Dynamic String Alignment

Panagiotis Charalampopoulos[1,2], Tomasz Kociumaka[3], and Shay Mozes[4]

[1]King's College London, United Kingdom

[2]University of Warsaw, Poland

[3]Bar-Ilan University, Israel

[4]The Interdisciplinary Center Herzliya, Israel

CPM 2020

June 17-19, 2020

Problem Definition

String Alignment

Input: two strings of total length n and weights

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Goal: compute an alignment with maximum weight.

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Goal: compute an alignment with maximum weight.

$$\begin{array}{ccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
S = & \mathtt{a} & - & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{b} \\
 & | & & | & | & | & & & | & \cdot & | \\
T = & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{a} & \mathtt{a} & - & - & \mathtt{c} & \mathtt{d} & \mathtt{b}
\end{array}$$

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Goal: compute an alignment with maximum weight.

$$\begin{array}{cccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ S = & \mathtt{a} & - & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{b} \\ & | & & | & | & | & & & | & \cdot & | \\ T = & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{a} & \mathtt{a} & - & - & \mathtt{c} & \mathtt{d} & \mathtt{b} \end{array}$$

Alignment's weight: $6w_{match} + w_{mis} + 3w_{gap}$.

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Goal: compute an alignment with maximum weight.

$$
\begin{array}{rcccccccccc}
 & {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} & {\scriptstyle 6} & {\scriptstyle 7} & {\scriptstyle 8} & {\scriptstyle 9} & {\scriptstyle 10} \\
S = & \mathtt{a} & - & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{b} \\
 & | & & | & | & | & & & | & \cdot & | \\
T = & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{a} & \mathtt{a} & - & - & \mathtt{c} & \mathtt{d} & \mathtt{b}
\end{array}
$$

Alignment's weight: $6w_{match} + w_{mis} + 3w_{gap}$.

Generalizes the Longest Common Subsequence problem and the Edit Distance problem.

Problem Definition

String Alignment

Input: two strings of total length n and weights

- w_{match} – for aligning a pair of matching letters,
- w_{mis} – for aligning a pair of mismatching letters,
- w_{gap} – for letters that are not aligned.

Goal: compute an alignment with maximum weight.

$S =$ a b a a b a c b b

$T =$ a c b a a c d b

Generalizes the Longest Common Subsequence problem and the Edit Distance problem.

Related Work

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk; Cybernetics 1968]

[Needleman-Wunsch; Journal of Molecular Biology 1970]

[Wagner-Fischer; Journal of the ACM 1974]

Related Work

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk; Cybernetics 1968]

[Needleman-Wunsch; Journal of Molecular Biology 1970]

[Wagner-Fischer; Journal of the ACM 1974]

Several works improved the complexity by polylogarithmic factors.

[Masek-Paterson; Journal of Computer and System Sciences 1980]

[Crochemore-Landau-Ziv-Ukelson; SIAM Journal on Computing 2003]

[Grabowski; Discrete Applied Mathematics 2016]

Related Work

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk; Cybernetics 1968]

[Needleman-Wunsch; Journal of Molecular Biology 1970]

[Wagner-Fischer; Journal of the ACM 1974]

Several works improved the complexity by polylogarithmic factors.

[Masek-Paterson; Journal of Computer and System Sciences 1980]

[Crochemore-Landau-Ziv-Ukelson; SIAM Journal on Computing 2003]

[Grabowski; Discrete Applied Mathematics 2016]

A strongly subquadratic-time algorithm would refute the Strong Exponential Time Hypothesis (SETH).

[Backurs-Indyk; SIAM Journal on Computing 2018]

[Bringmann-Künnemann; FOCS 2015]

[Abboud-Hansen-Vassilevska Williams-Williams; STOC 2016]

Related Work

The DP algorithm is online: it can handle appending a letter to either of the strings in $\mathcal{O}(n)$ time. It can also handle deleting the last letter of either of the strings.

Related Work

The DP algorithm is online: it can handle appending a letter to either of the strings in $\mathcal{O}(n)$ time. It can also handle deleting the last letter of either of the strings.

Several works considered prepending letters or deleting the first letter in either of the strings, culminating in an $\mathcal{O}(n)$-time algorithm.

[Landau-Myers-Schmidt; SIAM Journal on Computing 1998]

[Kim-Park; Journal of Discrete Algorithms 2004]

[Ishida-Inenaga-Shinohara-Takeda; FCT 2005]

[Tiskin; arxiv 2007]

[Hyyrö-Narisawa-Inenaga; JDA 2015]

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.
[Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.
[Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

The lower bound for the static version of the problem means that we cannot hope for $\mathcal{O}(n^{1-\epsilon})$ update time for any constant $\epsilon > 0$.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.
[Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

The lower bound for the static version of the problem means that we cannot hope for $\mathcal{O}(n^{1-\epsilon})$ update time for any constant $\epsilon > 0$.

Integer alignment weights $\leq w$: update time $\tilde{\mathcal{O}}(nw)$.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.
[Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

The lower bound for the static version of the problem means that we cannot hope for $\mathcal{O}(n^{1-\epsilon})$ update time for any constant $\epsilon > 0$.

Integer alignment weights $\leq w$: update time $\tilde{\mathcal{O}}(nw)$.

- Based on Tiskin's algorithm for efficient distance multiplication of simple unit-Monge matrices.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings. [Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

The lower bound for the static version of the problem means that we cannot hope for $\mathcal{O}(n^{1-\epsilon})$ update time for any constant $\epsilon > 0$.

Integer alignment weights $\leq w$: update time $\tilde{\mathcal{O}}(nw)$.

- Based on Tiskin's algorithm for efficient distance multiplication of simple unit-Monge matrices.

Alignment weights of size $n^{\mathcal{O}(1)}$: update time $\tilde{\mathcal{O}}(n\sqrt{n})$.

Our Results

We consider a dynamic setting, where letter updates (insertions, deletions, substitutions) are allowed anywhere in the strings.
[Hyyrö-Narisawa-Inenaga; JDA 2015]: practical, $\mathcal{O}(n^2)$ worst-case time.

The lower bound for the static version of the problem means that we cannot hope for $\mathcal{O}(n^{1-\epsilon})$ update time for any constant $\epsilon > 0$.

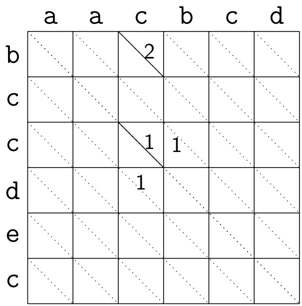
Integer alignment weights $\leq w$: update time $\tilde{\mathcal{O}}(nw)$.

- Based on Tiskin's algorithm for efficient distance multiplication of simple unit-Monge matrices.

Alignment weights of size $n^{\mathcal{O}(1)}$: update time $\tilde{\mathcal{O}}(n\sqrt{n})$.

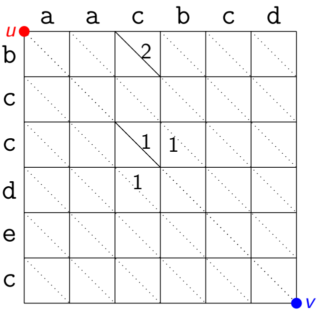
- Based on black-boxes from planar graphs.

Preliminaries



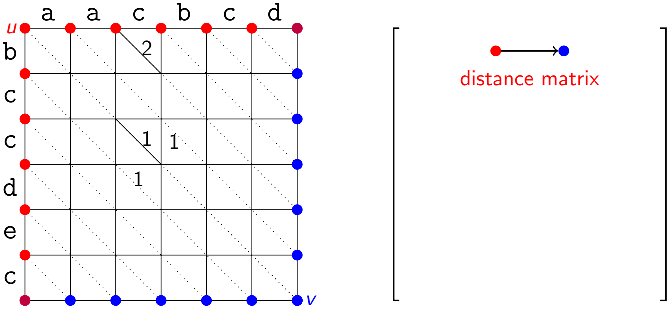
$w_{match} = 1$, $w_{mis} = 2$, and $w_{gap} = 1$.

Preliminaries



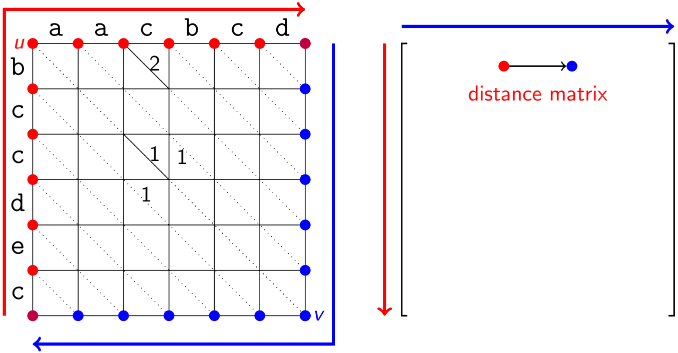
$$w_{match} = 1,\ w_{mis} = 2, \text{ and } w_{gap} = 1.$$

$$\mathtt{LCS}(S, T) = |S| + |T| - d(u, v).$$



$$w_{match} = 1,\ w_{mis} = 2, \text{ and } w_{gap} = 1.$$

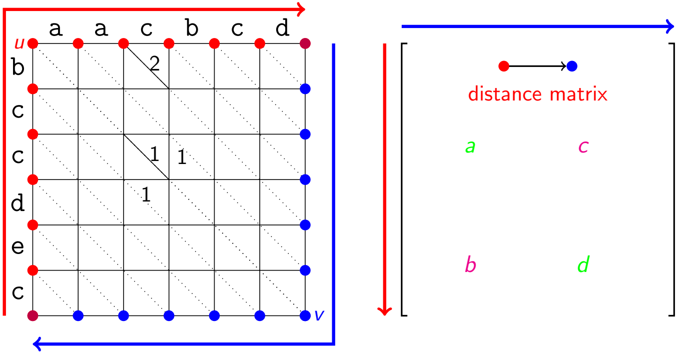
$$\mathtt{LCS}(S, T) = |S| + |T| - d(u, v).$$



$$w_{match} = 1,\ w_{mis} = 2, \text{ and } w_{gap} = 1.$$

$$\mathrm{LCS}(S, T) = |S| + |T| - d(u, v).$$

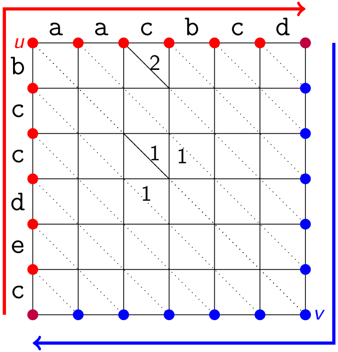
Preliminaries

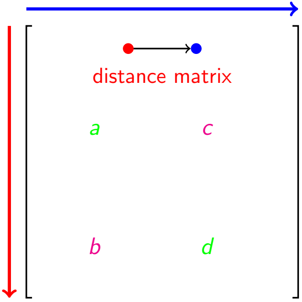


$$w_{match} = 1,\ w_{mis} = 2, \text{ and } w_{gap} = 1.$$

$$\mathtt{LCS}(S, T) = |S| + |T| - d(u, v).$$

A matrix M is *Monge* if $M[i,j] + M[i',j'] \le M[i',j] + M[i,j']$ for all $i < i'$ and $j < j'$.





$$w_{match} = 1,\ w_{mis} = 2, \text{ and } w_{gap} = 1.$$

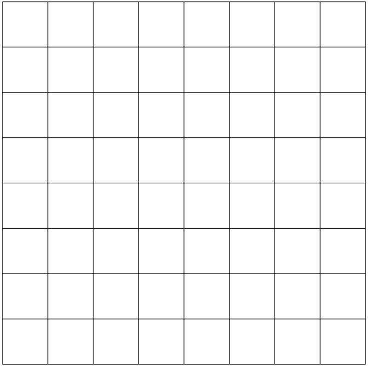
$$\mathtt{LCS}(S, T) = |S| + |T| - d(u, v).$$

A matrix M is *Monge* if $M[i,j] + M[i',j'] \leq M[i',j] + M[i,j']$ for all $i < i'$ and $j < j'$.

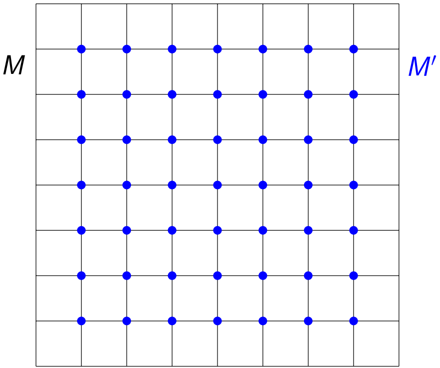
This distance matrix is Monge, and, in fact, unit-Monge.

Unit-Monge Matrices

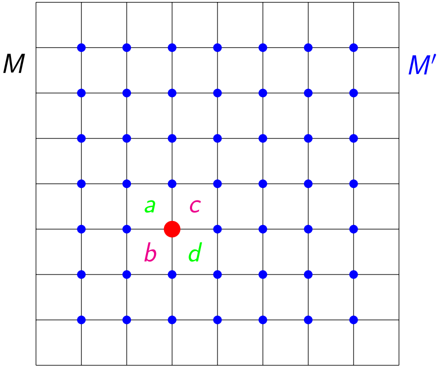
M



Unit-Monge Matrices

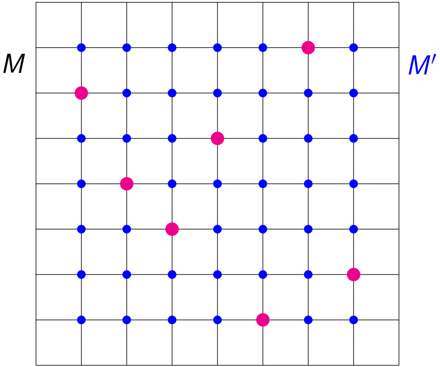


Unit-Monge Matrices



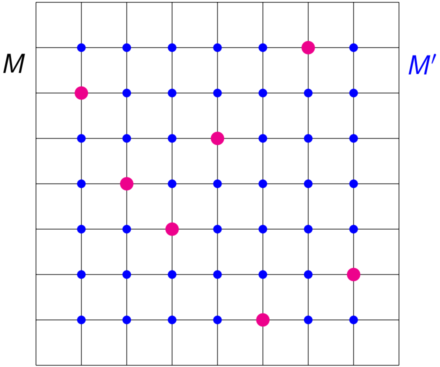
● $= b + c - a - d$

Unit-Monge Matrices



M is unit-Monge if and only if M' is a permutation matrix.

Unit-Monge Matrices



M is unit-Monge if and only if M' is a permutation matrix.

We can represent an $n \times n$ unit-Monge matrix M in $\tilde{\mathcal{O}}(n)$ space so that each entry can be retrieved in $\tilde{\mathcal{O}}(1)$ time.

Distance Product of Unit-Monge Matrices

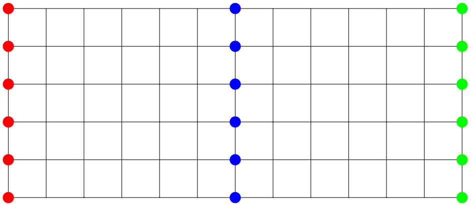
Distance Product

The $(\min, +)$ product or distance product of an $m \times k$ matrix A and a $k \times n$ matrix B, denoted by $A \odot B$ is an $m \times n$ matrix C, such that $C[i,j] = \min_{1 \le r \le k} \{A[i,r] + B[r,j]\}$.

Distance Product of Unit-Monge Matrices

Distance Product

The $(\min, +)$ product or distance product of an $m \times k$ matrix A and a $k \times n$ matrix B, denoted by $A \odot B$ is an $m \times n$ matrix C, such that $C[i,j] = \min_{1 \le r \le k} \{A[i,r] + B[r,j]\}$.



$A :=$ distance matrix

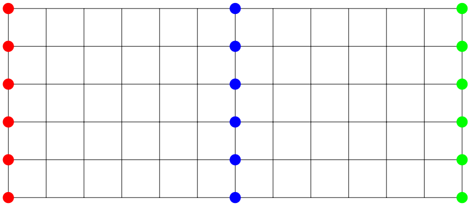
$B :=$ distance matrix

$C :=$ distance matrix

$C = A \odot B$

Distance Product

The $(\min, +)$ product or distance product of an $m \times k$ matrix A and a $k \times n$ matrix B, denoted by $A \odot B$ is an $m \times n$ matrix C, such that $C[i,j] = \min_{1 \le r \le k} \{A[i,r] + B[r,j]\}$.



$A :=$ distance matrix (red → blue)

$B :=$ distance matrix (blue → green)

$C :=$ distance matrix (red → green)

$C = A \odot B$

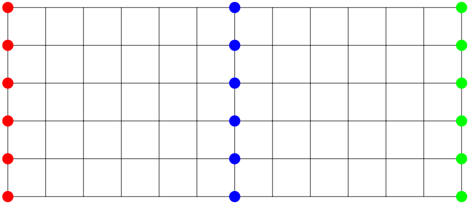
Efficient $(\min, +)$ multiplication [Tiskin; Algorithmica 2015]

The distance product of two $n \times n$ simple unit-Monge matrices can be computed in time $\mathcal{O}(n \log n)$.

Distance Product of Unit-Monge Matrices

Distance Product

The $(\min, +)$ product or distance product of an $m \times k$ matrix A and a $k \times n$ matrix B, denoted by $A \odot B$ is an $m \times n$ matrix C, such that $C[i,j] = \min_{1 \le r \le k} \{A[i,r] + B[r,j]\}$.



$A :=$ distance matrix (red → blue)

$B :=$ distance matrix (blue → green)

$C :=$ distance matrix (red → green)

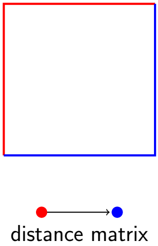
$C = A \odot B$

Efficient $(\min, +)$ multiplication [Tiskin; Algorithmica 2015]

The distance product of two $n \times n$ simple unit-Monge matrices can be computed in time $\mathcal{O}(n \log n)$.

Algorithm for substitutions

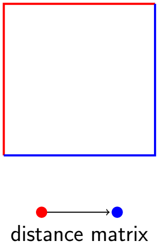
Goal: Maintain the distance matrix of the alignment graph.



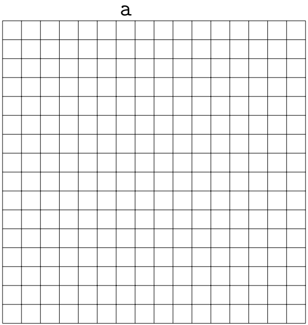
Algorithm for substitutions

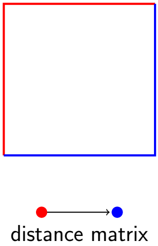
Goal: Maintain the distance matrix of the alignment graph.

We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.



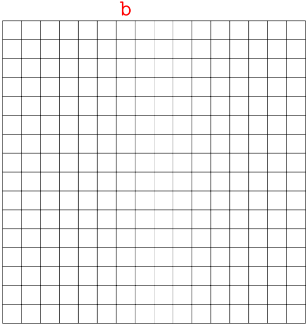
Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.

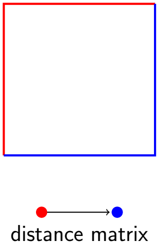




Algorithm for substitutions

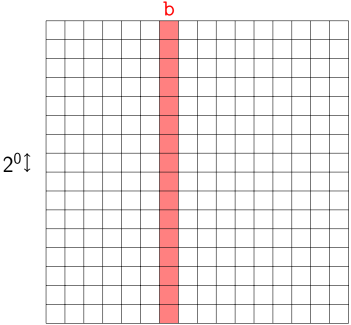
Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.

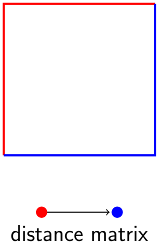




Algorithm for substitutions

Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.



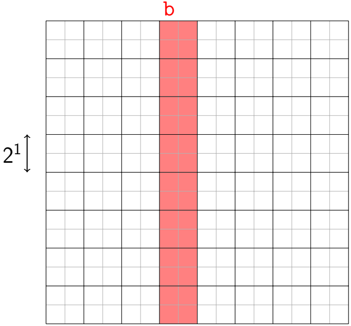


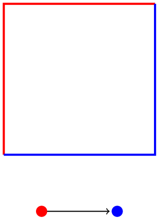
$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Algorithm for substitutions

Goal: Maintain the distance matrix of the alignment graph.

We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.

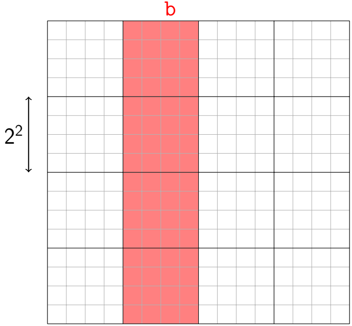


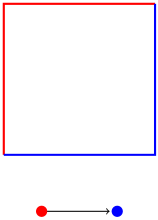


$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Algorithm for substitutions

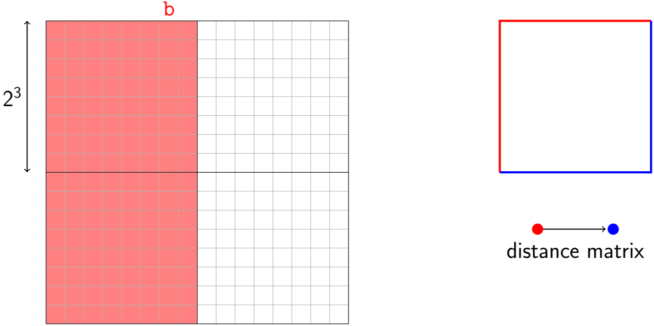
Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.





$\mathcal{O}(n/2^i)$ distance matrices change at level i.

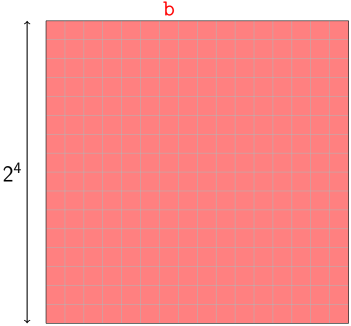
Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.

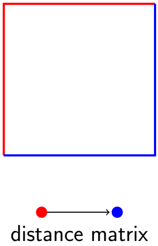


$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Algorithm for substitutions

Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.



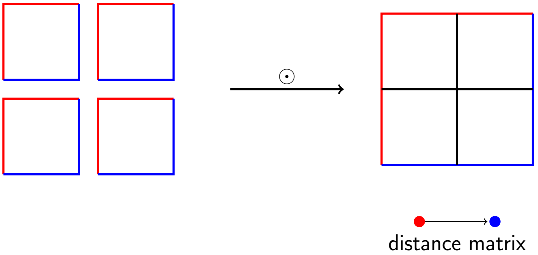


$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Algorithm for substitutions

Goal: Maintain the distance matrix of the alignment graph.

We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.

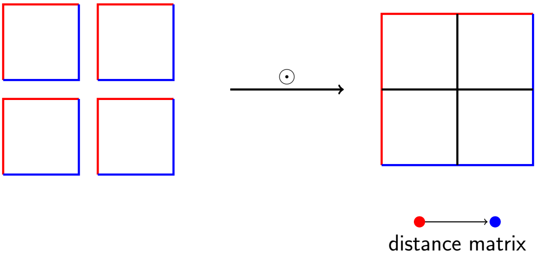


$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Each of them is recomputed from four distance matrices of the previous level in $\mathcal{O}(2^i \log(2^i))$ time using distance multiplication.

Algorithm for substitutions

Goal: Maintain the distance matrix of the alignment graph.
We maintain a hierarchy of decompositions of the alignment graph into $2^i \times 2^i$ blocks. For each block we maintain a distance matrix.



$\mathcal{O}(n/2^i)$ distance matrices change at level i.

Each of them is recomputed from four distance matrices of the previous level in $\mathcal{O}(2^i \log(2^i))$ time using distance multiplication.

The total update time is thus $\mathcal{O}(n \log^2 n)$.

Remarks

- Insertions and deletions can be handled by carefully resizing blocks.

Remarks

- Insertions and deletions can be handled by carefully resizing blocks.
- The actual LCS can be retrieved within the same time complexity by tracing back the computations.

Remarks

- Insertions and deletions can be handled by carefully resizing blocks.
- The actual LCS can be retrieved within the same time complexity by tracing back the computations.
- Fragment-to-fragment LCS queries can be answered in time $\mathcal{O}(n \log^2 n)$.

Remarks

- Insertions and deletions can be handled by carefully resizing blocks.
- The actual LCS can be retrieved within the same time complexity by tracing back the computations.
- Fragment-to-fragment LCS queries can be answered in time $\mathcal{O}(n \log^2 n)$.
- String alignment with integer weights $\leq w$ can be reduced to LCS by replacing each letter by a string of size $\mathcal{O}(w)$, as shown by Tiskin.

Remarks

- Insertions and deletions can be handled by carefully resizing blocks.
- The actual LCS can be retrieved within the same time complexity by tracing back the computations.
- Fragment-to-fragment LCS queries can be answered in time $\mathcal{O}(n \log^2 n)$.
- String alignment with integer weights $\leq w$ can be reduced to LCS by replacing each letter by a string of size $\mathcal{O}(w)$, as shown by Tiskin. Update time: $\tilde{\mathcal{O}}(nw)$.

Remarks

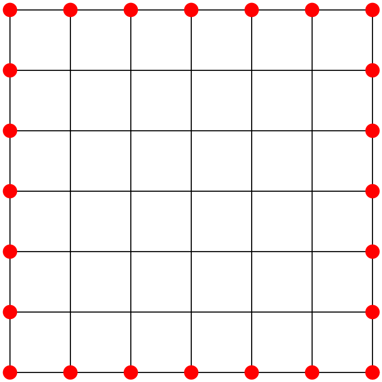
- Insertions and deletions can be handled by carefully resizing blocks.
- The actual LCS can be retrieved within the same time complexity by tracing back the computations.
- Fragment-to-fragment LCS queries can be answered in time $\mathcal{O}(n \log^2 n)$.
- String alignment with integer weights $\leq w$ can be reduced to LCS by replacing each letter by a string of size $\mathcal{O}(w)$, as shown by Tiskin. Update time: $\tilde{\mathcal{O}}(nw)$.

Next: An $\tilde{\mathcal{O}}(n\sqrt{n})$-time algorithm for integer weights of size $n^{\mathcal{O}(1)}$ using techniques for computing shortest paths in planar graphs.

MSSP

Multiple Source Shortest Paths (MSSP) [Klein; SODA 2005]

We can construct in nearly-linear time (in the size of the graph) a data structure that can report in logarithmic time the distance between any node on the infinite face and any node in the graph.



FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.

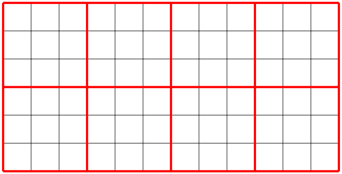
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



Before: distance product. Now: SSSP computations, many DDGs.

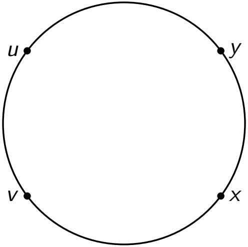
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



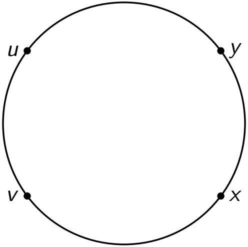
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



Monge Property:
$d(u,y) + d(v,x) \leq d(u,x) + d(v,y)$

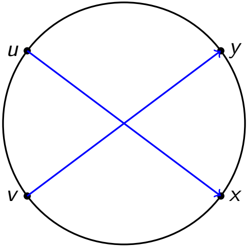
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



$$d(u,y) + d(v,x) \leq d(u,x) + d(v,y)$$

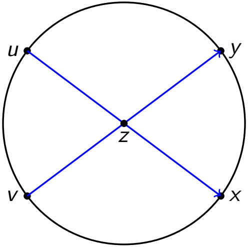
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



$$d(u, y) + d(v, x) \leq d(u, x) + d(v, y)$$

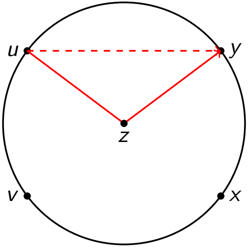
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



$$d(u,y) + d(v,x) \leq d(u,x) + d(v,y)$$

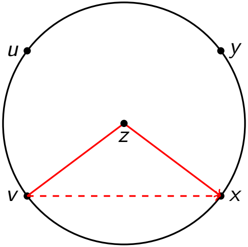
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.



$$d(u, y) + d(v, x) \leq d(u, x) + d(v, y)$$

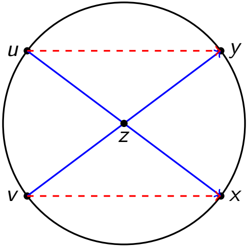
FR-Dijkstra

Dense Distance Graphs

The distance matrix capturing pairwise distances between the vertices of a set ∂H of vertices of a planar graph H, lying on a single face, can be computed in $\tilde{\mathcal{O}}(|H| + |\partial H|^2)$ time using MSSP.

FR-Dijkstra [Fakcharoenphol-Rao; JCSS 2006]

We can compute shortest paths from a single-source in a collection of DDGs with N vertices in total (with multiplicities) in $\tilde{\mathcal{O}}(N)$ time.

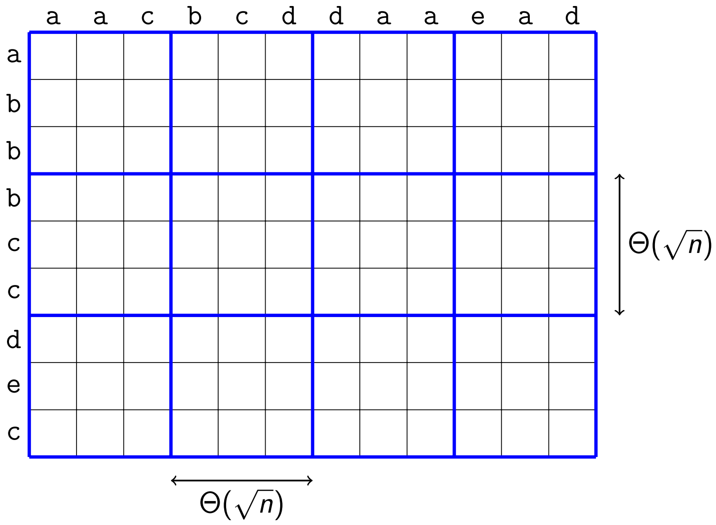


Monge Property:
$d(u,y) + d(v,x) \leq d(u,x) + d(v,y)$

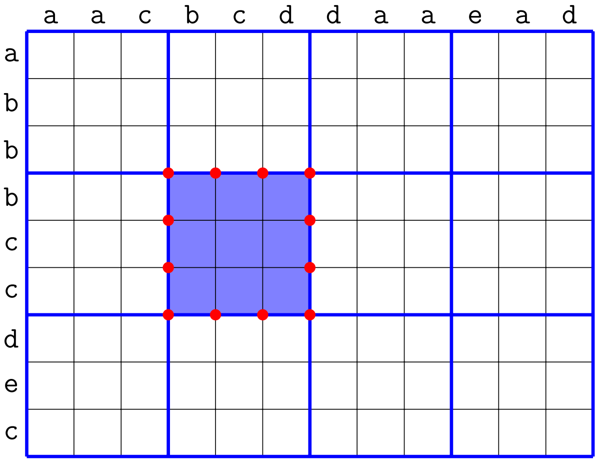
Algorithm for Large Weights

	a	a	c	b	c	d	d	a	a	e	a	d
a												
b												
b												
b												
c												
c												
d												
e												
c												

Algorithm for Large Weights

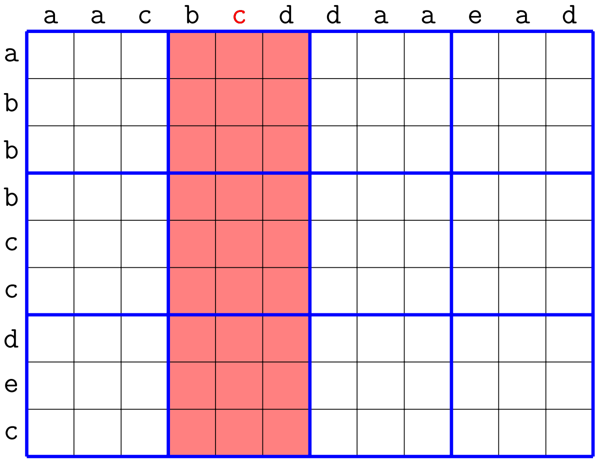


Algorithm for Large Weights



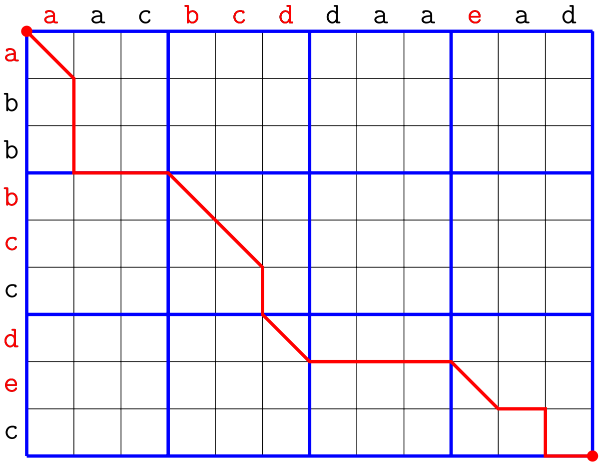
We maintain a DDG for each piece P with the set of "boundary" vertices as ∂P. $|P| = \Theta(n)$, $|\partial P| = \Theta(\sqrt{n})$.

Algorithm for Large Weights



Each update in one of the strings affects $\mathcal{O}(\sqrt{n})$ pieces. The DDG information for each piece is recomputed in $\tilde{\mathcal{O}}(n)$ time using MSSP.

Algorithm for Large Weights



We run FR-Dijkstra on the union of $\mathcal{O}(\sqrt{n} \cdot \sqrt{n}) = \mathcal{O}(n)$ DDGs.
The runtime is $\tilde{\mathcal{O}}(n\sqrt{n})$, since each DDG has $\mathcal{O}(\sqrt{n})$ vertices.

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?
- What if one string is given as a straight-line program (SLP)? [Tiskin; arxiv 2007]: The LCS of a standard string of length n and a string given by an SLP of size N can be computed in $\tilde{\mathcal{O}}(n \cdot N)$ time.

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?
- What if one string is given as a straight-line program (SLP)? [Tiskin; arxiv 2007]: The LCS of a standard string of length n and a string given by an SLP of size N can be computed in $\tilde{\mathcal{O}}(n \cdot N)$ time.

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?
- What if one string is given as a straight-line program (SLP)? [Tiskin; arxiv 2007]: The LCS of a standard string of length n and a string given by an SLP of size N can be computed in $\tilde{\mathcal{O}}(n \cdot N)$ time.
- How about maintaining an approximation of the edit distance/ LCS in the dynamic setting?

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?
- What if one string is given as a straight-line program (SLP)?
  [Tiskin; arxiv 2007]: The LCS of a standard string of length n and a string given by an SLP of size N can be computed in $\tilde{\mathcal{O}}(n \cdot N)$ time.
- How about maintaining an approximation of the edit distance/ LCS in the dynamic setting?
  [Andoni-Nosatzki; arxiv 2020]: The edit distance can be $\mathcal{O}(1)$-approximated in $\mathcal{O}(n^{1+\epsilon})$ time for any $\epsilon > 0$.

Final Remarks

Extension:

- We can in fact also handle copy-paste operations.

Open problems:

- Can we do better than $\tilde{\mathcal{O}}(n\sqrt{n})$ for large weights?
- What if one string is given as a straight-line program (SLP)? [Tiskin; arxiv 2007]: The LCS of a standard string of length n and a string given by an SLP of size N can be computed in $\tilde{\mathcal{O}}(n \cdot N)$ time.
- How about maintaining an approximation of the edit distance/ LCS in the dynamic setting? [Andoni-Nosatzki; arxiv 2020]: The edit distance can be $\mathcal{O}(1)$-approximated in $\mathcal{O}(n^{1+\epsilon})$ time for any $\epsilon > 0$. [Mitzenmacher-Seddighin; STOC 2020]: Dynamic LIS and distance to monotonicity.

Thank you for your attention!