Approximate Cartesian Tree Matching with Substitutions

Panagiotis Charalampopoulos, *Jonas Ellert*, Manal Mohamed

Cartesian tree matching with substitutions

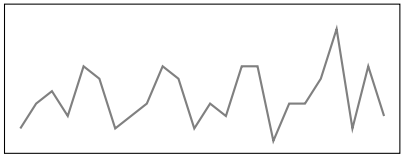
- not so much money in research ... maybe use algorithmic skills to exploit the stock market

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)

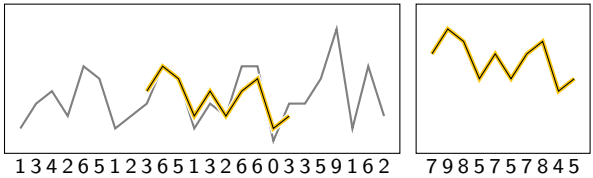


Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)



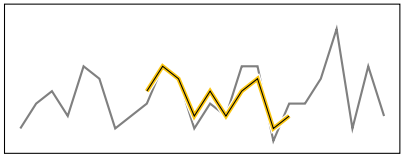
Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)

- exact pattern matching not OK
- only care about relative order

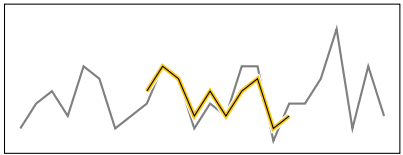


Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

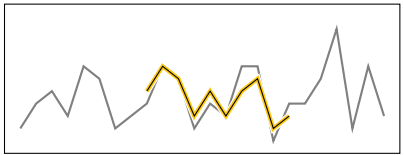
$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal

Previous-smaller-tree of an array:

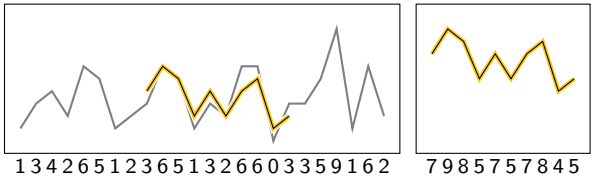
- dummy root node ($-\infty$)

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal

Previous-smaller-tree of an array:

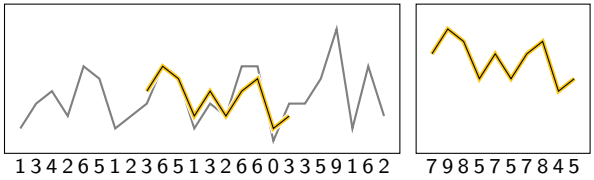
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal

Previous-smaller-tree of an array:

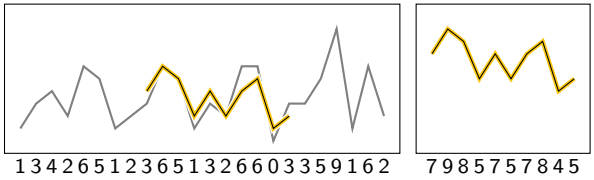
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

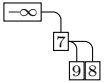
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

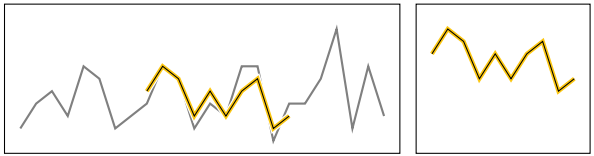
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

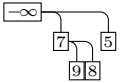
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

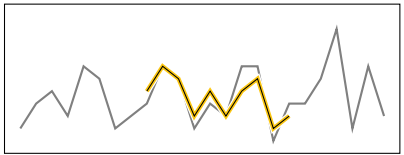
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

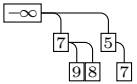
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

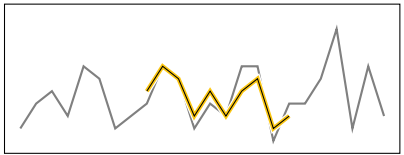
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

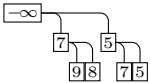
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

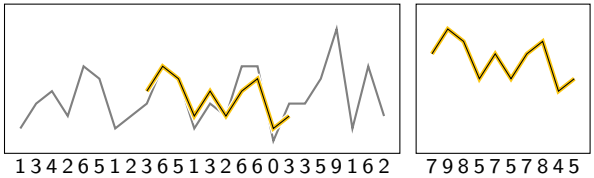
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

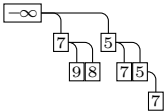
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

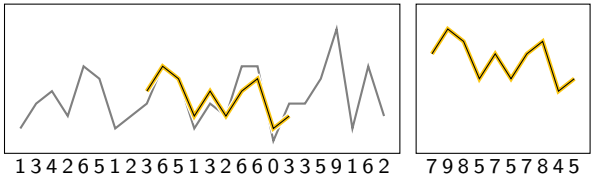
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

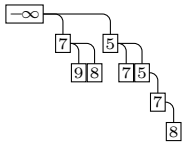
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

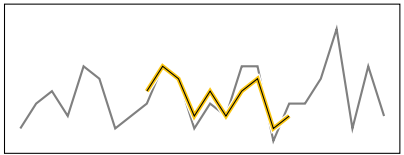
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

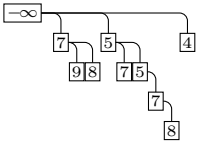
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

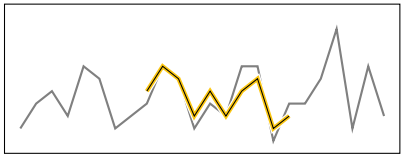
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

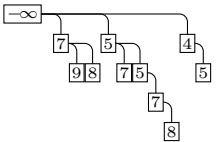
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

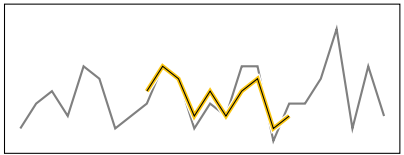
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

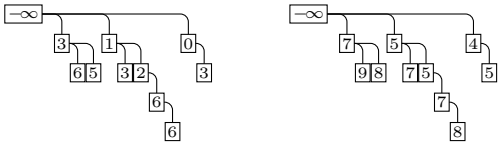
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

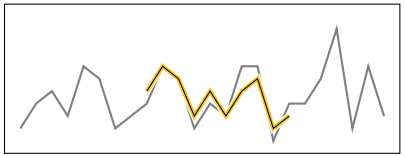
- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

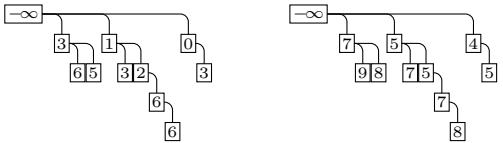
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

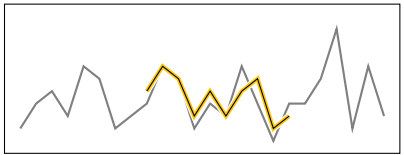
problem: not very robust...

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

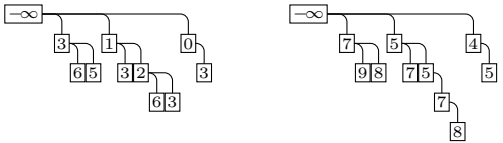
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

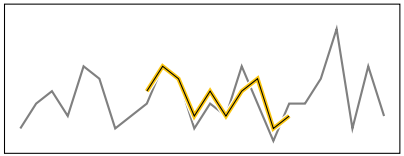
problem: not very robust...

Cartesian tree matching with substitutions

- not so much money in research ... maybe use algorithmic skills to exploit the stock market

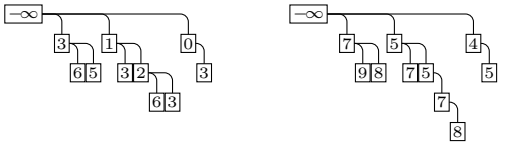
given:

- price history of stock (the text)
- interesting trend in price (the pattern)



- exact pattern matching not OK
- only care about relative order

$\Longrightarrow$ **Cartesian tree matching:**
match iff Cartesian trees are equal



Previous-smaller-tree of an array:

- dummy root node ($-\infty$)
- attach every value to closest value on the left that is smaller or equal

problem: not very robust...

solution: allow k substitutions in text *before* computing tree

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

A standard trick...

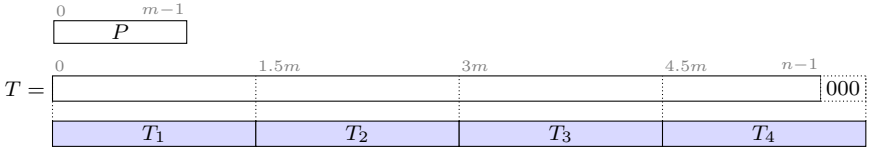
- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

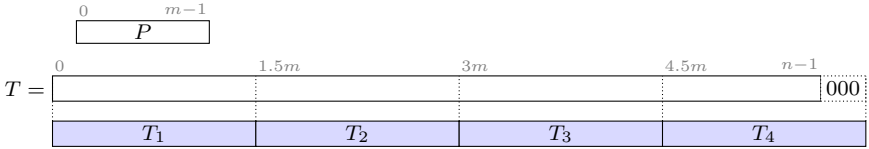


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

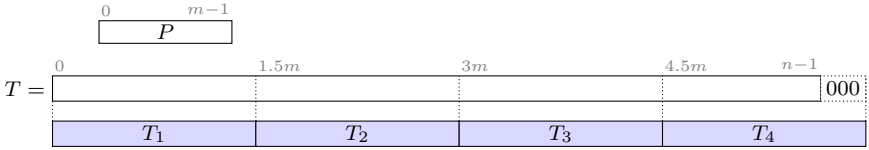


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

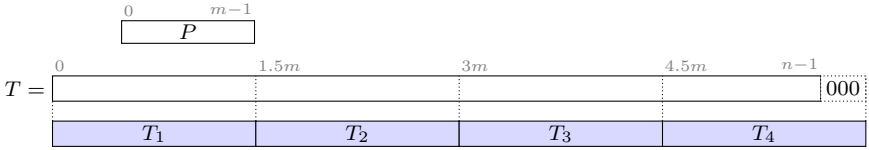


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

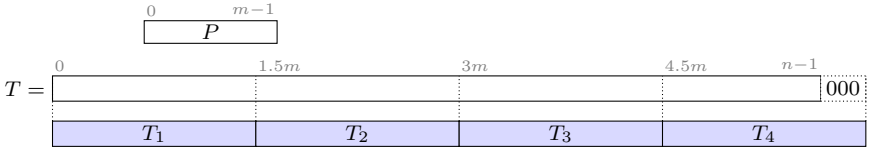


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

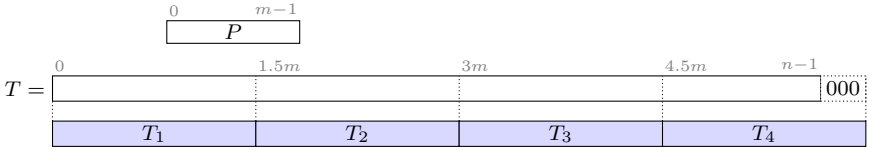


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

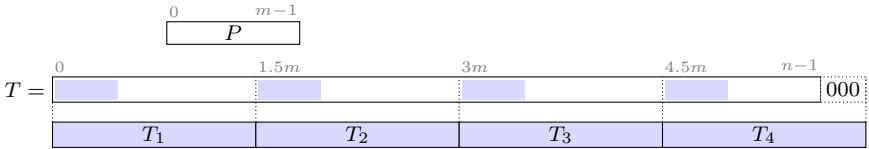


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

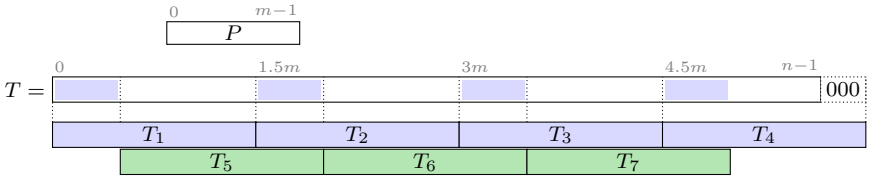


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

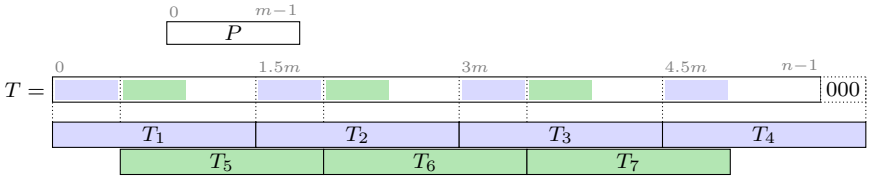


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

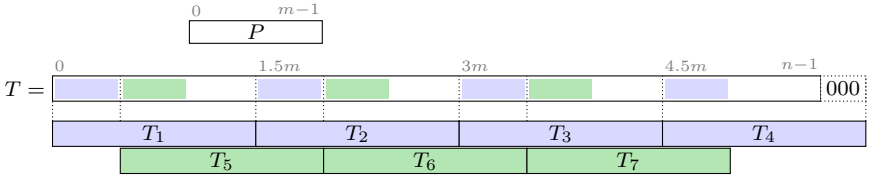


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

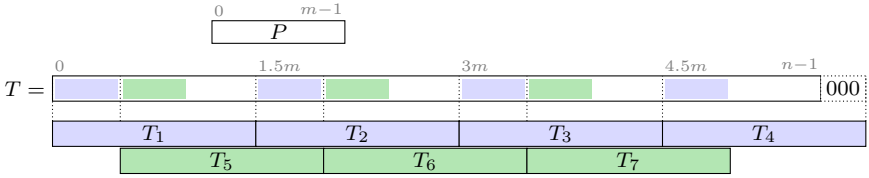


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

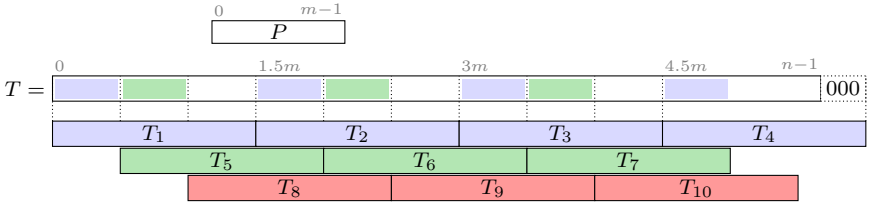


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

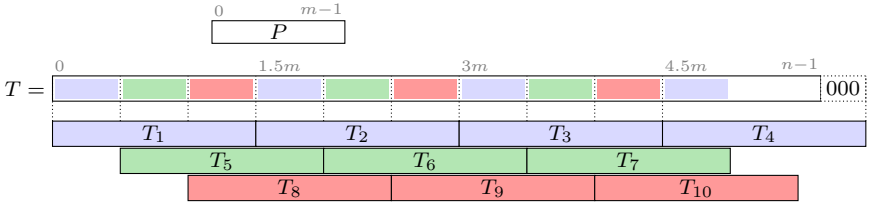


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

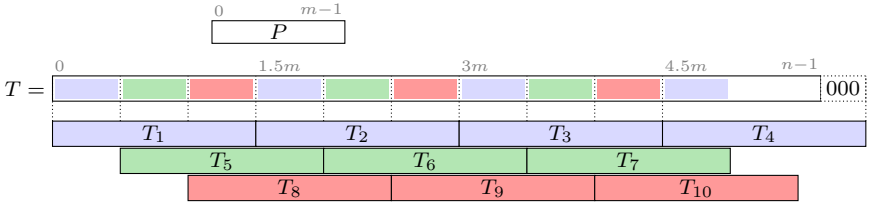


- cut T into blocks of size $1.5m$ (add padding if needed)

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$

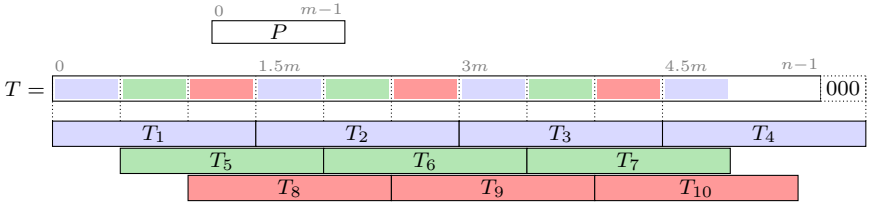


- cut T into blocks of size $1.5m$ (add padding if needed)
- let blocks start not only at multiples of $1.5m$, but at multiples of $0.5m$

A standard trick...

- search for CT matches with k substitutions, pattern of length $m := |P|$, text of length $n := |T|$
- complexity is function of $k, m, n...$ cumbersome in algorithm design and analysis

"standard trick": find matches of P in T $\implies$ find matches of P in $\mathcal{O}(n/m)$ texts, each of length $1.5 \cdot m$



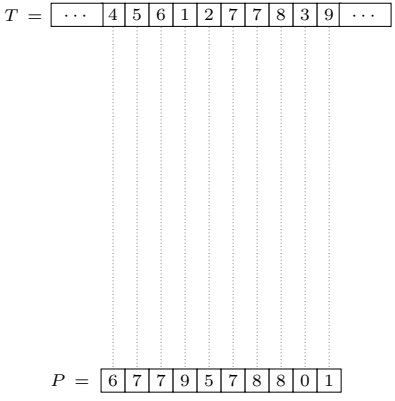
- cut T into blocks of size $1.5m$ (add padding if needed)
- let blocks start not only at multiples of $1.5m$, but at multiples of $0.5m$
- $\mathcal{O}(n/m)$ blocks, as promised!

Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P

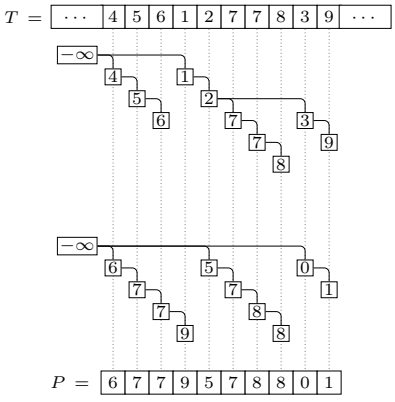
Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P



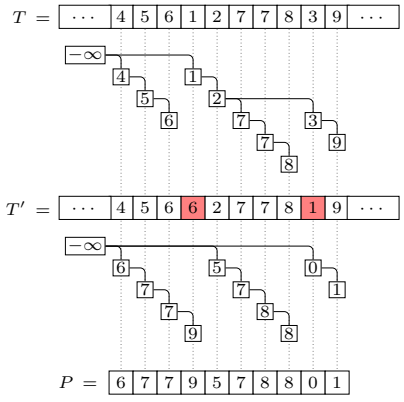
Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P



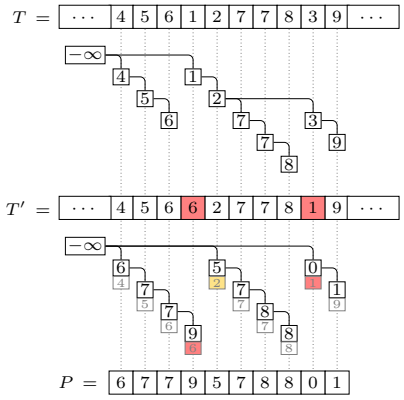
Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P



Cartesian tree matching with substitutions

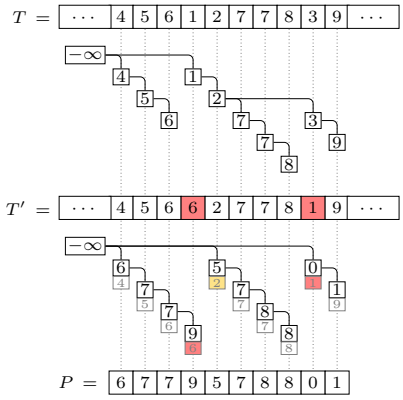
- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P



Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P

Fastest previous algorithm: $\mathcal{O}(m^2k)$ time
[Kim+Han '25].

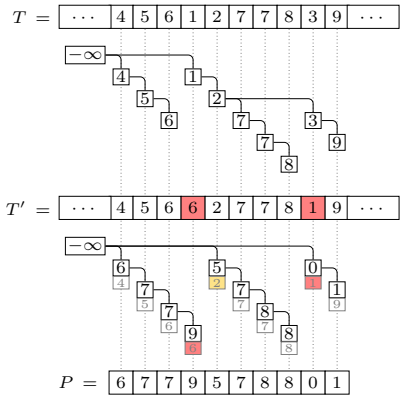


Cartesian tree matching with substitutions

- text $T[1\,..\,1.5m]$, pattern $P[1\,..\,m]$, threshold k
- output all fragments $T[i\,..\,i+m)$ at Hamming distance $\leq k$ to a string that has same CT as P

Fastest previous algorithm: $\mathcal{O}(m^2k)$ time
[Kim+Han '25].

New: $\mathcal{O}(m\sqrt{m}k^{2.5})$ time
[Charalampopoulos et al. '26].



Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

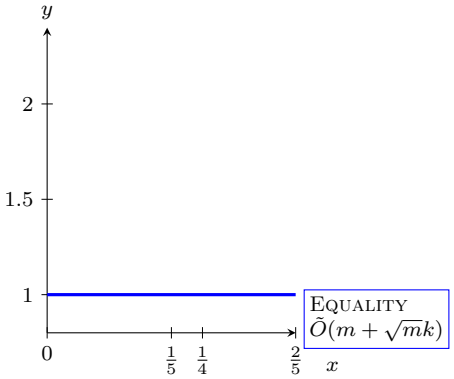
- not very well suited for finding trends in series data

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

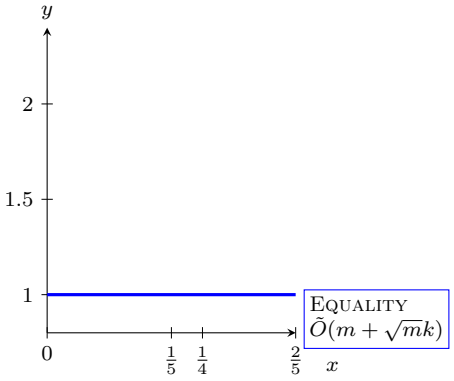
Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

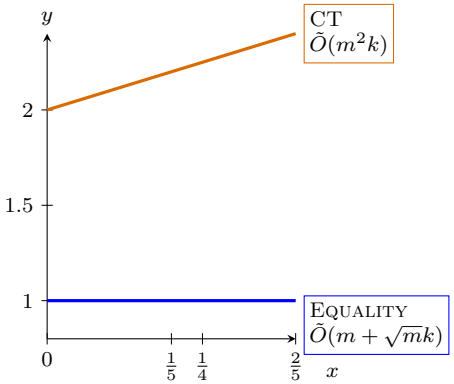
Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

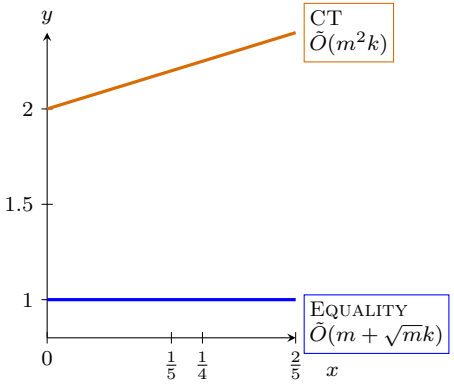
- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

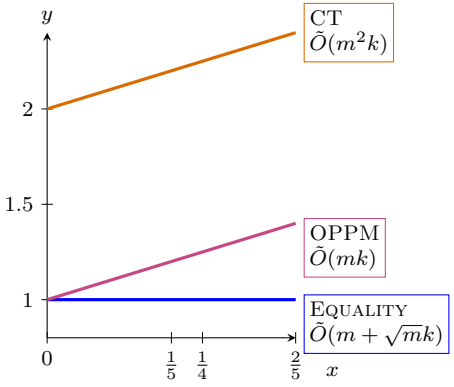
- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

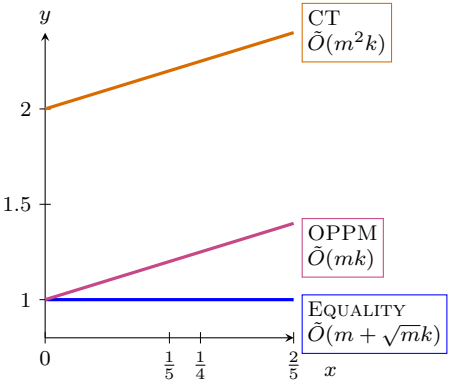
- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

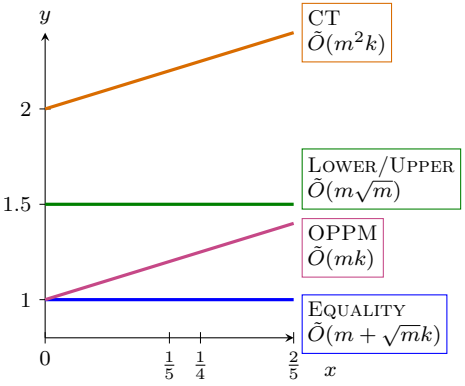
- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$



$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

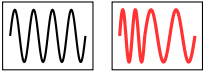
- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

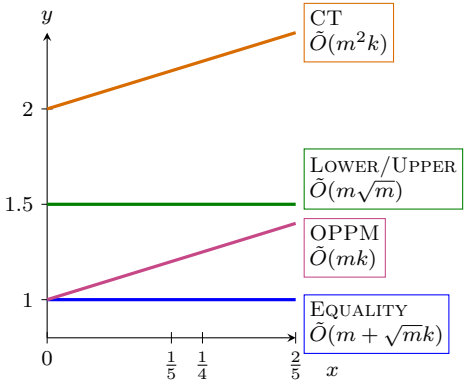
Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$

Dynamic time warping

- non-uniformly stretch and/or squeeze pattern to maximize similarity with text fragment





$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

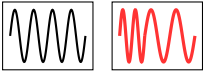
- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

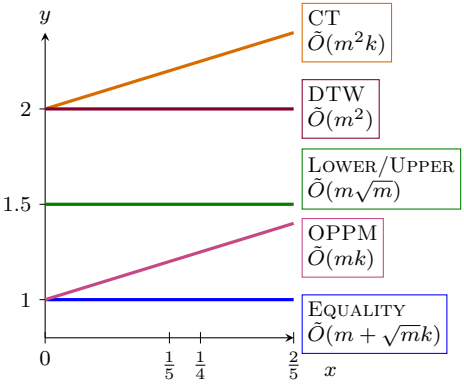
Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$

Dynamic time warping

- non-uniformly stretch and/or squeeze pattern to maximize similarity with text fragment





$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

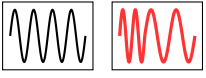
- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

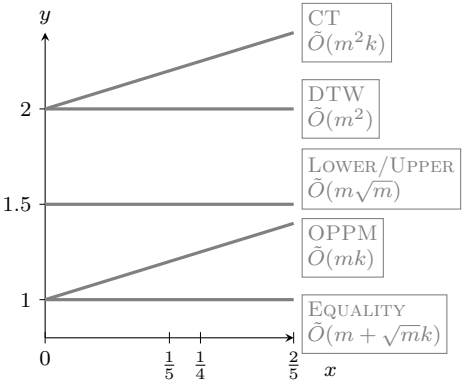
Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$

Dynamic time warping

- non-uniformly stretch and/or squeeze pattern to maximize similarity with text fragment





$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Some other flavors of this kind of pattern matching

Equality matching (with k mismatches)

- not very well suited for finding trends in series data

Cartesian tree matching (with k mismatches)

- match iff CT of P and $T[i\,..\,i+m)$ is the same

Order preserving matching (with k mismatches)

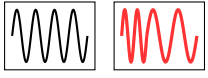
- match iff relative order of symbols in P and $T[i\,..\,i+m)$ is the same

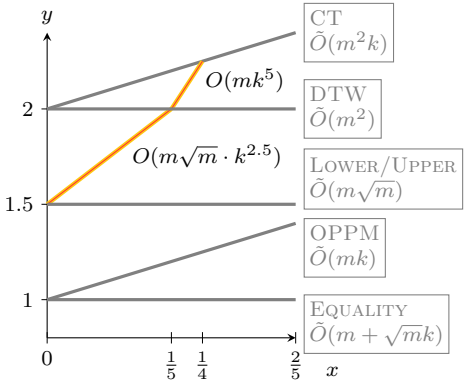
Matching with lower and upper bounds P_ℓ and P_u

- match iff $T[j] \in [P_\ell[j], P_u[j]]$ for $j \in T[i\,..\,i+m)$

Dynamic time warping

- non-uniformly stretch and/or squeeze pattern to maximize similarity with text fragment





$k = m^x,\ \text{time} = \tilde{\mathcal{O}}(m^y)$

[Gawrychowski+Uznanski '18; Kim+Han '25; Gawrychowski+Uznanski '16; Labib et al. '19; Boneh et al. '24]

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{mk})$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{mk}^{2.5})$ Cartesian tree matching. But why?

- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{mk})$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{mk}^{2.5})$ Cartesian tree matching. But why?

- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

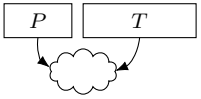
query for equality matching:
compute $\mathsf{hd}(P, T[i\,..\,i+m))$
up to threshold k

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

query for equality matching:
compute $\mathrm{hd}(P, T[i\,..\,i+m))$
up to threshold k



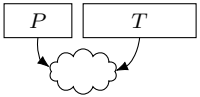
data structure
with $\mathcal{O}(m)$ constr. time
and $\mathcal{O}(k)$ query time

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

query for equality matching:
compute hd$(P, T[i\,..\,i+m))$
up to threshold k



data structure
with $\mathcal{O}(m)$ constr. time
and $\mathcal{O}(k)$ query time

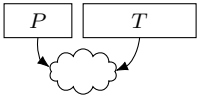
query for CT matching:
compute CT-hd$(P, T[i\,..\,i+m))$
up to threshold k

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

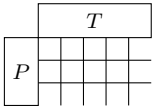
- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

query for equality matching:
compute hd$(P, T[i\,..\,i+m))$
up to threshold k



data structure
with $\mathcal{O}(m)$ constr. time
and $\mathcal{O}(k)$ query time

query for CT matching:
compute CT-hd$(P, T[i\,..\,i+m))$
up to threshold k



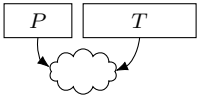
optimized DP in $\mathcal{O}(mk)$ time

Much slower than equality matching...

... $\tilde{\mathcal{O}}(m + \sqrt{m}k)$ equality matching vs $\tilde{\mathcal{O}}(m\sqrt{m}k^{2.5})$ Cartesian tree matching. But why?

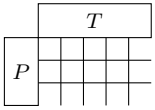
- equality matching can use FFT for small alphabets: $\tilde{\mathcal{O}}(m\,|\Sigma|)$ for pattern over alphabet Σ
- vastly different verifiers for occurrences

query for equality matching:
compute hd$(P, T[i\,..\,i+m))$
up to threshold k



data structure
with $\mathcal{O}(m)$ constr. time
and $\mathcal{O}(k)$ query time

query for CT matching:
compute CT-hd$(P, T[i\,..\,i+m))$
up to threshold k



optimized DP in $\mathcal{O}(mk)$ time

- algorithms for equality matching heavily exploit periodicity

Sometimes we can reduce to exact CT matching

Sometimes we can reduce to exact CT matching

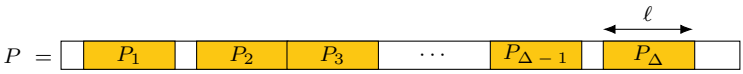
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P

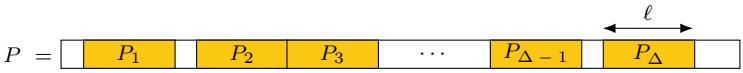
Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$



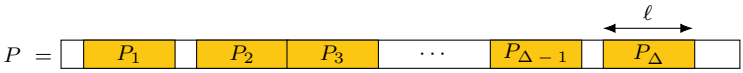
Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



Sometimes we can reduce to exact CT matching

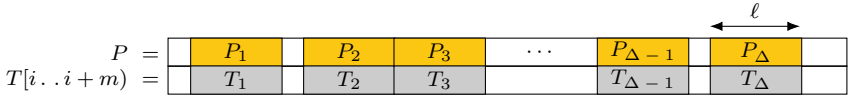
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- if $T[i\,..\,i+m)$ and P have same CT, then also P_i and T_i have same CT

Sometimes we can reduce to exact CT matching

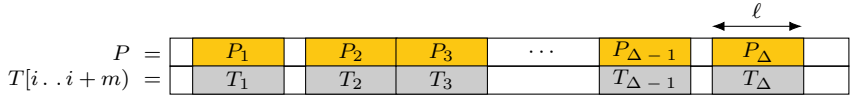
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- if $T[i \,..\, i+m)$ and P have same CT, then also P_i and T_i have same CT

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT

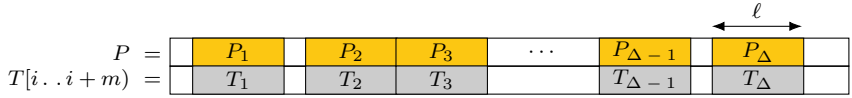


$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i \,..\, i+m)$ and P have same CT, then also P_i and T_i have same CT

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT

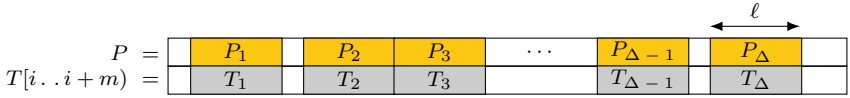


$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i\,..\,i+m)$ and P have same CT, then also P_i and T_i have same CT
- if $T[i\,..\,i+m)$ and P have same CT after k substitutions in T
  then P_i and T_i have same CT for at least $\Delta - k \geq \Delta/2$ choices of i

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



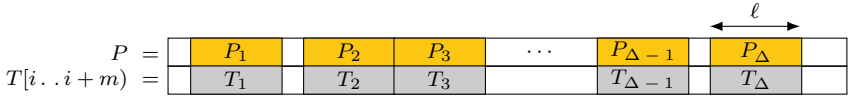
$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i\,..\,i+m)$ and P have same CT, then also P_i and T_i have same CT
- if $T[i\,..\,i+m)$ and P have same CT after k substitutions in T
  then P_i and T_i have same CT for at least $\Delta - k \geq \Delta/2$ choices of i

- use exact CT matching to find all occurrences of all P_i in T in $\tilde{\mathcal{O}}(m\Delta)$ time

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



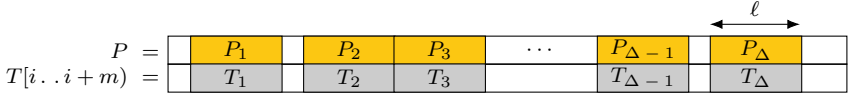
$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i\,..\,i+m)$ and P have same CT, then also P_i and T_i have same CT
- if $T[i\,..\,i+m)$ and P have same CT after k substitutions in T
  then P_i and T_i have same CT for at least $\Delta - k \geq \Delta/2$ choices of i

- use exact CT matching to find all occurrences of all P_i in T in $\tilde{\mathcal{O}}(m\Delta)$ time
- at most $\mathcal{O}(m)$ occurrences in total, at most $\mathcal{O}(m/\Delta)$ text fragments have enough matches

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



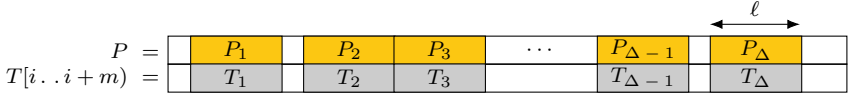
$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i\,..\,i+m)$ and P have same CT, then also P_i and T_i have same CT
- if $T[i\,..\,i+m)$ and P have same CT after k substitutions in T
  then P_i and T_i have same CT for at least $\Delta - k \geq \Delta/2$ choices of i

- use exact CT matching to find all occurrences of all P_i in T in $\tilde{\mathcal{O}}(m\Delta)$ time
- at most $\mathcal{O}(m)$ occurrences in total, at most $\mathcal{O}(m/\Delta)$ text fragments have enough matches
- verify each in $\mathcal{O}(mk)$ time

Sometimes we can reduce to exact CT matching

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



$CT(P_3) \neq CT(T_3)$? At least one substitution in T_3!

- if $T[i \,..\, i+m)$ and P have same CT, then also P_i and T_i have same CT
- if $T[i \,..\, i+m)$ and P have same CT after k substitutions in T
  then P_i and T_i have same CT for at least $\Delta - k \geq \Delta/2$ choices of i

- use exact CT matching to find all occurrences of all P_i in T in $\tilde{\mathcal{O}}(m\Delta)$ time
- at most $\mathcal{O}(m)$ occurrences in total, at most $\mathcal{O}(m/\Delta)$ text fragments have enough matches
- verify each in $\mathcal{O}(mk)$ time
- overall time $\tilde{\mathcal{O}}(m^2k/\Delta + m\Delta) = \tilde{\mathcal{O}}(m\sqrt{m} \cdot \text{poly}(k))$ using $\Delta = \Theta(\sqrt{m}/\text{poly}(k))$

What if we can't find enough anchors?

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT

What if we can't find enough anchors?

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT

- find as many anchors as possible

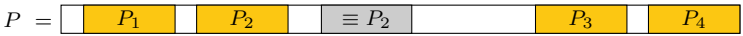
What if we can't find enough anchors?

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT

- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

What if we can't find enough anchors?

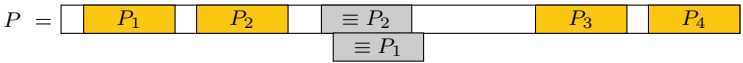
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

What if we can't find enough anchors?

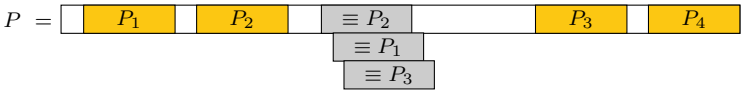
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

What if we can't find enough anchors?

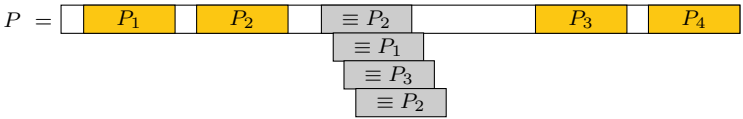
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

What if we can't find enough anchors?

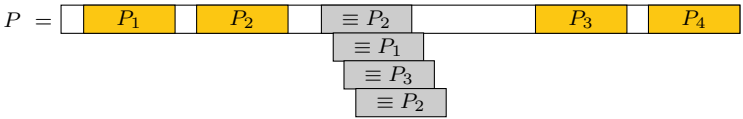
- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

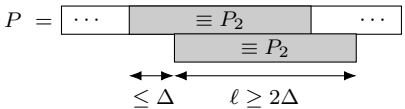
What if we can't find enough anchors?

- fix some parameter $\Delta \in [2k, \sqrt{m}/2]$ to define some anchor substrings of P
  - Δ non-overlapping fragments of P
  - each of length $\ell \approx m/(2\Delta)$
  - no two of them have the same CT



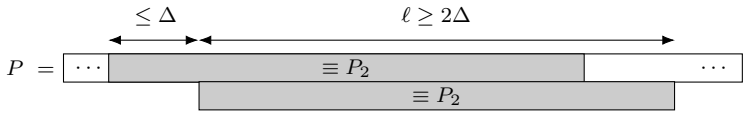
- find as many anchors as possible
- there must some lengthy fragment of P with no anchors

- zoom in...



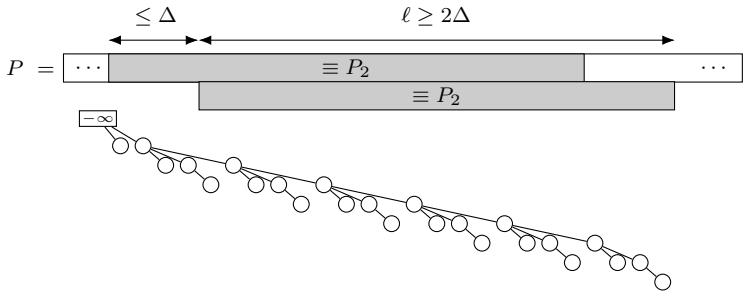
What if we can't find enough anchors?

- zoom in even further...



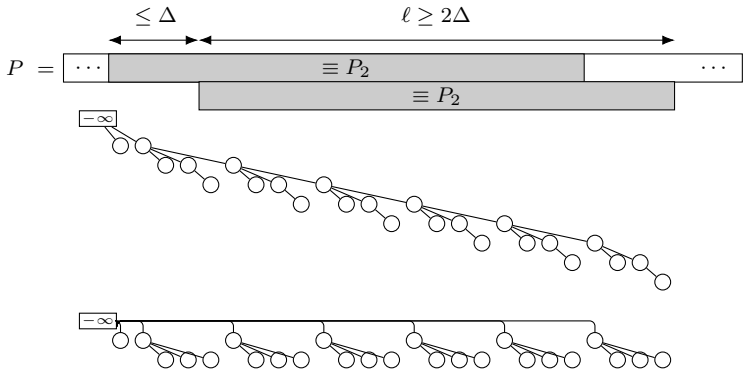
What if we can't find enough anchors?

- zoom in even further...



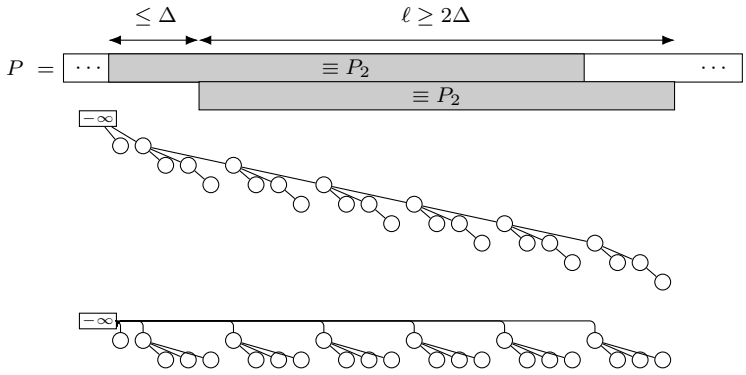
What if we can't find enough anchors?

- zoom in even further...



What if we can't find enough anchors?

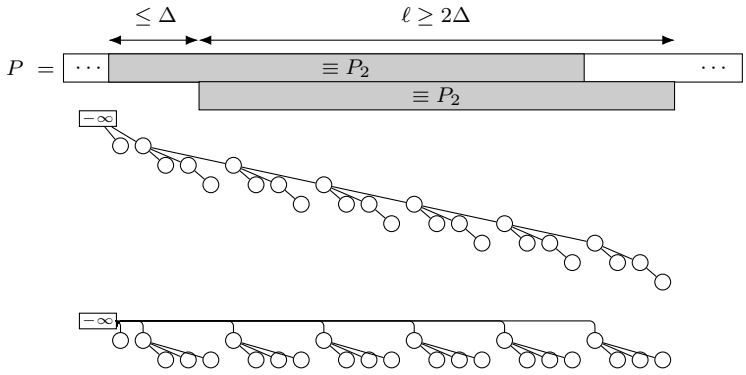
- zoom in even further...



- highly periodic with respect to structure of CT

What if we can't find enough anchors?

- zoom in even further...



- highly periodic with respect to structure of CT
- adapt all the tricks for periodic fragments from equality pattern matching

Conclusion

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching
- or find and exploit periodicity in pattern

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching
- or find and exploit periodicity in pattern

Open questions:

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching
- or find and exploit periodicity in pattern

Open questions:

- What is the correct complexity?

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching
- or find and exploit periodicity in pattern

Open questions:

- What is the correct complexity?
- Is there a faster verifier?

Conclusion

- CT matching with k mismatches in $\mathcal{O}(m\sqrt{m} \cdot k^{2.5})$ time
- either reduce to exact CT matching
- or find and exploit periodicity in pattern

Open questions:

- What is the correct complexity?
- Is there a faster verifier?
- CT/OPPM/DTW... how do they actually work on real data?

References


Boneh, Itai et al. (2024). "Optimal Dynamic Time Warping on Run-Length Encoded Strings". In: *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, Tallinn, Estonia, July 8-12, 2024*. Vol. 297. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:17. DOI: 10.4230/LIPICS.ICALP.2024.30.


Charalampopoulos, Panagiotis, Jonas Ellert, and Manal Mohamed (2026). "Approximate Cartesian Tree Matching with Substitutions". In: *43rd International Symposium on Theoretical Aspects of Computer Science, STACS 2026, Grenoble, France, March 9-13, 2026*. Vol. 364. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 26:1–26:21. DOI: 10.4230/LIPICS.STACS.2026.26.


Gawrychowski, Pawel and Przemyslaw Uznanski (2016). "Order-preserving pattern matching with k mismatches". In: *Theor. Comput. Sci.* 638, pp. 136–144. DOI: 10.1016/J.TCS.2015.08.022.


— (2018). "Towards Unified Approximate Pattern Matching for Hamming and L_1 Distance". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, July 9-13, 2018*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 62:1–62:13. DOI: 10.4230/LIPICS.ICALP.2018.62.


Kim, Sungmin and Yo-Sub Han (2025). "Approximate Cartesian tree pattern matching". In: *Theor. Comput. Sci.* 1056, Article 115506. DOI: 10.1016/J.TCS.2025.115506.


Labib, Karim, Przemyslaw Uznanski, and Daniel Wolleb-Graf (2019). "Hamming Distance Completeness". In: *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, Pisa, Italy, June 18-20, 2019*. Vol. 128. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:17. DOI: 10.4230/LIPICS.CPM.2019.14.