Parameterized Matching: Solutions & Extensions

Juan Mendivelso[1] & Yoan Pinzón[2]

[1] Fundación Universitaria Konrad Lorenz
[2] Universidad Nacional de Colombia

Prague Stringology Conference
2015

Outline

Background

String Comparison

- *X[1..m]* and *Y[1..m]* match if *X[i]* = *Y[i]* for all *i*.

X	a	b	a	c	a	b	a	c	a
Y	a	b	a	c	a	b	a	c	a

String Pattern Matching

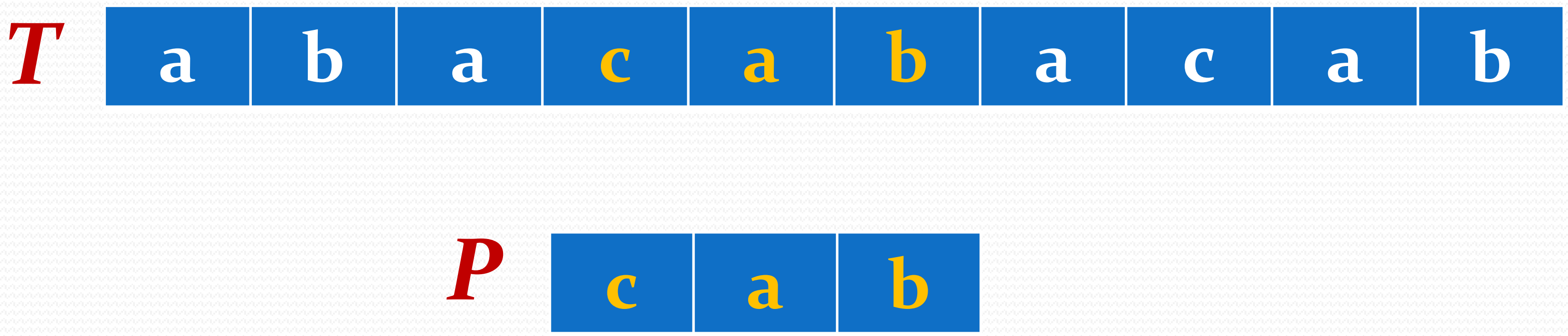
- Find the matches of a pattern *P[1..m]* within a string *T[1..n]*.

T	a	b	a	c	a	b	a	c	a	b

P	c	a	b

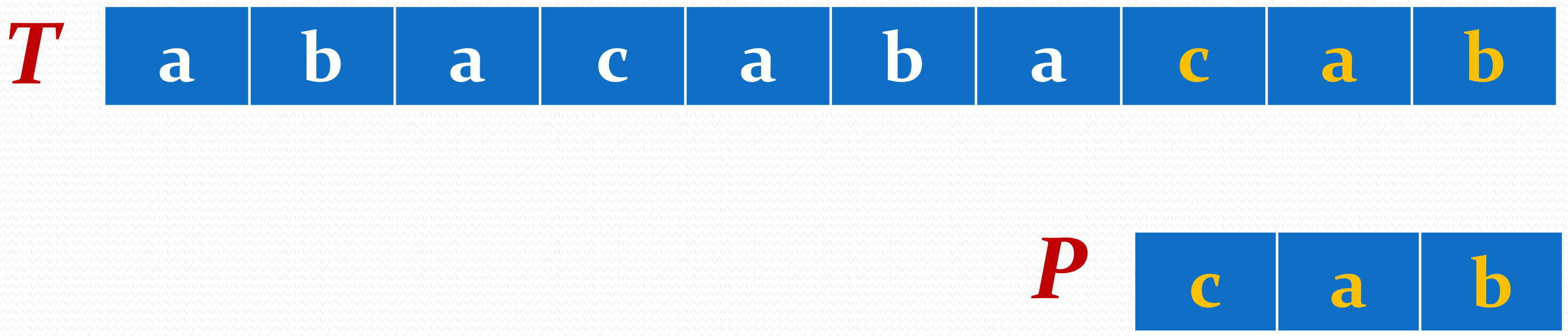
String Pattern Matching

- Find the matches of a pattern *P[1..m]* within a string *T[1..n]*.



String Pattern Matching

- Find the matches of a pattern *P[1..m]* within a string *T[1..n]*.



Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Extensions
- Applications
- Conclusions

Motivation for Parameterized Matching

- Software Maintenance Application
- Definition of Parameterized-Match (p-match)

Motivation for Parameterized Matching

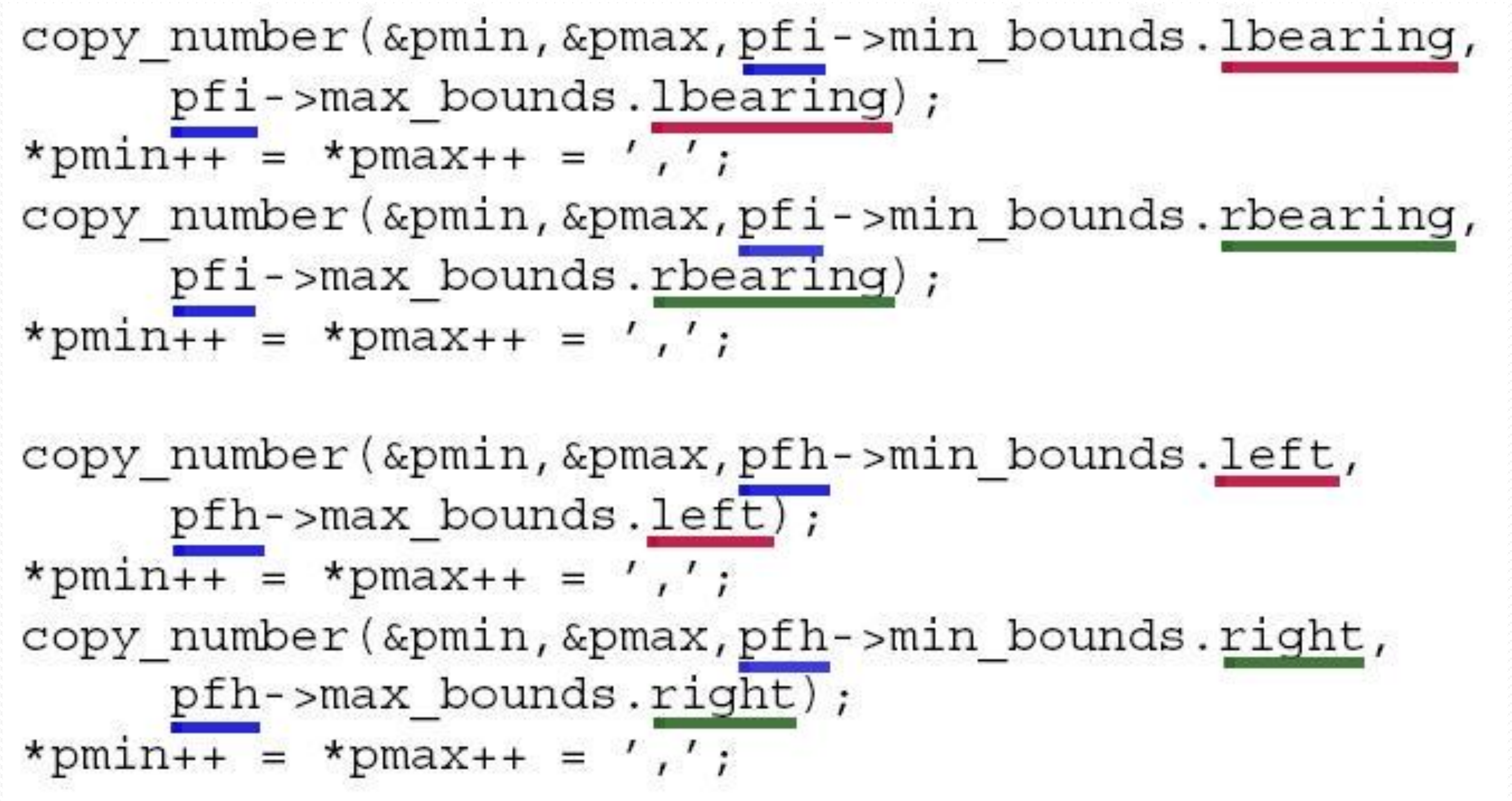
- Software Maintenance Application
- Definition of Parameterized-Match (p-match)

Software Maintenance Application

- Programmers introduce duplicate code in large software systems when they introduce new features or fix bugs.
- They copy and slightly modify the code to avoid the introduction of new bugs.
- The code can be seen as a sequence of tokens.
- Duplicate code can have tokens that remains the same and tokens that systematically change.

Finding duplicate code

- Baker developed interest in solving this problem.



```
copy_number(&pmin,&pmax,pfi->min_bounds.lbearing,
        pfi->max_bounds.lbearing);
*pmin++ = *pmax++ = ',';
copy_number(&pmin,&pmax,pfi->min_bounds.rbearing,
        pfi->max_bounds.rbearing);
*pmin++ = *pmax++ = ',';

copy_number(&pmin,&pmax,pfh->min_bounds.left,
        pfh->max_bounds.left);
*pmin++ = *pmax++ = ',';
copy_number(&pmin,&pmax,pfh->min_bounds.right,
        pfh->max_bounds.right);
*pmin++ = *pmax++ = ',';
```

Figure: [Baker, 1992]

Importance of the problem

- Code gets larger, more complex and more difficult to maintain.
- Fixing a new issue in one of the copies does not fix it in the other (unmonitored) copies.
- Experiments show that 22% of code may be duplicate [Baker, 1992].
- Finding such code can help using better programming techniques to eliminate duplication.

Motivation for Parameterized Matching

- Software Maintenance Application
- Definition of Parameterized-Match (p-match)

Then, Baker defined…

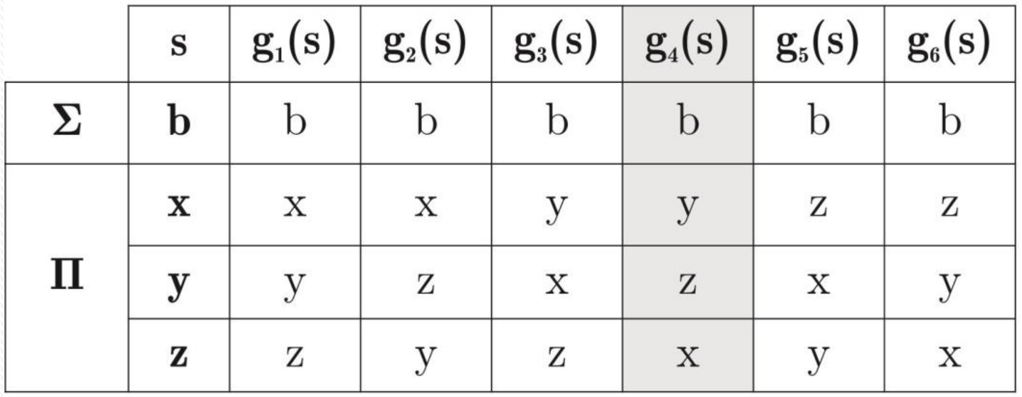
- Constant Alphabet *(∑)*
- Paramater Alphabet *(Π)*
- Parameterized-strings: defined over *(∑ U Π)*

$\Sigma = \{b\}$ $\Pi = \{x, y, z\}$

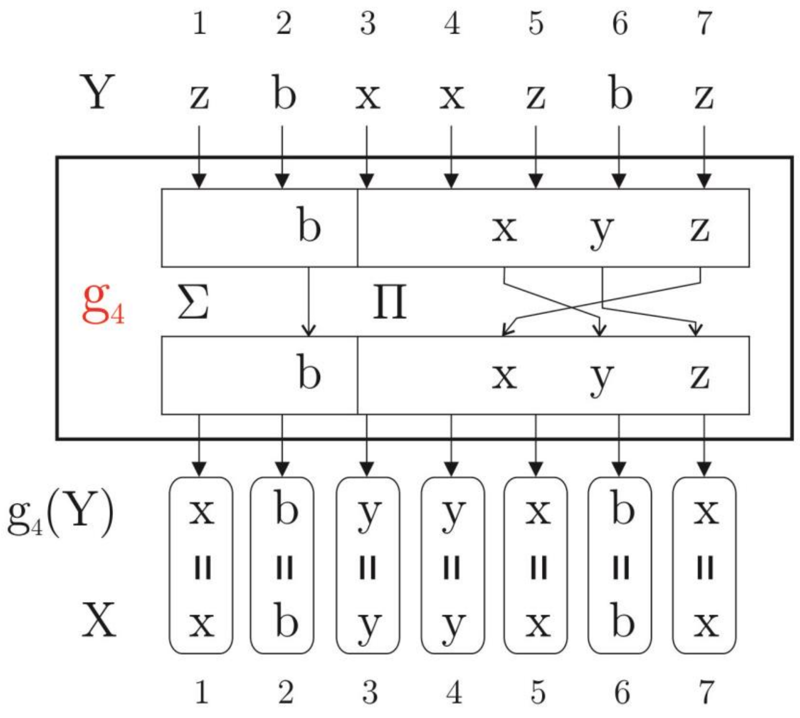
X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Parameterized-match (p-match)

- P-strings *X[1..m]* and *Y[1..m]* are a p-match if one can be mapped into the other through a bijection such that the mapping is identity for the symbols in ∑.

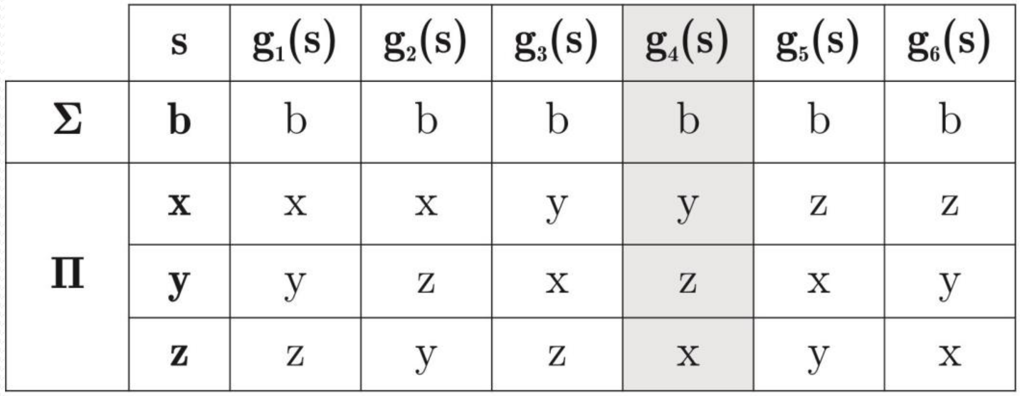


	s	$g_1(s)$	$g_2(s)$	$g_3(s)$	$g_4(s)$	$g_5(s)$	$g_6(s)$
Σ	**b**	b	b	b	b	b	b
Π	**x**	x	x	y	y	z	z
	y	y	z	x	z	x	y
	z	z	y	z	x	y	x

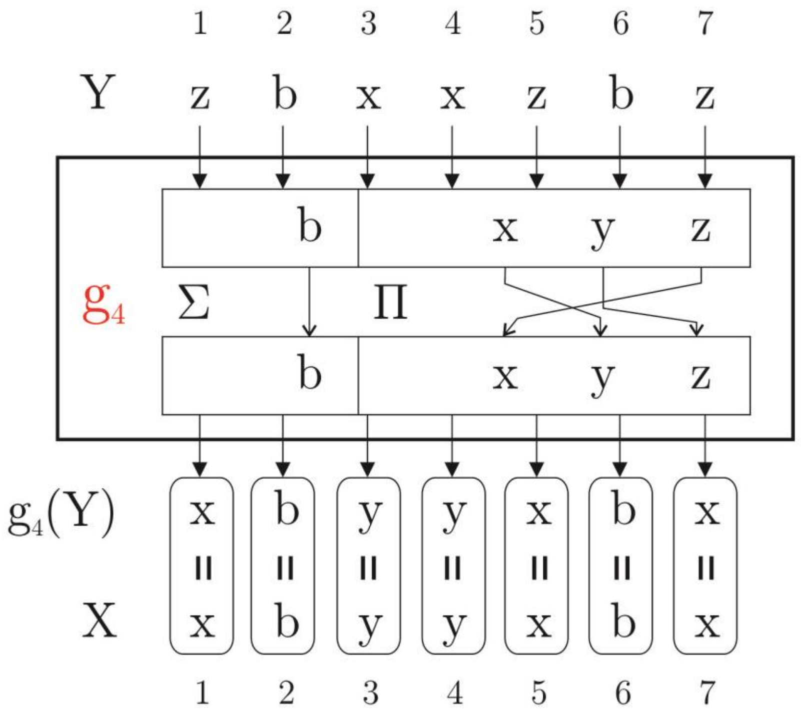


Parameterized-match (p-match)

- There are | ∏ |! possible bijections which makes parameterized matching an interesting combinatorial problem.

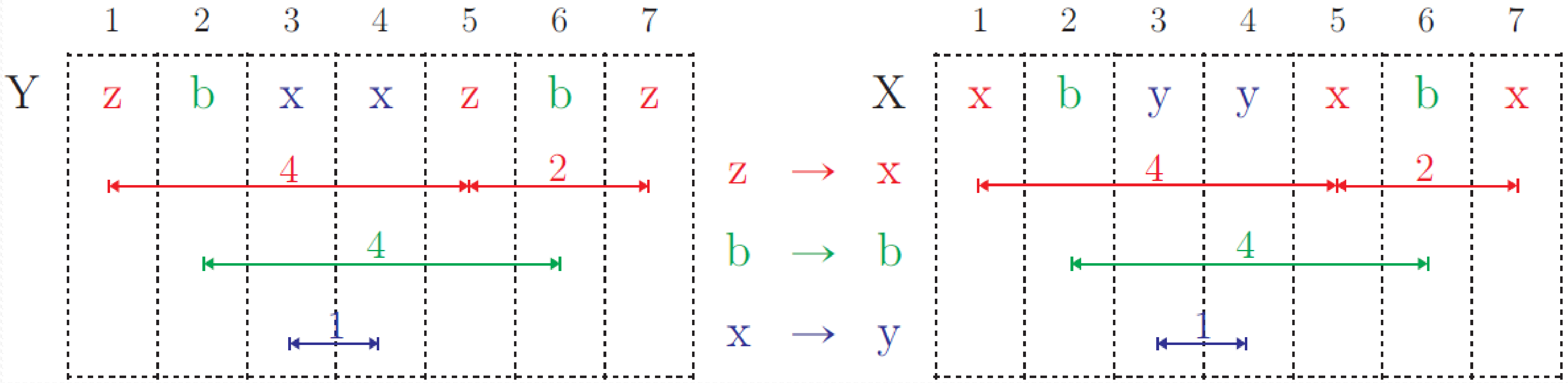


	s	$g_1(s)$	$g_2(s)$	$g_3(s)$	$g_4(s)$	$g_5(s)$	$g_6(s)$
Σ	**b**	b	b	b	b	b	b
Π	**x**	x	x	y	y	z	z
	y	y	z	x	z	x	y
	z	z	y	z	x	y	x



Similarity in structure

- Two p-strings that p-match…
  - … have the same number of distinct symbols.
  - … the occurrences of each distinct symbol take place in corresponding positions.



Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Extensions
- Applications
- Conclusions

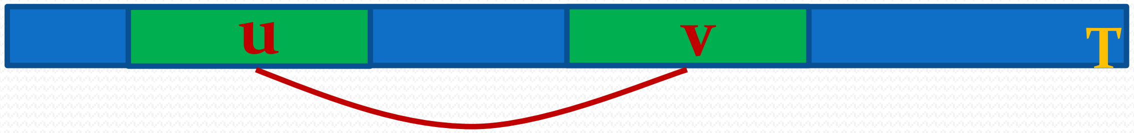
Basic Problems

- Maximal p-matches over a Threshold Length
- Parameterized Pattern Matching
- Parameterized Fixed Multiple Pattern Matching
- Parameterized Dynamic Dictionary Matching

Basic Problems

- **Maximal p-matches over a threshold length:**
  - **Input:** *T, k*

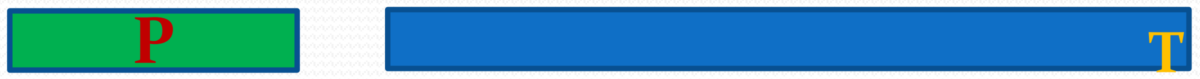
  - **Output:** pairs *(u,v)* of maximal parameterized matching substrings such that $|u| \geq k$.



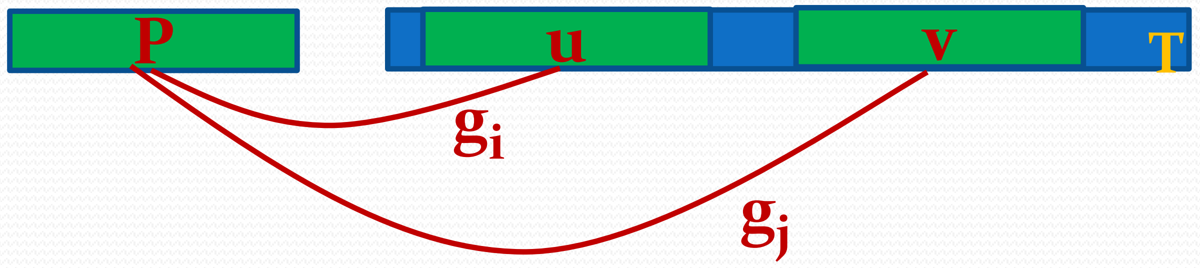
  - **Complexity:** *O(n+occ)* [Baker, 1997]

Basic Problems

- **Parameterized Fixed Pattern Matching:**
  - **Input:** *T[1..n], P[1..m]*



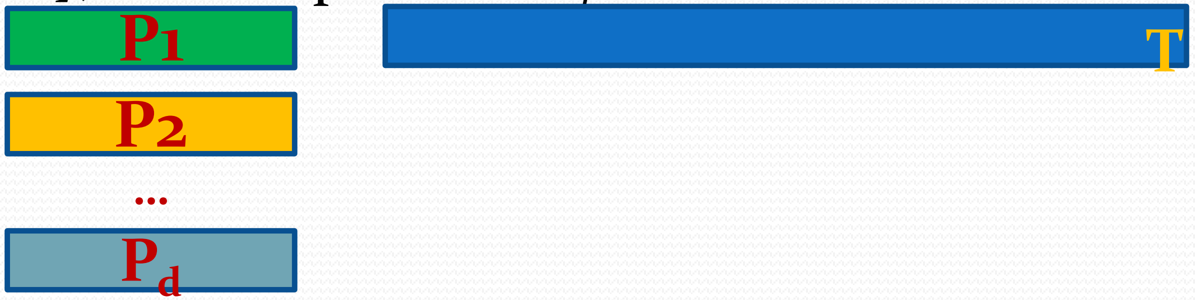
  - **Output:** substrings in *T* that parameterized-match *P*



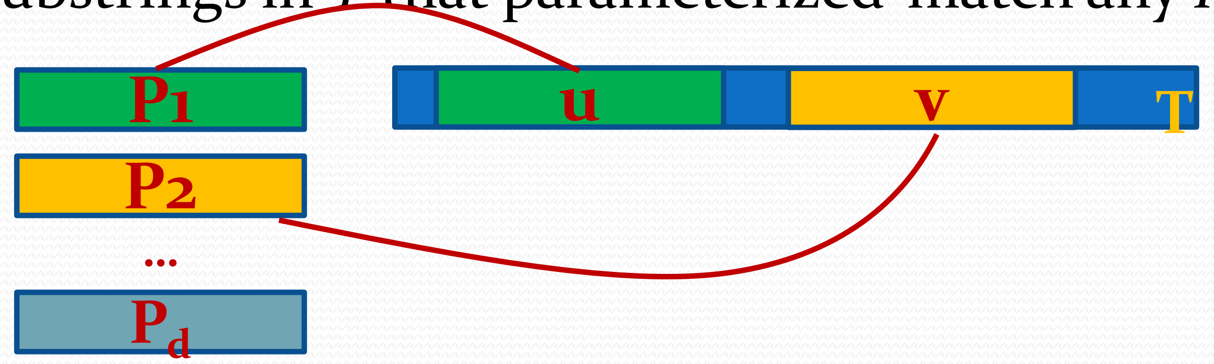
  - **Complexity:** *O(n log min (m,|Π|))* [Amir, 1994]

Basic Problems

- **Parameterized Fixed Multiple Pattern Matching:**
  - **Input:** $T[1..n]$, set of d patterns P_i



  - **Output:** substrings in T that parameterized-match any P_i



  - **Complexity:** $O(n \log |\Sigma| + occ)$ [Idury, 1996]

Basic Problems

- **Parameterized Dynamic Dictionary Matching:**
  - The same as Parameterized Fixed Pattern Matching, but new patterns can be inserted or removed from the set.
  - Complexity: $O((n+occ)(\log|\Sigma|+\log d))$ [Idury, 1996].
- Literature on parameterized matching includes solutions for all of these problems, as presented in next section.

Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Extensions
- Applications
- Conclusions

Solutions

Baker’s theory

Generalization of Exact Matching Algorithms

Solutions

Baker’s theory

Generalization of Exact Matching Algorithms

Baker's Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

Baker's Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

DUP Algorithm

- Proposed by [Baker, 1992].
- To find maximal p-matches over a threshold length.
- It works as follows:
  - Converts the parameters in a single symbol.
  - Looks for exact matches using a suffix tree.
  - Determines which of such matches are p-matches.
- Experimental results show that just few exact matches are p-matches.

Baker's Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

α	f(α)

$\Sigma = \{b\}$ $\Pi = \{x,y,z\}$

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

α	f(α)
x	z

$\Sigma = \{b\}$ $\Pi = \{x,y,z\}$

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

α	f(α)
x	z

$\Sigma = \{b\}$ $\Pi = \{x,y,z\}$

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

$\Sigma = \{b\}$ $\qquad$ $\Pi = \{x, y, z\}$

α	f(α)
x	z
y	x

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

α	f(α)
x	z
y	x

$\Sigma = \{b\}$ $\Pi = \{x, y, z\}$

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.

α	f(α)
x	z
y	x

$\Sigma = \{b\}$ $\Pi = \{x,y,z\}$

X	x	b	y	y	x	b	x
Y	z	b	x	x	z	b	z

Solution for String Comparison

- Straightforward solution [Baker, 1997].
- Construct a mapping table of the mapping while simultaneously traversing both strings until a mismatch is found.
- Time Complexity: $O(m)$.
- Space Complexity: $O(|\Pi|)$.

Baker’s Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

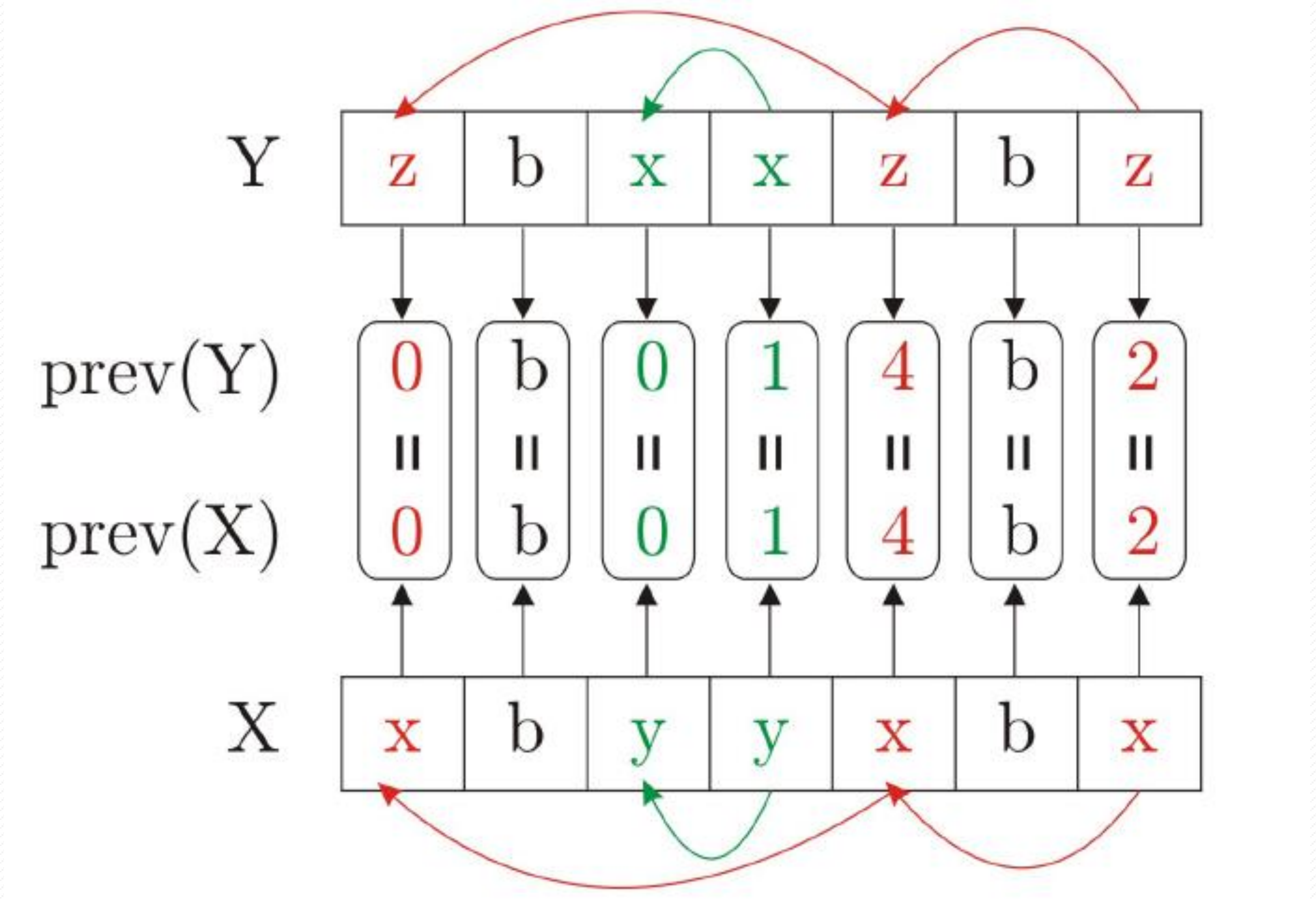
Maximal p-matches over a threshold length

Procedure *prev*

- Proposed by [Baker, 1997].
- Array encoding of a p-string *X[1..m]* where:
  - Every symbol in ∑ remains the same.
  - The first occurrence of each parameter becomes 0.
  - The other occurrences of each parameter becomes the distance to its previous occurrence (**parameter pointers**).
- It focuses on the string structure.

Procedure *prev*

- Then, *X* and *Y* are a p-match iff *prev(X)* = *prev(Y)*:



Complexity of Computing *prev*

- Time complexity: $O(m)$.
- Space complexity: $O(|\Pi|)$.
- String comparison using prev: $O(m)$.

Computing prev of a substring

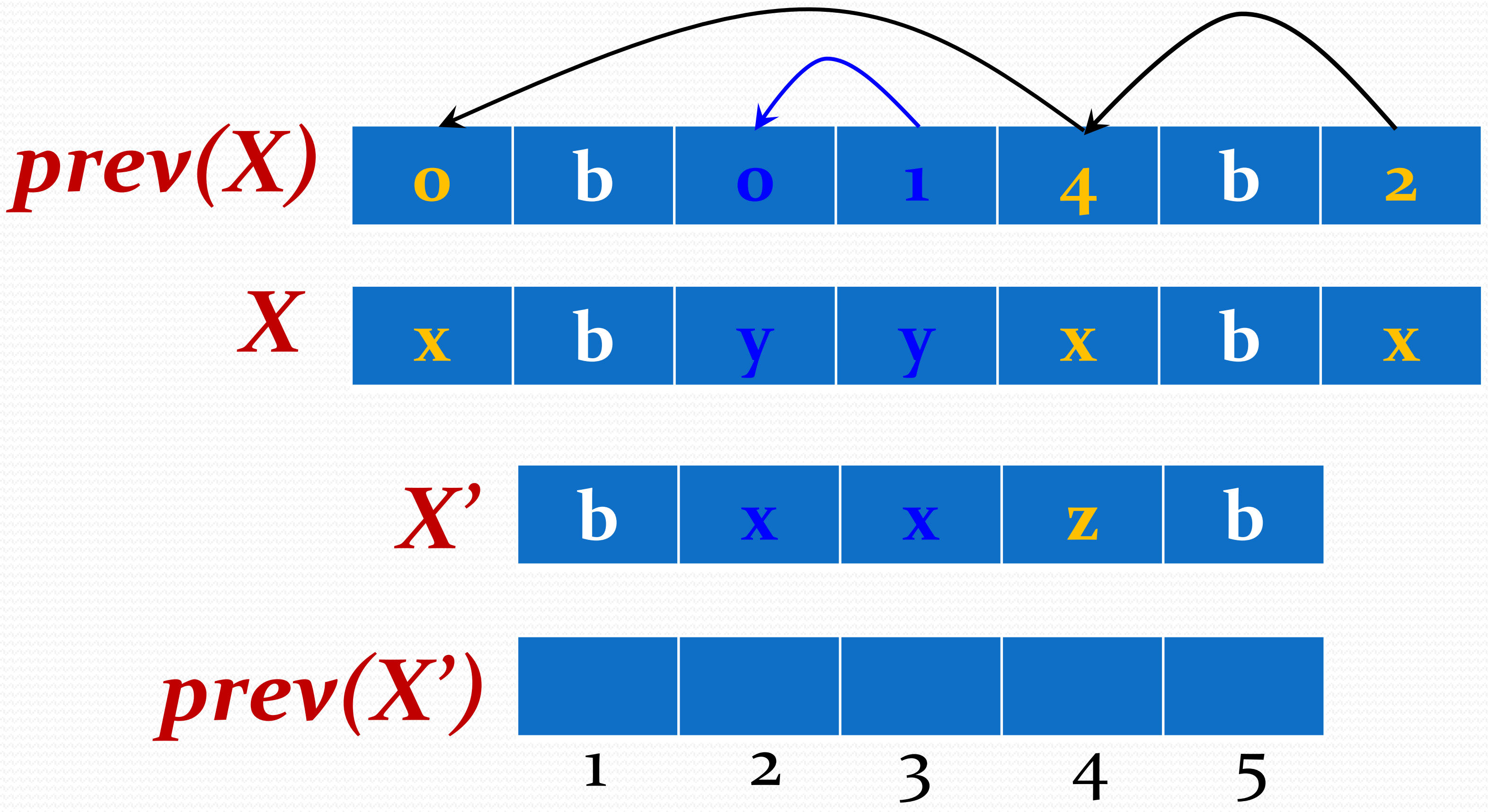
- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.
- Specifically,

$$prev(X[i..j])_k = \begin{cases} 0 & \text{if } prev(X[1..m])_{i+k-1} > k-1 \\ prev(X[1..m])_{i+k-1} & \text{otherwise} \end{cases}$$

- Essentially, this means that a parameter pointer becomes zero when it points outside of the substring.

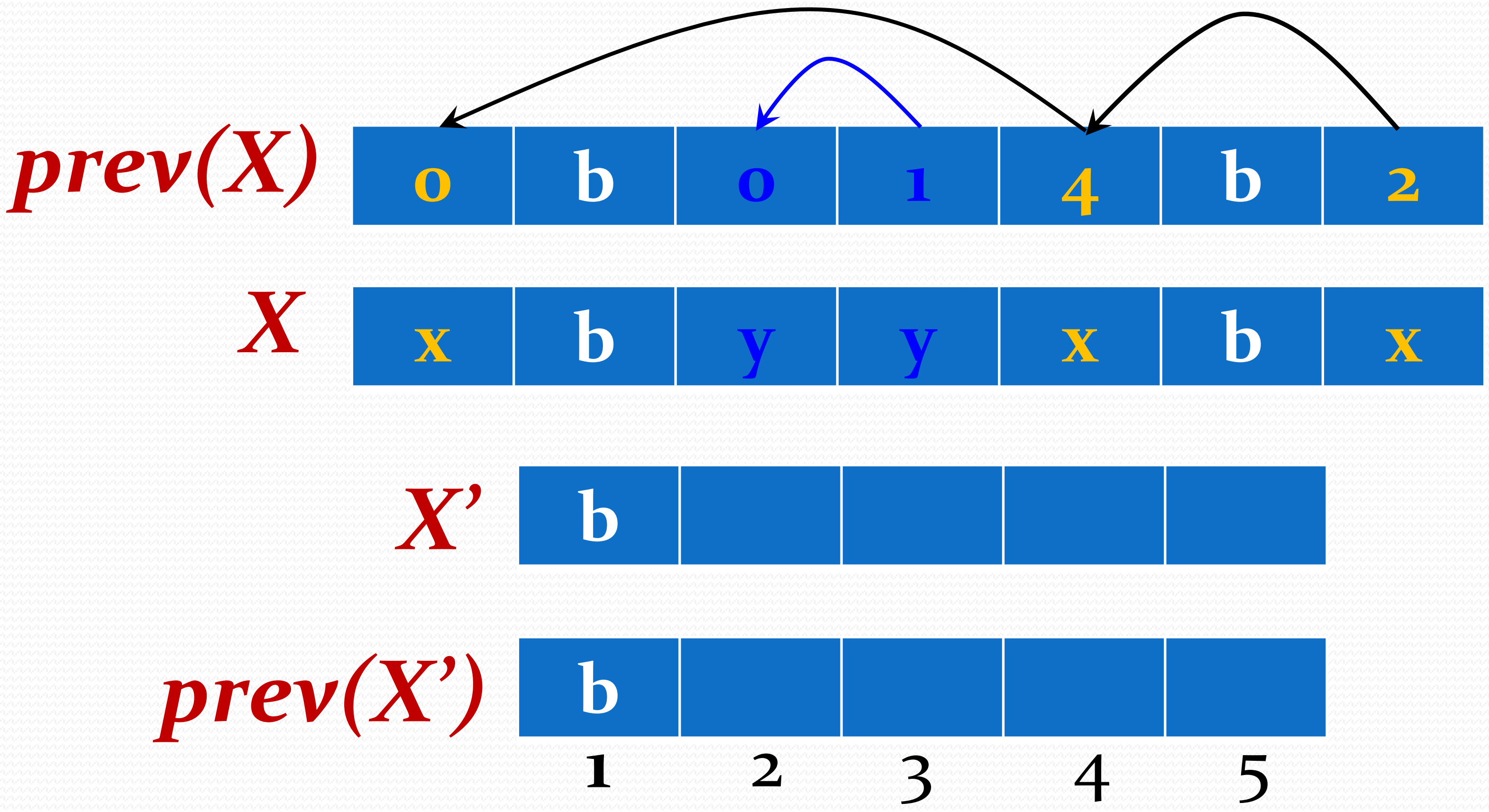
Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



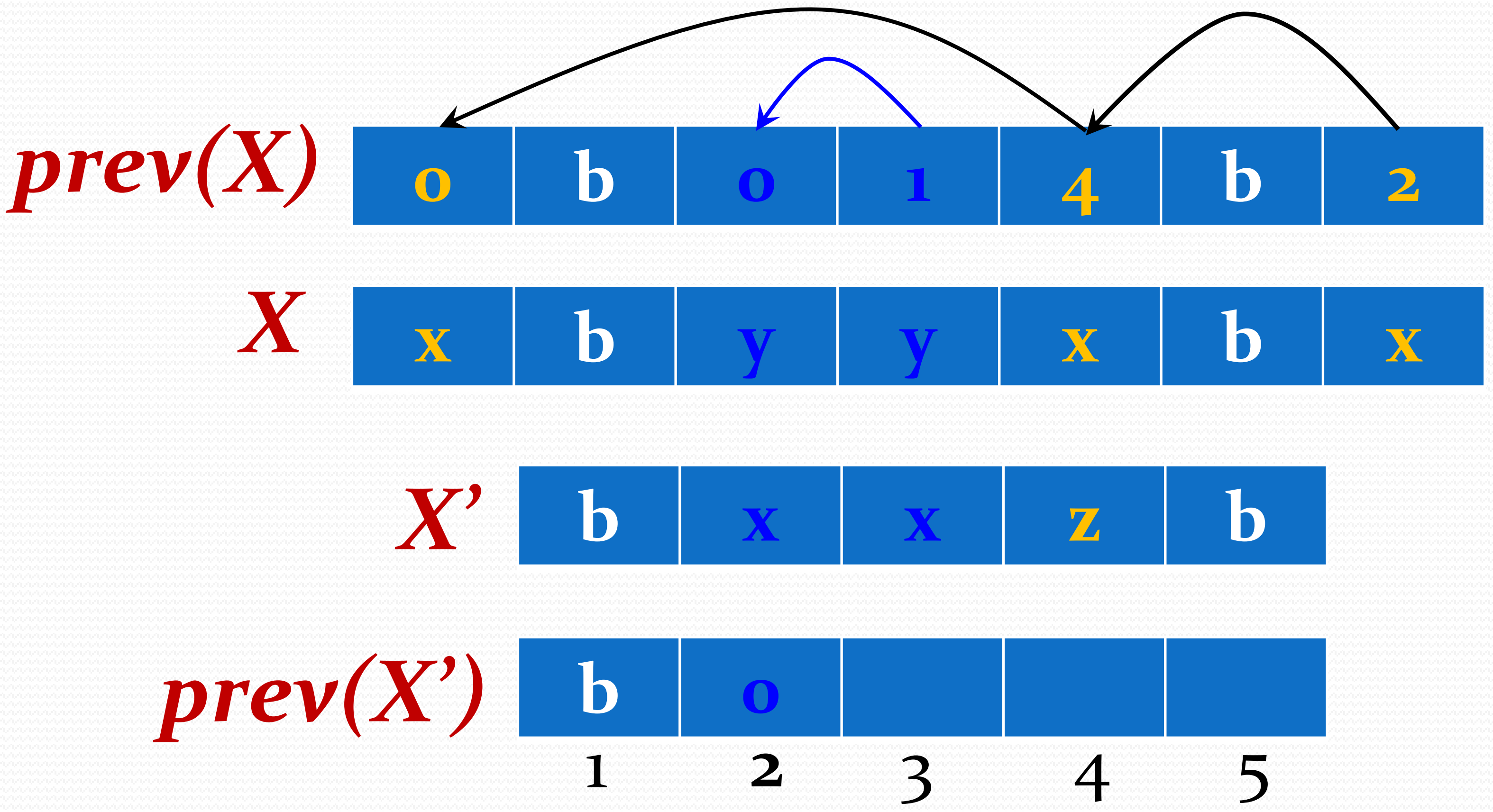
Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



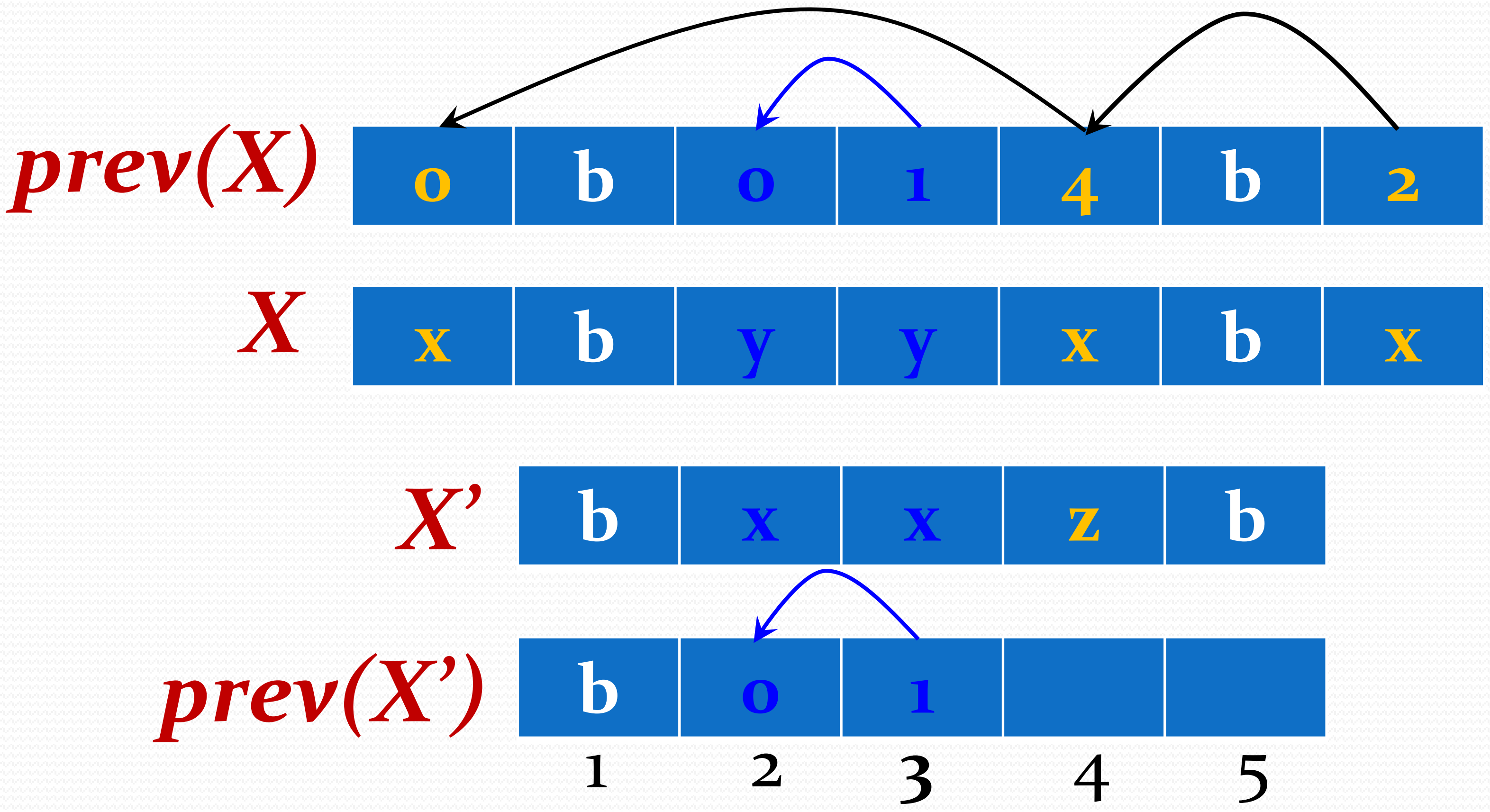
Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



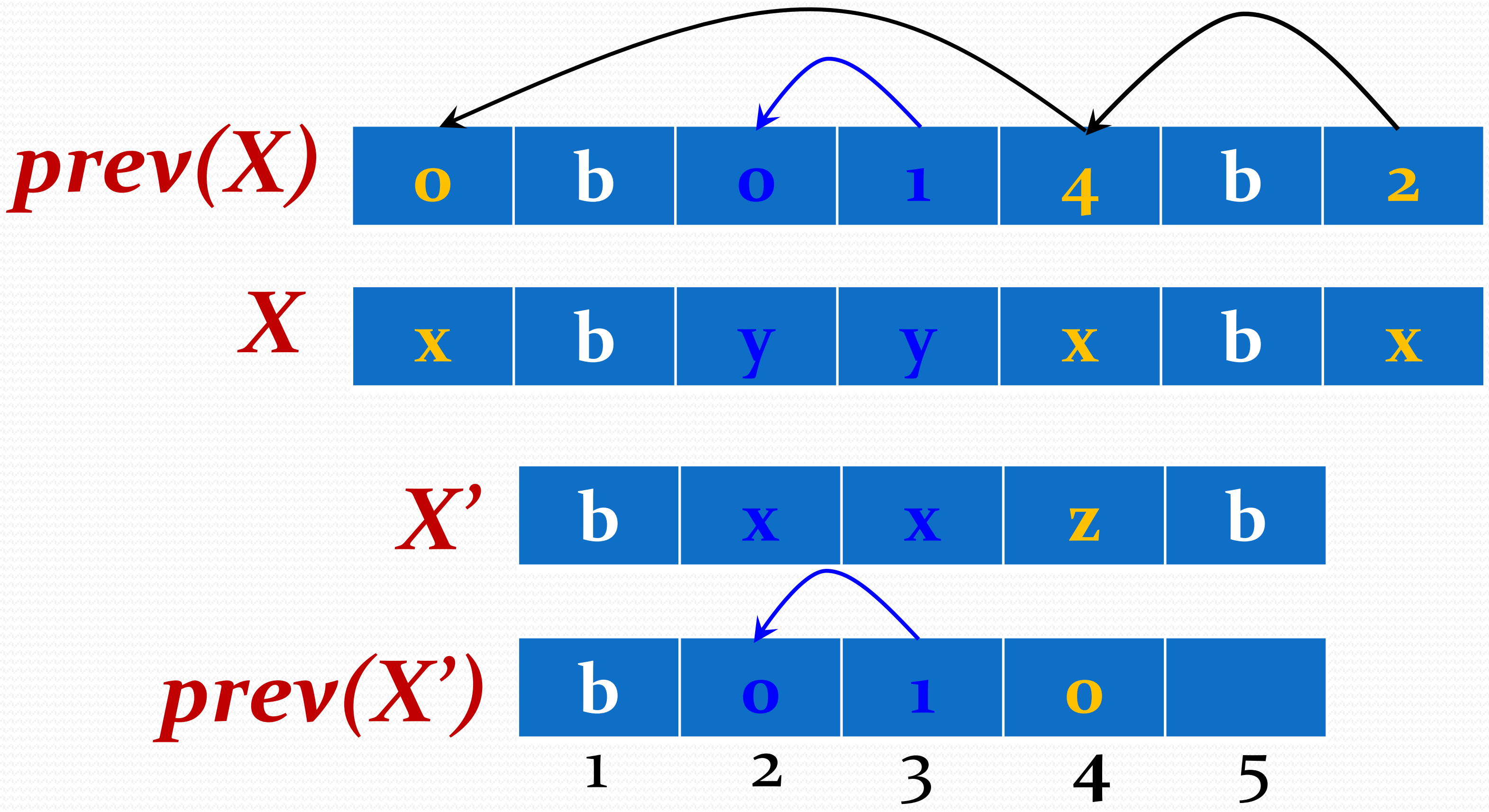
Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



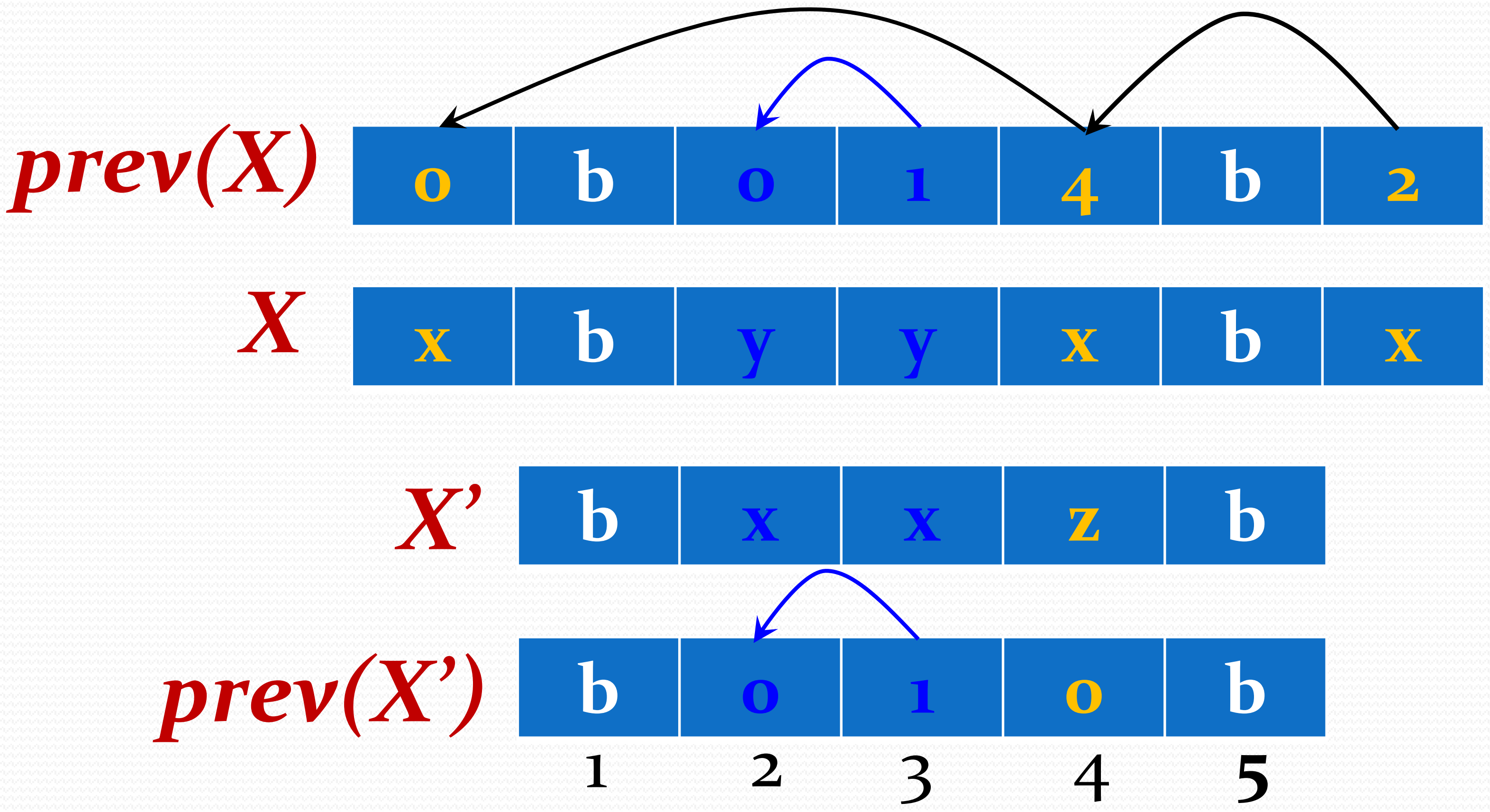
Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



Computing prev of a substring

- We can compute *prev(X[i..j])* based on *prev(X[1..m])*.



Baker’s Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

Parameterized-suffix (p-suffix)

- **P-suffixes** were also introduced by [Baker, 1997].
- *i*-th p-suffix of *X[1..m]*: *prev(X[i..m])*.
- Parameterized-suffix tree (**p-suffix tree**): compacted trie that stores all the p-suffixes of a p-string.
- Used as an aid to solve the parameterized pattern matching problem.

p-suffixes

- *Σ={b}, Π={x,y}*
- *T=xbyyxbx*
- *prev(T)= 0b014b2*

i	p-substring	p-suffix
1	*xbyyxbx*	0b014b2
2	*byyxbx*	b010b2
3	*yyxbx*	010b2
4	*yxbx*	00b2
5	*bx*	0b2
6	*bx*	b0
7	*x*	0

p-suffix Tree

- $\Sigma=\{b\}, \Pi=\{x,y\}$
- $T=xbyyxbx$

1	0b014b2
2	b010b2
3	010b2
4	00b2
5	0b2
6	b0
7	0

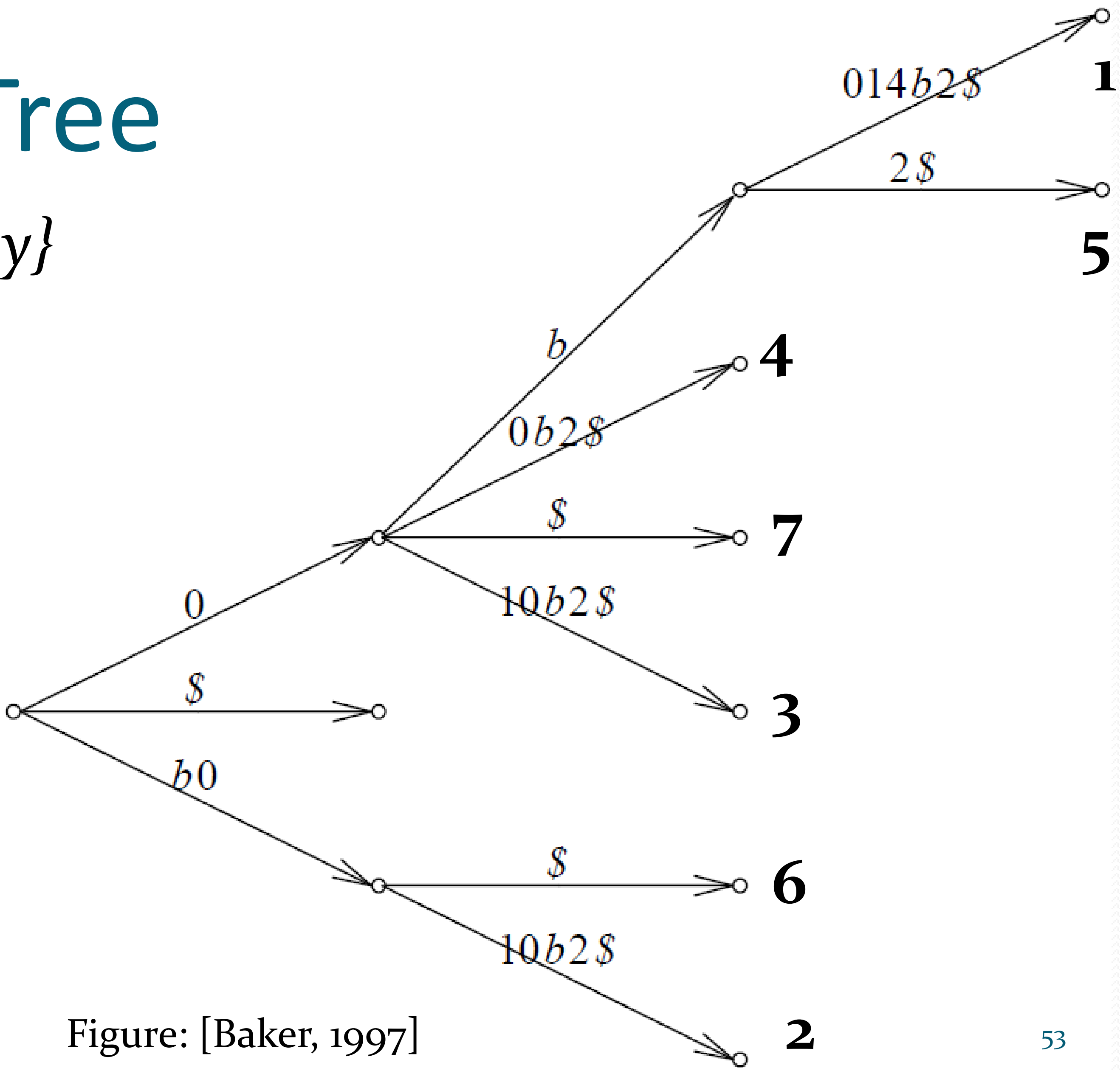


Figure: [Baker, 1997]

p-suffix Tree Construction

Algorithm	Time Complexity
[Baker, 1997] : Lazy	$O(n\|\Pi\| \log (\|\Sigma\|+\|\Pi\|))$
[Baker, 1993]: Eager	$O(n(\|\Pi\|+ \log (\|\Sigma\|+\|\Pi\|)))$
[Kosaraju, 1995]	$O(n \log (\|\Sigma\|+\|\Pi\|))$
[Lee, 2011]	Randomized $O(n)$

Baker's Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

Pattern Matching

- **Key idea:** if there is a p-match, *prev(P)* exactly matches the first part of a p-suffix of *T*.
- **Algorithm:**
  - Construct a p-suffix tree of *T*.
  - Calculate *prev(P)*.
  - Follow the path established by *prev(P)*.
  - The leaves under the path indicate the matching positions.
- **Complexity (fixed alphabets):**
  - Time: O(m+occ), Space: O(n)

Pattern Matching

- *Σ={b}, Π={x,y}*
- *T=xbyyxbx*
- *P=bxxyb*
- *prev(P) =b010b*

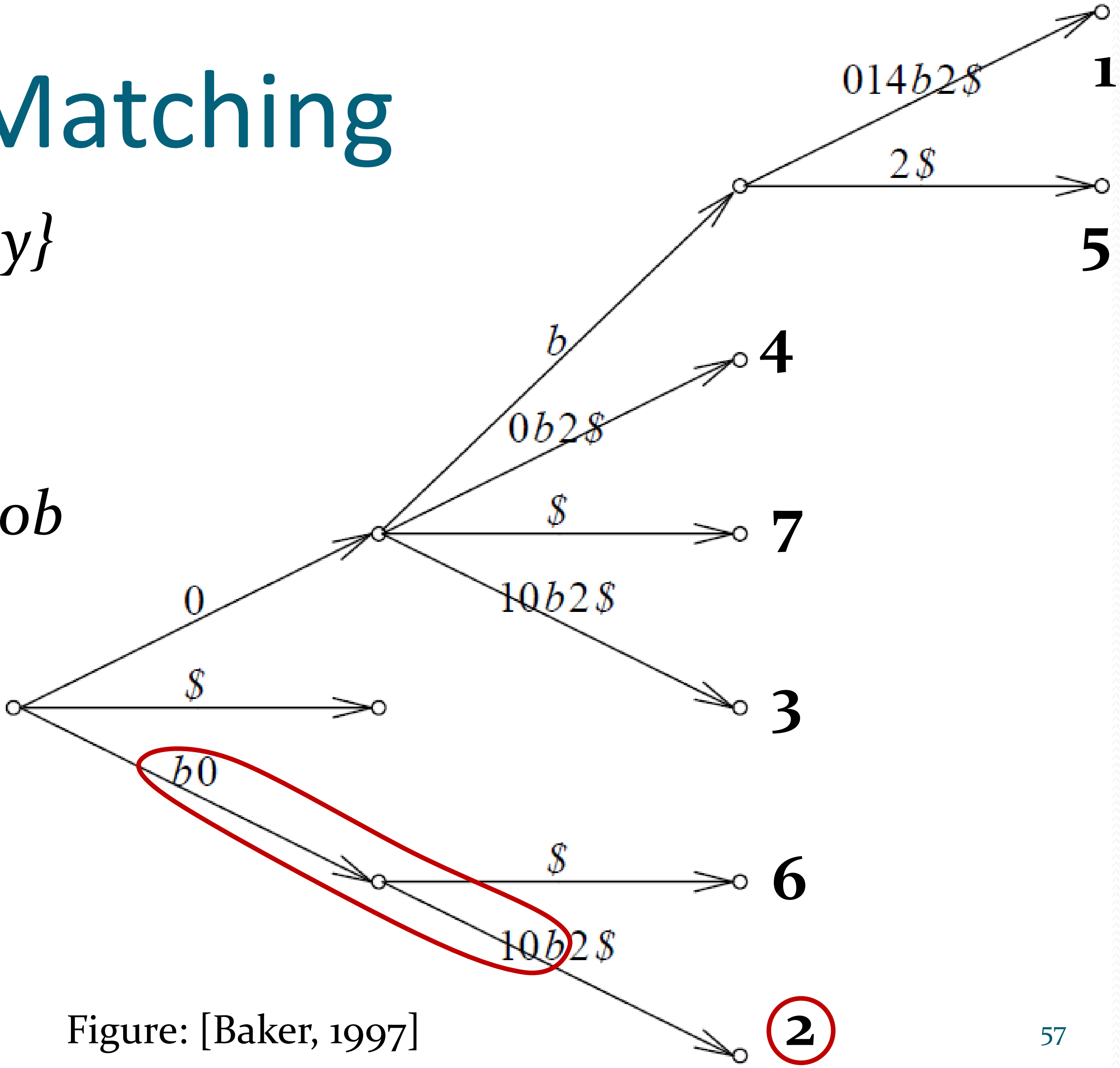


Figure: [Baker, 1997]

Pattern Matching

- *Σ={b}, Π={x,y}*
- *T=x***byyxb***x*
- *P=***bxxyb**
- *prev(P) =b010b*

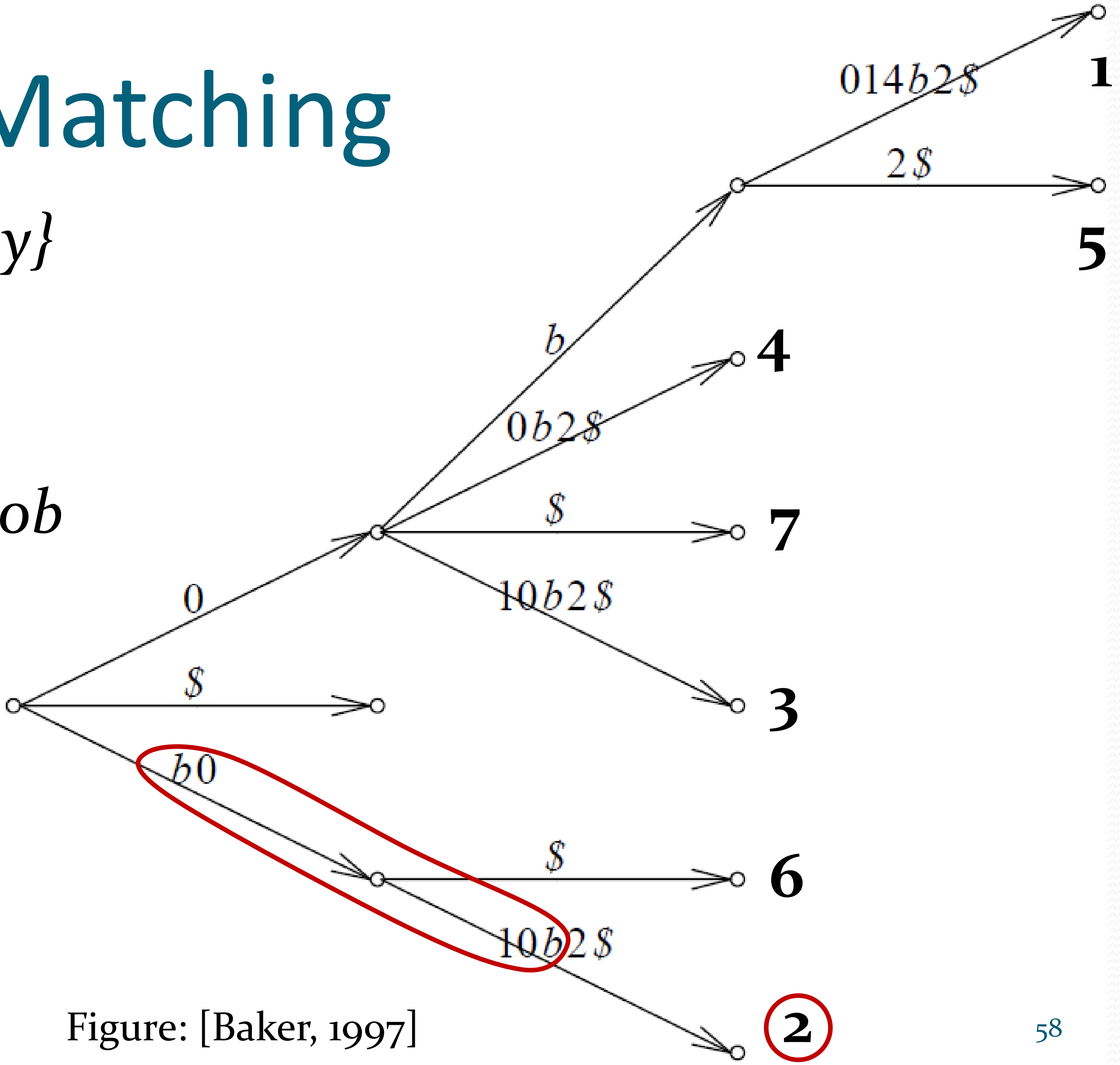


Figure: [Baker, 1997]

Baker's Theory

DUP

Solution for String Comparison

prev

p-suffix trees

Pattern matching

Maximal p-matches over a threshold length

Maximal p-matches

- DUP was generalized to pDUP [Baker, 1997].
- Instead of a suffix tree, it uses a p-suffix tree.
- It augments the p-suffix tree with lists that provide useful information to determine left-extensibility.
- **Complexity:** $O(n+occ)$ even for variable alphabets.

Solutions

Baker's theory

Generalization of Exact Matching Algorithms

Generalization of Exact Matching Algorithms

p-Suffix Arrays

p-KMP

p-TurboBM

p-AhoCorasick

PBTM

Parameterized Suffix Arrays

- Improve memory usage and access locality.
- Defined with respect to p-suffix trees in an analogous manner as suffix arrays are defined to suffix trees [Deguchi, 2008].
- **P-suffix arrays** and **p-LCP** (parameterized longest common prefix) can simulate the operation of p-suffix trees.
- Pattern matching can be solved with a binary search in $O(m+\log n+occ)$.

Construction of p-suffix Arrays

- Algorithms to construct a p-suffix array without constructing its corresponding p-suffix tree.
  - [Deguchi, 2008] for binary alphabets.
  - [I, 2009] for non-binary alphabets.

p-suffix Sorting

- Problem of lexicographically sorting the p-suffixes of a p-string.
- The dynamic nature of p-strings becomes a challenge.
- p-suffix sorting has been considered:
  - [I, 2009]:
    - $O(n^3)$ based on QuickSort
    - $O(n^2)$ based on Raddix Sort.
  - [Beal, 2012]: uses fingerprints and arithmetic codes. Worst case: $o(n^2)$; expected time: $O(n)$.

Other Insights on the Problem

- [Amir, 1994] defined an associated paradigm: **mapped matching** (where ∑ is empty).
- Notice that when ∏ is empty, parameterized matching is equivalent to exact pattern matching.
- Based on a reduction to the element distinctness problem, they proved that $\log \min(m, |\Pi|)$ is inherent to any parameterized matching algorithm.

Parameterized KMP

- [Amir, 1994] also proposed a parameterized version of the KMP algorithm: p-KMP.
- It runs in $O(n \log \min(m, |\Pi|))$.
- It is the first optimal algorithm.

Parameterized Boyer-Moore

- Later, [Baker, 1995] explored the generalization of Boyer-Moore algorithm to parameterized matching, but its worst-case performance was poor.
- Then, she generalized one of its variants: **TurboBM**.
- The resulting algorithm takes
  - Searching phase: $O(n \log \min(m, |\Pi|))$ so it's optimal.
  - Preprocessing phase: $O(m \log \min(m, |\Pi|))$
  - Space complexity: $O(n)$
  - Better for long patterns.

Parameterized Aho-Corasick

- [Idury, 1996] proposed multiple parameterized matching.
- They proposed an adaptation of the Aho Corasick algorithm that runs in $O(n \log (|\Sigma|+|\Pi|)+occ)$.
- A dynamic dictionary of patterns was also considered:
  - Searching for patterns: $O((n+occ)(\log (|\Sigma|+|\Pi|)+\log d))$
  - Inserting a pattern: $O(m \log (|\Sigma|+|\Pi|)+\log^2 d))$
  - Deleting a pattern: $O(m \log (|\Sigma|+|\Pi|)+\log d))$

Parameterized border arrays

- Parameterized version of traditional border arrays.
- The p-AhoCorasick algorithm led to their definition:
  - **pgoto**, **pfail** are the parameterized counterparts of **goto** and **fail** in traditional AhoCorasick.
  - When there is a single pattern, **pfail** can be implemented as a **p-border array**.
  - It can be computed in linear time [Idury, 1996].

Parameterized border arrays

- For binary alphabets [I, 2009a] proposed algorithms to:
  - Validate if an integer array is a valid p-border array. **Complexity**: $O(n)$.
  - Compute all the p-strings that share the same p-border array. **Complexity**: $O(n)$.
  - Compute all the border arrays shorter than a threshold length. **Complexity**: linear in the output reported.

Parameterized border arrays

- For unbounded alphabets, [I, 2009a] proposed an algorithm to verify if an integer array is valid p-border array. **Time**: $O(n^{1.5})$. **Space:** $O(n)$.
- Furthermore, they showed that the enumeration of all p-border arrays shorter than a threshold length can be done in $O(B^n n^{2.5})$.

p-Shift-OR

- [Fredriksson, 2006] makes use of Baker's theory to propose to algorithms: p-ShiftOR and PBTM.
- p-ShiftOR is a generalization of ShiftOR to p-strings.
- Time complexity:
  - Worst case: $O(n\lceil m/w \rceil)$
  - Average case: $O(n)$.

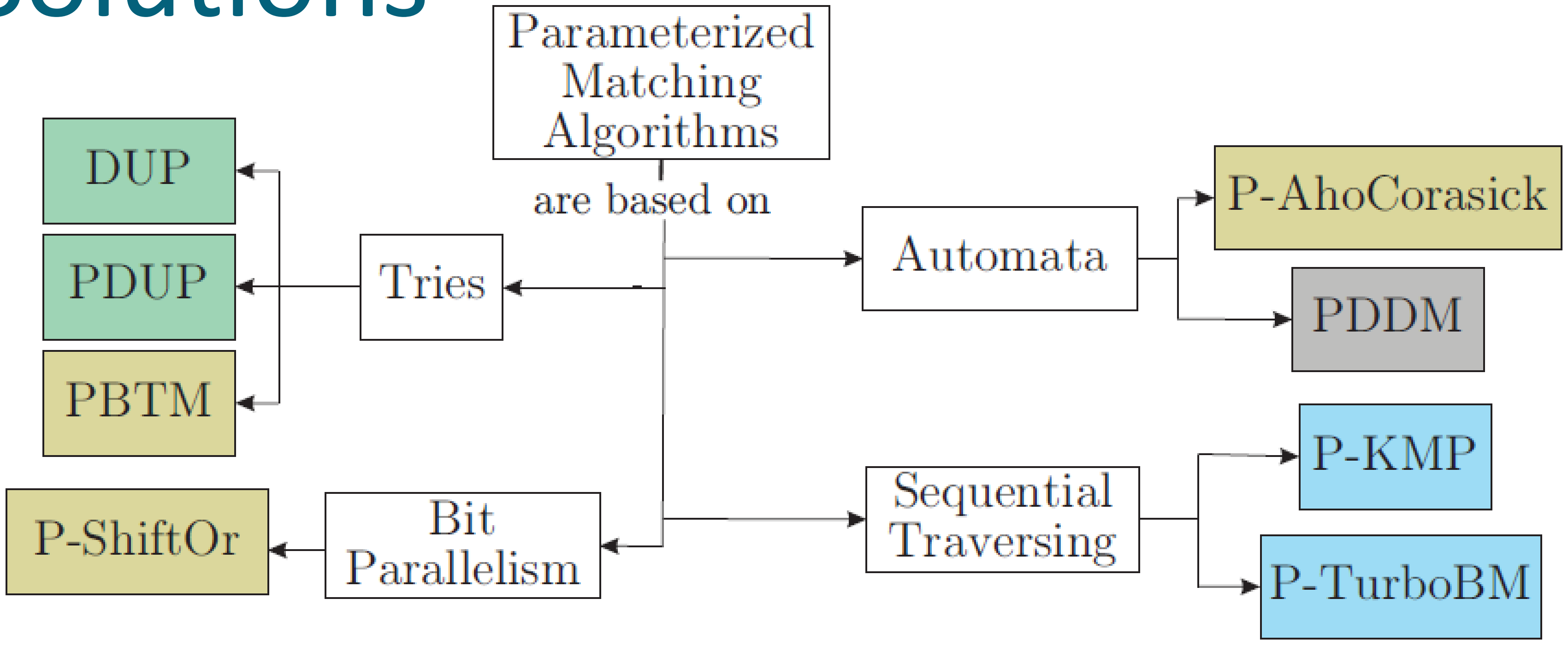
Parameterized Backward Trie Matching (PBTM)

- It is based on the Backward DAWG Matching (BDM) Algorithm and makes use of tries.
- Its average time complexity is *O(n log (m)/m)*.
- A variation that uses arrays instead of tries was also consider by [Fredriksson, 2006]; such variation is calles PBAM.

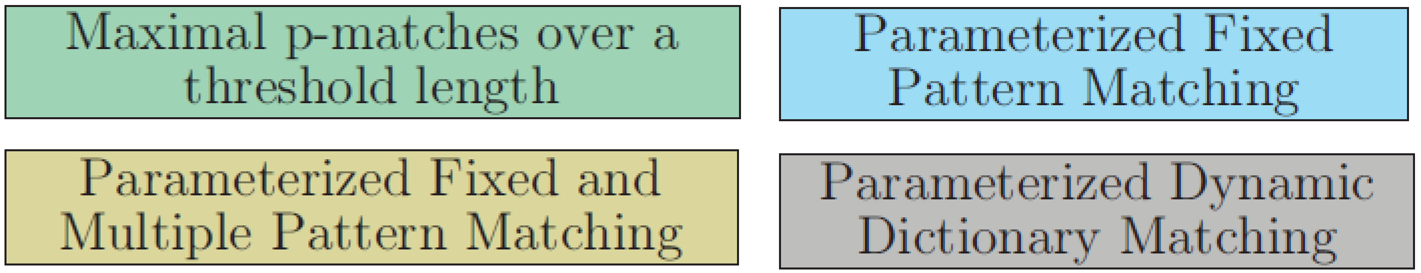
Average Case Analysis

- P-ShiftOR and PBTM were the first parameterized matching algorithms for which the average-case analysis was made.
- An algorithm that has sublinear average-case expected time was proposed by [Salmela, 2006]. It is based on Boyer-Moore.

Solutions



The background color of each algorithm indicates the problem it solves:


Maximal p-matches over a threshold length
Parameterized Fixed Pattern Matching
Parameterized Fixed and Multiple Pattern Matching
Parameterized Dynamic Dictionary Matching

Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Extensions
- Applications
- Conclusions

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Some properties

- **Relation with palindromes:** Two strings drawn from an alphabet of size 3 have the same set of maximal palindromes iff they are a p-match [I, 2010].
- **Periodicity and repetitions:** [Apostolico, 2008]
  - For binary alphabets, p-strings and strings behave in a similar manner.
  - For non-binary alphabets, there are significant differences between p-strings and strings.

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

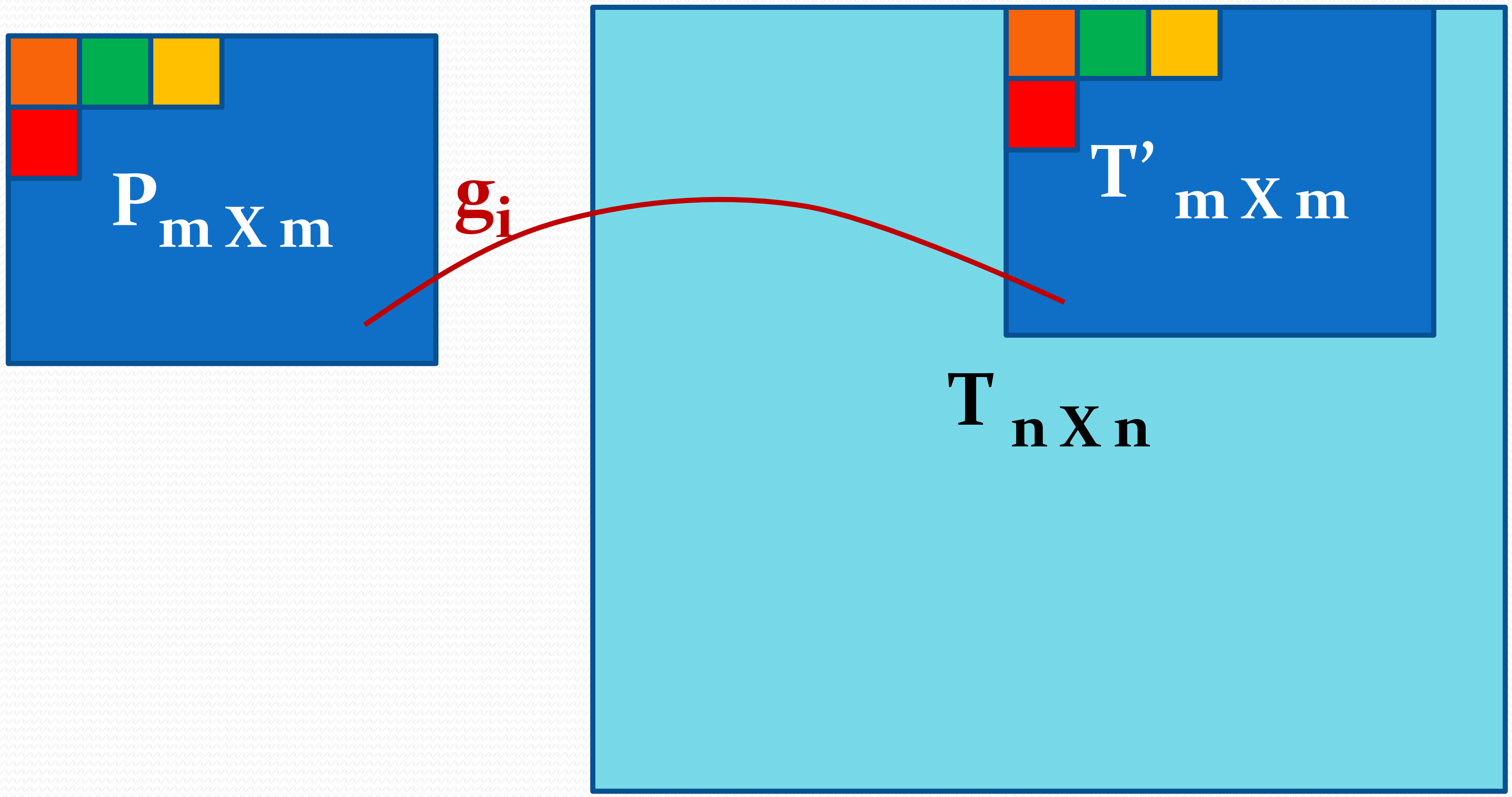
Parameterized Longest Previous Factor

Structural Matching

Function Matching

Two dimensional p-matching

- Find all the 2-dimensional p-matches:



Two-dimensional p-matching

- Deterministic solutions:
  - $O(n^2+m^{2.5}\ polylog\ m)$ by [Hazay, 2004].
  - $O(n^2 \log^2 m)$ by [Amir, 2003].
- Randomized Algorithm
  - $O(n^2 \log n)$ by [Amir, 2003] with error probability of $1/n^k$ (where k is a constant).

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Approximate Approaches

P-Edit distance

P-matching under the hamming distance

δγ-Parameterized Matching

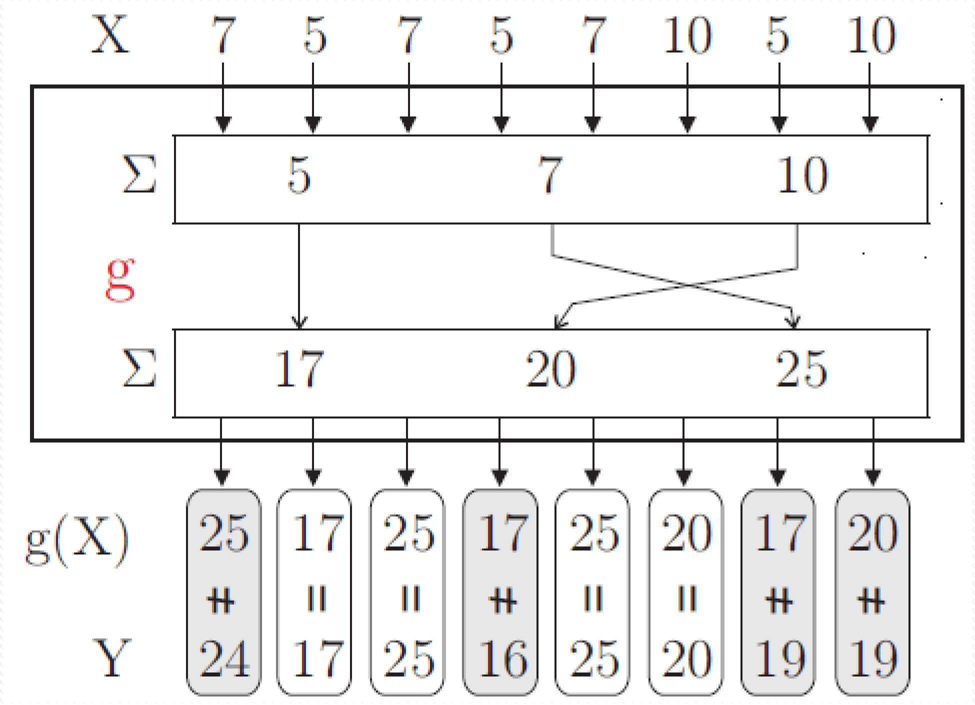
Longest Common Parameterized Subsequence

Parameterized edit distance

- **P-edit distance:** cost of a minimal script that transforms one p-strings into the other.
- Valid operations:
  - Insertions
  - Deletions
  - Parameterized replacements (replacement of a p-string with a p-string that matches it).
- $O(D(n+m))$-time algorithms proposed by [Baker, 1999].
  - Calculating the p-edit distance D.
  - Reporting the minimal p-edit script.

P-matching under the hamming distance

- For a given mapping *g* between to equal-length p-strings X and Y, the ***g*-match** is the number of matches between *X[i]* and *g(Y[i])*, for all *i*.



P-matching under the hamming distance

- **Approximate Parameterized Matching:** Find the maximal *g*-match between two equal-length p-strings.
- **Parameterized searching under the hamming distance:** For every length-*m* text window in the text find the maximal *g*-match.
- An algorithm for a run-length encoded pattern and text, where one of them is a binary p-string, was proposed by [Apostolico, 2007].

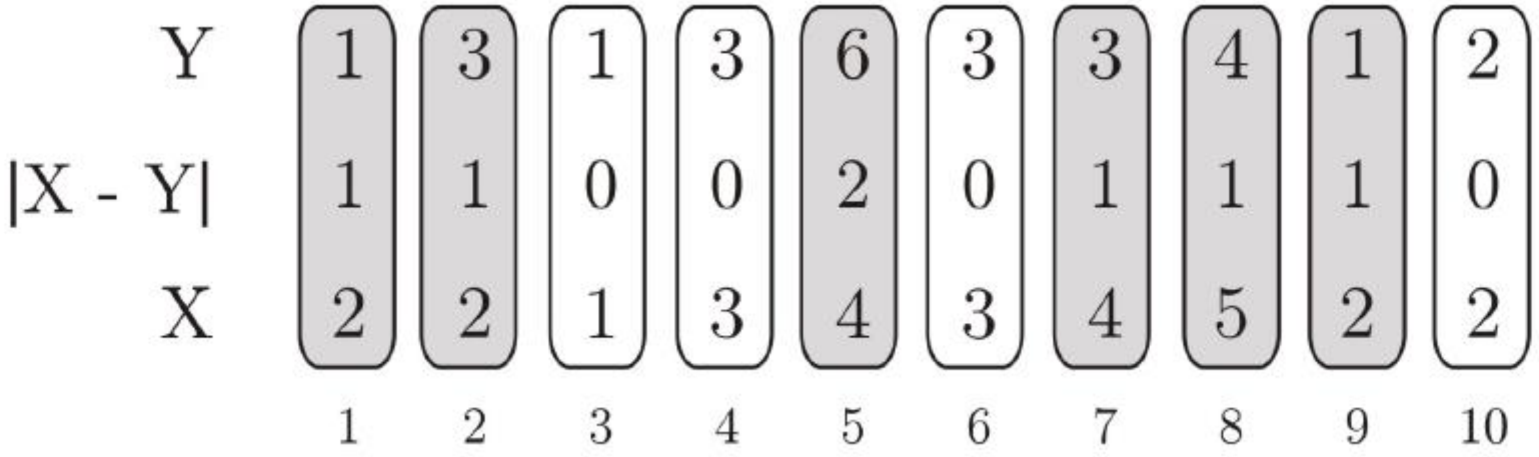
P-matching under the hamming distance

- **Parameterized matching with *k* mismatches:** Find all the text windows in the text that p-match the pattern with at most *k* mismatches.
- Algorithms proposed by [Hazay, 2007]:

Case	Time Complexity
String comparison	$O(m+k^{1.5})$
Pattern matching	$O(nk^{1.5}+mk \log m)$
2-Dimensional	$O(n^2mk^{1.5}+m^2k \log m)$

δγ-Parameterized Matching

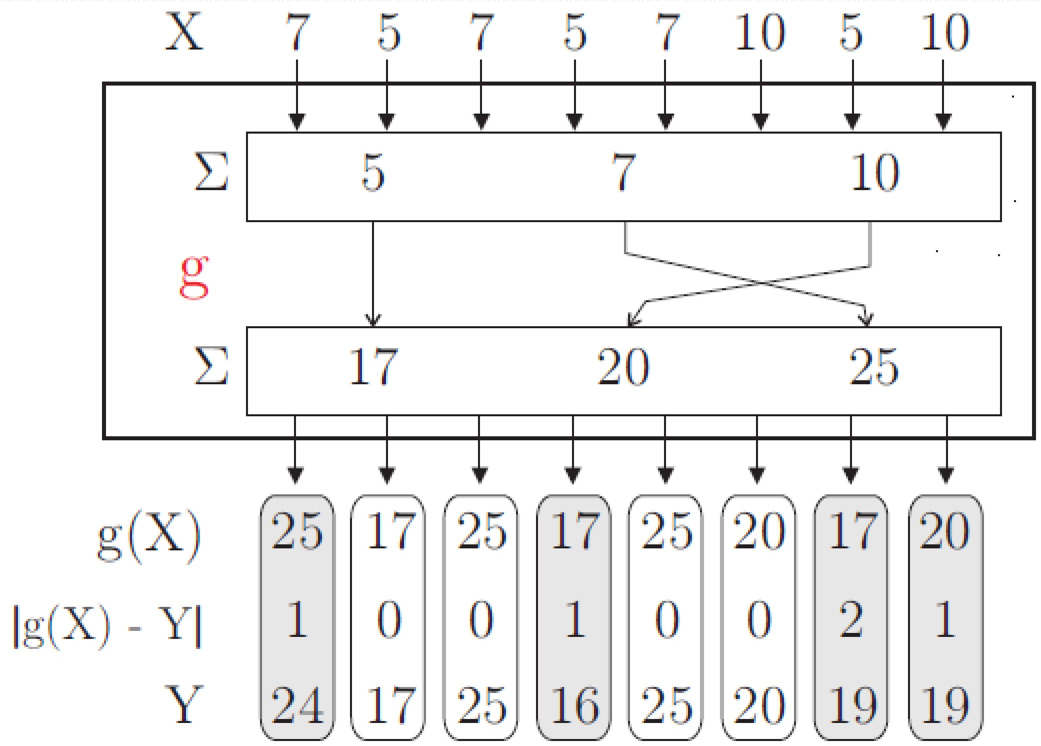
- In traditional integer strings, *X[1..m]* and *Y[1..m]*…
  - … δ-match iff $max_i\ |X[i]\text{-}Y[i]| \leq \delta$.
  - … γ-match iff $\sum_i |X[i]\text{-}Y[i]| \leq \gamma$.
- For example, the following strings $\delta\gamma$–match for δ=2 and γ=7:



Y	1	3	1	3	6	3	3	4	1	2
|X - Y|	1	1	0	0	2	0	1	1	1	0
X	2	2	1	3	4	3	4	5	2	2
	1	2	3	4	5	6	7	8	9	10

δγ-Parameterized Matching

- Integer p-strings *X[1..m]* and *Y[1..m]* δγ–parameterized match iff *X* can be transformed into *X'* via a bijection *g* such that X' δγ–matches Y.
- Example:
  - δ=2
  - γ=5


X 7 5 7 5 7 10 5 10
Σ 5 7 10
g
Σ 17 20 25
g(X) 25 17 25 17 25 20 17 20
|g(X) - Y| 1 0 0 1 0 0 2 1
Y 24 17 25 16 25 20 19 19

δγ-Parameterized Matching

- A *O(nm)* algorithm to find all the *δγ*–parameterized matches of a pattern in a text was proposed by [Mendivelso, 2010].
- It is based on a reduction to the Maximum Weight Perfect Matching problem in bipartite graphs.

Longest Common Parameterized Subsequence (LCPS)

- Given *X[1..n]* and *Y[1..m]*, find a subsequence I of *X* and a subsequence *J* of *Y* of maximum length such that *I* and *J* are a p-match.
- It's an NP-Hard problem.
- An approximate solution was proposed by [Keller, 2009].

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Parameterized Longest Previous Factor (p-LPF)

- For a p-string of, the p-LPF is calculated for each p-suffix starting at position *i* as the longest factor between such p-suffix and a p-suffix starting before.
- Used to study duplication and compression in p-strings.
- [Beal, 2012] proposed an expected linear time algorithm to compute the p-LPF, LPF, p-LCP, LCP.

Variants of the p-LPF

- [Beal, 2012a] proposed a taxonomy of classes of LPF problems that show the relation between p-LPF and traditional data structures.
- It is shown that p-LCP can be used to linearly construct the p-border array and the border array.
- The concept of permuted LCP is extended to p-strings.

Variants of the p-LPF

- [Beal,2012a] defined:
  - Parameterized Longest not-equal Factor (p-LneF)
  - Parameterized Longest reverse Factor (p-LrF)
  - Parameterized Longest Factor (p-LF)
- These structures can be calculated with the same framework of p-LPF by changing preprocessing and postprocessing.
- They have applications in clone detection, periodicity study and biological sequence compression.

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Structural Matching (s-matching)

- [Shibuya, 2004] defined it as parameterized matching but taking into account an injective complementary relation among a subset of the parameters.
- Additional constrain in the matching: if parameter *x* is mapped to parameter *y*, then the complement of *x* must be mapped to the complement of *y*.
- This is motivated by the application of RNA matching:
  - Adenine – Uracil
  - Cytosine – Guanine

Structural Suffix Trees

- Then, two s-strings that s-match have similar structures and, hence, similar functions.
- [Shibuya, 2004] proposed a solution based on **structural suffix trees**.
- He also proposed an $O(n(\log|\Sigma|+\log|\Pi|))$ online algorithm to construct a s-suffix tree.
- It is linear for RNA/DNA sequences.

Structural Suffix Arrays

- For better space utilization, [Beal, 2013 and 2015] defined:
  - S-suffix array
  - S-LCP
  - S-border array

Extensions

Some properties

Two-dimensional parameterized matching

Approximate Approaches

Parameterized Longest Previous Factor

Structural Matching

Function Matching

Function Matching

- Two equal-length strings function-match if one can be transformed into the other by means of a function.
- In pattern matching, many symbols in the pattern can be mapped to the same symbol in the text window.
- Solutions by [Amir, 2003]:
  - Deterministic Solution: $O(n|\Pi| \log m)$
  - Monte Carlo Algorithm: $O(n \log m)$ with $1/n^k$ failure probability.
-

Function Matching Extensions

- **2-dimensional Funcion Matching:** A $O(kn^2 \log n)$ randomized algorithm was proposed [Amir, 2003].
- **δγ-Function Matching:**
  - $X[1..m]$ and $Y[1..m]$ strings match if X can be transformed into X' by means of a function g such that X' δγ-matches Y.
  - A $O(nm)$ algorithm was proposed by [Mendivelso, 2012].

Generalized Function Matching with Don’t Cares

- The image of the mapping function any substring in $(\Sigma \cup \Pi)^*$.
- The **don’t care** symbol ϕ can be present in strings. It matches:
  - Any substring in the text if it is in the pattern.
  - Any symbol in the pattern if it is in the text.

Generalized Function Matching with Don’t Cares

- A polynomial-time algorithm for finite alphabets was devised [Amir, 2007].
- It was shown that for infinite alphabets, the problem is NP-Hard.
- It is the first problem for which there is a polynomial solutions for finite alphabets but not for infinite alphabets.

Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Extensions
- Applications
- Conclusions

Applications

Image Processing

Databases

Graph Isomorphism Solution

Applications

Image Processing

Databases

Graph Isomorphism Solution

Image Processing

- The problem of searching an icon in the screen [Hazay, 2007].
- It can be solved with:
  - Exact matching
  - Parameterized matching
  - Approximate parameterized matching (hamming, p-edit, δγ distance)
  - Function matching

Applications

Image Processing

Databases

Graph Isomorphism Solution

Databases

- In a database of URL’s, parameterized queries can be used to improve the ergonomy of the site and finding the best places for advertisement ads.
- In computational biology, it can be used to find amino acid strings that follow a determined structure.

Applications

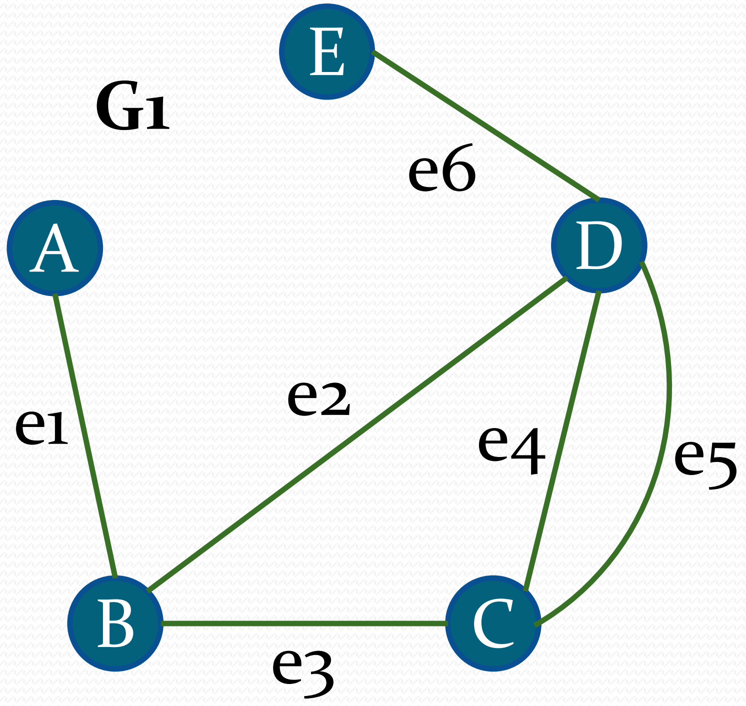
Image Processing

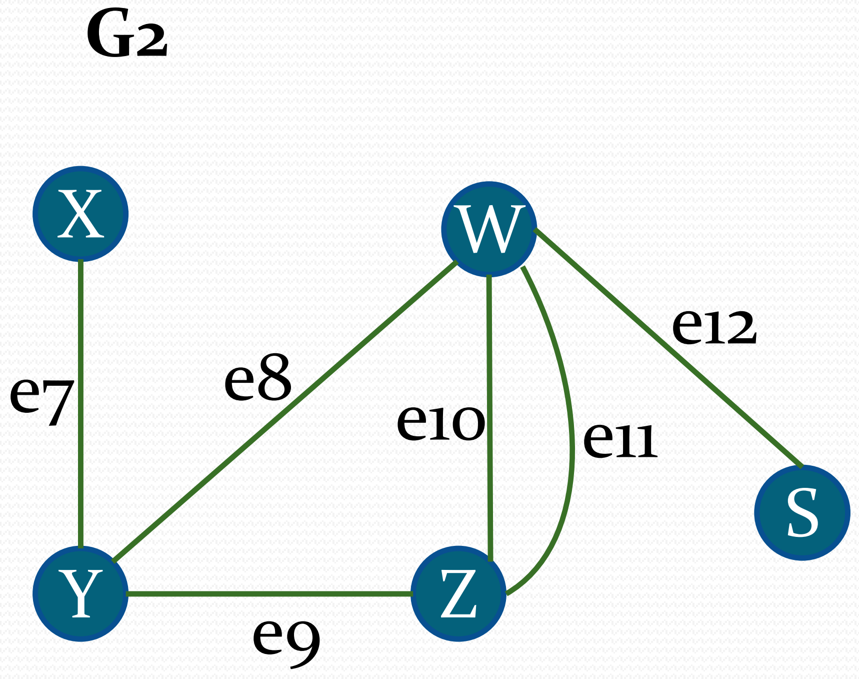
Databases

Graph Isomorphism Solution

Graph Isomorphism

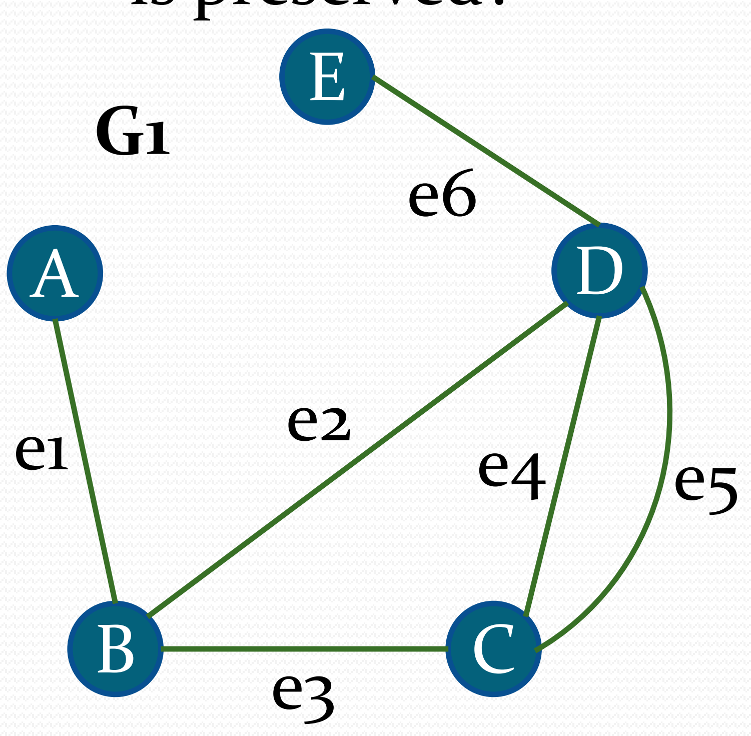
- Is there a bijection **f** that maps the nodes/edges of *G1* to the nodes/edges in *G2* so that the adjacency relation is preserved?

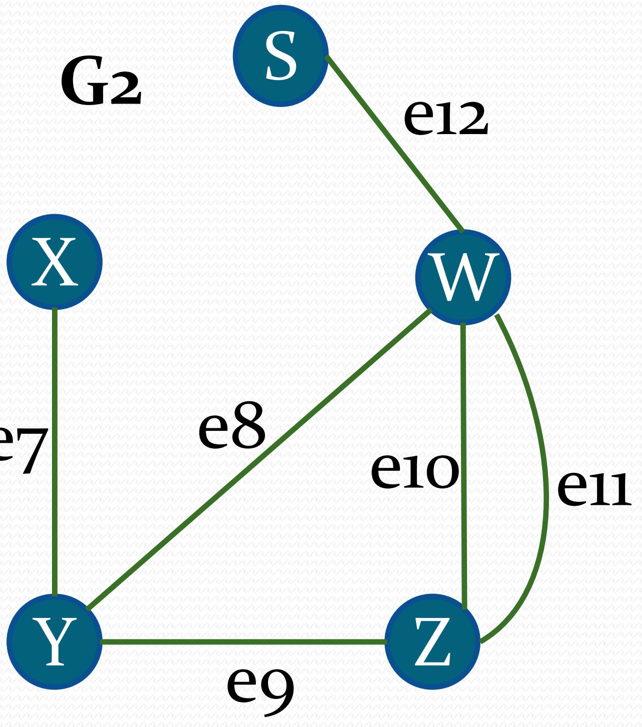




Graph Isomorphism

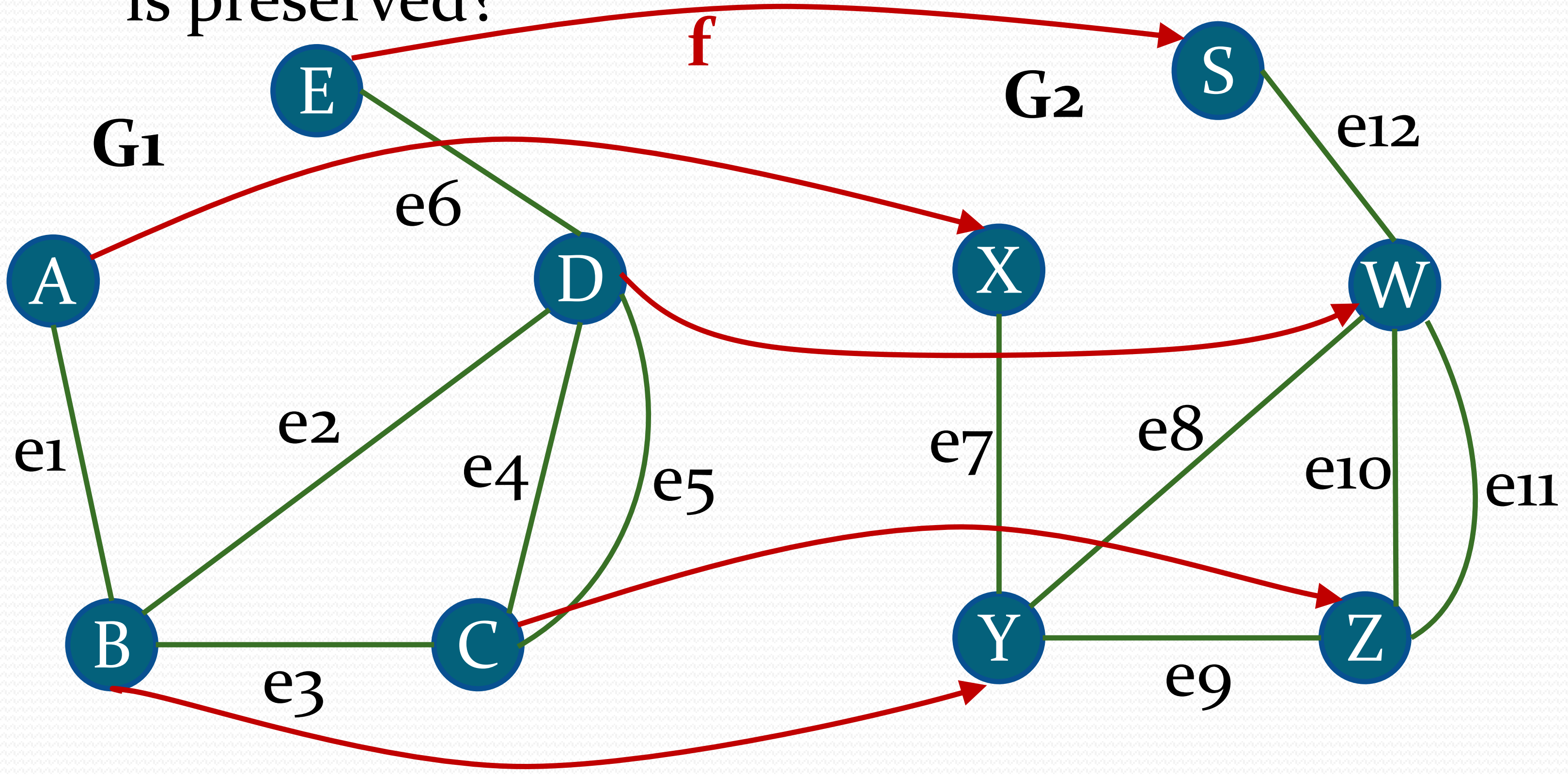
- Is there a bijection **f** that maps the nodes/edges of *G1* to the nodes/edges in *G2* so that the adjacency relation is preserved?





Graph Isomorphism

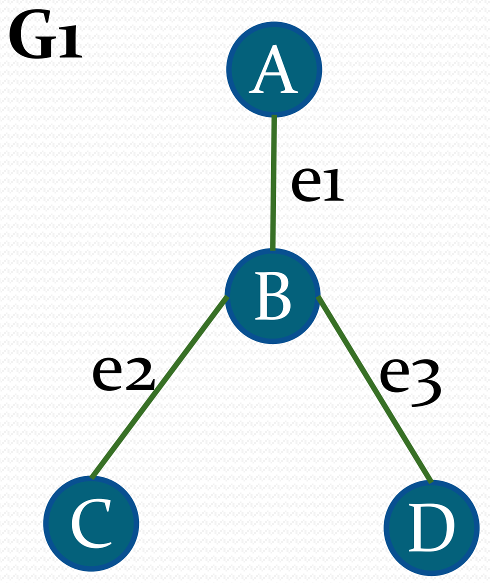
- Is there a bijection **f** that maps the nodes/edges of *G1* to the nodes/edges in *G2* so that the adjacency relation is preserved?



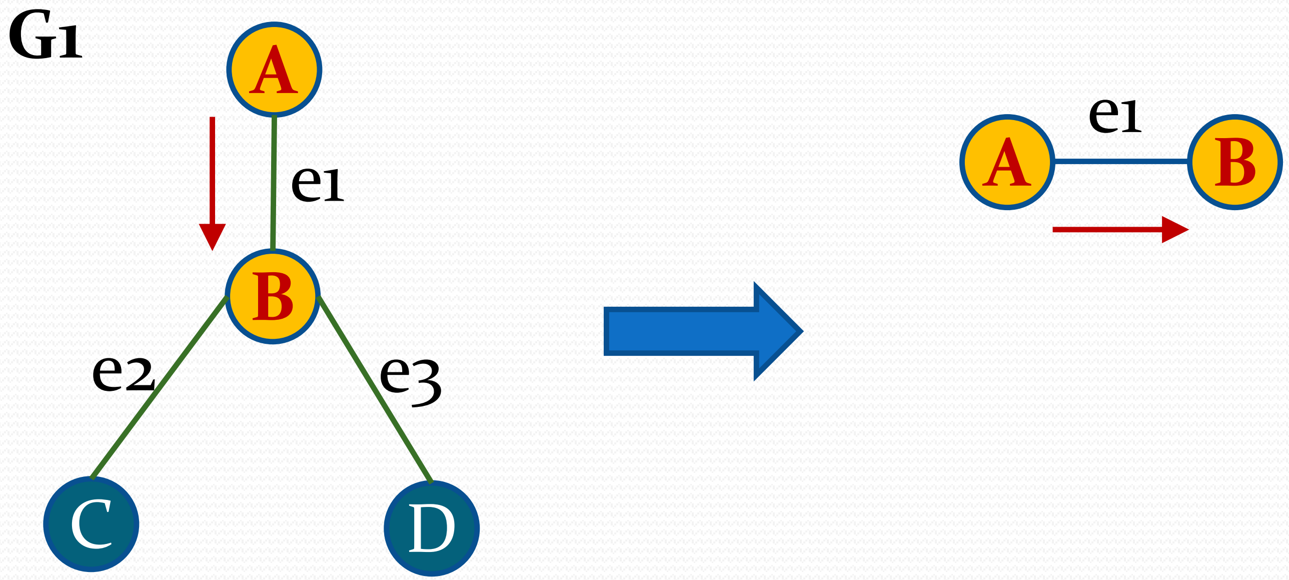
Graph Linearization

- It represents the structure of a graph in a linear manner.
- Specifically, our linearization is a walk on the graph that contains all its nodes and edges at least once.
- Then, we evaluate graph isomorphism by comparing walks rather than graphs.

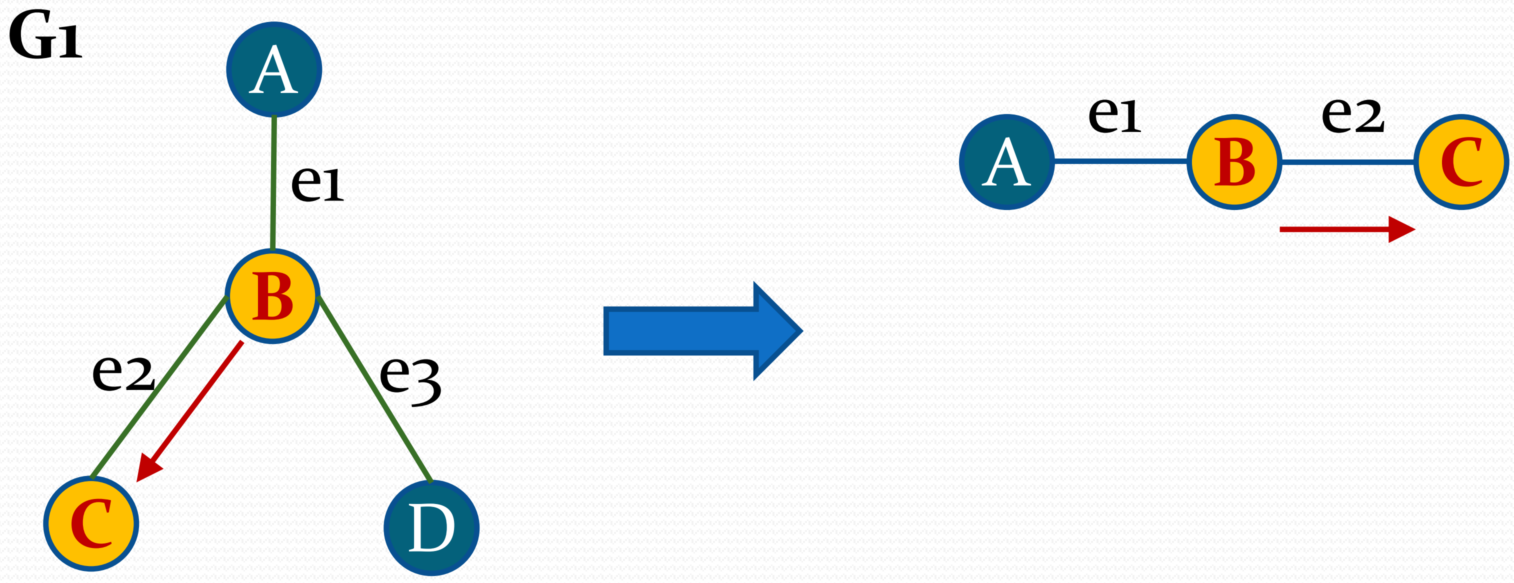
How do we linearize a graph?



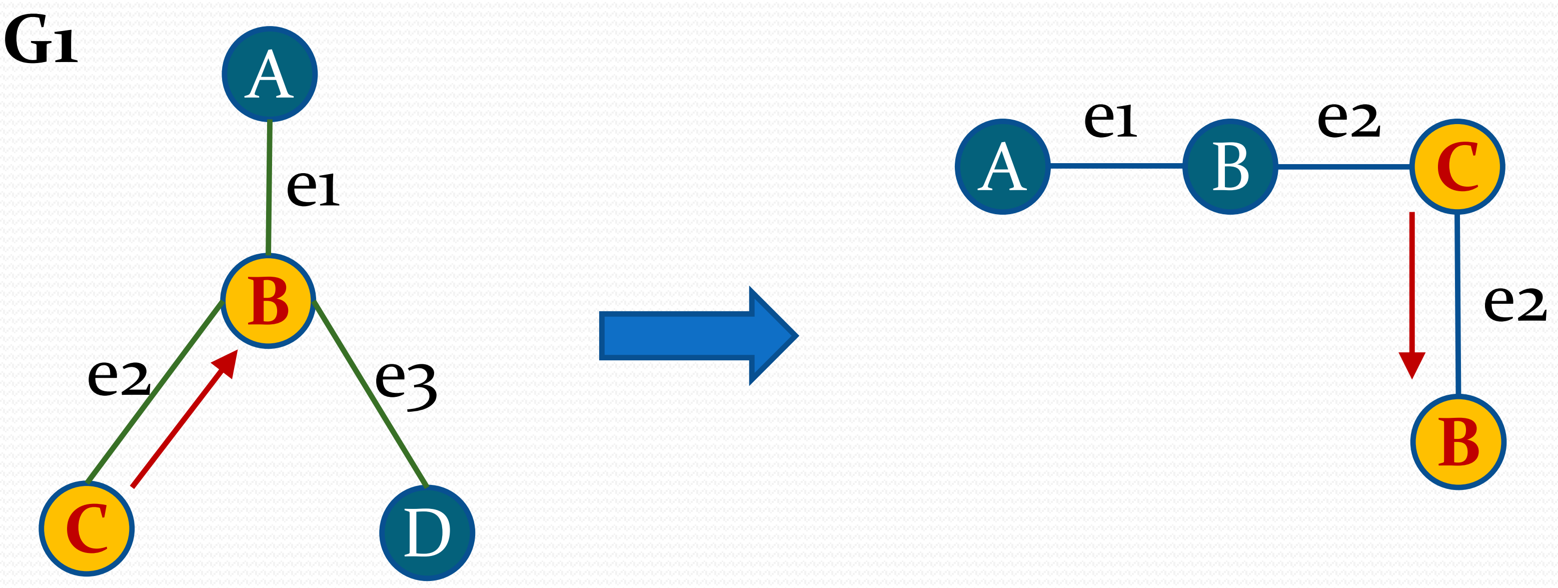
How do we linearize a graph?



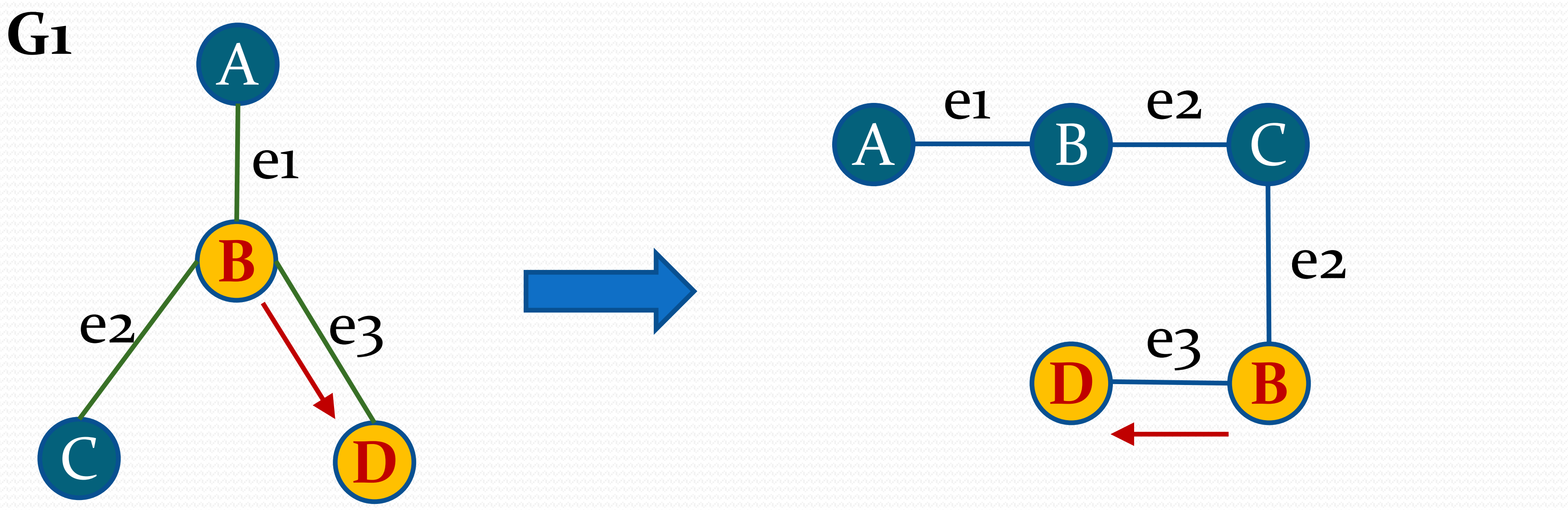
How do we linearize a graph?



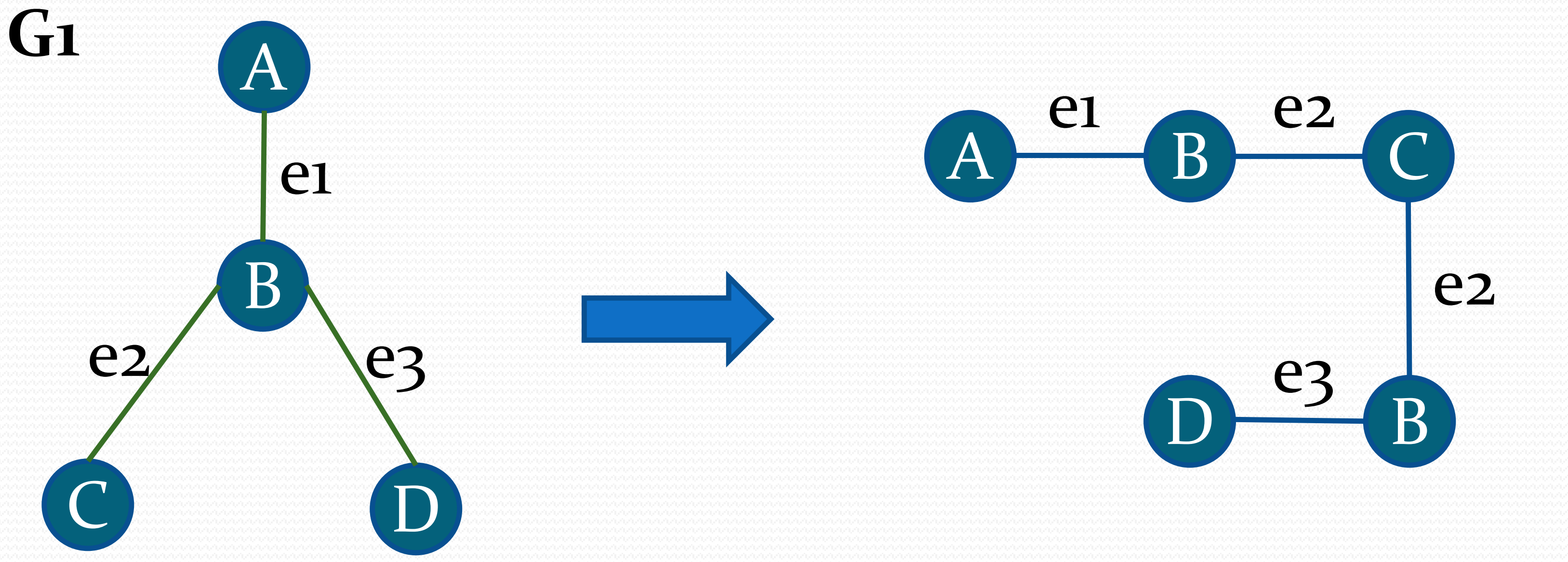
How do we linearize a graph?



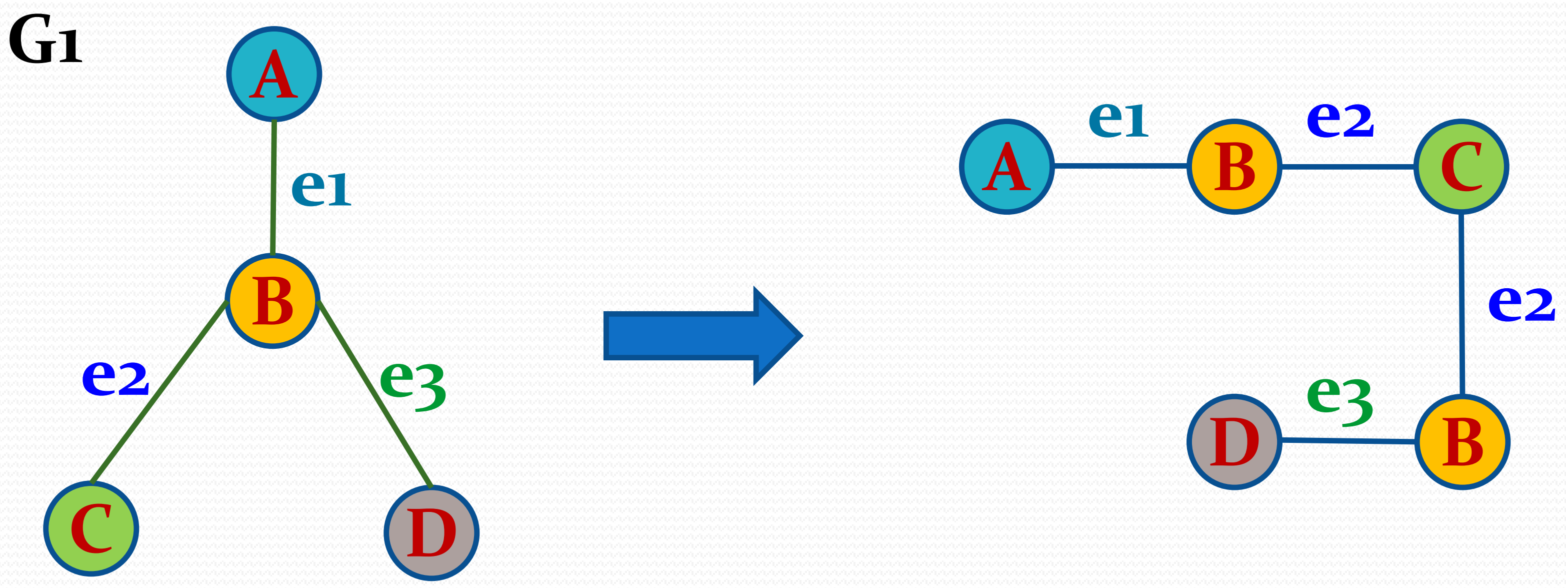
How do we linearize a graph?



How do we linearize a graph?

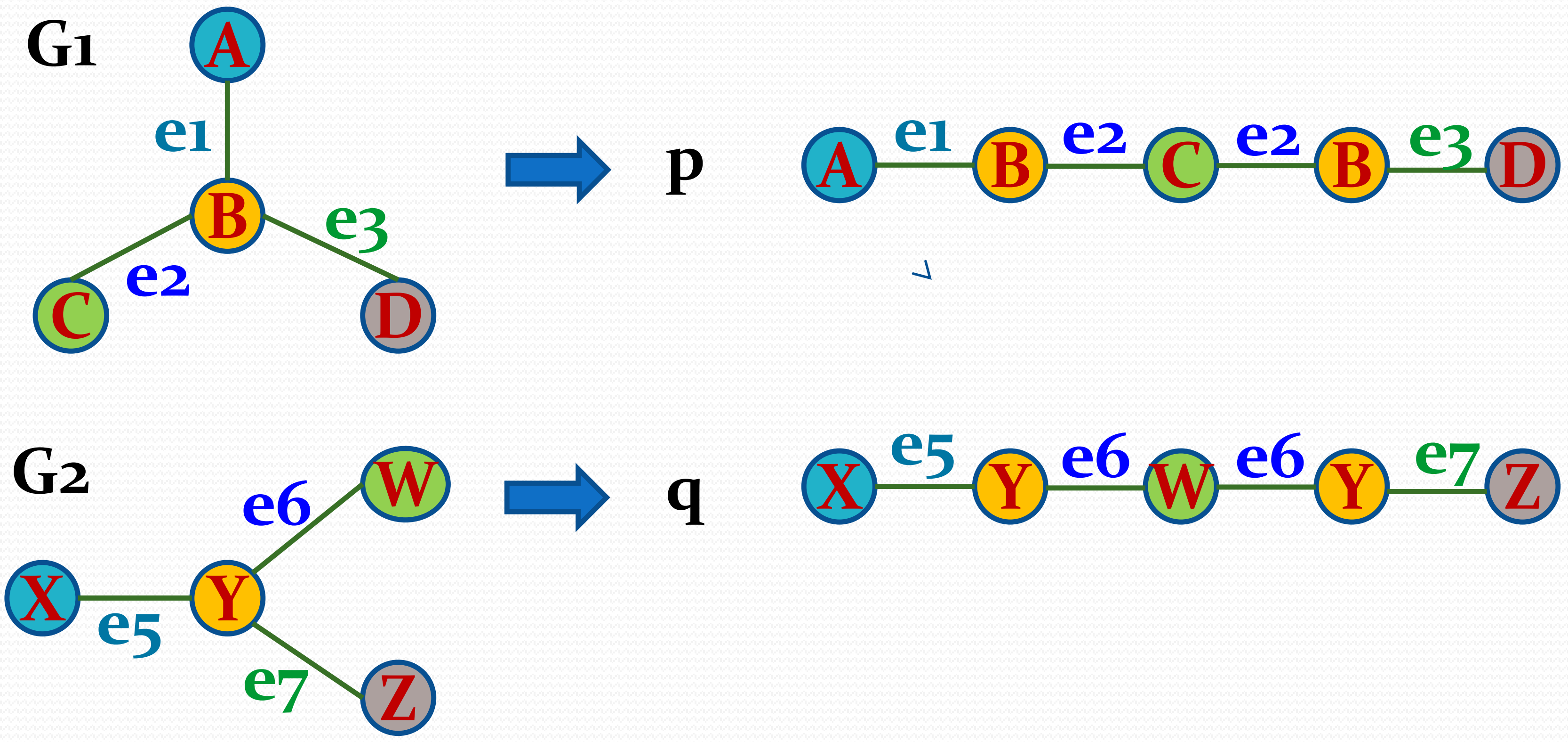


The parameters



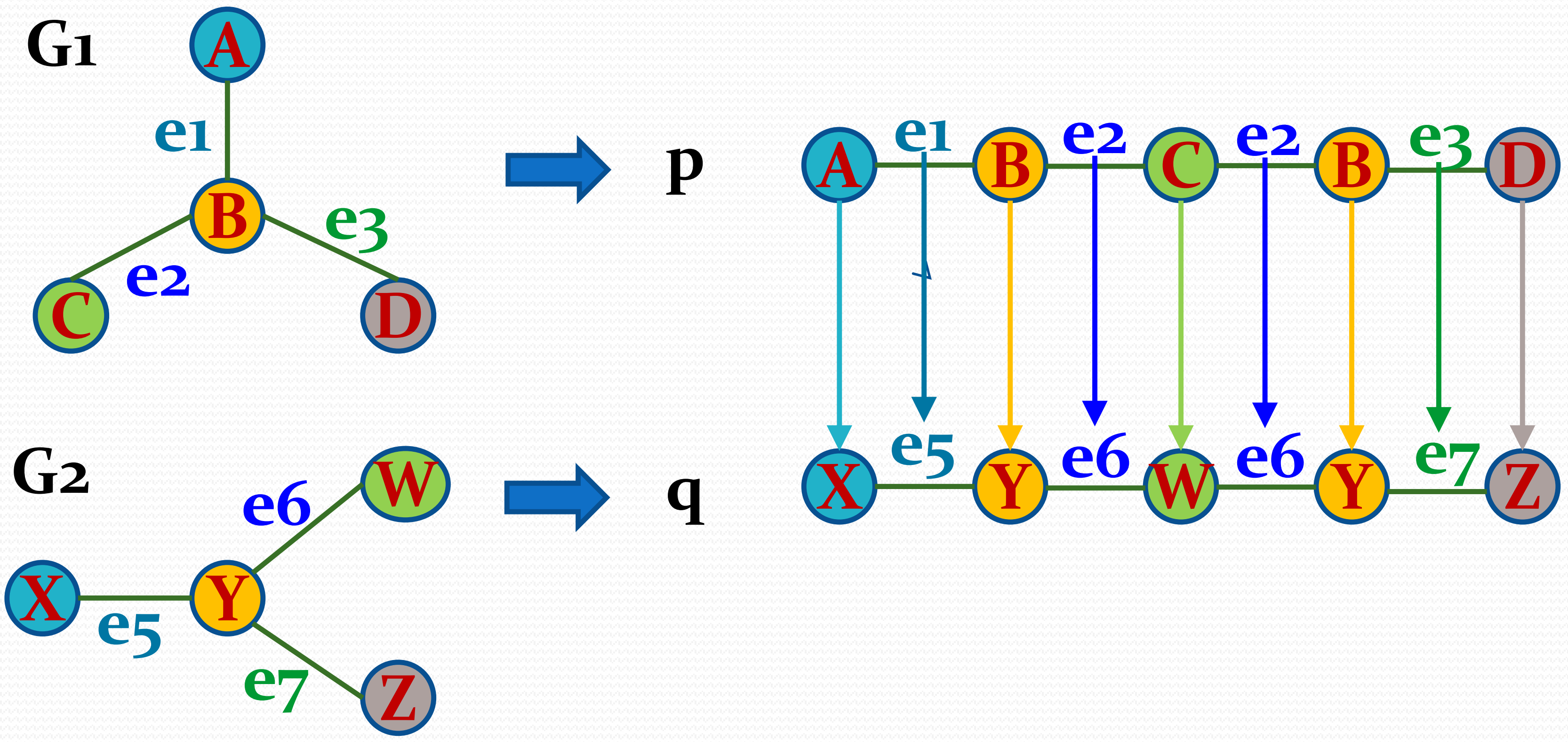
How to use our linearizations to match graphs?

- *G1* and *G2* are isomorphic if **there is** a linearization of *G2* that parameterized-matches the linearization of *G1*.



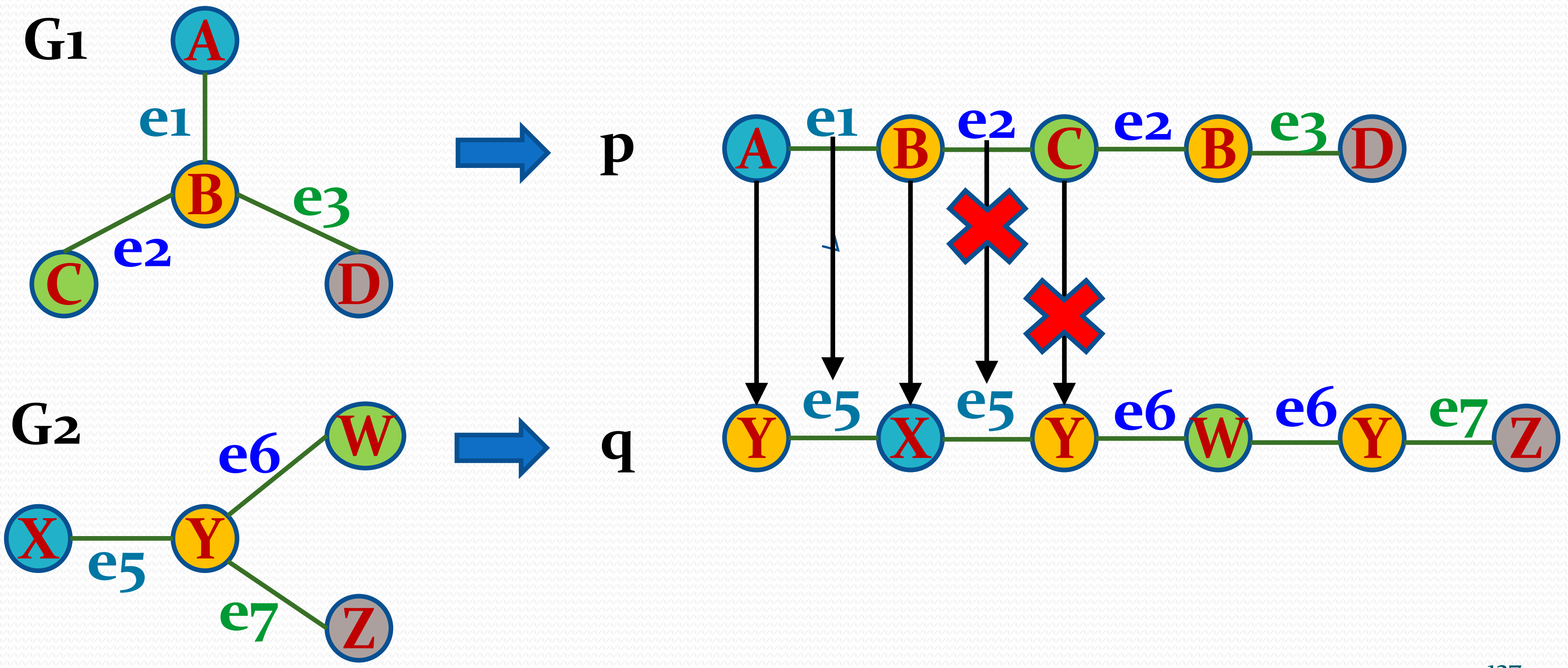
How to use our linearizations to match graphs?

- *G1* and *G2* are isomorphic if **there is** a linearization of *G2* that parameterized-matches the linearization of *G1*.



How to use our linearizations to match graphs?

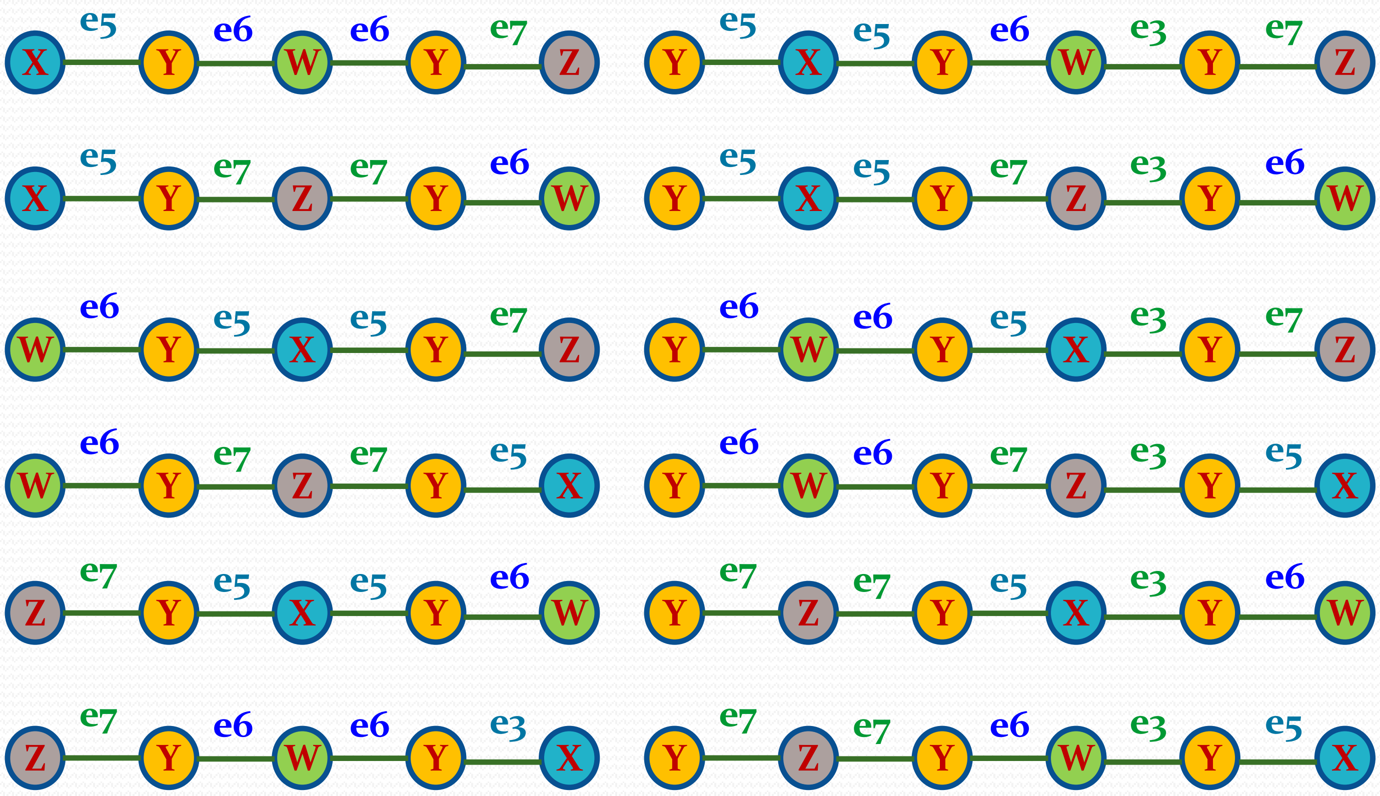
- But what if we had calculated the following *q*?
- We need to check all the possible linearizations *q*.



How to use our linearizations to match graphs?

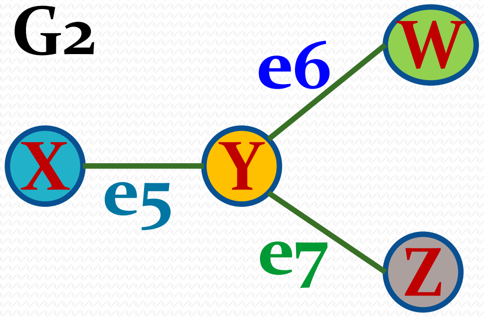
- But there may be *Ω(max(n!,m!))* linearizations of a graph.

q can be any of:



X e5 Y e6 W e6 Y e7 Z
Y e5 X e5 Y e6 W e3 Y e7 Z
X e5 Y e7 Z e7 Y e6 W
Y e5 X e5 Y e7 Z e3 Y e6 W
W e6 Y e5 X e5 Y e7 Z
Y e6 W e6 Y e5 X e3 Y e7 Z
W e6 Y e7 Z e7 Y e5 X
Y e6 W e6 Y e7 Z e3 Y e5 X
Z e7 Y e5 X e5 Y e6 W
Y e7 Z e7 Y e5 X e3 Y e6 W
Z e7 Y e6 W e6 Y e3 X
Y e7 Z e7 Y e6 W e3 Y e5 X

G2



Proposed Solution

- [Mendivelso, 2013] proposed a solution to determine if $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ are isomorphic. It consists of two steps:
  1. Calculating a linearization p of G_1.
  2. Determining whether there exists a walk q in G_2 that parameterized-matches p.

Proposed Solution

- The total time complexity is:

$$O(dm \log d + nd^{\ell/2}) = O(nd^{\ell/2})$$

- Experimental results show that this solution is efficient especially for Miyazaki graphs which constitute a hard case for graph isomorphism algorithms [Mendivelso, 2015].

Outline

- Background
- Motivation for Parameterized Matching
- Basic Problems
- Solutions
- Applications
- Conclusions

Conclusions

Conclusions

- Parameterized matching allows to find strings with similar structure.
- It has important applications in different areas such as software maintenance, image processing, computational biology, to name some.
- There has been extensive research for the last decades.
- New insights include the definition of new data structures, the extension to RNA matching and its application to solve graph isomorphism.

Bibliography (I)

- [Amir, 1994] **Amir, Farach and Muthukrishnan**. *"Alphabet dependence in parameterized matching"*. Information Processing Letters, 49(3):111–115, 1994.
- [Amir, 2003] **Amir, Aumann, Cole, Lewenstein and Porat**. *"Function matching: Algorithms, applications and a lower bound"*. Proc. of the 30th International Colloquium on Automata, Languages and Programming, 2003.
- [Amir, 2007] **Amir and Nor**. *"Generalized function matching"*. Journal of Discrete Algorithms, 2003.
- [Apostolico, 2007] **Apostolico Erdos and Lewenstain**. *"Parameterized matching with mismatches"*. Journal of Discrete Algorithms, 5(1):135-140, 2007.
- [Apostolico, 2008] **Apostolico and Giancarlo**. *"Periodicity and repetition in parameterized strings"*. Discrete Applied Mathematics, 156(9):1389-1398, 2008.
- [Baker, 1992] **Baker**. *"A program for identifying duplicated code"*. Computing Science and Statistics: Proc. of the 24th Symp. on the Interface, 1992.
- [Baker, 1993] **Baker**. *"A theory of parameterized pattern matching: algorithms and applications"*. Proc. of the 25th Annual ACM Symp. on Theory of Computing, 1993.
- [Baker, 1995] **Baker**. *"Parameterized pattern matching by boyer-moore type algorithms*. Proc. of the 6th Annual ACM-SIAM Symp. on Discrete Algorithms, 1995.
- [Baker, 1997] **Baker**. *"Parameterized duplication in strings: Algorithms and an application to software maintenance"*. SIAM Journal on Computing, 26(5):1343–1362, 1997.
- [Baker, 1999] **Baker**. *"Parameterized diff"*. Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, 1999.
- [Beal, 2012] **Beal and Adjeroh**. *"Parameterized longest previous factor"*. Theoretical Computer Science, 2012.

Bibliography (II)

- [Beal, 2012a] **Beal and Adjeroh**. *"Variations of the parameterized longest previous factor"*. Journal of Discrete Algorithms, 2012.
- [Beal, 2013] **Beal and Adjeroh**. *"The structural border array"*. Journal of Discrete Algorithms, 2013.
- [Beal, 2015] **Beal and Adjeroh**. *"Efficient pattern matching for rna secondary structures"*. Theoretical Computer Science, 2015.
- [Cole, 2004] **Cole and Hariharan**. *"Faster suffix tree construction with missing suffix links"*. SIAM Journal on Computing, 33(1):26-42, 2004.
- [Cordella, 2004] **Cordella, Foggia, Sansone and Vento**. *"A (sub) graph isomorphism algorithm for matching large graphs"*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(10):1367–1372, 2004.
- [Deguchi, 2008] **Deguchi, Bannai, Inenaga and Takeda**. *"Parameterized suffix arrays for binary strings"*. Proc. of Prague Stringology Conference, 2008.
- [Fredriksson, 2006] **Fredriksson and Mozgovoy**. *"Efficient Parmaeterized String Matching"*. Information Processing Letters, 100(3):91-96, 2006.
- [Hazay, 2004] **Hazay**. *"Parameterized matching"*. Master's thesis, Bar-Ilan University, 2004.
- [Hazay, 2007] **Hazay**. *"Approximate parameterized matching"*. ACM Transactions on Algorithms, 3(3):29, 2007.
- [I, 2009] **I, Deguchi, Bannai, Inenaga and Takeda**. *"Lightweight parameterized suffix array construction"*. Proc. of IWOCA, 2009.
- [I, 2009a] **I, Deguchi, Bannai and Takeda**. *"Verifying and enumerated parameterized border arrays"*. Theoretical Computer Science, 2009.
- [I, 2010] **I, Inenaga, Bannai, and Takeda**. *"Counting and verifying maximal palindromes"*. Proc. of SPIRE, 2010.

Bibliography (III)

- [Idury, 1996] **Idury and Schäffer**. *"Multiple matching of parameterized patterns"*. Theoretical Computer Science, 154(2):203-224, 1996.
- [Keller, 2009] **Keller, Kopelowitz and Lewenstein**. *"On the longest common parameterized subsequence"*. Theoretical Computer Science, 410(51):5347-5353, 2009.
- [Kosaraju, 1995] **Kosaraju**. *"Faster algorithms for the construction of parameterized suffix trees"*. Proc. of the 36th Annual Symp. on Foundations of Computer Science, IEEE, 1995.
- [Lee, 2008] **Lee, Mendivelso and Pinzón**. *"δγ-parameterized matching"*. Proc. of SPIRE, 2008.
- [Lee, 2011] **Lee, Na, Park**. *"On-line construction of parameterized suffix trees for large alphabets"*. Information Processing Letters, 111(5):201-207, 2011.
- [Mendivelso 2010] **Mendivelso**. *"Definition and solution of a new string searching variant termed δγ-parameterized matching"*. Master's thesis, Universidad Nacional de Colombia, 2010.
- [Mendivelso 2010] **Mendivelso, Lee and Pinzón**. *"Function matching under δγ- distances"*. Proc. of SPIRE, 2012.
- [Mendivelso 2013] **Mendivelso, Kim, Elnilkety, He, Hwang and Pinzón**. *"Solving graph isomorphism using parameterized matching"*. Proc. of SPIRE, 2013.
- [Mendivelso 2015] **Mendivelso**. *"The graph pattern matching problem through parameterized matching"*. PhD thesis, Universidad Nacional de Colombia, 2015.
- [Salmela, 2006] **Salmela and Tarhio**. *"Sublinear algorithms for parameterized matching"*. Proc. of CPM, 2006.
- [Shibuya, 2004] **Shibuya**. *"Generalization of a suffix tree for rna structural pattern matching".* Algorithmica 39(1):1-19, 2004.

Thank you !

Any questions?