mathbb{P}_{blue}$ be (finite) subsets of $\mathbb{R}^d$ with $d \in \mathbb{N}$. Then a *dominating pair* is defined to be a tuple $(p, q) \in \mathbb{P}_{red} \times \mathbb{P}_{blue}$ such that it holds

$$\forall k = 1, \dots, d : p_k \leq q_k,$$

where p_k, and q_k are the k-th coordinate of p, and q respectively.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(nˆ3/log(n)) Time"

Dominating Pairs

Let $\mathbb{P}_{red}, \mathbb{P}_{blue}$ be (finite) subsets of $\mathbb{R}^d$ with $d \in \mathbb{N}$. Then a *dominating pair* is defined to be a tuple $(p, q) \in \mathbb{P}_{red} \times \mathbb{P}_{blue}$ such that it holds

$$\forall k = 1, \ldots, d : p_k \leq q_k,$$

where p_k, and q_k are the k-th coordinate of p, and q respectively.

Notes

The choice of the names "red" and "blue" is arbitrary.

Dominating Pairs

Dominating Pairs

Let $\mathbb{P}_{red}, \mathbb{P}_{blue}$ be (finite) subsets of $\mathbb{R}^d$ with $d \in \mathbb{N}$. Then a *dominating pair* is defined to be a tuple $(p, q) \in \mathbb{P}_{red} \times \mathbb{P}_{blue}$ such that it holds

$$\forall k = 1, \ldots, d : p_k \leq q_k,$$

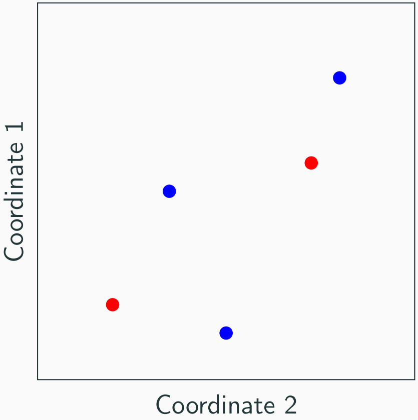
where p_k, and q_k are the k-th coordinate of p, and q respectively.

Notes

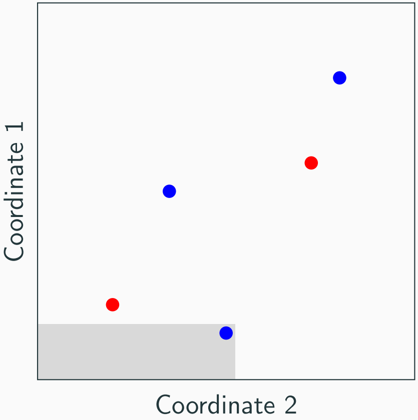
The choice of the names "red" and "blue" is arbitrary.

This implies that for any dominating pair, the first point will always be the one with smaller coordinates. This can, to a certain degree, be seen as an "ordering" of $\mathbb{P}_{red}$, and $\mathbb{P}_{blue}$.

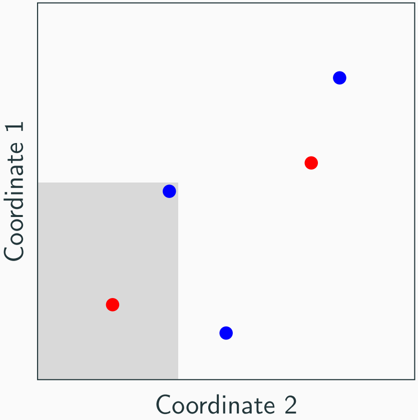
Dominating Pairs



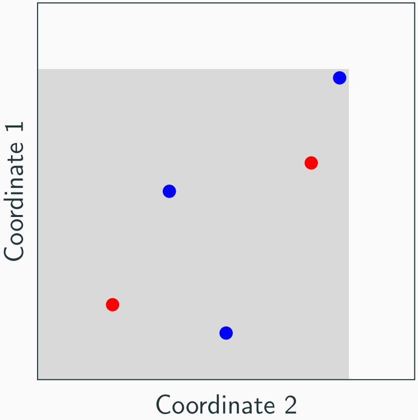
Dominating Pairs



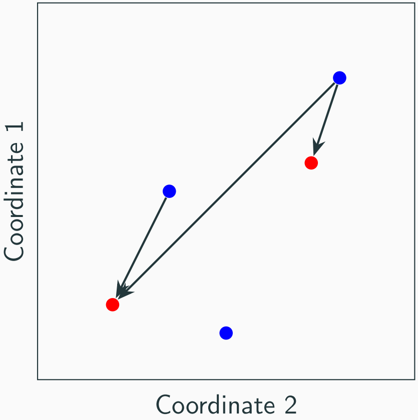
Dominating Pairs



Dominating Pairs



Dominating Pairs



Lemma 1

Let $\mathbb{P} \subset \mathbb{R}^d$ be a set of red or blue points, partitioned into $\mathbb{P}_{red}$ and $\mathbb{P}_{blue}$ such that $n = |\mathbb{P}_{red}| + |\mathbb{P}_{blue}|$, and $\varepsilon \in (0, 1)$. Then we can find all k dominating pairs in a time of $\mathcal{O}\left(c_\varepsilon^d n^{1+\varepsilon} + k\right)$, where we define $c_\varepsilon := \frac{2^\varepsilon}{2^\varepsilon - 1}$.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Matrix Products

Min-Plus/Distance Product[1]

We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Matrix Products

Min-Plus/Distance Product

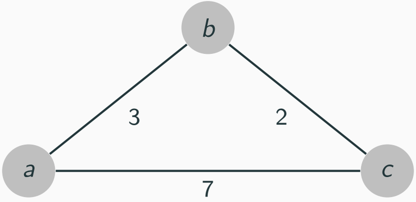
We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n \times d} \times \mathbb{R}^{d \times m} \to \mathbb{R}^{n \times m}, \qquad (A \otimes B)_{i,j} := \min_{k \in \{1,\dots,d\}} a_{i,k} + b_{k,j}.$$

Computing Distances

This product is useful, because it mimics the computation of shortest distances:

$$\textbf{if } d(a,c) > d(a,b) + d(b,c) \textbf{ then}$$
$$d(a,c) := d(a,b) + d(b,c)$$



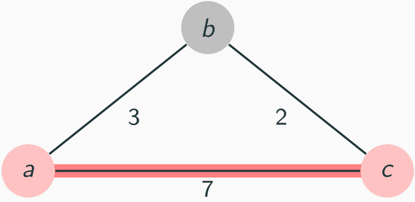
Matrix Products

Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$

Computing Distances
This product is useful, because it mimics the computation of shortest distances:

$$\textbf{if } d(a,c) > d(a,b) + d(b,c) \textbf{ then}$$
$$d(a,c) := d(a,b) + d(b,c)$$



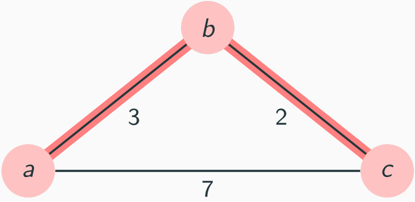
Matrix Products

Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\ldots,d\}} a_{i,k} + b_{k,j}.$$

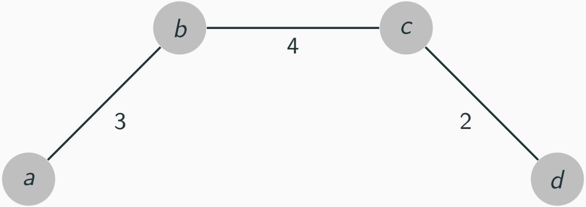
Computing Distances
This product is useful, because it mimics the computation of shortest distances:

if $d(a,c) > d(a,b) + d(b,c)$ **then**
$d(a,c) := d(a,b) + d(b,c)$



Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$

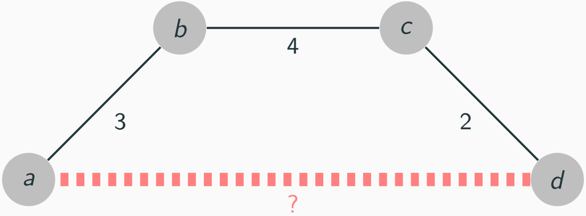


Matrix Products

Min-Plus/Distance Product

We define the *min-plus* or *distance product* as the operator

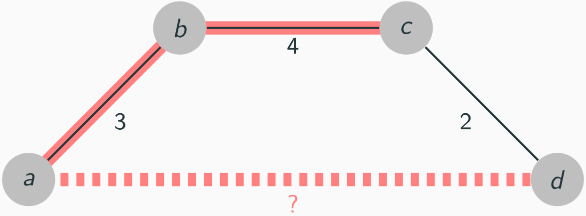
$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$



Matrix Products

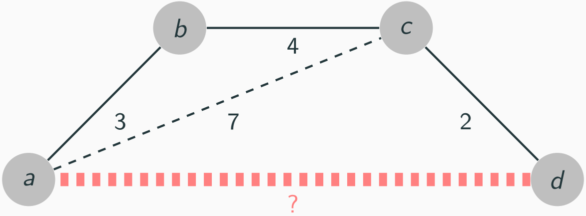
Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\ldots,d\}} a_{i,k} + b_{k,j}.$$



Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

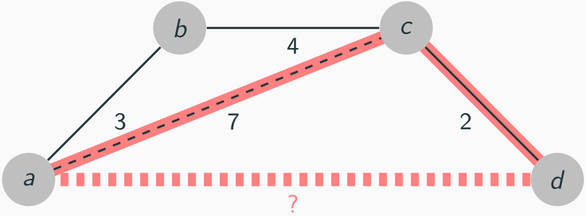
$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\ldots,d\}} a_{i,k} + b_{k,j}.$$



Min-Plus/Distance Product

We define the *min-plus* or *distance product* as the operator

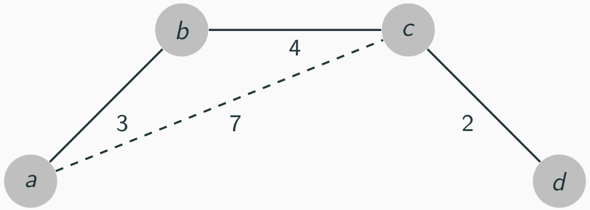
$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$



Matrix Products

Min-Plus/Distance Product
We define the *min-plus* or *distance product* as the operator

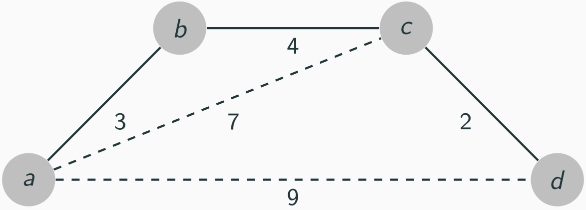
$$\otimes : \mathbb{R}^{n \times d} \times \mathbb{R}^{d \times m} \to \mathbb{R}^{n \times m}, \qquad (A \otimes B)_{i,j} := \min_{k \in \{1,\dots,d\}} a_{i,k} + b_{k,j}.$$



Min-Plus/Distance Product

We define the *min-plus* or *distance product* as the operator

$$\otimes : \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{n\times m}, \qquad (A \otimes B)_{i,j} := \min_{k\in\{1,\dots,d\}} a_{i,k} + b_{k,j}.$$



Lemma 3.1[1]

Lemma 2

Given two matrices $A \in \mathbb{R}^{n\times d}$, $B \in \mathbb{R}^{d\times n}$, *and* $\varepsilon \in (0,1)$, $c_\varepsilon := \dfrac{2^\varepsilon}{2^\varepsilon - 1}$, *we can compute their min-plus product* $A \otimes B$ *in* $\mathcal{O}\left(dc_\varepsilon^d n^{1+\varepsilon} + n^2\right)$ *time.*

Here, ε *and* c_ε *are the same variables as in Lemma 1.*

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method[1])

[1] Alfred V Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Design and Analysis of Computer Algorithms*. Addison-Wesley Pub. Co, 1974, Section 6.2

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method)

$$C = AB$$

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method)

$$C = AB$$

$$\begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method)

$$C = AB$$

$$\begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \implies \begin{cases} C_1 = A_1 B_1 + A_2 B_3 \\ C_2 = A_1 B_2 + A_2 B_4 \\ C_3 = A_3 B_1 + A_4 B_3 \\ C_4 = A_3 B_2 + A_4 B_4 \end{cases}$$

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method)

$$C = AB$$

$$\begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

Splitting Quadratic Matrices

Q: How do we go about multiplying two matrices efficiently?

A: Split and multiply individually! (Strassen's method)

$$C = AB$$

$$\begin{pmatrix} C_1 & C_2 \\ C_3 & C_4 \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix}$$

$$C_1 = M_1 + M_2 - M_4 + M_6$$

$$C_2 = M_4 + M_5$$

$$C_3 = M_6 + M_7$$

$$C_4 = M_2 - M_3 + M_5 - M_7$$

$$M_1 = (A_2 - A_4)(B_3 + B_4)$$

$$M_2 = (A_1 + A_4)(B_1 + B_4)$$

$$M_3 = (A_1 - A_3)(B_1 + B_2)$$

$$M_4 = (A_1 + A_2)\,B_4$$

$$M_5 = A_1\,(B_2 - B_3)$$

$$M_6 = A_4\,(B_3 - B_1)$$

$$M_7 = (A_3 + A_4)\,B_1$$

Multipyling Matrices and Computing their Closure

Problem

Strassen's method does not work for closed semirings.[1] However we want to work with the min-plus product which induces a closed semiring.[2]

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

[2] Aho, Hopcroft, and Ullman, *Design and Analysis of Computer Algorithms*, Example 5.9

Multipyling Matrices and Computing their Closure

Problem

Strassen's method does not work for closed semirings. However we want to work with the min-plus product which induces a closed semiring.

We can instead make use of the closure of the matrix defined as $A^* := \sum_{i=0}^{\infty} A^i$.

Problem
Strassen's method does not work for closed semirings. However we want to work with the min-plus product which induces a closed semiring.

We can instead make use of the closure of the matrix defined as $A^* := \sum_{i=0}^{\infty} A^i$.

It can then be shown that under some conditions on the computational time needed to multiply two square matrices and the time needed to compute the closure of a square matrix, both times are of the same order.[1]

[1] Aho, Hopcroft, and Ullman, *Design and Analysis of Computer Algorithms*, Section 5.9

Multipying Matrices and Computing their Closure

Problem

Strassen's method does not work for closed semirings. However we want to work with the min-plus product which induces a closed semiring.

We can instead make use of the closure of the matrix defined as $A^* := \sum_{i=0}^{\infty} A^i$.

It can then be shown that under some conditions on the computational time needed to multiply two square matrices and the time needed to compute the closure of a square matrix, both times are of the same order.

This can then be related to the APSP problem via the closure under the min-plus product.[1]

[1] Aho, Hopcroft, and Ullman, *Design and Analysis of Computer Algorithms*, Section 5.9, Corollary 2

Multiplying Matrices[1]

We will make use of splitting matrices into intermediaries whilst avoiding Strassen's method (although they ressemble each other closely):

$$A = \begin{pmatrix} A_1 & A_2 & \cdots & A_d \end{pmatrix} \qquad B = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_d \end{pmatrix}$$

$$\implies \left(C = A \otimes B \iff c_{i,j} = \min_{m=1,\ldots,d} (A_m \otimes B_m)_{i,j} \right)$$

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(nˆ3/log(n)) Time"

Theorem 3.2[1]

Theorem 3

Given any two matrices $A, B \in \mathbb{R}^{n \times n}$ *we can compute their min-plus (distance) product in a time of* $\mathcal{O}\left(n^3 / \log(n)\right)$.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(nˆ3/log(n)) Time"

Theorem 3.2[1] & Corollary 3.3[1]

Theorem 3

Given any two matrices $A, B \in \mathbb{R}^{n\times n}$ we can compute their min-plus (distance) product in a time of $\mathcal{O}\left(n^3/\log(n)\right)$.

Corollary 4

We can solve the all pairs shortest paths problem for a graph $G = (V, E)$ with $|V| = n$ vertices in $\mathcal{O}\left(n^3/\log(n)\right)$ time.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Computing the Runtime

Proof of Lemma 1[1]

Lemma 1

Let $\mathbb{P} \subset \mathbb{R}^d$ be a set of red or blue points, partitioned into $\mathbb{P}_{red}$ and $\mathbb{P}_{blue}$ such that $n = |\mathbb{P}_{red}| + |\mathbb{P}_{blue}|$, and $\varepsilon \in (0, 1)$. Then we can find all k dominating pairs in a time of $\mathcal{O}\left(c_\varepsilon^d n^{1+\varepsilon} + k\right)$, where we define $c_\varepsilon := \frac{2^\varepsilon}{2^\varepsilon - 1}$.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time", Lemma 2.1

Proof of Lemma 1

Assumption
All values are presorted by the d coordinates.

Proof of Lemma 1

Idea: Divide and Conquer!

We split each set $\mathbb{P}_{red}$, $\mathbb{P}_{blue}$ along the median d-th coordinate into two sets each:

$$\mathbb{P}_{red} \rightsquigarrow \mathbb{P}_{red,left}, \mathbb{P}_{red,right} \qquad \mathbb{P}_{blue} \rightsquigarrow \mathbb{P}_{blue,left}, \mathbb{P}_{blue,right}$$

Proof of Lemma 1

Idea: Divide and Conquer!

We split each set $\mathbb{P}_{red}$, $\mathbb{P}_{blue}$ along the median d-th coordinate into two sets each:

$$\mathbb{P}_{red} \rightsquigarrow \mathbb{P}_{red,left}, \mathbb{P}_{red,right} \qquad \mathbb{P}_{blue} \rightsquigarrow \mathbb{P}_{blue,left}, \mathbb{P}_{blue,right}$$

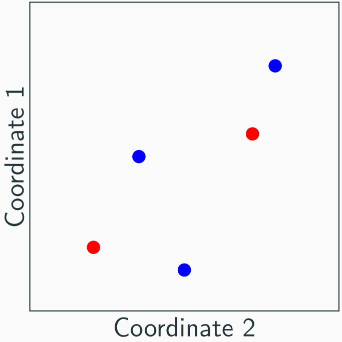
such that it holds for all $p \in \mathbb{P}_{red}, q \in \mathbb{P}_{blue}$:

$$p \in \begin{cases} \mathbb{P}_{red,left}, & p_d \leq m_d \\ \mathbb{P}_{red,right}, & else \end{cases} \qquad q \in \begin{cases} \mathbb{P}_{blue,left}, & q_d \leq m_d \\ \mathbb{P}_{blue,right}, & else \end{cases}$$

Here, m_d denote the median d-th coordinate for $\mathbb{P}_{red} \cup \mathbb{P}_{blue}$.

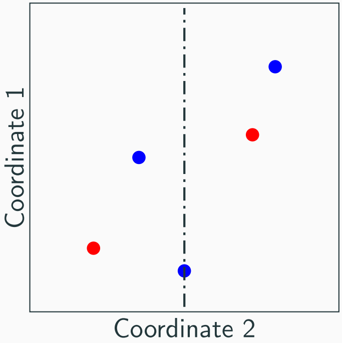
Proof of Lemma 1

Idea: Divide and Conquer!



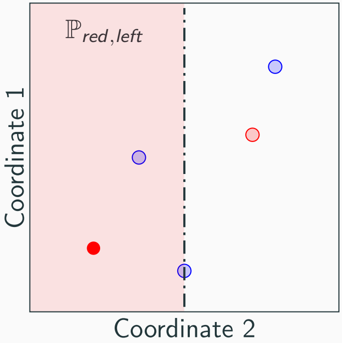
Proof of Lemma 1

Idea: Divide and Conquer!



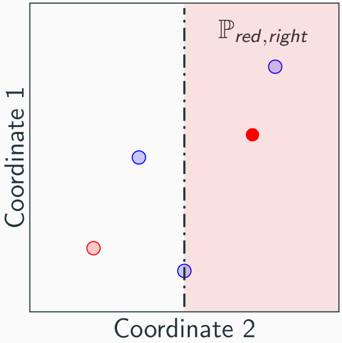
Proof of Lemma 1

Idea: Divide and Conquer!



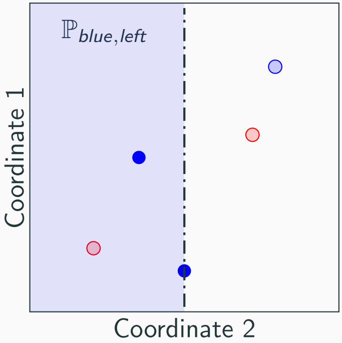
Proof of Lemma 1

Idea: Divide and Conquer!



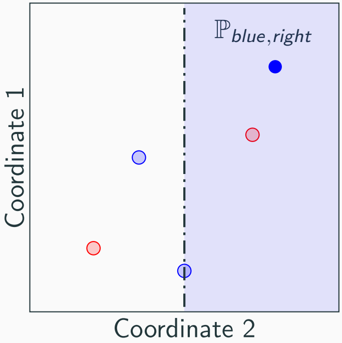
Proof of Lemma 1

Idea: Divide and Conquer!



Proof of Lemma 1

Idea: Divide and Conquer!



Proof of Lemma 1

We solve the dominating pairs problem on the sets

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}.$$

Proof of Lemma 1

We solve the dominating pairs problem on the sets

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}.$$

For $\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$ we only need to consider the first $d-1$ coordinates.

Proof of Lemma 1

We solve the dominating pairs problem on the sets

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}.$$

For $\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$ we only need to consider the first $d-1$ coordinates.

We do not consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$, because $\forall p \in \mathbb{P}_{red,right}, q \in \mathbb{P}_{blue,left} : q_d \leq m_d < p_d$.
Hence no q can never dominate any p.

Proof of Lemma 1

We solve the dominating pairs problem on the sets

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}.$$

For $\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$ we only need to consider the first $d-1$ coordinates.

We do not consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$, because $\forall p \in \mathbb{P}_{red,right}, q \in \mathbb{P}_{blue,left} : q_d \leq m_d < p_d$. Hence no q can never dominate any p.

We stop dividing if the number of points left to consider is 1; we output all pairs of red and blue points if the number of dimensions left is 0.

Proof of Lemma 1

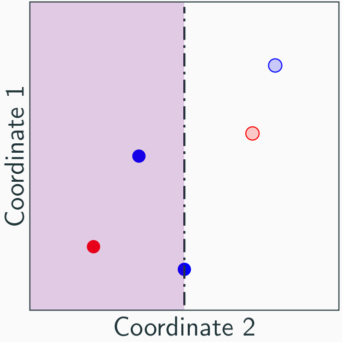
$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$$

We do not need to consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$.

Proof of Lemma 1

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$$

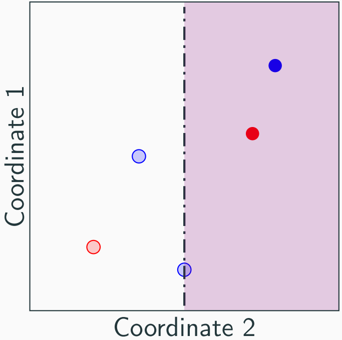
We do not need to consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$.



Proof of Lemma 1

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$$

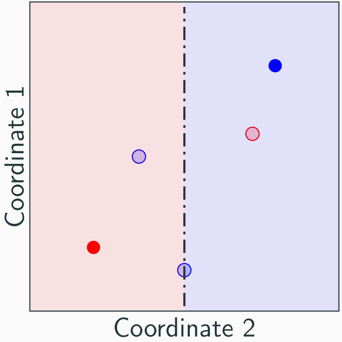
We do not need to consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$.



Proof of Lemma 1

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$$

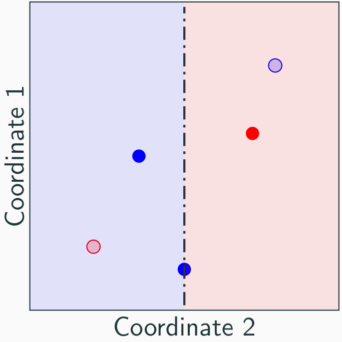
We do not need to consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$.



Proof of Lemma 1

$$\mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \qquad \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}, \qquad \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$$

We do not need to consider $\mathbb{P}_{red,right} \cup \mathbb{P}_{blue,left}$.



Proof of Lemma 1

Without the output costs of $\mathcal{O}(k)$, we get the recurrence relation

$$T_d(n) \leq \underbrace{2T_d\left(\frac{n}{2}\right)}_{(A)} + \underbrace{T_{d-1}(n)}_{(B)} + \underbrace{\mathcal{O}(n)}_{(C)}.$$

Proof of Lemma 1

Without the output costs of $\mathcal{O}(k)$, we get the recurrence relation

$$T_d(n) \le \underbrace{2T_d\left(\frac{n}{2}\right)}_{(A)} + \underbrace{T_{d-1}(n)}_{(B)} + \underbrace{\mathcal{O}(n)}_{(C)}.$$

(A): Two subproblems for half of the data each. $\rightsquigarrow \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}$

(B): Subproblem where d-th coordinate does not matter. $\rightsquigarrow \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$

(C): Time to split current point set about the median.[1]

[1] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer New York, 1985, Section 2.3.2

Proof of Lemma 1

Without the output costs of $\mathcal{O}(k)$, we get the recurrence relation

$$T_d(n) \leq \underbrace{2T_d\left(\frac{n}{2}\right)}_{(A)} + \underbrace{T_{d-1}(n)}_{(B)} + \underbrace{\mathcal{O}(n)}_{(C)}.$$

(A): Two subproblems for half of the data each. $\rightsquigarrow \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,left}, \mathbb{P}_{red,right} \cup \mathbb{P}_{blue,right}$

(B): Subproblem where d-th coordinate does not matter. $\rightsquigarrow \mathbb{P}_{red,left} \cup \mathbb{P}_{blue,right}$

(C): Time to split current point set about the median.

From the termination criteria we get

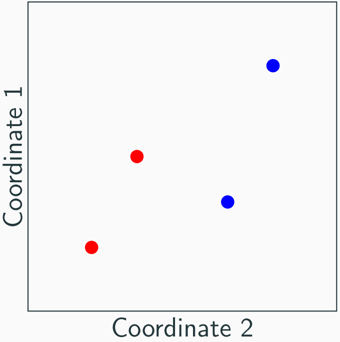
$$T_d(1) = \mathcal{O}(1), \qquad T_0(n) = \mathcal{O}(n).$$

Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)?$$

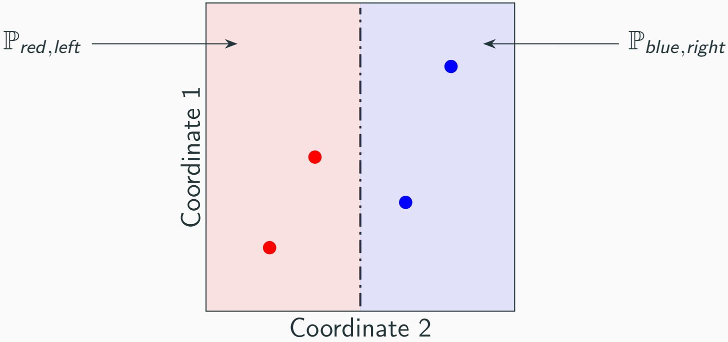
Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)?$$



Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)?$$



Proof of Lemma 1

$$T_0(n) = \mathcal{O}(n)?$$

"If $d = 0$, we just output all pairs of red and blue points."[1]

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time", Lemma 2.1

Proof of Lemma 1

$$T_0(n) = \mathcal{O}(n)?$$

"If $d = 0$, we just output all pairs of red and blue points."

$\rightsquigarrow$ Output: n^2 pairs in the worst case

Proof of Lemma 1

$$T_0(n) = \mathcal{O}(n)?$$

"If $d = 0$, we just output all pairs of red and blue points."

$\rightsquigarrow$ Output: n^2 pairs in the worst case $\implies T_0(n) = \mathcal{O}\left(n^2\right)$ ↯

Proof of Lemma 1

$$T_0(n) = \mathcal{O}(n)?$$

"If $d = 0$, we just output all pairs of red and blue points."

$\rightsquigarrow$ Output: n^2 pairs in the worst case $\implies T_0(n) = \mathcal{O}\left(n^2\right)$ ↯

$\rightsquigarrow$ In the recurrence relation we're completely ignoring the output cost the pairs.

Proof of Lemma 1

$$T_0(n) = \mathcal{O}\left(n\right)?$$

"If $d = 0$, we just output all pairs of red and blue points."

$\rightsquigarrow$ Output: n^2 pairs in the worst case $\implies T_0(n) = \mathcal{O}\left(n^2\right)$ ⚡

$\rightsquigarrow$ In the recurrence relation we're completely ignoring the output cost the pairs. $\implies \mathcal{O}(k)$

Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)$$

A first solution to this equation is that $T_d(n) = \mathcal{O}\left(n\log(n)^d\right)$, yielding an algorithmic runtime of $\mathcal{O}\left(n\log(n)^d + k\right)$. (Additional logarithmic factors can be safed by handling the cases $d = 1$ and $d = 2$ independently.[1])

However, we can do better...

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)$$

Let b be fixed, and define $T'(N) := \max\left\{T_k(i) \mid i = 1, \ldots, n; k = 1, \ldots, d : b^k i \leq N\right\}$.

Substituting into the previous reccurence formula thus yields for some constant c:

$$T'(N) \leq 2T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN.$$

Proof of Lemma 1

$$T_d(n) \leq 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + \mathcal{O}(n)$$

Let b be fixed, and define $T'(N) := \max\left\{ T_k(i) \mid i = 1, \dots, n; k = 1, \dots, d : b^k i \leq N \right\}$.

Substituting into the previous reccurence formula thus yields for some constant c:

$$T'(N) \leq 2T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN.$$

$\implies$ What does this reccurence evaluate to and how do we need to choose b?

Proof of Lemma 1

$$T'(N) \leq 2\,T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN$$

Proof of Lemma 1

$$T'(N) \leq 2T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN$$

By guessing that $T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right)$, we get

$$T'(N) \leq 2\tilde{c}\left(\left[\frac{N}{2}\right]^{1+\varepsilon} - \frac{N}{2}\right) + \tilde{c}\left(\left[\frac{N}{b}\right]^{1+\varepsilon} - \frac{N}{b}\right) + cN$$

Proof of Lemma 1

$$T'(N) \leq 2T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN$$

By guessing that $T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right)$, we get

$$\begin{aligned} T'(N) &\leq 2\tilde{c}\left(\left[\frac{N}{2}\right]^{1+\varepsilon} - \frac{N}{2}\right) + \tilde{c}\left(\left[\frac{N}{b}\right]^{1+\varepsilon} - \frac{N}{b}\right) + cN \\ &= \tilde{c}\left(\frac{2}{2^{1+\varepsilon}}N^{1+\varepsilon} - N\right) + \tilde{c}\left(\frac{1}{b^{1+\varepsilon}}N^{1+\varepsilon} - \frac{1}{b}N\right) + cN \end{aligned}$$

Proof of Lemma 1

$$T'(N) \leq 2T'\left(\frac{N}{2}\right) + T'\left(\frac{N}{b}\right) + cN$$

By guessing that $T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right)$, we get

$$\begin{aligned} T'(N) &\leq 2\tilde{c}\left(\left[\frac{N}{2}\right]^{1+\varepsilon} - \frac{N}{2}\right) + \tilde{c}\left(\left[\frac{N}{b}\right]^{1+\varepsilon} - \frac{N}{b}\right) + cN \\ &= \tilde{c}\left(\frac{2}{2^{1+\varepsilon}}N^{1+\varepsilon} - N\right) + \tilde{c}\left(\frac{1}{b^{1+\varepsilon}}N^{1+\varepsilon} - \frac{1}{b}N\right) + cN \\ &= \left(\frac{2}{2^{1+\varepsilon}} + \frac{1}{b^{1+\varepsilon}}\right)\tilde{c}N^{1+\varepsilon} - \tilde{c}N - \frac{\tilde{c}}{b}N + cN. \end{aligned}$$

Proof of Lemma 1

Thus far we computed

$$T'(N) \leq \underbrace{\left(\frac{2}{2^{1+\varepsilon}} + \frac{1}{b^{1+\varepsilon}}\right)}_{(A)} \tilde{c}N^{1+\varepsilon} - \tilde{c}N - \underbrace{\frac{\tilde{c}}{b}N + cN}_{(B)} .$$

Proof of Lemma 1

Thus far we computed

$$T'(N) \leq \underbrace{\left(\frac{2}{2^{1+\varepsilon}} + \frac{1}{b^{1+\varepsilon}}\right)}_{(A)} \tilde{c}N^{1+\varepsilon} - \tilde{c}N - \underbrace{\frac{\tilde{c}}{b}N + cN}_{(B)}.$$

If (A): $1 = \dfrac{2}{2^{1+\varepsilon}} + \dfrac{1}{b^{1+\varepsilon}}$, and (B): $c \leq \frac{\tilde{c}}{b}$ are fulfilled we get

$$T'(N) \leq \tilde{c}N^{1+\varepsilon} - \tilde{c}N$$

Proof of Lemma 1

Thus far we computed

$$T'(N) \le \underbrace{\left(\frac{2}{2^{1+\varepsilon}} + \frac{1}{b^{1+\varepsilon}}\right)}_{(A)} \tilde{c}N^{1+\varepsilon} - \tilde{c}N - \underbrace{\frac{\tilde{c}}{b}N + cN}_{(B)}.$$

If (A): $1 = \dfrac{2}{2^{1+\varepsilon}} + \dfrac{1}{b^{1+\varepsilon}}$, and (B): $c \le \frac{\tilde{c}}{b}$ are fulfilled we get

$$T'(N) \le \tilde{c}N^{1+\varepsilon} - \tilde{c}N \implies T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right).$$

Proof of Lemma 1

Thus far we computed

$$T'(N) \leq \underbrace{\left(\frac{2}{2^{1+\varepsilon}} + \frac{1}{b^{1+\varepsilon}}\right)}_{(A)} \tilde{c}N^{1+\varepsilon} - \tilde{c}N - \underbrace{\frac{\tilde{c}}{b}N + cN}_{(B)}.$$

If (A): $1 = \dfrac{2}{2^{1+\varepsilon}} + \dfrac{1}{b^{1+\varepsilon}}$, and (B): $c \leq \frac{\tilde{c}}{b}$ are fulfilled we get

$$T'(N) \leq \tilde{c}N^{1+\varepsilon} - \tilde{c}N \implies T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right).$$

It remains to calculate b.

Proof of Lemma 1

If (A): $1 = \dfrac{2}{2^{1+\varepsilon}} + \dfrac{1}{b^{1+\varepsilon}}$, and (B): $c \le \frac{\tilde{c}}{b}$ are fulfilled we get

$$T'(N) \le \tilde{c}N^{1+\varepsilon} - \tilde{c}N \implies T'(N) = \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right).$$

In (A) we get $1 = \dfrac{2}{2^{1+\varepsilon}} + \dfrac{1}{b^{1+\varepsilon}} \iff b^{1+\varepsilon} = b^{1+\varepsilon}\dfrac{1}{2^{\varepsilon}} + 1 \iff -1 = \dfrac{b^{1+\varepsilon}}{2^{\varepsilon}} - b^{1+\varepsilon}$

$$\iff -1 = \frac{1-2^{\varepsilon}}{2^{\varepsilon}} b^{1+\varepsilon} \iff b^{1+\varepsilon} = \frac{2^{\varepsilon}}{2^{\varepsilon}-1} =: c_{\varepsilon}.$$

Proof of Lemma 1

We can now resubstitute:

$$
\begin{aligned}
T'(N) &= \mathcal{O}\left(N^{1+\varepsilon} - N\right) = \mathcal{O}\left(N^{1+\varepsilon}\right) \\
\implies\ T_d(n) &= \mathcal{O}\left(\left(b^d n\right)^{1+\varepsilon}\right) = \mathcal{O}\left(c_\varepsilon^d n^{1+\varepsilon}\right),
\end{aligned}
$$

because $T'(N) := \max\left\{ T_k(i) \mid i = 1, \ldots, n; k = 1, \ldots, d : b^k i \leq N \right\}$.

Finally, outputing all k dominating pairs requires $\mathcal{O}(k)$ time. $\square$

Proof of Lemma 2[1]

Lemma 2

Given two matrices $A \in \mathbb{R}^{n \times d}$, $B \in \mathbb{R}^{d \times n}$, *and* $\varepsilon \in (0,1)$, $c_\varepsilon := \dfrac{2^\varepsilon}{2^\varepsilon - 1}$, *we can compute their min-plus product* $A \otimes B$ *in* $\mathcal{O}\left(d c_\varepsilon^d n^{1+\varepsilon} + n^2\right)$ *time.*

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time", Lemma 3.1

Proof of Lemma 2

Note
We need to evaluate the inequality $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$.

For $k \neq \tilde{k}$ we need to break ties such as $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$ where "$\leq$" is fulfilled by "$=$" to obtain a uniquely defined minimum. W are considered to be "$<$" if $k < \tilde{k}$.

Proof of Lemma 2

Reminder
We compute $C = A \otimes B$ elementwise through $c_{i,j} = \min_k a_{i,k} + b_{k,j}$.

Proof of Lemma 2

Reminder
We compute $C = A \otimes B$ elementwise through $c_{i,j} = \min_k a_{i,k} + b_{k,j}$.

Consider A to have entries $(a_{i,j})_{i,j=1}^{n,d}$, and B to have entries $(b_{i,j})_{i,j=1}^{d,n}$. We want to compute the pairs

$$X_k = \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$$

Proof of Lemma 2

Reminder
We compute $C = A \otimes B$ elementwise through $c_{i,j} = \min_k a_{i,k} + b_{k,j}$.

Consider A to have entries $(a_{i,j})_{i,j=1}^{n,d}$, and B to have entries $(b_{i,j})_{i,j=1}^{d,n}$. We want to compute the pairs

$$X_k = \{(i,j) \,|\, \forall k' = 1, \dots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$$

After computing these X_k, we can then set $C = A \otimes B$ elementwise as follows:

$$(i,j) \in X_k \implies c_{i,j} = a_{i,k} + b_{k,j}$$

Proof of Lemma 2

$$X_k = \{(i,j) \mid \forall k' = 1,\ldots,d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$$

Proof of Lemma 2

$$\begin{aligned} X_k &= \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\} \\ &= \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} - a_{i,k'} \leq b_{k',j} - b_{k,j}\} \end{aligned}$$

Proof of Lemma 2

$$
\begin{aligned}
X_k &= \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\} \\
&= \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} - a_{i,k'} \leq b_{k',j} - b_{k,j}\}
\end{aligned}
$$

This effectively amounts to computing all dominant pairs between the sets

$$
\begin{aligned}
\mathcal{A}_k &:= \{(a_{i,k} - a_{i,1}), (a_{i,k} - a_{i,2}), \ldots, (a_{i,k} - a_{i,d})\}_{i=1}^{n}, \text{and} \\
\mathcal{B}_k &:= \{(b_{1,j} - b_{k,j}), (b_{2,j} - b_{k,j}), \ldots, (b_{d,j} - b_{k,j})\}_{j=1}^{n},
\end{aligned}
$$

where $\mathcal{A}_k$ takes the role of red points and $\mathcal{B}_k$ acts as the set of blue points.

Proof of Lemma 2

$$
\begin{aligned}
X_k &= \{(i,j) \mid \forall k' = 1, \dots, d : a_{i,k} + b_{k,j} \le a_{i,k'} + b_{k',j}\} \\
&= \{(i,j) \mid \forall k' = 1, \dots, d : a_{i,k} - a_{i,k'} \le b_{k',j} - b_{k,j}\}
\end{aligned}
$$

This effectively amounts to computing all dominant pairs between the sets

$$
\begin{aligned}
\mathcal{A}_k &:= \{(a_{i,k} - a_{i,1}), (a_{i,k} - a_{i,2}), \dots, (a_{i,k} - a_{i,d})\}_{i=1}^{n}, \text{and} \\
\mathcal{B}_k &:= \{(b_{1,j} - b_{k,j}), (b_{2,j} - b_{k,j}), \dots, (b_{d,j} - b_{k,j})\}_{j=1}^{n},
\end{aligned}
$$

where $\mathcal{A}_k$ takes the role of red points and $\mathcal{B}_k$ acts as the set of blue points.

By Lemma 1, this takes an effort of $\mathcal{O}\left(c_\varepsilon^d n^{1+\varepsilon} + |X_k|\right)$.

Proof of Lemma 2

The penultimate step is to compute that

$$\mathcal{O}\left(\sum_{k=1}^{d}\left(c_\varepsilon^d n^{1+\varepsilon} + |X_k|\right)\right) = \mathcal{O}\left(d c_\varepsilon^d n^{1+\varepsilon} + \sum_{k=1}^{d}|X_k|\right),$$

leaving only to compute that $\displaystyle\sum_{k=1}^{d}|X_k| = n^2$.

Proof of Lemma 2

The penultimate step is to compute that

$$\mathcal{O}\left(\sum_{k=1}^{d}\left(c_\varepsilon^d n^{1+\varepsilon} + |X_k|\right)\right) = \mathcal{O}\left(d c_\varepsilon^d n^{1+\varepsilon} + \sum_{k=1}^{d} |X_k|\right),$$

leaving only to compute that $\displaystyle\sum_{k=1}^{d} |X_k| = n^2$.

Note

It is possible to recover the shortest paths from the construction of the X_ks. We will see more on this later.

Proof of Lemma 2

Suppose an index pair (i, j) were to be included in X_k, and in $X_{\tilde{k}}$, with $k \neq \tilde{k}$.

Proof of Lemma 2

Suppose an index pair (i, j) were to be included in X_k, and in $X_{\tilde{k}}$, with $k \neq \tilde{k}$.

Recall the definition of X_k:

$$X_k = \{(i, j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}.$$

Thus we get that $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$, because $(i, j) \in X_k$.

Proof of Lemma 2

Suppose an index pair (i, j) were to be included in X_k, and in $X_{\tilde{k}}$, with $k \neq \tilde{k}$.

Recall the definition of X_k:

$$X_k = \{(i, j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}.$$

Thus we get that $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$, because $(i, j) \in X_k$.

Applying the definition of $X_{\tilde{k}}$, we then also get $a_{i,\tilde{k}} + b_{\tilde{k},j} \leq a_{i,k} + b_{k,j}$.

Proof of Lemma 2

Suppose an index pair (i, j) were to be included in X_k, and in $X_{\tilde{k}}$, with $k \neq \tilde{k}$.

Recall the definition of X_k:

$$X_k = \{(i, j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}.$$

Thus we get that $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$, because $(i, j) \in X_k$.

Applying the definition of $X_{\tilde{k}}$, we then also get $a_{i,\tilde{k}} + b_{\tilde{k},j} \leq a_{i,k} + b_{k,j}$.

This means that $a_{i,\tilde{k}} + b_{\tilde{k},j} = a_{i,k} + b_{k,j}$.

To break this tie, we can w.l.o.g. assume $k < \tilde{k}$, which would result in $(i, j) \notin X_{\tilde{k}}$.

Proof of Lemma 2

Suppose an index pair (i, j) were to be included in X_k, and in $X_{\tilde{k}}$, with $k \neq \tilde{k}$.

Recall the definition of X_k:

$$X_k = \{(i, j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}.$$

Thus we get that $a_{i,k} + b_{k,j} \leq a_{i,\tilde{k}} + b_{\tilde{k},j}$, because $(i, j) \in X_k$.

Applying the definition of $X_{\tilde{k}}$, we then also get $a_{i,\tilde{k}} + b_{\tilde{k},j} \leq a_{i,k} + b_{k,j}$.

This means that $a_{i,\tilde{k}} + b_{\tilde{k},j} = a_{i,k} + b_{k,j}$.

To break this tie, we can w.l.o.g. assume $k < \tilde{k}$, which would result in $(i, j) \notin X_{\tilde{k}}$.

$\implies$ Every index pair (i, j) can only be included in at most one X_k.

Proof of Lemma 2

Now assume that there would exist an index pair (i, j) that is contained in none of the X_k.
(Recall $X_k = \{(i, j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$.)

Proof of Lemma 2

Now assume that there would exist an index pair (i, j) that is contained in none of the X_k.
(Recall $X_k = \{(i, j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \le a_{i,k'} + b_{k',j}\}$.)

This is equivalent to the condition that

$$\forall k = 1, \ldots, d : \exists k' = 1, \ldots, d : a_{i,k} + b_{k,j} > a_{i,k'} + b_{k',j}.$$

Proof of Lemma 2

Now assume that there would exist an index pair (i, j) that is contained in none of the X_k.
(Recall $X_k = \{(i, j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$.)

This is equivalent to the condition that

$$\forall k = 1, \ldots, d : \exists k' = 1, \ldots, d : a_{i,k} + b_{k,j} > a_{i,k'} + b_{k',j}.$$

Because the set $\{1, \ldots, d\}$ is finite we can choose $\hat{k} := \underset{m=1,\ldots,d}{\arg\min}\, a_{i,m} + b_{m,j}$.

Proof of Lemma 2

Now assume that there would exist an index pair (i, j) that is contained in none of the X_k.
(Recall $X_k = \{(i,j) \mid \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$.)

This is equivalent to the condition that

$$\forall k = 1, \ldots, d : \exists k' = 1, \ldots, d : a_{i,k} + b_{k,j} > a_{i,k'} + b_{k',j}.$$

Because the set $\{1, \ldots, d\}$ is finite we can choose $\hat{k} := \operatorname*{arg\,min}_{m=1,\ldots,d} a_{i,m} + b_{m,j}$.

Hence, choosing $k = \hat{k}$ yields $\forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} = a_{i,\hat{k}} + b_{\hat{k},j} \leq a_{i,k'} + b_{k',j}$. $\lightning$

Proof of Lemma 2

Now assume that there would exist an index pair (i, j) that is contained in none of the X_k.
(Recall $X_k = \{(i,j) \,|\, \forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} \leq a_{i,k'} + b_{k',j}\}$.)

This is equivalent to the condition that

$$\forall k = 1, \ldots, d : \exists k' = 1, \ldots, d : a_{i,k} + b_{k,j} > a_{i,k'} + b_{k',j}.$$

Because the set $\{1, \ldots, d\}$ is finite we can choose $\hat{k} := \underset{m=1,\ldots,d}{\arg\min}\, a_{i,m} + b_{m,j}$.

Hence, choosing $k = \hat{k}$ yields $\forall k' = 1, \ldots, d : a_{i,k} + b_{k,j} = a_{i,\hat{k}} + b_{\hat{k},j} \leq a_{i,k'} + b_{k',j}$. ↯

$\Longrightarrow$ Every index pair (i, j) has to be included in one X_k.

Proof of Lemma 2

Recall
Every index pair (i, j) can only be included in at most one X_k.

Every index pair (i, j) has to be included in one X_k.

$\implies$ Over all (disjoint) $X_k, k = 1, \ldots, d$, every index pair (i, j) is encountered exactly once.

Proof of Lemma 2

Recall
Every index pair (i, j) can only be included in at most one X_k.

Every index pair (i, j) has to be included in one X_k.

$\Longrightarrow$ Over all (disjoint) X_k, $k = 1, \ldots, d$, every index pair (i, j) is encountered exactly once.

$$\Longrightarrow \sum_{k=1}^{d} |X_k| = n^2$$

$$\Longrightarrow \mathcal{O}\left(\sum_{k=1}^{d} \left(c_\varepsilon^d n^{1+\varepsilon} + |X_k|\right)\right) = \mathcal{O}\left(dc_\varepsilon^d n^{1+\varepsilon} + n^2\right) \qquad \square$$

Theorem 3

Given any two matrices $A, B \in \mathbb{R}^{n \times n}$ we can compute their min-plus (distance) product in a time of $\mathcal{O}\left(n^3 / \log(n)\right)$.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(nˆ3/log(n)) Time", Theorem 3.2

Proof of Theorem 3

We recall the idea of splitting matrices to multiply them:

$$\begin{pmatrix} A_1 & A_2 & \cdots & A_d \end{pmatrix} \qquad \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_d \end{pmatrix}$$

$\Longrightarrow$ Strassen is not applicable here — we need to make use of the relation between matrix multiplication and matrix closure.

Proof of Theorem 3

We split our matrices A and B into $\frac{n}{d}$ blocks, that is $\forall m = 1, \dots, \frac{n}{d} : A_m \in \mathbb{R}^{n \times d}, B_m \in \mathbb{R}^{d \times n}$ for some fixed d. (If necessary we round $\frac{n}{d}$ and adjust the number of blocks accordingly.)

Proof of Theorem 3

We split our matrices A and B into $\frac{n}{d}$ blocks, that is $\forall m = 1, \dots, \frac{n}{d} : A_m \in \mathbb{R}^{n \times d}, B_m \in \mathbb{R}^{d \times n}$ for some fixed d. (If necessary we round $\frac{n}{d}$ and adjust the number of blocks accordingly.)

We then compute the distance products $A_i \otimes B_i$ for all $i = 1, \dots, \frac{n}{d}$, and set the product to be defined by the element-wise minimum, i.e. $c_{i,j} := \min_{m=1,\dots,\frac{n}{d}} (A_m \otimes B_m)_{i,j}$, where $i, j = 1, \dots, n$.

Proof of Theorem 3

We split our matrices A and B into $\frac{n}{d}$ blocks, that is $\forall m = 1, \dots, \frac{n}{d} : A_m \in \mathbb{R}^{n \times d}, B_m \in \mathbb{R}^{d \times n}$ for some fixed d. (If necessary we round $\frac{n}{d}$ and adjust the number of blocks accordingly.)

We then compute the distance products $A_i \otimes B_i$ for all $i = 1, \dots, \frac{n}{d}$, and set the product to be defined by the element-wise minimum, i.e. $c_{i,j} := \min_{m=1,\dots,\frac{n}{d}} (A_m \otimes B_m)_{i,j}$, where $i, j = 1, \dots, n$.

By Lemma 2, this procedure requires $\mathcal{O}\left(\frac{n}{d}\left(dc_\varepsilon^d n^{1+\varepsilon} + n^2\right)\right) = \mathcal{O}\left(c_\varepsilon^d n^{2+\varepsilon} + \frac{n^3}{d}\right)$ time.

Proof of Theorem 3

We split our matrices A and B into $\frac{n}{d}$ blocks, that is $\forall m = 1, \ldots, \frac{n}{d} : A_m \in \mathbb{R}^{n \times d}, B_m \in \mathbb{R}^{d \times n}$ for some fixed d. (If necessary we round $\frac{n}{d}$ and adjust the number of blocks accordingly.)

We then compute the distance products $A_i \otimes B_i$ for all $i = 1, \ldots, \frac{n}{d}$, and set the product to be defined by the element-wise minimum, i.e. $c_{i,j} := \min_{m=1,\ldots,\frac{n}{d}} (A_m \otimes B_m)_{i,j}$, where $i, j = 1, \ldots, n$.

By Lemma 2, this procedure requires $\mathcal{O}\left(\frac{n}{d}\left(dc_\varepsilon^d n^{1+\varepsilon} + n^2\right)\right) = \mathcal{O}\left(c_\varepsilon^d n^{2+\varepsilon} + \frac{n^3}{d}\right)$ time.

It now only remains to choose the constant d.

Proof of Theorem 3

We want to assure $\dfrac{n^3}{d} > c_\varepsilon^d n^{2+\varepsilon}$.

Proof of Theorem 3

We want to assure $\frac{n^3}{d} > c_\varepsilon^d n^{2+\varepsilon}$.

An obvious choice is a term like $d = n^{1-\varepsilon}$, however this result in exponential growth:

$$c_\varepsilon^d = c_\varepsilon^{n^{\mathbf{1}-\varepsilon}} \sim \mathfrak{c}^n \lightning$$

Proof of Theorem 3

We want to assure $\dfrac{n^3}{d} > c_\varepsilon^d n^{2+\varepsilon}$.

An obvious choice is a term like $d = n^{1-\varepsilon}$, however this result in exponential growth:

$$c_\varepsilon^d = c_\varepsilon^{n^{1-\varepsilon}} \sim \mathfrak{c}^n \lightning$$

It is better to choose $d = \tilde{c}\log(n)$ with $\tilde{c}$ sufficiently small and depending on ε:

$$c_\varepsilon^d = c_\varepsilon^{\tilde{c}\log(n)} \sim n^{\mathfrak{c}}$$

Proof of Theorem 3

We want to assure $\dfrac{n^3}{d} > c_\varepsilon^d n^{2+\varepsilon}$.

An obvious choice is a term like $d = n^{1-\varepsilon}$, however this result in exponential growth:

$$c_\varepsilon^d = c_\varepsilon^{n^{\mathbf{1}-\varepsilon}} \sim \mathfrak{c}^n \lightning$$

It is better to choose $d = \tilde{c}\log(n)$ with $\tilde{c}$ sufficiently small and depending on ε:

$$c_\varepsilon^d = c_\varepsilon^{\tilde{c}\log(n)} \sim n^{\mathfrak{c}}$$

$$\implies \mathcal{O}\left(c_\varepsilon^d n^{2+\varepsilon} + \tfrac{n^3}{d}\right) = \mathcal{O}\left(\tfrac{n^3}{\log(n)}\right). \qquad \square$$

Proof of Corollary 4[1]

Corollary 4
We can solve the all pairs shortest paths problem for a graph $G = (V, E)$ with $|V| = n$ nodes in $\mathcal{O}\left(n^3/\log(n)\right)$ time.

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time", Corollary 3.3

Proof of Corollary 4

We consider A and B to be the matrices defined by $w_{i,j} := \begin{cases} w(e), & \exists e \in E : e = (i,j) \\ \infty, & else \end{cases}$.

The corollary then follows by applying Theorem 3. $\square$

Recovery of Shortest Paths and Setting Product Matrix Entries

Recovery of Shortest Paths

We want to recover a shortest path from vertex i to vertex j. Consider $(i, j) \in X_k$.

Recovery of Shortest Paths

We want to recover a shortest path from vertex i to vertex j. Consider $(i, j) \in X_k$.

This means that a shortest path from i to j must go through k because
$\forall k' = 1, \ldots, n : w_{i,k} + w_{k,j} \leq w_{i,k'} + w_{k',j}$.

(*Taking a path through any vertex other than k increases the weight.*)

Recovery of Shortest Paths

We want to recover a shortest path from vertex i to vertex j. Consider $(i, j) \in X_k$.

This means that a shortest path from i to j must go through k because
$\forall k' = 1, \ldots, n : w_{i,k} + w_{k,j} \leq w_{i,k'} + w_{k',j}$.

(*Taking a path through any vertex other than k increases the weight.*)

For neighbouring vertices, where a shortest path is the edge directly between them, we get that
$(i, j) \in X_m$, with $m = \min\{i, j\}$ because (assuming $i < j$)
$\forall k' = 1, \ldots, n : w_{i,i} + w_{i,j} \leq w_{i,k'} + w_{k',j}$.

(*Direct shortest paths between neighbouring vertices i and j fall into X_i or X_j.*)

Recovery of Shortest Paths

Data: The sets X_k, source vertex i, target vertex j

def $get\_shortest\_path(i, j)$

  Set k such that $(i, j) \in X_k$;

  if $k \notin \{i, j\}$ **then**

    Set $\mathfrak{p} := get\_shortest\_path(i, k) \oplus get\_shortest\_path(k, j)$;

  else

    Set $\mathfrak{p} := (i, j)$;

  return $\mathfrak{p}$

With the usual definition $(a, \ldots, b) \oplus (b, \ldots, c) := (a, \ldots, b, \ldots, c)$.

Setting the Product Matrix' Entries $c_{i,j}$[1]

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.[1]

[1] Chan, "All-Pairs Shortest Paths with Real Weights in O(n^3/log(n)) Time"

Setting the Product Matrix' Entries $c_{i,j}$

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.

Instead we consider a "bucket" $\mathcal{B}_i$ for every $i = 1, \ldots, n$ and an additional "slot" matrix S of dimension $n \times n$.

Setting the Product Matrix' Entries $c_{i,j}$

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.

Instead we consider a "bucket" $\mathcal{B}_i$ for every $i = 1, \ldots, n$ and an additional "slot" matrix S of dimension $n \times n$.

For $(i, j) \in X_k$ we insert (j, k) into $\mathcal{B}_i$.

Setting the Product Matrix' Entries $c_{i,j}$

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.

Instead we consider a "bucket" $\mathcal{B}_i$ for every $i = 1, \ldots, n$ and an additional "slot" matrix S of dimension $n \times n$.

For $(i, j) \in X_k$ we insert (j, k) into $\mathcal{B}_i$.

For every $i = 1, \ldots, n$ we presort the bucket $\mathcal{B}_i$ with respect to the first index. (*This corresponds to the j from the index pairs above.*)

Setting the Product Matrix' Entries $c_{i,j}$

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.

Instead we consider a "bucket" $\mathcal{B}_i$ for every $i = 1, \ldots, n$ and an additional "slot" matrix S of dimension $n \times n$.

For $(i, j) \in X_k$ we insert (j, k) into $\mathcal{B}_i$.

For every $i = 1, \ldots, n$ we presort the bucket $\mathcal{B}_i$ with respect to the first index. (*This corresponds to the j from the index pairs above.*)

We then set the entry $s_{i,j}$ to k for every $(j, k) \in \mathcal{B}_i$.

Setting the Product Matrix' Entries $c_{i,j}$

Setting $c_{i,j} = a_{i,k} + b_{k,j}$ directly is not going to work due to random access constraints.

Instead we consider a "bucket" $\mathcal{B}_i$ for every $i = 1, \ldots, n$ and an additional "slot" matrix S of dimension $n \times n$.

For $(i, j) \in X_k$ we insert (j, k) into $\mathcal{B}_i$.

For every $i = 1, \ldots, n$ we presort the bucket $\mathcal{B}_i$ with respect to the first index. (*This corresponds to the j from the index pairs above.*)

We then set the entry $s_{i,j}$ to k for every $(j, k) \in \mathcal{B}_i$.

Finally, we can set $c_{i,j} = a_{i,s_{i,j}} + b_{s_{i,j},j}$ in $\mathcal{O}\left(n^2\right)$ time.

Summary & Discussion

Summary & Discussion

1. Computing dominating pairs $\mathcal{O}\left(c_\varepsilon^d n^{1+\varepsilon} + k\right)$
2. Computing $A \otimes B$ for rectangular matrices $\mathcal{O}\left(dc_\varepsilon^d n^{1+\varepsilon} + n^2\right)$
3. Computing $A \otimes B$ for quadratic matrices $\mathcal{O}\left(\frac{n^3}{\log(n)}\right)$
4. Making the jump to APSP problems

Data: Weight matrix W
Split W into $W_1, \ldots, W_{\frac{n}{d}}$;
for $i = 1, \ldots, \frac{n}{d}$ **do**
  Compute the min-plus products $W_i \otimes W_i$;
    Create the index sets X_k;
    Recover the product matrix' entries via bins/buckets;
Set the final shortest paths matrix elementwise as the minimum over the $W_i \otimes W_i$;