1. Sequence comparisons

Slides for the Lecture on
Foundations of Sequence Analysis
Winter 2022/2023
Stefan Kurtz

October 20, 2022

An overview of sequence comparison models

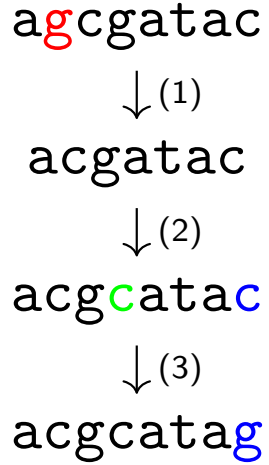
- The most important model for sequence comparison is the model of edit distance.
- It measures the distance between sequences in terms of edit operations, that is, deletions, insertions, and replacements of single characters.
- Two sequences are compared by determining a sequence of edit operations that converts one sequence into the other and minimizes the sum of the operations' costs.
- Here is an example:

The sequence $u = \texttt{agcgatac}$ can be converted to $v = \texttt{acgcatag}$ as follows:

(1) delete the first g in u gives acgatac

(2) insert c between g and a gives acgcatac

(3) replace the last c by g gives v

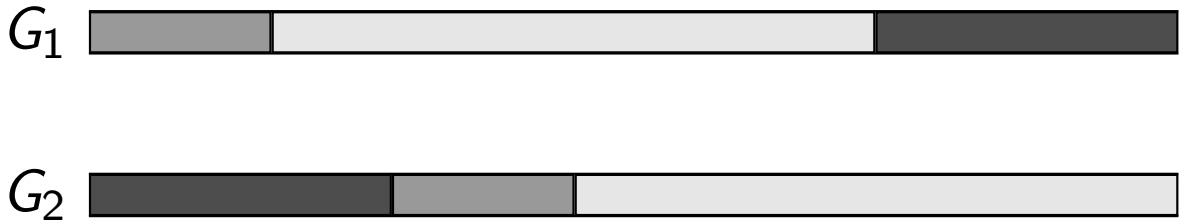


An overview of sequence comparison models

- Such a sequence of edit operations (1,2,4) (also called alignment) can be computed in $O(mn)$ time (where m and n are the lengths of the two sequences), using the technique of dynamic programming (discussed later in full detail).
- The edit distance is a measure of local similarities in which matches between substrings are highly dependent on their relative positions in the sequences.
- There are situations where this property is not desired.

An overview of sequence comparison models

- Suppose one wants to consider sequences as similar which differ only by an exchange of large substrings.
- This occurs, for instance, if a bacterial genome that has evolved from an ancestral genome by transposition of large sections of DNA, see the following illustration of two genomes G_1 and G_2, such that blocks of the same grey-scale are considered to be highly similar:



- In such a case, the edit distance model should not be used since it gives a large edit distance, thus not revealing the similarity.
- There are many alternatives, so called alignment-free sequence comparison models, that are more appropriate for this case.
- In this lecture we consider three of these models: the maximal matches model, the q-gram model and a model involving minHashes.

An overview of sequence comparison models

- When comparing biological sequences, the edit distance computation is often too expensive, while the order of the sequence characters is still important.
- Therefore heuristics have been developed, which approximate the edit distance model.
- These heuristics have been implemented in very popular programs like Fasta and Blast.
- In the following, we first consider the issue of sequence comparison in general.
- Then we describe the three models of sequence comparison in details and give algorithms to compute the respective distances.
- For the rest of this section let u and v be sequences of length m and n, respectively.
- These are the sequences we compare.

The sequence comparison problem

The trivial method to compare two sequences is to compare them character by character:

$$u \text{ and } v \text{ are equal} \Leftrightarrow |u| = |v| \text{ and } u[i] = v[i] \text{ for all } i,\ 1 \leq i \leq n.$$

Thus, the equality can be determined in $O(n)$ time. However, this comparison model is too restrictive as it does not consider the requirements when comparing biological sequences:

- tolerating errors in the sequencing technology
- variant calling: mapping reads which differ from the reference genome due to SNP or short indels
- homology search: searching for a protein with unknown function, a "similar" and not necessarily identical protein sequence, whose biological function is known.
- gene prediction: using the genes of a well-annotated genome to annotate a newly sequenced genome of a closely related species

The sequence comparison problem

- To generalize the trivial sequence comparison method, one defines a function $f : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}$, which delivers a qualitative measure of distance/similarity.
- Note that there is a duality in the notions *distance* and *similarity*: the smaller the distance, the larger the similarity and vice versa.
- Sequence comparison models based on distances minimize these.
- Sequence comparison models based on similarities maximize these.
- In the following, we define three distance models for sequences, the euclidean distance, the block distance and the hamming distance.
- The first two of these require integer alphabets.

The sequence comparison problem

Definition 1

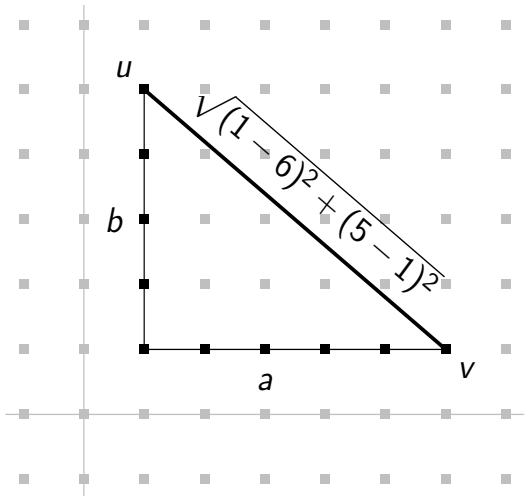
Let $\mathcal{A}$ be a finite subset of the integers. Suppose $n > 0$ and $M = \mathcal{A}^n$. Then we define the following distance notions:

$$\text{euclidean distance:} \quad f_e(u, v) = \sqrt{\sum_{i=1}^{n} (u[i] - v[i])^2}$$

$$\text{block distance:} \quad f_b(u, v) = \sum_{i=1}^{n} |u[i] - v[i]| \quad \square$$

- These distance notions only make sense for sequences of the same length and they require that the characters of the alphabet can be subtracted, which explains why we require that $\mathcal{A}$ consists of integers.
- The euclidean distance is the distance of the points (represented by fixed-length sequences of integers) in euclidean space of n dimensions.

Figure 1: Illustration of the euclidean distance for $n = 2$, $u = (1, 5)$ and $v = (6, 1)$. so $f_e(u, v) = \sqrt{(1-6)^2 + (5-1)^2} \approx 6.4$.



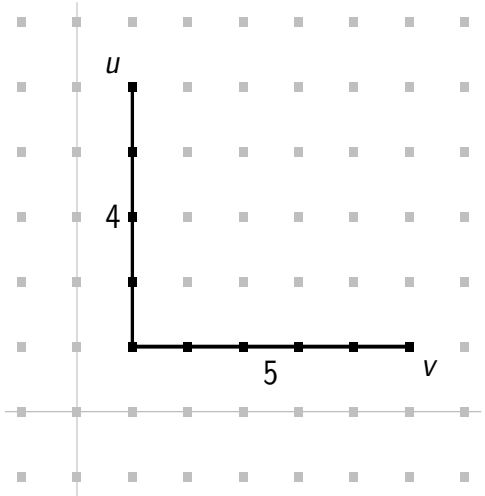
$$f_e(u, v) = \sqrt{\sum_{i=1}^{n} (u[i] - v[i])^2}$$

Example 1

- Let $n = 2$, $a = u[1] - v[1]$, $b = u[2] - v[2]$, and c be the distance of u and v in $\mathbb{Z}^2$.
- Then, by the Pythagorean theorem, we have $a^2 + b^2 = c^2$ and

$$c = \sqrt{a^2 + b^2} = \sqrt{(u[1] - v[1])^2 + (u[2] - v[2])^2} = f_e(u, v)$$

Figure 2: Illustration of the block distance for $n = 2$, $u = (1, 5)$ and $v = (6, 1)$. So $f_b(u, v) = |1 - 6| + |5 - 1| = 5 + 4 = 9$.



$$f_b(u, v) = \sum_{i=1}^{n} |u[i] - v[i]|$$

Example 2

- For $n = 2$, the block-distance assumes a topology of the space in which one can only go left, right, up or down.
- So the block-distance is the number of vertical movements (left or right) plus the number of horizontal movements (up or down) to reach v when starting at u.

The sequence comparison problem

The euclidean distance and the block distance are rarely used for biological sequences, while the following is more common.

Definition 2

Let $\mathcal{A}$ be an alphabet, let $n > 0$ and $M = \mathcal{A}^n$. Then we define the hamming distance f_h as follows:

$$f_h(u, v) = |\{i \mid 1 \leq i \leq n, u[i] \neq v[i]\}|$$

The hamming distance counts the number of positions at which the sequences are different. It is only defined for sequences of the same length. The distance notions we consider in the rest of this section are also defined for sequences of different length.

The edit distance model

The notion of edit operations, introduced in the early 1970's is the key to the edit distance model.

Definition 3

An *edit operation* is a pair $(\alpha, \beta) \in (\mathcal{A}^1 \cup \{\varepsilon\}) \times (\mathcal{A}^1 \cup \{\varepsilon\}) \setminus \{(\varepsilon, \varepsilon)\}$. □

- In other words, an edit operation is a pair (α, β) where $\alpha, \beta \in \mathcal{A}^1 \cup \{\varepsilon\}$ and $(\alpha, \beta) \neq (\varepsilon, \varepsilon)$.
- α and β denote *sequences* of length ≤ 1.
- However, each such string, whenever of length 1, is identified with the character it consists of.
- An edit operation (α, β) is usually written as $\alpha \rightarrow \beta$.
- This reflects the operational view which considers edit operations as rewrite rules transforming a source sequence into a target sequence, step by step

In particular, there are three kinds of edit operations:

- $a \to \varepsilon$ denotes the *deletion* of the character $a \in \mathcal{A}$,
- $\varepsilon \to b$ denotes the *insertion* of the character $b \in \mathcal{A}$,
- $a \to b$ denotes the *replacement* of the character $a \in \mathcal{A}$ by the character $b \in \mathcal{A}$.

Notice that $\varepsilon \to \varepsilon$ is not an edit operation. Insertions and deletions are sometimes referred to collectively as *indels*.

Example 3

As in a previous example, we consider the transformation of $u = \texttt{agcgatac}$ into $v = \texttt{acgcatag}$, this time using our notation for edit operations.

$$\begin{array}{c} \texttt{agcgatac} \\ \downarrow \texttt{g} \to \varepsilon \\ \texttt{acgatac} \\ \downarrow \varepsilon \to \texttt{c} \\ \texttt{acgcatac} \\ \downarrow \texttt{c} \to \texttt{g} \\ \texttt{acgcatag} \end{array}$$

The edit distance model

- Sometimes sequence comparison just means to measure how different or similar sequences are.
- Often it is additionally of interest to represent the differences between two sequences as a collection of individual elementary differences [Kruskal and Sankoff, 1983].
- The most important mode of such analyses is an alignment.

Definition 4

An *alignment* A of u and v is a sequence

$$(\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$$

of edit operations such that

$$u = \alpha_1 \ldots \alpha_h \text{ and}$$
$$v = \beta_1 \ldots \beta_h.$$

- Note that the unique alignment of ε and ε is the empty alignment, that is, the empty sequence of edit operations.
- An alignment is usually written by placing the characters of the two aligned sequences on different lines, with inserted dashes denoting ε.
- In such a representation, every column represents an edit operation.

Example 4

The alignment $A = (\varepsilon \to \mathtt{t}, \mathtt{g} \to \mathtt{g}, \mathtt{c} \to \mathtt{a}, \varepsilon \to \mathtt{t}, \mathtt{a} \to \mathtt{a}, \mathtt{c} \to \varepsilon, \mathtt{t} \to \mathtt{t})$ of the sequences `gcact` and `tgatat` is written as follows:

$$\begin{pmatrix} \mathtt{-} & \mathtt{g} & \mathtt{c} & \mathtt{-} & \mathtt{a} & \mathtt{c} & \mathtt{t} \\ \mathtt{t} & \mathtt{g} & \mathtt{a} & \mathtt{t} & \mathtt{a} & \mathtt{-} & \mathtt{t} \end{pmatrix}$$

Example 5

Five alignments of $u = \texttt{GATH}$ and $v = \texttt{GCDHT}$:

$$A_1 = \begin{pmatrix} \texttt{G}-\texttt{A}\,\texttt{T}-\texttt{H}- \\ \texttt{G}\,\texttt{C}--\texttt{D}\,\texttt{H}\,\texttt{T} \end{pmatrix} \quad A_2 = \begin{pmatrix} \texttt{G}-\texttt{A}-\texttt{T}\,\texttt{H}- \\ \texttt{G}\,\texttt{C}-\texttt{D}-\texttt{H}\,\texttt{T} \end{pmatrix} \quad A_3 = \begin{pmatrix} \texttt{G}--\texttt{A}\,\texttt{T}\,\texttt{H}- \\ \texttt{G}\,\texttt{C}\,\texttt{D}--\texttt{H}\,\texttt{T} \end{pmatrix}$$

$$A_4 = \begin{pmatrix} \texttt{G}\,\texttt{A}--\texttt{T}\,\texttt{H} \\ \texttt{G}\,\texttt{C}\,\texttt{D}\,\texttt{H}\,\texttt{T}- \end{pmatrix} \quad A_5 = \begin{pmatrix} \texttt{G}\,\texttt{A}\,\texttt{T}\,\texttt{H}- \\ \texttt{G}\,\texttt{C}\,\texttt{D}\,\texttt{H}\,\texttt{T} \end{pmatrix}$$

The following table shows different key values of these alignments, where id is the number of indels and r is the number of replacements in the respective alignment.

	A_1	A_2	A_3	A_4	A_5
length	7	7	7	6	5
id	5	5	5	3	1
r	2	2	2	3	4
$id + 2 \cdot r$	9	9	9	9	9

Lemma 1

Let A be an alignment of u and v. Let id be the number of insertions and deletions, and r be the number of replacements in A. Then $|u| + |v| = id + 2 \cdot r$.

Proof.

- *Each of the $|u| + |v|$ characters in u and v must occur in A.*
- *Moreover, all characters in A are from the sequences u and v.*
- *Hence the number of characters in A and in u and v must be identical.*
- *In each indel one character occurs.*
- *In each replacement two of these characters occur.*
- *Hence the number of characters in A is $id + 2 \cdot r$.*
- *By the argumentation above, this number must be equal to $|u| + |v|$.*
- *So the equation holds.*

Here is another lemma characterizing the relation between alignment length and sequence lengths.

Lemma 2

Let u and v be sequences of length m and n, respectively. Let $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ be an alignment of u and v. Then $m + n \geq h \geq \max\{m, n\}$.

Proof

The alignment

$$\begin{pmatrix} u[1] & u[2] & \ldots & u[m] & - & - & \ldots & - \\ - & - & \ldots & - & v[1] & v[2] & \ldots & v[n] \end{pmatrix}$$

of u and v has maximal length among all alignments of u and v. Its length is $m + n$. Hence the length of A cannot be larger than $m + n$ and so $m + n \geq h$.

Proof

1. Let $m \geq n$. Then

$$\begin{pmatrix} u[1]\ u[2] \ldots u[n]\ u[n+1] \ldots u[m] \\ v[1]\ v[2] \ldots v[n]\ \ - \ \ \ldots \ \ - \end{pmatrix}$$

   is an alignment of u and v of minimal length. That is, no alignment of u and v can be shorter than m. Hence $h \geq m = \max\{m, n\}$.

2. Let $m < n$. Then

$$\begin{pmatrix} u[1]\ u[2] \ldots u[m]\ \ - \ \ \ldots \ \ - \\ v[1]\ v[2] \ldots v[m]\ v[m+1] \ldots v[n] \end{pmatrix}$$

   is an alignment of u and v of minimal length. That is, no alignment of u and v can be shorter than n. Hence $h \geq n = \max\{m, n\}$.

For all $m, n \geq 0$ let $\mathit{Aligns}(m, n)$ be the number of alignments of two fixed sequences, say u and v, of length m and n.

Example 6

Here are all alignments of ab and ca (omitting the boundary brackets):

```
--ab   -a-b   a--b   a-b   -ab   -ab-   a-b-   ab-   ab--   ab-   -ab   a-b   ab
ca--   c-a-   -ca-   ca-   ca-   c--a   -c-a   c-a   --ca   -ca   c-a   -ca   ca
```

Hence $\mathit{Aligns}(2, 2) = 13$. Note that the number of alignments for two other sequences of the same lengths is also 13. This property holds in general, that is, the number of alignments only depends on the length of the sequences to be aligned, not on their content.

We will derive a recursive equation for $Aligns(m, n)$ by case distinction:
(1) Let $m = 0$ or $n = 0$. According to Lemma 2, an alignment of two sequences of length m and n has length h, s.t. $m + n \geq h \geq \max\{m, n\}$.

- If $m = n = 0$ we conclude $0 \geq h \geq 0$, i.e. $h = 0$. That is, an alignment of two empty sequences has length 0, and there is exactly one such alignment, namely the empty alignment.
- If $m = 0$ and $n > 0$, then $m + n = n \geq h \geq n = \max\{m, n\}$, which implies $h = n$. That is, all alignments of ε and v are of length n. They must consist of exactly n insertions. Of course, there is exactly one such alignment of ε and v.

Example 7

Let $v = \mathtt{cd}$. The only alignment of ε and v is $\begin{pmatrix} - & - \\ \mathtt{c} & \mathtt{d} \end{pmatrix}$

– If $m > 0$ and $n = 0$, then $m + n = m \geq h \geq m = \max\{m, n\}$, which implies $h = m$. That is, all alignments of u and ε are of length m. They must consist of exactly m deletions. Of course, there is exactly one such alignment.

Example 8

Let $u = \mathtt{ab}$. The only alignment of u and ε is $\begin{pmatrix} \mathtt{a}\,\mathtt{b} \\ \mathtt{-}\,\mathtt{-} \end{pmatrix}$

Thus in case (1) ($m = 0$ or $n = 0$) we obtain $\mathit{Aligns}(m, n) = 1$.

The number of alignments (1/2)

(2) Now let $m > 0$ and $n > 0$, i.e. u and v are both not empty. An alignment of u and v contains at least one edit operation. We make a case distinction about the kind of the last edit operation:

- Consider all alignments of u and v ending with a deletion.
- They all delete the last character of u, i.e. they end with the same deletion.
- Thus the number of alignments of u and v ending with a deletion is the same as the number of all alignments of $u[1 \ldots m-1]$ and v.
- This is $\mathit{Aligns}(m-1, n)$.
- Thus there are $\mathit{Aligns}(m-1, n)$ alignments of u and v ending with a deletion.

The number of alignments (2/2)

- Consider all alignments of u and v ending with an insertion.
- They all insert the last character of v, i.e. they end with the same insertion.
- Thus the number of alignments of u and v ending with an insertion is the same as the number of all alignments of u and $v[1 \ldots n-1]$.
- By definition, this is $\mathit{Aligns}(m, n-1)$.
- Thus there are $\mathit{Aligns}(m, n-1)$ alignments of u and v ending with an insertion.

The number of alignments (1/2)

- Consider all alignments of u and v ending with a replacement.
- They all replace the last character of u with the last character of v, i.e. they end with the same replacement.
- Thus the number of alignments of u and v ending with a replacement is the same as the number of all alignments of $u[1 \ldots m-1]$ and $v[1 \ldots n-1]$.
- By definition, this is $\mathit{Aligns}(m-1, n-1)$.
- Thus there are $\mathit{Aligns}(m-1, n-1)$ alignments ending with a replacement.

The number of alignments (2/2)

So we have derived the following numbers of alignments in the 3 subcases:

- deletion case: $Aligns(m-1, n)$ alignments
- insertion case: $Aligns(m, n-1)$ alignments
- replacement case: $Aligns(m-1, n-1)$ alignments

As we exactly have three cases to obtain the total number we have to add them. So we conclude that the number $Aligns(m, n)$ of alignments of two sequences of length n and n , respectively, is

$$Aligns(m-1, n) + Aligns(m, n-1) + Aligns(m-1, n-1).$$

Altogether, we obtain the following recurrence:

$$Aligns(m, n) = \begin{cases} 1 & \text{if } m = 0 \text{ or } n = 0 \\ Aligns(m-1, n) + & \\ Aligns(m-1, n-1) + & \\ Aligns(m, n-1) & \text{otherwise} \end{cases}$$

Example 9

$$
\begin{aligned}
\mathit{Aligns}(2,2) &= \mathit{Aligns}(1,2) + \mathit{Aligns}(1,1) + \mathit{Aligns}(2,1) \\
&= (\mathit{Aligns}(0,2) + \mathit{Aligns}(0,1) + \mathit{Aligns}(1,1)) + \\
&\quad (\mathit{Aligns}(0,1) + \mathit{Aligns}(0,0) + \mathit{Aligns}(1,0)) + \\
&\quad (\mathit{Aligns}(1,1) + \mathit{Aligns}(1,0) + \mathit{Aligns}(2,0)) \\
&= (1 + 1 + \mathit{Aligns}(1,1)) + \\
&\quad (1 + 1 + 1) + \\
&\quad (\mathit{Aligns}(1,1) + 1 + 1) \\
&= 7 + 2 \cdot \mathit{Aligns}(1,1) \\
&= 7 + 2 \cdot (\mathit{Aligns}(0,1) + \mathit{Aligns}(0,0) + \mathit{Aligns}(1,0)) \\
&= 7 + 2 \cdot (1 + 1 + 1) \\
&= 7 + 2 \cdot 3 \\
&= 13
\end{aligned}
$$

$Aligns(n, n)$ can be approximated by the Stanton-Cowan-Numbers:

$$Aligns(n, n) \approx \left(1 + \sqrt{2}\right)^{2n+1} \cdot \sqrt{n}$$

As $\sqrt{n} \geq 1$ and the expression $1 + \sqrt{2} \approx 2.4142$ is strictly greater than 1, one can state that the number of alignments grows exponentially with the length of the aligned sequences.

Example 10

For $n = 1\,000$ we have $Aligns(n, n) \approx \left(1 + \sqrt{2}\right)^{2\,001} \cdot \sqrt{1\,000} = 10^{767.4}$.

- If one considers sets of alignments of the same sequence, one quickly recognizes that the order of insertions and deletions immediately following each other in an alignment is not important.
- So all alignments which differ only by the order of consecutive indels should be considered equivalent.

Example 11

The two alignments

$$\begin{pmatrix} \mathtt{a} & \mathtt{-} \\ \mathtt{-} & \mathtt{c} \end{pmatrix} \text{ and } \begin{pmatrix} \mathtt{-} & \mathtt{a} \\ \mathtt{c} & \mathtt{-} \end{pmatrix} \tag{1}$$

should be considered equivalent.
The three alignments

$$A_1 = \begin{pmatrix} \mathtt{G-AT-H-} \\ \mathtt{GC--DHT} \end{pmatrix} A_2 = \begin{pmatrix} \mathtt{G-A-TH-} \\ \mathtt{GC-D-HT} \end{pmatrix} A_3 = \begin{pmatrix} \mathtt{G--ATH-} \\ \mathtt{GCD--HT} \end{pmatrix}$$

should be considered equivalent.

The number of subsequences

Definition 5

A *subsequence* of u and v is a sequence of index pairs

$$(i_1, j_1), \ldots, (i_r, j_r)$$

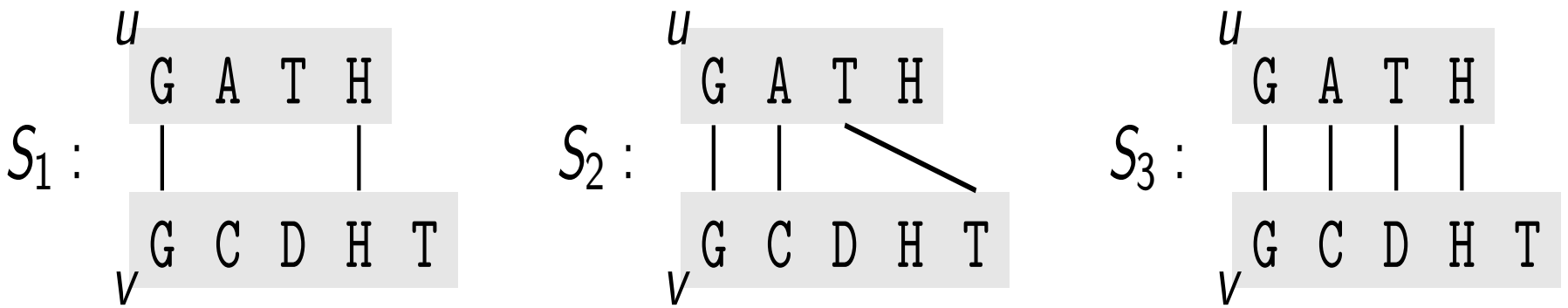
such that

$$1 \leq i_1 < \ldots < i_r \leq m \text{ and}$$
$$1 \leq j_1 < \ldots < j_r \leq n. \quad \square$$

- The index pair (i_h, j_h) stands for the replacement $u[i_h] \rightarrow v[j_h]$.
- All characters in u and v at positions not occurring in a given subsequence of u and v are considered to be deleted in u or v (without specifying the order in which the indels appear).
- So the empty subsequence stands for $\begin{pmatrix} a & - \\ - & c \end{pmatrix}$ and $\begin{pmatrix} - & a \\ c & - \end{pmatrix}$.
- In a graphical representation, the index pairs of the subsequence appear as lines connecting the characters in the subsequence.

Example 12

S_1=(1,1),(4,4), S_2=(1,1),(2,2),(3,5), S_3=(1,1),(2,2),(3,3),(4,4) are subsequences of $u = \texttt{GATH}$ and $v = \texttt{GCDHT}$, depicted as follows



They represent the alignments

$$A_1 = \begin{pmatrix} \texttt{G-AT-H-} \\ \texttt{GC--DHT} \end{pmatrix} \quad A_2 = \begin{pmatrix} \texttt{G-A-TH-} \\ \texttt{GC-D-HT} \end{pmatrix} \quad A_3 = \begin{pmatrix} \texttt{G--ATH-} \\ \texttt{GCD--HT} \end{pmatrix}$$

$$A_4 = \begin{pmatrix} \texttt{GA--TH} \\ \texttt{GCDHT-} \end{pmatrix} \quad A_5 = \begin{pmatrix} \texttt{GATH-} \\ \texttt{GCDHT} \end{pmatrix}$$

S_1 represents A_1, A_2, and A_3. S_2 represents A_4, and S_3 represents A_5.

Lemma 3

Let $Subseqs(m, n)$ be the number of subsequences of two fixed sequences of length m and n. Then

$$Subseqs(m, n) = \sum_{r=0}^{\min(m,n)} \binom{m}{r} \cdot \binom{n}{r} \tag{2}$$

Proof.

- *Let r, $0 \leq r \leq \min\{m, n\}$ be arbitrary but fixed.*
- *For the ordered selection of the indices $i_1, \ldots, i_r$ at m positions there are $\binom{m}{r}$ possibilities; for the ordered selection of the indices $j_1, \ldots, j_r$ at n positions there are $\binom{n}{r}$ possibilities.*
- *All these possibilities have to be combined which gives $\binom{m}{r} \cdot \binom{n}{r}$ combinations.*
- *Summing over all possible r in the range from 0 to $\min\{m, n\}$ gives Equation (2).*

Table 1: Number of alignments and number of subsequences for two sequences of length n for n in the range from 0 to 10.

n	$Aligns(n, n)$	$Subseqs(n, n)$	$\frac{Aligns(n,n)}{Subseqs(n,n)}$
0	1	1	1.00
1	3	2	1.50
2	13	6	2.17
3	63	20	3.15
4	321	70	4.59
5	1 683	252	6.68
6	8 989	924	9.73
7	48 639	3 432	14.17
8	265 729	12 870	20.65
9	1 462 563	48 620	30.08
10	8 097 453	184 756	43.83

The number of subsequences

– $Subseqs(n, n)$ can be approximated by $2^{2n}\left(4 \cdot \sqrt{n\pi}\right)^{-1}$, e.g.

$$Subseqs(1\,000, 1\,000) \approx 10^{600}$$

So we conclude that the number of alignments is much larger than the number of subsequences (by a factor of about 10^{167}).

– As both numbers grow exponentially with the sequence length, this difference does not really matter in practice.

The edit distance problem

Definition 6

- A *cost function* δ assigns non-negative costs to all edit operations such that $\delta(a \to b) \geq 0$ for all replacements $a \to b$ and $\delta(\alpha \to \beta) > 0$ for all indels $\alpha \to \beta$.
- If $\delta(\alpha \to \beta) = \delta(\beta \to \alpha)$ for all edit operations $\alpha \to \beta$ and $\beta \to \alpha$, then δ is *symmetric*.
- If $\delta(\alpha \to \beta) = 0$ if and only if $\alpha = \beta$ for all edit operations $\alpha \to \beta$, then δ satisfies the *zero-property*.
- If $\delta(\alpha \to \beta) \leq \delta(\alpha \to \gamma) + \delta(\gamma \to \beta)$ for all edit operations $\alpha \to \beta$, $\alpha \to \gamma$, and $\gamma \to \beta$, then δ satisfies the *triangle inequality*.

δ is extended to alignments in a straightforward way: The cost $\delta(A)$ of an alignment $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ is

$$\delta(A) = \sum_{i=1}^{h} \delta(\alpha_i \to \beta_i). \ \square$$

Example 13

Let

$$\delta(\alpha \to \beta) = \begin{cases} 0 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha = \beta \\ 1 & \text{otherwise} \end{cases}$$

Then δ is the *unit cost*.

Example 14

Let

$$\delta(\alpha \to \beta) = \begin{cases} 0 & \text{if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha = \beta \\ 1 & \text{else if } \alpha, \beta \in \mathcal{A} \text{ and } \alpha \neq \beta \\ \infty & \text{otherwise} \end{cases}$$

Then δ is the *hamming cost*. If A is the only alignment of u and v without indels, then $\delta(A) = f_h(u, v)$. For an alignment A with indels, $\delta(A) = \infty$.

Example 15

Suppose δ is given by the following table:

δ	ε	A	C	G	T
ε		3	3	3	3
A	3	0	2	1	2
C	3	2	0	2	1
G	3	1	2	0	2
T	3	2	1	2	0

representing

$$\delta(a \to \varepsilon) = 3 \text{ for } a \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$$

$$\delta(\varepsilon \to b) = 3 \text{ for } b \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$$

$$\delta(a \to b) = 1 \text{ for } (a, b) \in \{(\mathtt{A}, \mathtt{G}), (\mathtt{G}, \mathtt{A}), (\mathtt{T}, \mathtt{C}), (\mathtt{C}, \mathtt{T})\}$$

$$\delta(a \to b) = 2 \text{ for } (a, b) \in \{(\mathtt{A}, \mathtt{T}), (\mathtt{T}, \mathtt{A}), (\mathtt{C}, \mathtt{G}), (\mathtt{G}, \mathtt{C})\}$$

- Then δ is the transversion/transition cost function.
- Bases A and G are called *purine*.
- Bases C and T are called *pyrimidine*.
- The transversion/transition cost function reflects the biological fact that a purine/purine and a pyrimidine/pyrimidine replacement is more likely to occur than a purine/pyrimidine replacement.
- Moreover, it takes into account that indels occur less often than replacements and thus assigns cost 3 for an indel.

Example 16

The following tables shows costs for all replacements of amino acids, as suggested by Willy Taylor. As it is symmetric only the lower triangle of the matrix is shown. Assuming positive indel costs, the cost function satisfies the zero-property.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	0																			
R	14	0																		
N	7	11	0																	
D	9	15	6	0																
C	20	26	23	25	0															
Q	9	11	5	6	26	0														
E	8	14	6	3	25	5	0													
G	7	17	10	9	21	12	9	0												
H	12	11	7	11	25	7	10	15	0											
I	13	18	16	19	22	17	17	17	17	0										
L	17	19	19	22	26	20	21	21	19	9	0									
K	11	7	7	10	25	8	10	13	10	17	19	0								
M	14	16	16	19	26	16	17	18	17	8	7	15	0							
F	24	25	25	29	26	27	28	27	24	17	14	26	18	0						
P	7	13	10	11	22	10	11	10	13	16	20	11	17	27	0					
S	6	12	7	11	20	11	10	9	11	14	19	10	16	24	9	0				
T	4	13	7	10	21	10	9	9	11	12	17	10	13	23	8	5	0			
W	32	25	30	34	34	32	34	33	30	31	27	29	29	24	32	29	31	0		
Y	23	24	23	27	22	26	26	26	21	18	18	25	20	8	26	22	22	25	0	
V	11	18	15	17	21	16	16	15	16	4	10	16	8	19	14	13	10	32	20	0

Definition 7

The *edit distance* of u and v, denoted by $edist_\delta(u, v)$, is the minimum possible cost of an alignment of u and v. That is,

$$edist_\delta(u, v) = \min\{\delta(A) \mid A \text{ is an alignment of } u \text{ and } v\}.$$

An alignment A of u and v is *optimal* if $\delta(A) = edist_\delta(u, v)$. If δ is the unit cost function, then $edist_\delta(u, v)$ is the *unit edit distance* between u and v. $\square$

Example 17

Let $u = ab$, $v = ca$ and δ be the unit cost function. Here are the alignments of u and v with the unit cost of each alignment shown below it:

--ab	-a-b	a--b	a-b	-ab	-ab-	a-b-	ab-	ab--	ab-	-ab	a-b	ab
ca--	c-a-	-ca-	ca-	ca-	c--a	-c-a	c-a	--ca	-ca	c-a	-ca	ca
4	4	4	3	2*	4	4	3	4	3	3	3	2*

As the smallest value is 2, we have $edist_\delta(u, v) = 2$. There are two alignments with cost 2, the optimal alignments (marked with $*$).

The edit distance problem

One important property of the edit distance is that it is a metric on $\mathcal{A}^*$.

Definition 8

Let M be a set and $f : M \times M \to \mathbb{R}^+$ be a function. f is a *metric* on M if for all $x, y, z \in M$ the following properties hold:

$$f(x,y) = 0 \iff x = y \qquad \text{zero property}$$
$$f(x,y) = f(y,x) \qquad \text{symmetry}$$
$$f(x,y) \leq f(x,z) + f(z,y) \qquad \text{triangle inequality}$$

If the symmetry and triangle inequality holds for f, and also $x = y \Rightarrow f(x,y) = 0$, then f is a *pseudo-metric* on M.

- If δ has the zero property, then so does $edist_\delta$.
- If δ is symmetric and satisfies the zero property as well as the triangle inequality, then $edist_\delta$ is a metric on $\mathcal{A}^*$, as shown in [Wagner and Fischer, 1974].

Lemma 4

For any cost function δ and any two sequences $u, v \in \mathcal{A}^*$ we have:

$$edist_\delta(u, v) = edist_\delta(u^{-1}, v^{-1})$$

Proof.

- *Let $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ be an optimal alignment of u and v.*
- *Then $A^{-1} = (\alpha_h \to \beta_h, \ldots, \alpha_1 \to \beta_1)$ is an alignment of u^{-1} and v^{-1}.*
- *Suppose there is an alignment X of u^{-1} and v^{-1} s.t. $\delta(X) < \delta(A^{-1})$.*
- *That is, A^{-1} is not an optimal alignment of u^{-1} and v^{-1}.*
- *Now X^{-1} is an alignment of u and v and we have $\delta(X^{-1}) = \delta(X) < \delta(A^{-1}) = \delta(A)$.*
- *Thus A is not an optimal alignment, which is a contradiction.*
- *Hence our assumption above was wrong, i.e. there is no alignment X of u^{-1} and v^{-1} with $\delta(X) < \delta(A^{-1})$, so A^{-1} is optimal.*
- *Hence $edist_\delta(u, v) = \delta(A) = \delta(A^{-1}) = edist_\delta(u^{-1}, v^{-1})$.*

The edit distance problem

Definition 9

The *edit distance problem* is to compute the edit distance and all optimal alignments. □

By specifying a concrete cost function, we obtain a special form of the edit distance:

Definition 10

If δ is the unit cost, then $edist_\delta$ is the *unit edit distance* or *Levenshtein distance*.

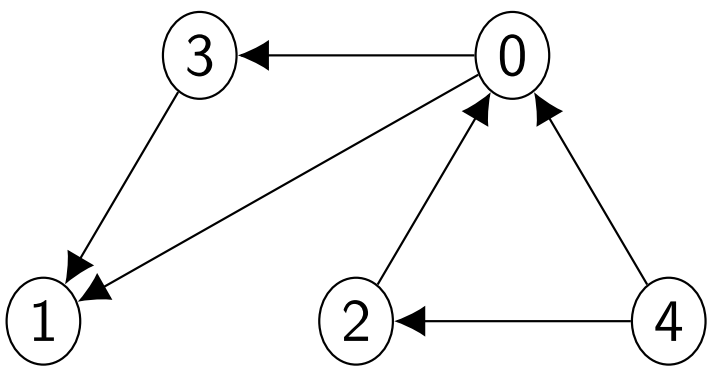
We will cast the edit distance problem as a graph theoretic problem and so we introduce some basic notions on graphs. A graph is a means of representing pairwise relations of any kind of items.

Definition 11

A graph consists of a set V of nodes (sometimes also called vertices) and a set $E \subseteq V \times V$ of edges. If for all $x, y \in V$, $(x, y) \in E$ implies $(y, x) \in E$, the graph is undirected.

Example 18

The graph (V, E) with $V = \{0, 1, 2, 3, 4\}$ and $E = \{(2, 0), (0, 1), (0, 3), (3, 1), (4, 0), (4, 2)\}$ is usually drawn as follows

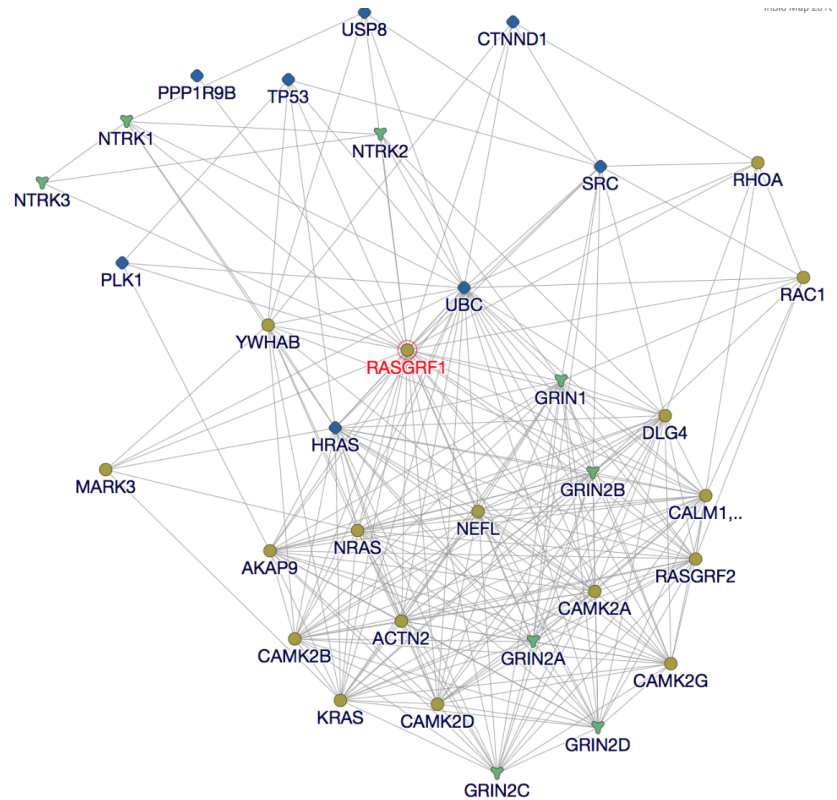


node labeled graph: nodes have labels (shown inside or besides node);
edge labeled graph: edges have labels (shown above or below edges);
graph is directed (the order of pairs in E matters); direction is expressed by arrows, i.e. $(x, y) \in E$ is written as $x \to y$

the placement of the nodes is arbitrary, i.e. it does not mean anything

An example of a graph representing biological data is given in Figure 3.

Figure 3: An undirected graph created by the web server InBioMap (`https://www.intomics.com`) when searching the term RASGRF1. A protein is represented by a node. Different node symbols represent different subcellular locations. A protein-interaction is represented by an edge. An edge label (i.e. a confidence score for the corresponding interaction) becomes visible with a mouse-over event.



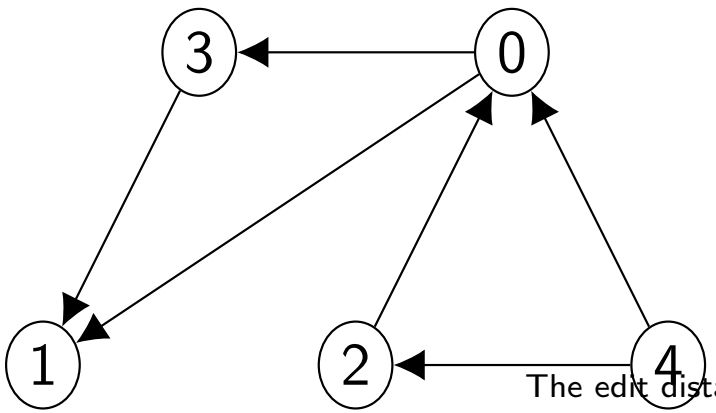
Definition 12

A *path* in a graph (V, E) is a sequence of nodes $v_0, v_1, \ldots, v_k$ for $k \geq 0$ such that for all i, $0 \leq i \leq k-1$ we have $(v_i, v_{i+1}) \in E$. An empty path satisfies $k = 0$, i.e. it has no edges and consists of a single node. A path starts with v_0 and ends with v_k and its length is k. So we state that it is a path from v_0 to v_k. Such a path is often written as

$$v_0 \to v_1 \to \cdots \to v_{k-1} \text{ (in directed graph)}$$
$$v_0 - v_1 - \cdots - v_{k-1} \text{ (in undirected graph)}$$

Example 19

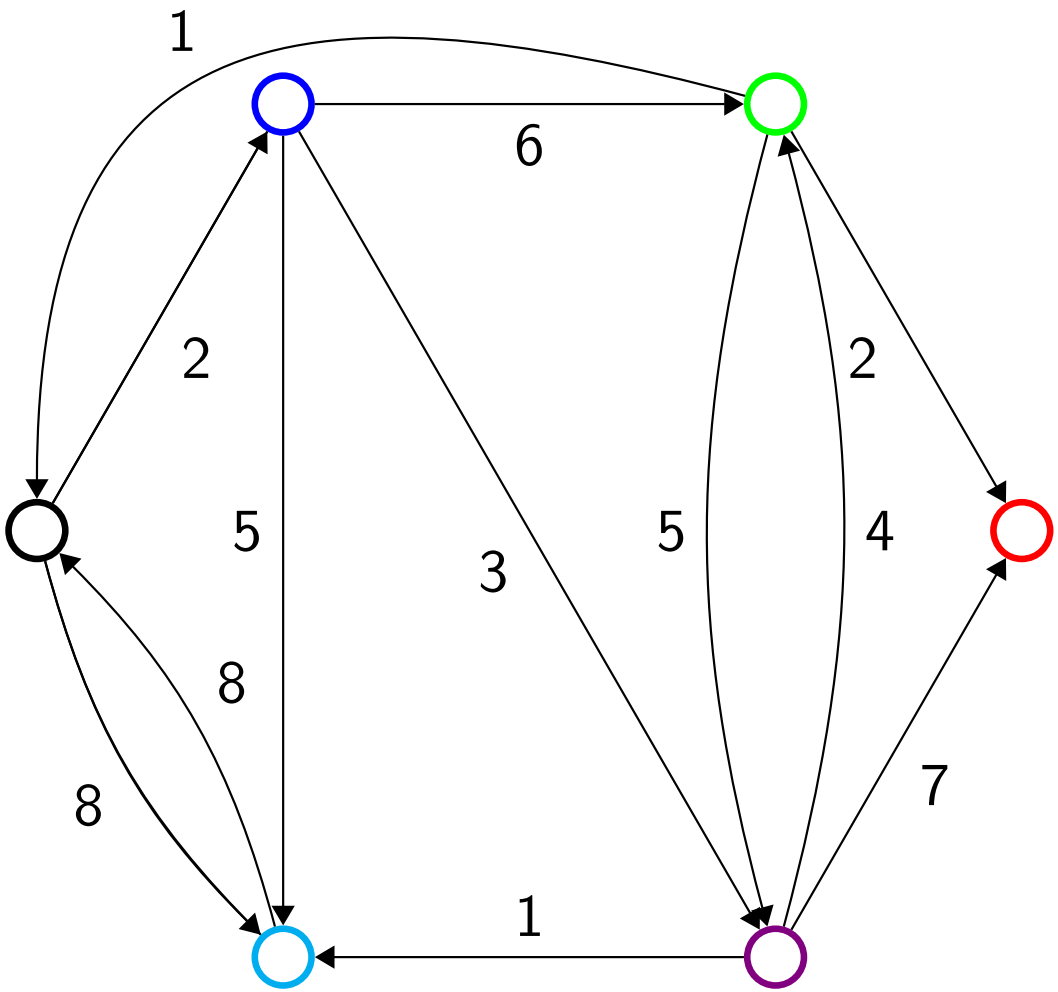
Reconsider the directed graph (V, E) from Example 18.



$4 \to 2 \to 0 \to 1$
is a path of length 3 from 4 to 1
$4 \to 0 \to 3$
is a path of length 2 from 4 to 3
$3 \to 1$ is a path of length 1 from 3 to 1
$0 \to 1 \to 3$ is not a path, as $(1, 3) \notin E$

Example 20

Here is a directed graph with unlabeled nodes. To better distinguish the nodes, they are shown in different colors. The edges are labeled and represent a weight function $w : E \to \mathbb{R}$.



such a graph can model:

nodes	edge labels
airports	flight times
social network users	1 = has communicated
currencies	exchange rates
strings	costs of edit operations
genes	interaction level

Basic definitions related to graphs

– for an edge labeled graph it makes sense to define the weight of a path by adding up the weights of the edges it consists of

Definition 13

Consider a graph (V, E) with weight function $w : E \to \mathbb{R}$. For any path $p = v_0 \to v_1 \to \cdots \to v_k$, we define the weight $w(p)$ of p by

$$w(p) = \sum_{i=0}^{k-1} w(v_i \to v_{i+1})$$

– a path of length 0 has weight 0

– when there is more than one path between two nodes, one is usually interested in the path of minimum weight (thus interpreting the weight as a distance)

– this leads to an important optimization problem:

Basic definitions related to graphs

Definition 14

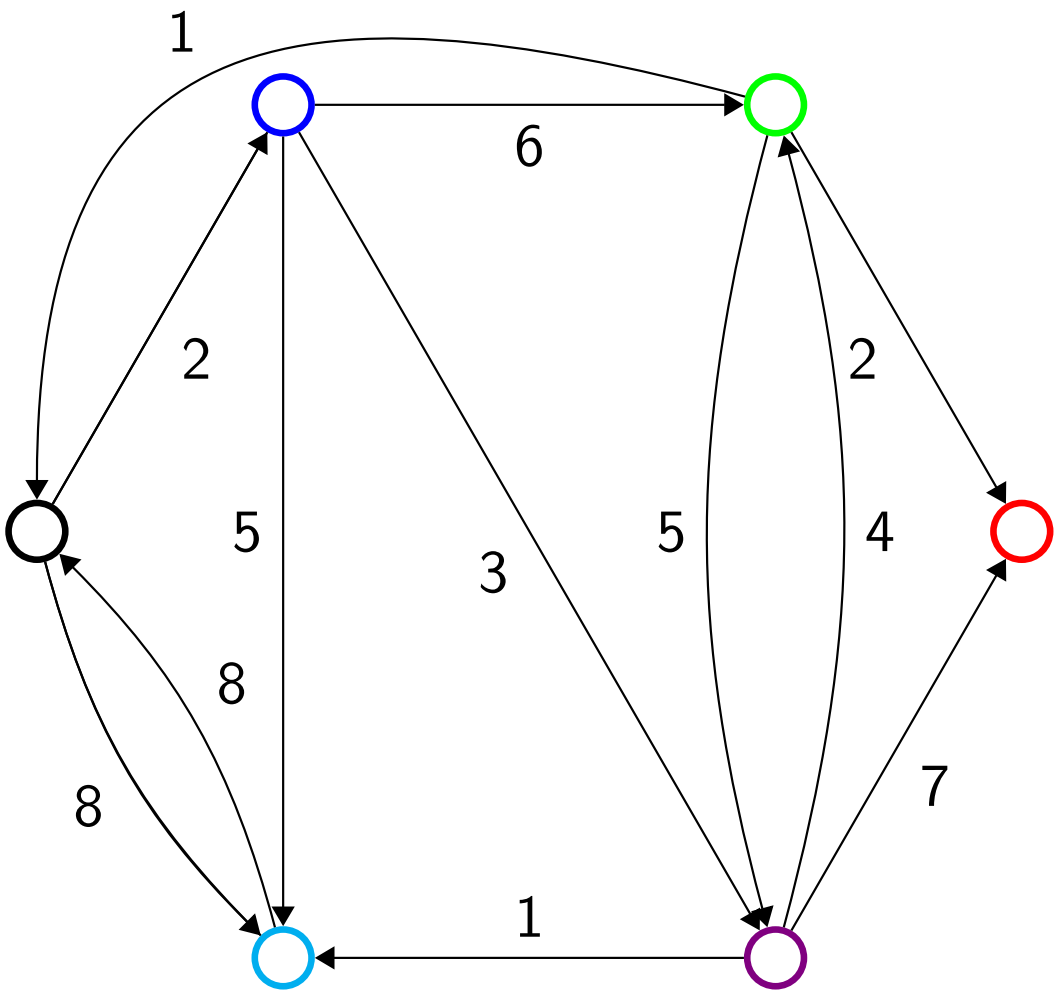
Let $G = (V, E)$ be a graph with edge weights $w : E \to \mathbb{R}$. Let $s, x \in V$ and define

$$\Delta(s, x) = \begin{cases} \min\{w(p) \mid p \text{ is a path from } s \text{ to } x\} & \text{a path from } s \\ & \text{to } x \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

The *shortest-path problem* for s and x consists of computing $\Delta(s, x)$ and a path p from s to x such that $w(p) = \Delta(s, x)$. Such a path is called shortest or optimal path from s to x.

Example 21

Reconsider the graph from Example 20.

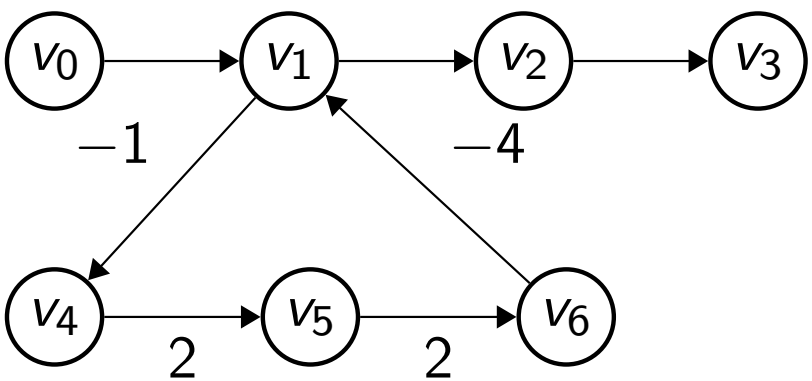


clockwise numbering of nodes from 0 to 5; start node s (0, black)

x	$\Delta(s,x)$	shortest path
0	0	0
1	2	$0 \to 1$
2	8	$0 \to 1 \to 2$
3	10	$0 \to 1 \to 2 \to 3$
4	5	$0 \to 1 \to 4$
5	6	$0 \to 1 \to 4 \to 5$

Existence of shortest paths

- suppose there is a path between s and u in graph G
- it may be surprising, that a shortest path does not always exist
- consider graph with a node x and non-empty path
  - from x to x (cycle),
  - with negative weight,
  - crossing the path from s to u
- then one can add additional cycles around x and decrease the weight
- any shortest path can be made shorter
- here is an example with $s = v_0$, $x = v_1$, and $u = v_3$



graph with negative weights appear e.g. in chemistry:

- compounds: nodes
- reaction: edge
- energy consumed ($\mathbb{R}_-$) or produced ($\mathbb{R}_+$): edge label

Non-negative weights and optimal subpaths

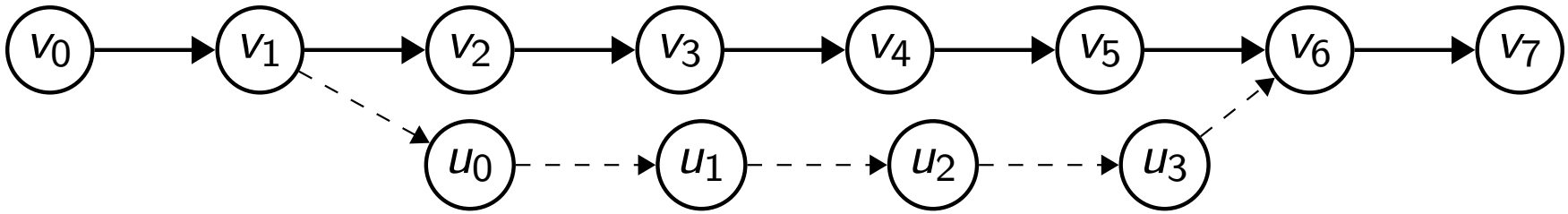
- in many cases, we can shift weights (by adding $-\min\{w(e) \mid e \in E\}$)
- so we restrict to the case of non-negative weights
- the key to many algorithms solving the shortest path problem is the fact that any subpath of a shortest path has minimum weight

Theorem 15

Consider a shortest path from s to u which includes a path from x to y. Then this path is a shortest path from x to y.

Non-negative weights and optimal subpaths

– we do not give a formal proof, but instead consider an example with a shortest path $p = v_0 \to v_1 \to v_2 \to \ldots \to v_5 \to v_6 \to v_7$:



– now look at the subpath $p' = v_1 \to v_2 \to \ldots \to v_5 \to v_6$

– assume that this is not the shortest path from v_1 to v_6 $\quad (*)$

– so there exists a path p'' from v_1 to v_6, say along the dashed edges, such that $w(p'') < w(p')$

– but then we could replace the subpath p' in p by p'' and obtain a path from v_0 to v_7 with total weight smaller than $w(p)$

– as p is the shortest path, this is not possible

$\Rightarrow$ assumption $(*)$ was wrong, i.e. the subpath p' is a shortest path

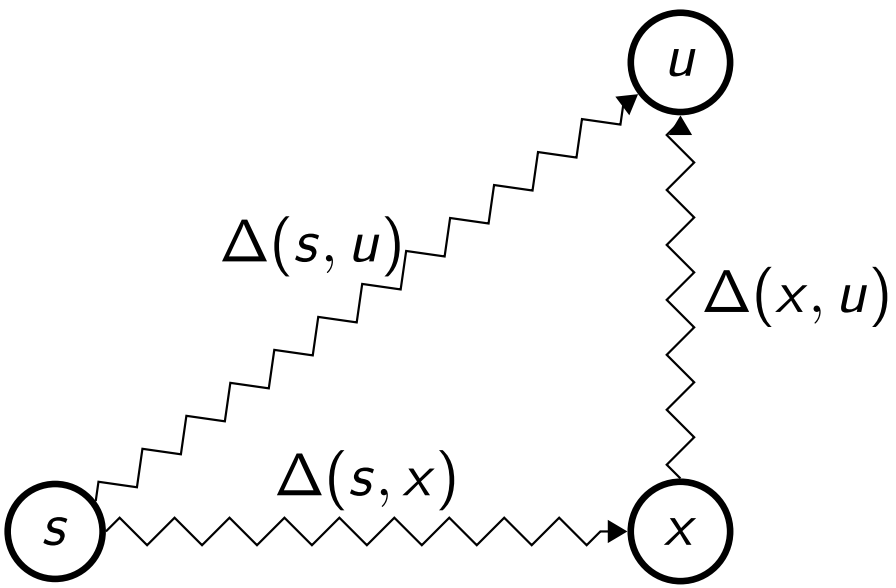
Triangle inequality

– the *optimality of subpath*-property is based on a general property which holds for many mathematical structures: the triangle inequality

Theorem 16

For any nodes s, u, $x \in V$ we have $\Delta(s,u) \leq \Delta(s,x) + \Delta(x,u)$.

– again, we do not give a formal proof, but just consider the following picture, in which the zigzag-lines represent possibly long paths involving many nodes



– if we have a shortest path from s to u, any path from s to u involving some particular node x will not be shorter, even if the path from s to x and from x to u are both a shortest path

– so formally
$\Delta(s,u) \leq \Delta(s,x) + \Delta(x,u)$

Single source shortest path problem

- recall that the shortest path problem we had considered involves two particular node s and u
- so lets denote it by $PathP(s, u)$, where $PathP$ stands for path problem
- a generalization of this is the single source shortest path problem:

  for a given node s (the source node) determine $\Delta(s, x)$ (and shortest path from s to x) for all $x \in V$

- let us denote this problem by $PathP(s, V)$
- with the most efficient algorithms known today, it is (in general) not easier to solve $PathP(s, u)$ for a particular $u \in V$ than to solve $PathP(s, V)$
- so consider how to solve $PathP(s, V)$
- if we have a solution for this problem, we can lookup the solution for a particular target node u

Directed acyclic graphs

Definition 17

A directed acyclic graph $G = (V, E)$ (often abbreviated as DAG) is a finite graph without non-empty cycles, i.e. for any node $s \in V$ there is no non-empty path from s to s.

A DAG can be used to model different kinds of data:

- dependencies of software consisting of many modules
- version history in version control systems
- logic blocks in electronic circuits design
- dependencies of events in Bayesian networks
- historical dependencies in family trees
- dependencies in a system of equations (recurrences)

We are in particular interested in

1. solving the shortest path problem for DAGs and
2. applying the solution to solve the edit distance problem

Let us first consider how to solve the first problem.

Directed acyclic graphs

- we have seen that a shortest path from node s to x consists of subpaths each of which is a shortest path for the nodes it connects
- this of course holds for DAGs as a special form of graphs
- suppose $s \neq x$
- then the shortest path from s to x must contain a least one edge
- so consider the last edge of the shortest path, i.e. an edge $y \rightarrow x \in E$ for some node y
- by the subpath-optimality principle the subpath of our shortest path from s to x which connects s with y must be a shortest path, too.
- so we can conclude $\Delta(s, x) = \Delta(s, y) + w(y, x)$
- of course, we do not know in advance which incoming edge to x will be the last edge of a shortest path, but there will be at least one such edge
- so we just minimize over all incoming edges, i.e. we can compute

$$\Delta(s, x) = \min\{\Delta(s, y) + w(y, x) \mid y \rightarrow x \in E\}$$

Directed acyclic graphs

- this is the central equation allowing to solve the shortest path problem
- to apply it we have to solve one subproblem:
- we can only evaluate

$$\Delta(s,x) = \min\{\Delta(s,y) + w(y,x) \mid y \to x \in E\}$$

  if we already know $\Delta(s,y)$
- so we have to compute the values $\Delta(s,\_)$ in an order which respects the edges, i.e. if there is an edge $y \to x \in E$ we have to compute $\Delta(s,y)$ before $\Delta(s,x)$
- this is an order of the nodes in the graph called *topological order*

Definition 18

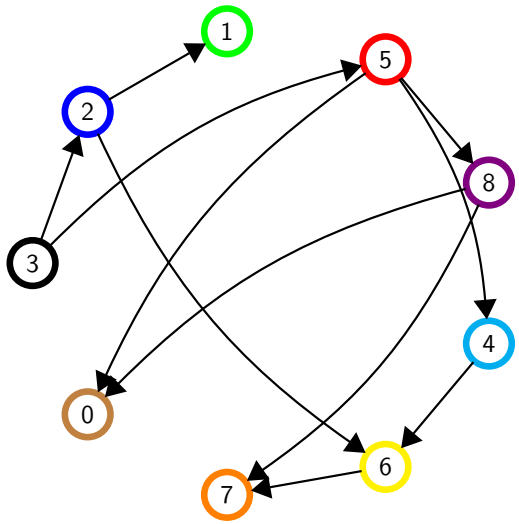
For a given directed graph $G = (V,E)$ a topological order $\prec$ of V satisfies:

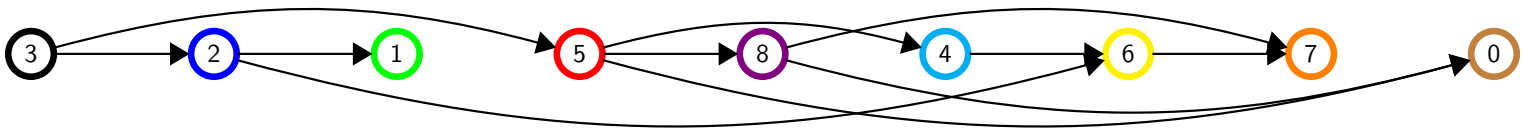
$$y \to x \in E \Rightarrow y \prec x$$

- note: there is at least one topological order for a DAG and usually more than one possible such orders

Example 22

- consider the graph $G = (V, E)$ with a layout of the nodes on a circle:
- here is a different layout of the graph in a linear form:





- as the edges only go from left to right, the layout implies the topological order

$$3 \prec 2 \prec 1 \prec 5 \prec 8 \prec 4 \prec 6 \prec 7 \prec 0$$

Directed acyclic graphs

Conclusion: solving shortest path problem on DAG $G = (V, E)$

- compute a topological order $\prec$ on G (runtime: $O(|V| + |E|)$)
- for given start node s eliminate all nodes $x \in V$, s.t. $x \prec s$ and all nodes outgoing from x
- for all nodes $x \in V$, $s \prec x$ in topological order, compute

$$\Delta(s, x) = \min\{\Delta(s, y) + w(y, x) \mid y \to x \in E\}$$

  and store it in a table for looking it up later
- as $y \to x \in E$ implies $y \prec x$, $\Delta(s, y)$ is available (and can be looked up in constant time) when $\Delta(s, x)$ is evaluated
- overall runtime: $O(|V| + |E|)$

- to solve the edit distance problem using this method we introduce the notion of edit graphs, as a special form of graphs

Edit graphs

Definition 19

The *edit graph* $G(u, v)$ of u and v is an edge labeled directed graph. The nodes are the pairs (i, j), $0 \leq i \leq m$, $0 \leq j \leq n$. The edges are given as follows:

- For $1 \leq i \leq m, 0 \leq j \leq n$ there is a deletion edge

$$(i-1, j) \xrightarrow{u[i] \mapsto \varepsilon} (i, j)$$

- For $0 \leq i \leq m, 1 \leq j \leq n$ there is an insertion edge

$$(i, j-1) \xrightarrow{\varepsilon \mapsto v[j]} (i, j)$$

- For $1 \leq i \leq m, 1 \leq j \leq n$ there is a replacement edge

$$(i-1, j-1) \xrightarrow{u[i] \mapsto v[j]} (i, j)$$

□

Figure 4: A part of the edit graph $G(u, v)$

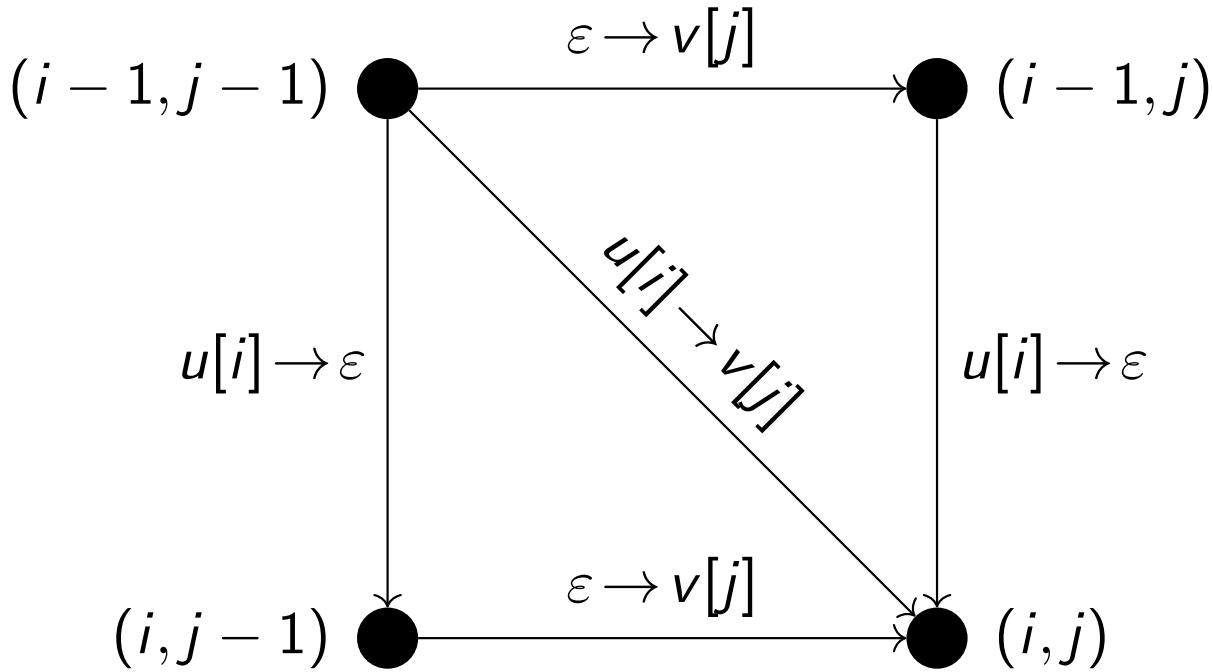
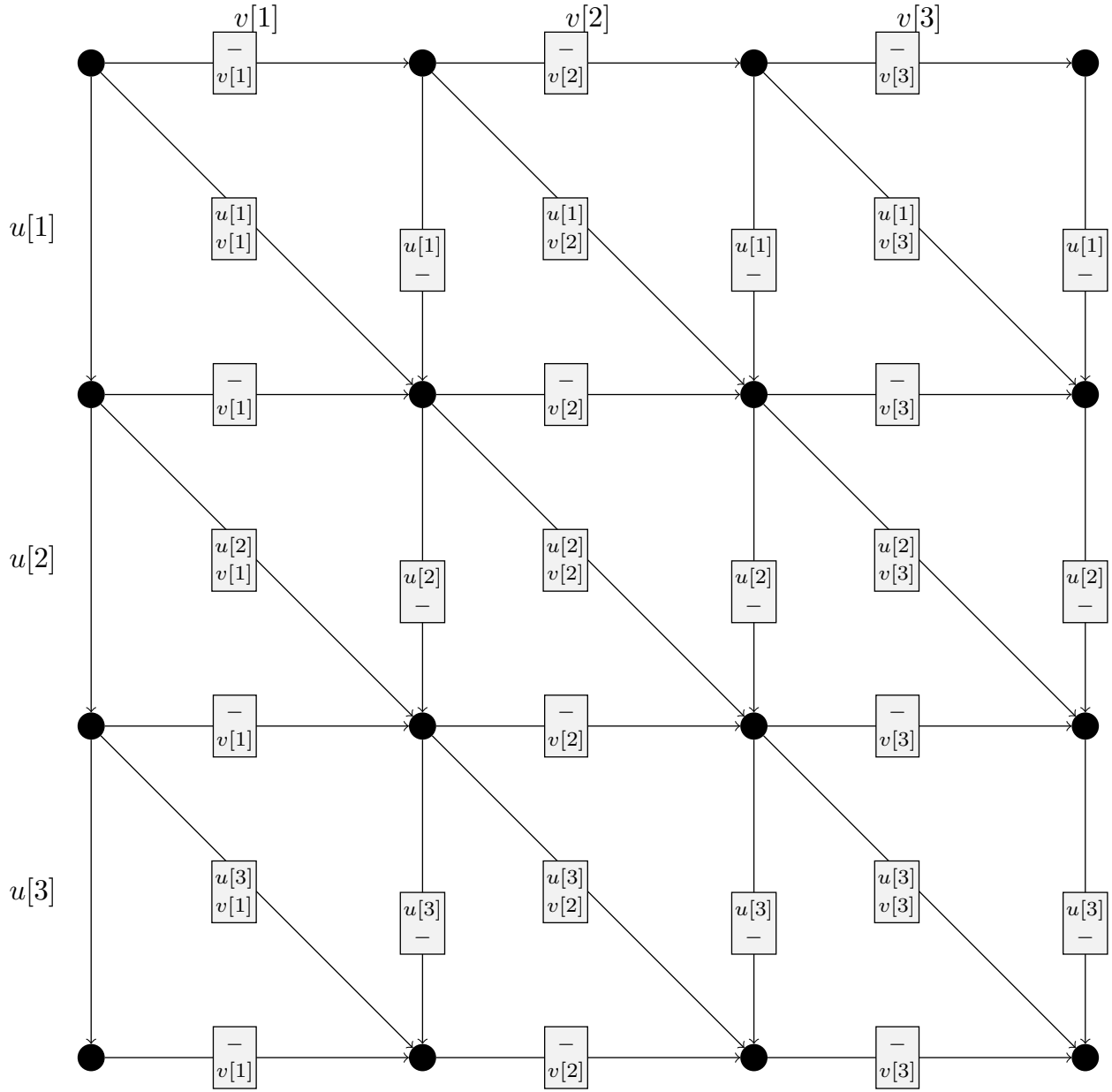


Figure 5: An edit graph $G(u, v)$ where $|u| = |v| = 3$



Edit graphs

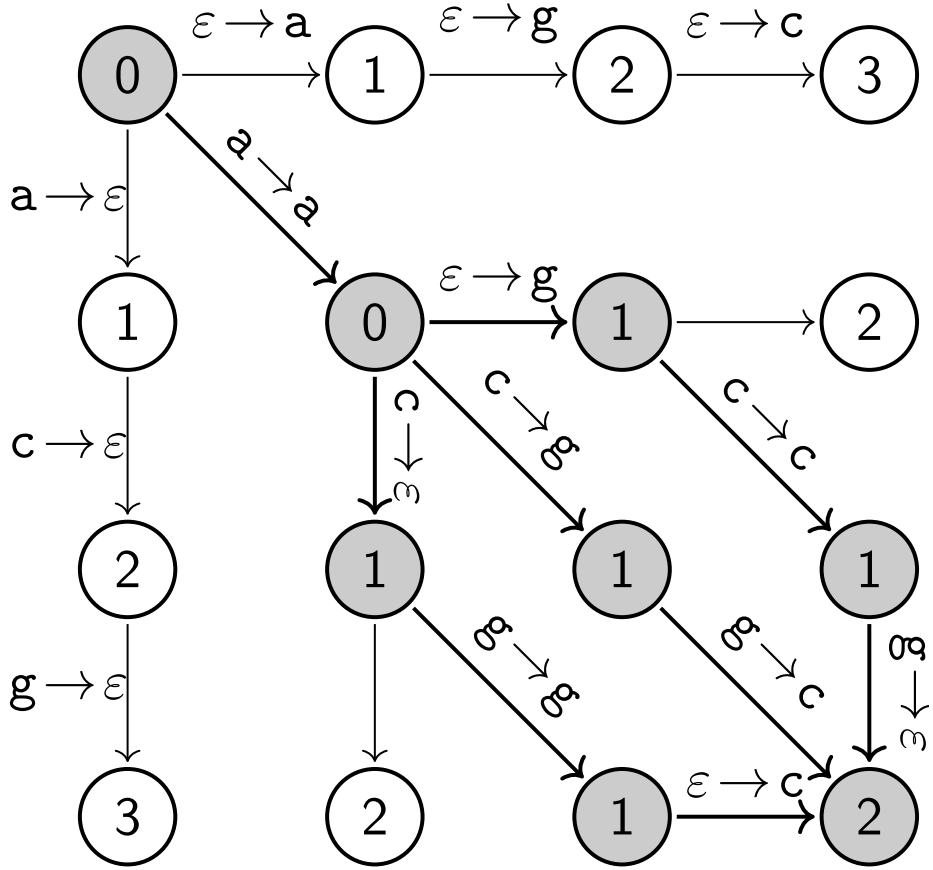
- The central feature of $G(u, v)$ is that each path from (i', j') to (i, j) is labeled by an alignment of $u[i' + 1 \ldots i]$ and $v[j' + 1 \ldots j]$, and a different path is labeled by a different alignment.
- So there is a one-to-one correspondence of paths and alignments.
- When applying the cost function δ to the edges of $G(u, v)$ one obtains weights, i.e. $w((i', j') \xrightarrow{\alpha \to \beta} (i, j)) = \delta(\alpha \to \beta)$.
- So each path represents an alignment and the weight of the path is the cost of the alignment.
- Thus, a shortest path from (i', j') to (i, j) is labeled by an optimal alignment of $u[i' + 1 \ldots i]$ and $v[j' + 1 \ldots j]$.
- In particular, a shortest path from $(0, 0)$ to (i, j) is labeled by an optimal alignment of $u[1 \ldots i]$ and $v[1 \ldots j]$.

Example 23

Edit graph for $(u, v) = (\mathsf{acg}, \mathsf{agc})$ restricted to the edges on shortest paths from $(0, 0)$. The
paths from $(1, 1)$ to $(3, 3)$ of thick nodes and edges represents the alignments

$(\varepsilon \to \mathsf{g}, \mathsf{c} \to \mathsf{c}, \mathsf{g} \to \varepsilon)$,
$(\mathsf{c} \to \mathsf{g}, \mathsf{g} \to \mathsf{c})$,
$(\mathsf{c} \to \varepsilon, \mathsf{g} \to \mathsf{g}, \varepsilon \to \mathsf{c})$, of
$u[2 \ldots 3] = \mathsf{cg}$ and
$v[2 \ldots 3] = \mathsf{gc}$.

- node (i, j) labeled by the cost of an optimal alignment of $u[1 \ldots i]$ and $v[1 \ldots j]$, i.e. the edit distance of these sequences



Edit graphs

- From the above, we deduce that a shortest path from $(0,0)$ to (m,n) is labeled by an optimal alignment of $u = u[1 \ldots m]$ and $v = v[1 \ldots n]$.
- So computing optimal alignments means to enumerate shortest paths from $(0,0)$ to (m,n) in $G(u,v)$.

Conclusion

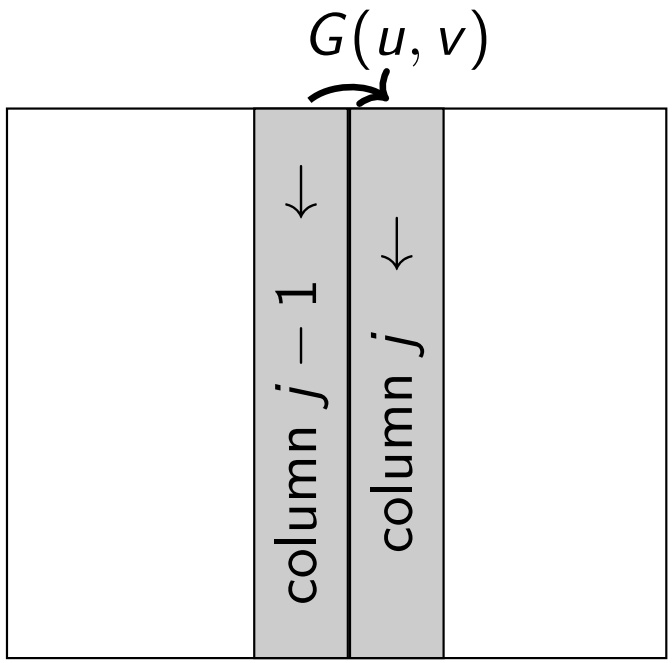
- We have formulated the edit distance problem for u and v as a shortest path problem for $G(u,v)$.
- we are in particular interested in the shortest path from $(0,0)$ to (m,n).

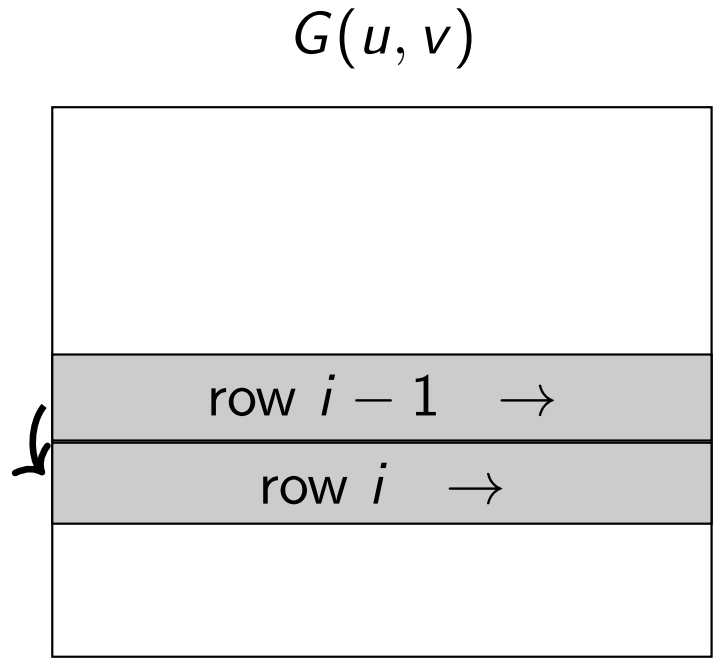
- As $G(u,v)$ only contains edges of the form $(i',j') \xrightarrow{\alpha \mapsto \beta} (i,j)$ such that $i' < i$ or $j' < j$, there are no cycles.
- So $G(u,v)$ is a directed acyclic graph, a DAG.
- We already know how to solve shortest path problems on DAGs: We first have to determine a topological order of the nodes in $G(u,v)$.
- There are basically two main topological orders in $G(u,v)$:

Edit graphs

columnwise topological order:
$(i', j') \prec (i, j) \iff$
$j' < j$ or $(j' = j$ and $i' < i)$

rowwise topological order:
$(i', j') \prec (i, j) \iff$
$i' < i$ or $(i' = i$ and $j' < j)$





– in both cases, we have $(i, j-1) \prec (i, j)$, $(i-1, j-1) \prec (i, j)$, and $(i-1, j) \prec (i, j)$.

From edit graphs to edit matrices

- recall the general recurrence from frame 59:
  $\Delta(s,x) = \min\{\Delta(s,y) + w(y,x) \mid y \to x \in E\}$
- in our case the nodes are pairs (i,j) and the edges are labeled by edit operations, to which we apply δ to obtain weights

Conclusion: solving the edit distance problem

- compute $G(u,v)$
- for all nodes $(i,j) \in G(u,v)$ in topological order, compute

$$\Delta((0,0),(i,j)) = \min\{\Delta((0,0),(i',j')) + \delta(\alpha \to \beta) \mid (i',j') \xrightarrow{\alpha \to \beta} (i,j) \text{ is edge in } G(u,v)\}$$

- to simplify the notation, we introduce the abbreviation

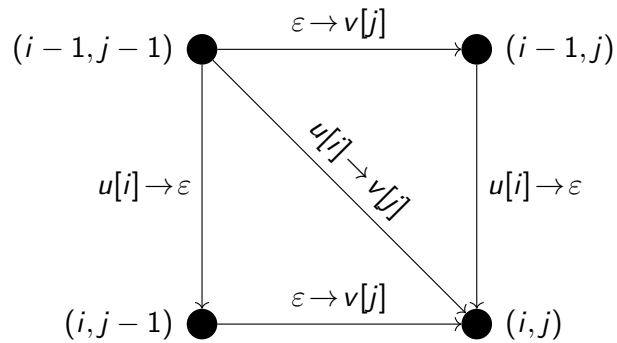
$$E_\delta(i,j) = \Delta((0,0),(i,j))$$

- and so the equation above simplifies to

$$E_\delta(i,j) = \min\{E_\delta(i',j') + \delta(\alpha \to \beta) \mid (i',j') \xrightarrow{\alpha \to \beta} (i,j) \text{ is edge in } G(u,v)\}$$

From edit graphs to edit matrices

- recall the simple structure of the edit graph
- if $i > 0$ and $j > 0$, node (i, j) has three incoming edges, as already illustrated previously:



- so we can rewrite

$$E_\delta(i,j) = \min\{E_\delta(i',j') + \delta(\alpha \to \beta) \mid (i',j') \xrightarrow{\alpha \to \beta} (i,j) \text{ is edge in } G(u,v)\}$$

$$\text{as } E_\delta(i,j) = \min \left\{ \begin{array}{l} E_\delta(i-1,j) + \delta(u[i] \to \varepsilon) \\ E_\delta(i,j-1) + \delta(\varepsilon \to v[j]) \\ E_\delta(i-1,j-1) + \delta(u[i] \to v[j]) \end{array} \right\}$$

From edit graphs to edit matrices

- for $i > 0$ and $j = 0$, node (i, j) only has one incoming edge $(i-1, j) \xrightarrow{u[i] \to \varepsilon} (i, j)$ and so we simplify

$$E_\delta(i,j) = \min\{E_\delta(i',j') + \delta(\alpha \to \beta) \mid (i',j') \xrightarrow{\alpha \to \beta} (i,j) \text{ is edge in } G(u,v)\}$$

$$\text{as } E_\delta(i,j) = E_\delta(i-1,j) + \delta(u[i] \to \varepsilon)$$

- for $i = 0$ and $j > 0$, node (i, j) only has one incoming edge $(i, j-1) \xrightarrow{\varepsilon \to v[j]} (i, j)$ and so

$$E_\delta(i,j) = E_\delta(i,j-1) + \delta(\varepsilon \to v[j])$$

- for $i = 0$ and $j = 0$, node (i, j) has no incoming edge and $E_\delta(i, j)$ is the weight of the shortest path from $(0, 0)$ to $(0, 0)$ which is 0. So $E_\delta(i, j) = 0$.

Solving the edit distance problem

Combining these equations, we obtain the following recurrences for E_δ:

$$E_\delta(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ E_\delta(0,j-1) + \delta(\varepsilon \to v[j]) & \text{if } i = 0 \text{ and } j > 0 \\ E_\delta(i-1,0) + \delta(u[i] \to \varepsilon) & \text{if } i > 0 \text{ and } j = 0 \\ \min \left\{ \begin{array}{l} E_\delta(i-1,j) + \delta(u[i] \to \varepsilon) \\ E_\delta(i,j-1) + \delta(\varepsilon \to v[j]) \\ E_\delta(i-1,j-1) + \delta(u[i] \to v[j]) \end{array} \right\} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

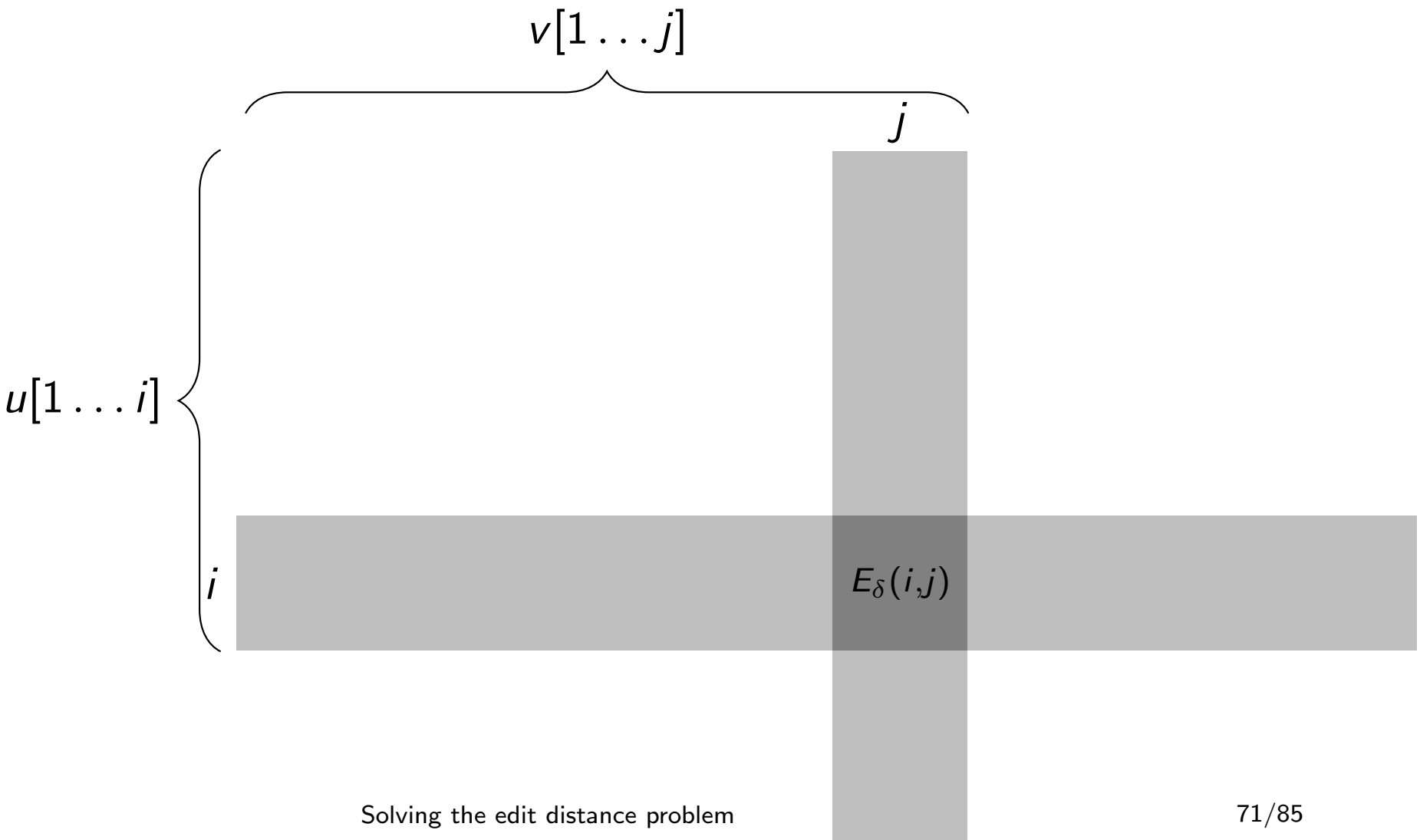
Recall that $E_\delta(i,j)$ is

- the weight of the shortest path from $(0,0)$ to (i,j) in $G(u,v)$ and equivalently
- also the cost of an optimal alignment of $u[1 \ldots i]$ and $v[1 \ldots j]$.

In the following we neglect $G(u,v)$ for a while and view E_δ as a matrix with $m+1$ rows (indexed from 0 to m) and $n+1$ columns (indexed from 0 to n) of numeric values. Such a matrix is called $(m+1) \times (n+1)$-matrix.

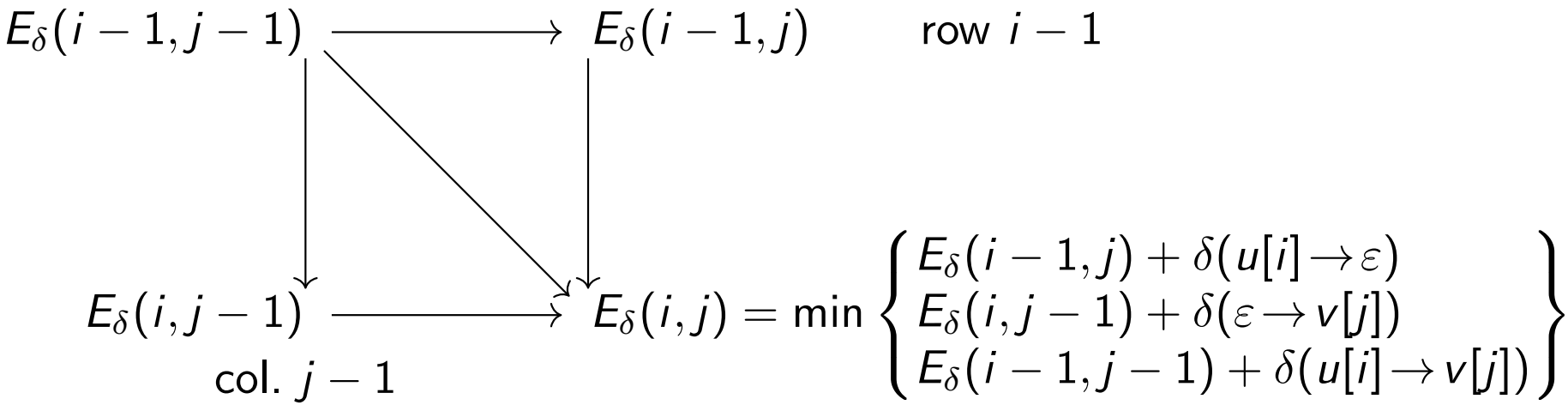
The index i and j are not only row and column numbers of the matrix, but also refer to prefixes of u and v, respectively, see Figure 6.

Figure 6: The curly brackets mark the substrings of u and v an entry in E_δ refers to. In particular, $E_\delta(i,j)$ refers to the pair of prefixes $u[1\ldots i]$ and $v[1\ldots j]$.



Solving the edit distance problem

- By definition, $E_\delta(m,n) = edist_\delta(u[1\ldots m], v[1\ldots n]) = edist_\delta(u,v)$, i.e. it gives the edit distance of u and v.
- According to the recurrence above, $E_\delta(i,j)$ depends on the previous value $E_\delta(i-1,j)$ in the current column j and on two values $E_\delta(i-1,j-1)$ and $E_\delta(i,j-1)$ in the previous column.
- See the following illustration, in which a directed edge $E_\delta(i',j') \to E_\delta(i,j)$ means that $E_\delta(i,j)$ depends on $E_\delta(i',j')$:



- The values in E_δ need to be computed in topological order of the nodes in the underlying graph $G(u,v)$.
- We compute E_δ columnwise and each column from top to bottom.

Algorithm 1 (Computation of the edit distance)

Input: sequences $u = u[1 \ldots m]$ and $v = v[1 \ldots n]$
cost function δ

Output: $edist_\delta(u, v)$

1: $E_\delta(0,0) \leftarrow 0$
2: **for** $i \leftarrow 1$ **upto** m **do**
3: $\quad E_\delta(i,0) \leftarrow E_\delta(i-1,0) + \delta(u[i] \rightarrow \varepsilon)$
4: **end for**
5: **for** $j \leftarrow 1$ **upto** n **do**
6: $\quad E_\delta(0,j) \leftarrow E_\delta(0,j-1) + \delta(\varepsilon \rightarrow v[j])$
7: $\quad$ **for** $i \leftarrow 1$ **upto** m **do**
8: $$E_\delta(i,j) \leftarrow \min \left\{ \begin{array}{l} E_\delta(i-1,j) + \delta(u[i] \rightarrow \varepsilon) \\ E_\delta(i,j-1) + \delta(\varepsilon \rightarrow v[j]) \\ E_\delta(i-1,j-1) + \delta(u[i] \rightarrow v[j]) \end{array} \right\}$$
9: $\quad$ **end for**
10: **end for**
11: **print** $E_\delta(m,n)$

Example 24

Let $u = \texttt{bcacd}$, $v = \texttt{dbadad}$, and assume that δ is the unit cost function. Then E_δ is as follows, with the index range for i and j shown in the second column and second row.

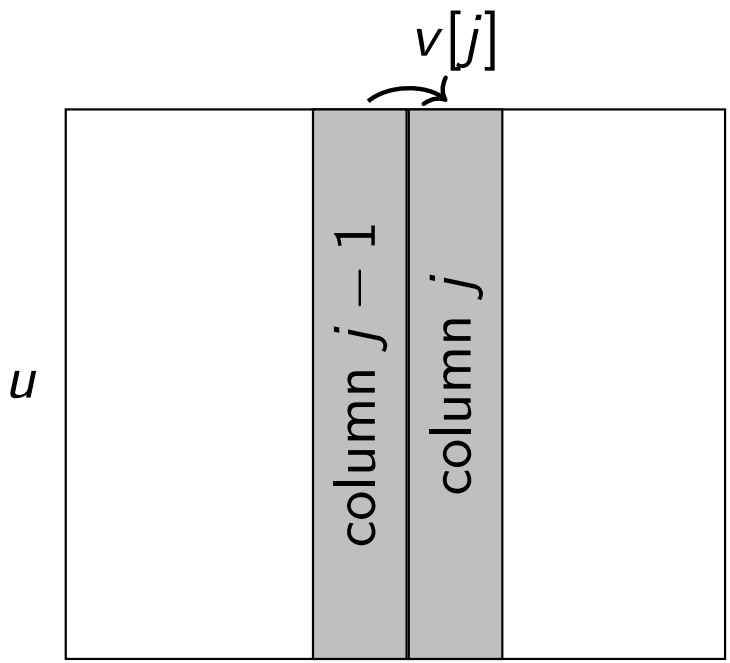
	i		d	b	a	d	a	d
j		0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
b	1	1	1	1	2	3	4	5
c	2	2	2	2	2	3	4	5
a	3	3	3	3	2	3	3	4
c	4	4	4	4	3	3	**4**	4
d	5	5	4	5	4	3	4	$\boxed{4}$

$$\boxed{4} = \min \begin{Bmatrix} 4+1 \\ 4+1 \\ \mathbf{4}+0 \end{Bmatrix}$$

Hence the edit distance of u and v is $E_\delta(5,6) = 4$. $\square$

Solving the edit distance problem

- Each entry in E_δ is computed in constant time.
- As there are $(m+1) \cdot (n+1)$ entries, the running time is $O(mn)$.
- If we only want to compute the edit distance, we only need the last column and its last value.
- Obviously, the values in each column only depend on the values of the current and the previous column, see the following illustration.



  - Hence, it suffices to store only two columns, and thus the so called "distance-only algorithm" requires only $O(m)$ space.
  - It is even possible to arrange the computation in such a way, that the space for only one column is required, plus two extra scalars (integers or floats, depending on the cost function).

Solving the edit distance problem

In molecular biology, the above algorithm is often called "the dynamic programming algorithm". However, dynamic programming (DP, for short) is a general programming paradigm. A problem can be solved by DP, if the following holds:

- optimal solutions to the problem can be derived from optimal solutions to subproblems.
- the optimal solutions can efficiently be determined, if a table of solutions for subproblems of increasing sizes are computed.

For the edit distance problem we derive the optimal solution $edist_\delta(u, v)$ from the optimal solution of subproblems, i.e. the edit distance of each pair of prefixes of u and v, respectively. And the optimal solutions to the subproblems are tabulated in E_δ, such that
$E_\delta(i, j) = edist_\delta(u[1 \ldots i], v[1 \ldots j])$.

Solving the edit distance problem

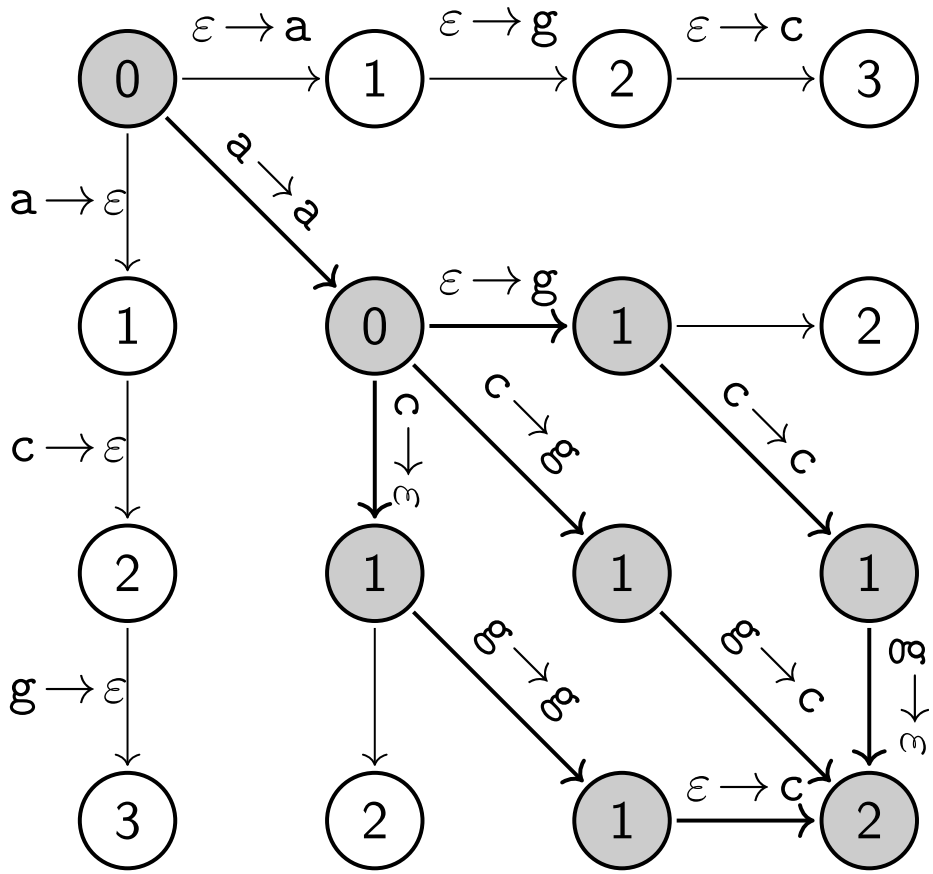
- To completely solve the edit distance problem, we also have to compute the optimal alignments.
- An optimal alignment is recovered by tracing back from the entry $E_\delta(m, n)$ to an entry in its three-way minimum that yielded it, determining which entry gave rise to that entry, and so on back to the entry $E_\delta(0, 0)$.
- This requires saving the entire table, giving an algorithm that takes $O(mn)$ space.
- This traceback algorithm can best be explained by referring to the edit graph and the notion of minimizing edges.

Definition 20

An edge $(i', j') \xrightarrow{\alpha \to \beta} (i, j)$ in $G(u, v)$ is *minimizing* if $E_\delta(i, j)$ equals $E_\delta(i', j') + \delta(\alpha \to \beta)$.

Example 25

The edit graph for $u = acg$ and $v = agc$ restricted to minimizing edges. The thick edges are those which are on the path from $(0, 0)$ to $(3, 3)$. Node (i, j) is labeled by $E_\delta(i, j)$.

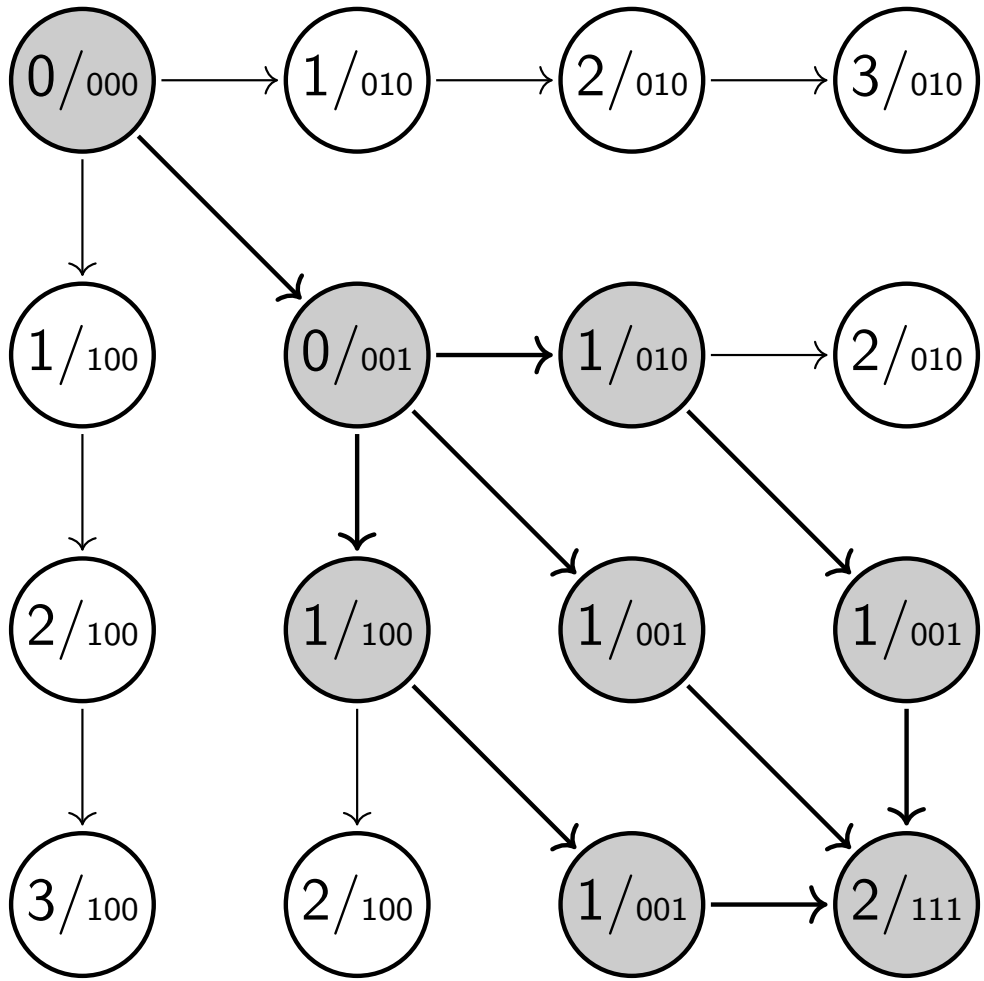


Solving the edit distance problem

- The traceback starts at node (m, n) and traces the minimizing edges back to node $(0, 0)$.
- The traceback method can be organized in such a way that each optimal alignment A of u and v is computed in $O(|A|)$ time.
- To facilitate the traceback (and implicitly storing the minimizing edges of the graph), we maintain with each entry $E_\delta(i, j)$ three bits, one for each type of edit operation.
- Each of these bits tells us whether the incoming edge into (i, j) is minimizing or not.

Example 26

Here we show the previous edit graph, in which each node (i, j) is labeled by $E_\delta(i, j)$ and the three bits representing the incoming minimizing edges.[a]



[a] bits = $d\ i\ r$, d: deletion, i: insertion, r: replacement.

Solving the edit distance problem

Theorem 21

The edit distance problem for two sequences u of length m and v of length n can be solved in $O(mn + z)$ time, where z is the total length of all alignments of u and v. □

Example 27

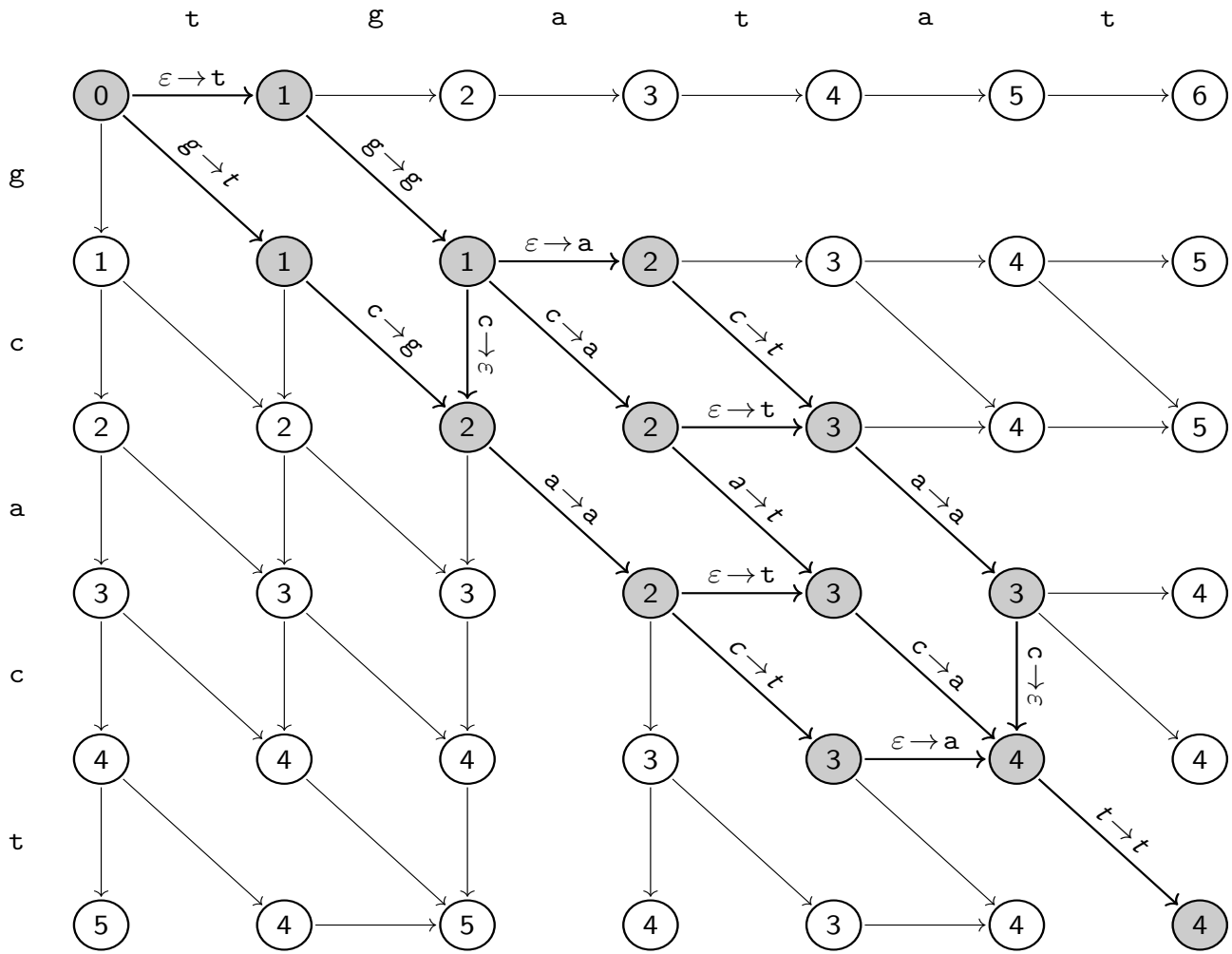
Let $u = \texttt{gcact}$ and $v = \texttt{tgatat}$. Suppose δ is the unit cost function. Then we have $edist_\delta(u, v) = 4$ and there are the following optimal alignments of u and v.

$$\begin{pmatrix} \texttt{g c a c - t} \\ \texttt{t g a t a t} \end{pmatrix} \begin{pmatrix} \texttt{- g c a c - t} \\ \texttt{t g - a t a t} \end{pmatrix} \begin{pmatrix} \texttt{g c a - c t} \\ \texttt{t g a t a t} \end{pmatrix} \begin{pmatrix} \texttt{- g c a - c t} \\ \texttt{t g - a t a t} \end{pmatrix}$$

$$\begin{pmatrix} \texttt{- g c a c t} \\ \texttt{t g a t a t} \end{pmatrix} \begin{pmatrix} \texttt{- g c - a c t} \\ \texttt{t g a t a - t} \end{pmatrix} \begin{pmatrix} \texttt{- g - c a c t} \\ \texttt{t g a t a - t} \end{pmatrix}$$

Figure 7 shows $G(u, v)$ with all minimizing edges. The minimizing paths are given by the thick edges. Each node is marked by the corresponding edit distance. It is straightforward to read the optimal alignments of u and v from the edit graph. □

Figure 7: All nodes and all minimizing edges in the edit graph $G(u, v)$ for $u = \texttt{gcact}$ and $v = \texttt{tgatat}$. The thick edges (marked with their edge labels) form the minimizing paths and all nodes on the minimizing paths are shown in gray.



$$\begin{pmatrix} \texttt{g}\,\texttt{c}\,\texttt{a}\,\texttt{c}\,\texttt{-}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{t} \end{pmatrix} \quad \begin{pmatrix} \texttt{-}\,\texttt{g}\,\texttt{c}\,\texttt{a}\,\texttt{c}\,\texttt{-}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{-}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{t} \end{pmatrix} \quad \begin{pmatrix} \texttt{g}\,\texttt{c}\,\texttt{a}\,\texttt{-}\,\texttt{c}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{t} \end{pmatrix} \quad \begin{pmatrix} \texttt{-}\,\texttt{g}\,\texttt{c}\,\texttt{a}\,\texttt{-}\,\texttt{c}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{-}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{t} \end{pmatrix}$$

$$\begin{pmatrix} \texttt{-}\,\texttt{g}\,\texttt{c}\,\texttt{a}\,\texttt{c}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{t} \end{pmatrix} \quad \begin{pmatrix} \texttt{-}\,\texttt{g}\,\texttt{c}\,\texttt{-}\,\texttt{a}\,\texttt{c}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{-}\,\texttt{t} \end{pmatrix} \quad \begin{pmatrix} \texttt{-}\,\texttt{g}\,\texttt{-}\,\texttt{c}\,\texttt{a}\,\texttt{c}\,\texttt{t} \\ \texttt{t}\,\texttt{g}\,\texttt{a}\,\texttt{t}\,\texttt{a}\,\texttt{-}\,\texttt{t} \end{pmatrix}$$

Solving the edit distance problem

- The space requirement for the above method is $O(mn)$.
- Using a distance-only algorithm in a recursive strategy, one obtains a divide and conquer algorithm that can determine each optimal alignment in $O(\min\{m, n\})$ space and $O(mn)$ time.
- These algorithms are very important, since space, not time, is often the limiting factor when computing optimal alignments between large sequences.


Kruskal, J. and Sankoff, D. (1983).
Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.
Addison-Wesley, Reading, MA.


Wagner, R. and Fischer, M. (1974).
The String to String Correction Problem.
JACM, 21(1):168–173.