Pattern Matching for Permutations

Stéphane Vialette

[2]CNRS & LIGM, Université Paris-Est Marne-la-Vallée, France

Permutation Pattern 2013, Paris

Outline

Pattern matching for permutations

Pattern containment / involvement / avoidance

A permutation π is said to **contain** another permutation σ, in symbols $\sigma \preceq \pi$, if there exists a subsequence of entries of π that has the same relative order as σ, and in this case σ is said to be a **pattern** of π.

Otherwise, π is said to **avoid** the permutation σ.

Example

A permutation contains the pattern 123 (resp. 321) if it has an increasing (resp. decreasing) subsequence of length 3.

Pattern matching for permutations

Two (deliberately vague) problems we are interested in

Pattern matching

Given two permutations π and σ (we may have constraints on π and/or σ), how fast can we decide whether σ is involved in π?

Common pattern

Given a collection $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of n permutations (we may have constraints on $\pi_1, \pi_2, \ldots, \pi_n$) and a "constraint" C, find the largest permutation σ that satisfies C and that is involved in every permutation in Π.

We may be interested in returning only the size of the largest common permutation.

Theorem ([Bose, Buss, Lubiw 98])

For two permutations π and σ, deciding whether $\sigma \preceq \pi$ is **NP***-complete.*

Remarks

- The problem is ascribed to H. Wilf in [Bose, Buss, Lubiw 98].
- Reduction from 3-Satisfiability.

Matching diagrams

Definition

A **matching diagram** is a graph G such that $\mathbf{V}(G)$ is equipped with a total order and $\mathbf{E}(G)$ is a perfect matching.

Restricted matching diagrams

- A matching diagram G is said to be **precedence-free** if there do not exist edges (i, j) and (k, ℓ) in G such that $i < j < k < \ell$ or $k < \ell < i < j$.
- A matching diagram G is said to be **crossing-free** if there do not exist edges (i, j) and (k, ℓ) in G such that $i < k < j < \ell$ or $k < i < \ell < j$.
- A matching diagram G is said to be **inclusion-free** if there do not exist edges (i, j) and (k, ℓ) in G such that $i < k < \ell < j$ or $k < i < j < \ell$.

Theorem ([Folklore])

Precedence-free matching diagrams of size $2n$ are in one-to-one correspondence with permutations of length n

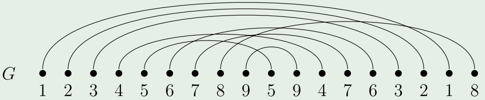
Remarks

- The vertices of G which are left endpoints of edges are labeled $\{1, 2, \ldots, n\}$.
- The vertices of G which are right endpoints of edges are labeled $\{n+1, n+2, \ldots, 2n\}$.
- The permutation π corresponding to G is defined by $\pi(j-n) = i$ if and only if $(i, j) \in \mathbf{E}(G)$.

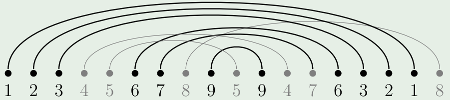
Pattern matching for separable patterns

Matching diagram

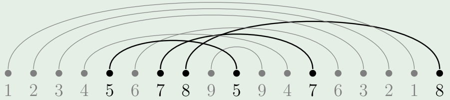
Examples



$\pi_G = 5\,9\,4\,7\,6\,3\,2\,1\,8$



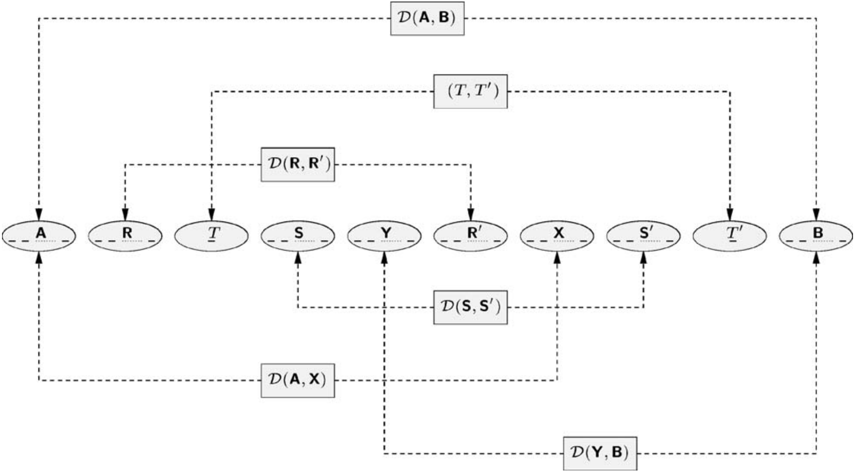
Decreasing subsequence 9 6 3 2 1



Increasing subsequence 5 7 8

Pattern matching for permutations

Proving hardness of pattern involvement using matching diagrams [V. 04]



Pattern matching for permutations

But I really need to answer my "*does σ occur in π?*" question !

Sage (combinat/permutation.py)

```
def has_pattern(self, patt):
  r"""
  Returns the boolean answering the question 'Is patt a pattern
  appearing in permutation p?'

  EXAMPLES::

      sage: Permutation([3,5,1,4,6,2]).has_pattern([1,3,2])
      True
  """
  p = self
  n = len(p)
  l = len(patt)
  if l > n:
      return False
  for pos in subword.Subwords(range(n),l):
      if to_standard(map(lambda z: p[z] , pos)) == patt:
          return True
  return False
```

Theorem ([Ahal, Rabinovich 08])

Let $\pi \in S_n$ and $\sigma \in S_m$. One can decide whether σ is involved in π in $O(n^{0.47m+o(m)})$ time.

Remarks

- The authors introduce two naturally defined (related) permutation complexity measures $C(\pi)$ and a somewhat finer $C^{\mathbf{T}}(\pi)$.
- They show that the algorithms run in time $O(n^{1+C(\sigma)})$ and $O(n^{2C^{\mathbf{T}}(\sigma)})$.
- In the general case, $C(\sigma) \leq 0.47k + o(m)$.

Theorem ([Bruner, Lackner 12])

Let $\pi \in S_n$ and $\sigma \in S_m$. One can decide whether σ is involved in π in $O(1.79^{\text{run}(\pi)})$ or $O^*((n^2/2\,\text{run}(\sigma))^{\text{run}(\sigma)})$ time.

Remarks

- Ahal and Rabinovich's $O(n^{0.47m+o(m)})$ time algorithm is $O(n^{1+\text{run}(\sigma)})$ time.
- Deciding whether σ is involved in π is **W[1]**-hard w.r.t. the parameter $\text{run}(\sigma)$.

Alternating permutations

Definition (Alternating permutations)

A permutation $\pi = \pi_1\,\pi_2\,\ldots\pi_n \in S_n$ is **alternating** if $\pi_1 > \pi_2 < \pi_3 > \ldots$, and **reverse alternating** if $\pi_1 < \pi_2 > \pi_3 < \ldots$.

Alternating permutations

Theorem ([Rizzi, V. 2013])

Deciding whether σ is involved in π is **NP***-complete even if both π and σ are alternating.*

Proof (Key idea).

Let $\pi \in S_n$ and $\sigma \in S_m$.

Define

$$\pi' = (2n+1)\ \pi_1\ (2n)\ \pi_2\ \ldots\ (n+2)\ \pi_k\ (n+1)$$
$$\sigma' = (2m+1)\ \sigma_1\ (2km)\ \sigma_2\ \ldots\ (m+2)\ \sigma_m\ (m+1)$$

Claim: σ is involved in π if and only if σ' is involved in π'. □

Finding a largest common permutations

Theorem ([Bose, Buss, Lubiw 98])

Given a collection $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of n permutations and a positive integer m, deciding whether there exists a permutation $\sigma \in S_m$ that is involved in every permutation in Π is **NP***-complete.*

Remarks

- The problem is at least as hard as deciding whether a given permutation σ is involved in another given permutation π.
- The problem is **NP**-complete for $n \geq 2$.
- This naturally reduces to an optimization problem.

Finding a largest common permutations

Definition

Let G be a precedence-free matching diagram.

- A **tower** is a set of pairwise nested edges. The **height** of G is defined to be the size of the maximum cardinality tower in G.
- A **staircase** is a set of pairwise crossing edges. The **depth** of G is defined to be the size of the maximum cardinality staircase in G.

The matching diagram G is called

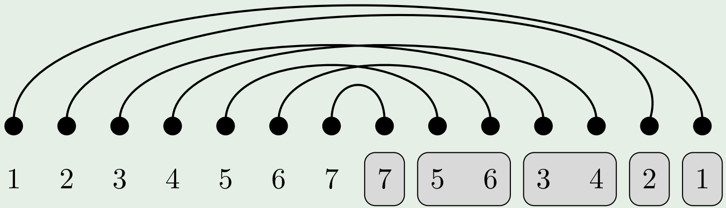
- a **tower of staircases** if any two maximal staircases do not share an edge (it is furthermore called **balanced** if all its maximal staircases are of equal cardinality),
- a **staircase of towers** if any two maximal towers do not share an edge (it is furthermore called **balanced** if all its maximal towers are of equal cardinality)

Finding a largest common permutations

Theorem ([Fertin, Hermelin, Rizzi, V. 10])

Let $G_1, G_2, \ldots, G_n$ be a collection of towers of staircases of depth at most 2, and ℓ be a positive integers. Deciding whether there exists a matching diagram of size ℓ that occurs in every tower of staircases G_i, $1 \leq i \leq n$, is **NP***-complete.*

Example



Finding a largest common permutations

Theorem ([Fertin, Hermelin, Rizzi, V. 10])

Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a collection of permutations of size at most m. The problem of computing the largest permutation that is involved in every permutation in Π is approximable within ratio $\sqrt{\mathbf{opt}}$ in $O(nm^{1.5})$ time, where **opt** *is the size of an optimal solution.*

This is the limit of our approach …

Lemma ([Fertin, Hermelin, Rizzi, V. 10])

For every collection $\Pi \subseteq S_n$, $n \in \mathbb{N}$ and $|\Pi| \leq 2^n$, there exists $\sigma \in S_K$, $K = \Omega(k^2)$, which avoids all permutations in Π.

A quick parenthesis

Theorem ([Fertin, Hermelin, Rizzi, V. 10])

*Let $\mathcal{G} = (G_1, G_2, \ldots, G_n)$ be a collection of linear graphs of maximum size m. There exists an algorithm with approximation ratio $O(\sqrt{\mathbf{opt} \log \mathbf{opt}})$ that runs in $O(nm^{3.5} \log m)$ time and returns a linear graph that occurs in every linear graph in $\mathcal{G}$, where **opt** is the size of an optimal solution*

Remarks

- Precedence-free matching diagrams remains the bottleneck.
- Any matching diagram of size n contains either a precedence-free matching diagram, an inclusion-free matching diagram, or a crossing-free matching diagram of size $\frac{\sqrt{17}-1}{8} n^{2/3}$.

Outline

Increasing patterns

Theorem ([Crochemore, Porat 10])

Let $\pi \in S_n$ and $\sigma = 1\ 2\ \ldots\ m$. On can decide whether σ is involved in π in $O(n \log\log m)$ time.

Remarks

- This improves the previous 30-year bound of $O(n \log m)$. (The algorithm also improves on the previous $O(n \log\log n)$ bound.)
- Having π to be sequence of integers (*i.e.*, multiple occurrences are allowed) does not change the result.
- A direct $O(n \log n)$ time solution for computing a longest increasing subsequence was proposed in [Fredman 75] ($n \log n - n \log\log n + O(n)$ comparisons in the worst case). The solution is optimal if the elements are drawn from an arbitrary set due to the $\Omega(n \log n)$ lower bound for sorting n elements.

Increasing patterns

Core algorithm

```
procedure LIS(π = π_1 π_2 ... π_n)
    Q ← EmptyPriorityQueue()
    k ← 0
    for i = 1 to n do
        Insert(Q, π_i)
        if Successor(Q, π_i) exists then
            delete(Q, Successor(Q, π_i))
        else
            k ← k + 1
        end if
    end for
    return(k)
end procedure
```

Increasing patterns

Example for $\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$

$\pi = \blacktriangledown\ 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = \emptyset$
$\pi = \overset{\blacktriangledown}{12}\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (12)$
$\pi = 12\ \overset{\blacktriangledown}{8}\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (8)$
$\pi = 12\ 8\ \overset{\blacktriangledown}{9}\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (8, 9)$
$\pi = 12\ 8\ 9\ \overset{\blacktriangledown}{1}\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (1, 9)$
$\pi = 12\ 8\ 9\ 1\ \overset{\blacktriangledown}{11}\ 6\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (1, 9, 11)$
$\pi = 12\ 8\ 9\ 1\ 11\ \overset{\blacktriangledown}{6}\ 7\ 2\ 10\ 4\ 5\ 3$	$Q = (1, 6, 11)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ \overset{\blacktriangledown}{7}\ 2\ 10\ 4\ 5\ 3$	$Q = (1, 6, 7)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ \overset{\blacktriangledown}{2}\ 10\ 4\ 5\ 3$	$Q = (1, 2, 7)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ \overset{\blacktriangledown}{10}\ 4\ 5\ 3$	$Q = (1, 2, 7, 10)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ \overset{\blacktriangledown}{4}\ 5\ 3$	$Q = (1, 2, 4, 10)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ \overset{\blacktriangledown}{5}\ 3$	$Q = (1, 2, 4, 5)$
$\pi = 12\ 8\ 9\ 1\ 11\ 6\ 7\ 2\ 10\ 4\ 5\ \overset{\blacktriangledown}{3}$	$Q = (1, 2, 3, 5)$

Pattern matching for 123-avoiding permutations

Theorem ([Guillemot, V. 09])

Let $\pi \in S_n$ and $\sigma \in S_m$ be two 123-avoiding permutations. One can decide whether σ is involved in π in $O(m^2\, n^6)$ time.

Theorem ([Guillemot, V. 09])

Let $\pi \in S_n$ and $\sigma \in S_m$. If σ is 123-avoiding and π is not, one can decide whether σ is involved in π in $O(m\, n^{4\sqrt{m}+12})$ time.

Remark

Deciding whether σ is involved in π is polynomial-time solvable if σ avoids 132, 312, 213 or 231 (since σ is clearly separable in this case).

Pattern matching for 123-avoiding permutations

Theorem ([Rizzi, V. 13])

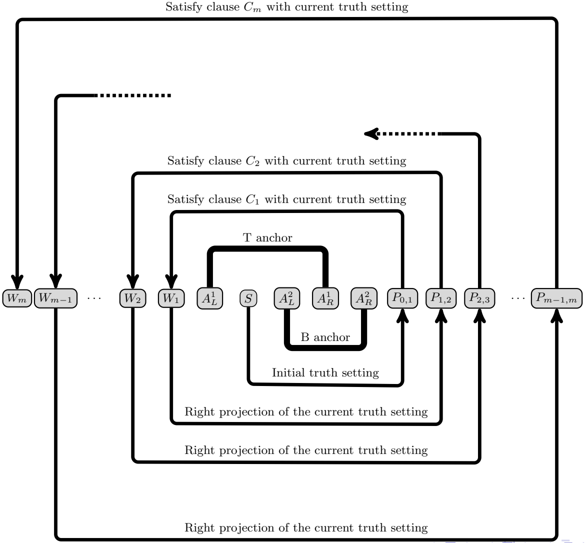
Let $\pi \in S_n$ and $\sigma \in S_m$. If σ is 123-avoiding and π is not, deciding whether σ is involved in π is **NP***-complete.*

Remarks

- If σ is 123-avoiding then its associated matching diagram does not contain three pairwise crossing edges.
- Reduction from 3-Satisfiability.

Pattern matching for 123-avoid permutations

The big picture



Vincular patterns

Definition

A **vincular pattern** of length m is a pair (σ, X) where σ is a permutation in S_m and $X \subseteq \{0\} \cup [m]$ is a set of adjacencies.

Definition

A permutation $\pi \in S_n$ contains the vincular pattern (σ, X) if there is a m-tuple $1 \leq i_1 \leq i_2 \leq \ldots \leq i_m \leq n$ such that the following three criteria are satisfied:

- $\mathrm{red}(\pi_{i_1}\pi_{i_2}\ldots\pi_{i_k}) = \sigma$,
- $i_{j+1} = i_j + 1$ for each $j \in X \setminus \{0, k\}$, and
- $i_1 = 1$ if $0 \in X$, and $i_k = n$ if $k \in X$.

Vincular patterns

Examples

Example of occurrences of vincular patterns in $\pi = 241563$:

Pattern	Occurrences in $\pi = 241563$
$(\sigma = 231, X = \emptyset)$	$241, 453, 463, 563$
$(\sigma = 231, X = \{1\})$	$241, 563$
$(\sigma = 231, X = \{2\})$	$241, 563$
$(\sigma = 231, X = \{0, 1, 2\})$	241
$(\sigma = 231, X = \{1, 2, 3\})$	563
$(\sigma = 231, X = \{3\})$	$453, 463, 563$

Vincular patterns

Theorem ([Bruner, Lackner 11])

Let π be a permutation and σ be a vincular pattern. Deciding whether σ is involved in π is $\mathbf{W[1]}$*-hard.*

Remarks

- Reduction from Independent Set, standard parameterization.
- Probably the first parameterized result in this area.

Outline

Size-3 patterns

Theorem

For $\sigma \in S_3$ *and* $\pi \in S_n$*, deciding whether* $\sigma \preceq \pi$ *is solvable in* $O(n)$ *time.*

Remarks

- Stack algorithm.
- Size-3 increasing patterns.

Size-4 patterns

Theorem ([Albert, Aldred, Atkinson, Holton. 01])

For $\sigma \in S_4$ and $\pi \in S_n$, deciding whether $\sigma \preceq \pi$ is solvable in $O(n \log n)$ time.

Remarks

- Symmetries reduce the bumber of cases that have to be considered to 7:

$$\sigma = 1234, 2134, 2341, 2314, 1324, 2143, 2413$$

- Tree-based data structures.

Size-4 patterns

Theorem ([Rizzi, V. 2013])

For $\sigma \in S_4$ and $\pi \in S_n$, deciding whether $\sigma \preceq \pi$ is solvable in $O(n \log \log n)$ time.

Remarks

- 7 algorithms (combination of point location like procedures) for 7 different cases.
- Van Emde Boas trees.
- Color based algorithms.

Outline

Separable permutations

Definition

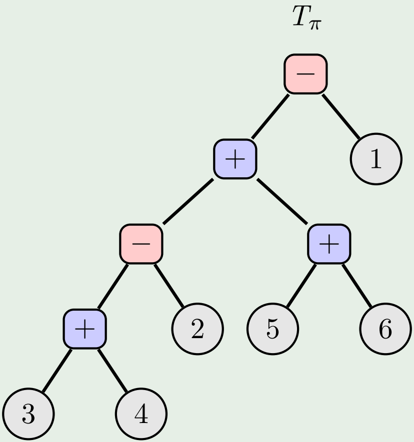
A permutation is separable if it contains neither 2413 nor 3142.

Remarks

- Enumerated by the Schröder numbers (sequence A006318 in OEIS).
- Permutations whose permutation graphs are cographs (*i.e.* P_4-free graphs).
- permutations that can be obtained from the trivial permutation 1 by *direct sums* and *skew sums*.

Separating trees

Example. $\pi = 3\,4\,2\,5\,6\,1$



Pattern matching for separable patterns

Theorem ([Ibarra 97])

Let $\pi \in S_n$ and $\sigma \in S_m$, σ begin separable. One can decide whether σ is involved in π in $O(mn^4)$ time and $O(mn^3)$ space.

Remarks

- Bottom up dynamic programming on the separating tree.
- $O(mn^6)$ time and $O(mn^4)$ space [Bose, Buss, Lubiw 98].

Pattern matching for separable patterns

Definition

The **bottom point** $\downarrow(s)$ of a match s of $\sigma(v)$ into S is the minimum value occurring in the sequence s.

The **upmost point** $\uparrow(s)$ of a match s of $\sigma(v)$ into S is the maximum value occurring in s.

Subproblems

For every node v of T_σ, every two $i, j \in [n]$ with $i \leq j$, and every upper bound $\text{ub} \in [n]$, we have the subproblem $\hat{\downarrow}_{v,i,j}[\text{ub}]$, where the semantic is the following.

$$\hat{\downarrow}_{v,i,j}[\text{ub}] \triangleq \max\{\downarrow(s) : s \text{ is a match of } \sigma(v) \text{ into } \pi[i,j] \text{ with } \uparrow(s) \leq \text{ub}\}.$$

Base

If v is a leaf of T_σ then

$$\hat{\downarrow}_{v,i,j}[\mathrm{ub}] := \max\{\pi[\iota] : \pi[\iota] \leq \mathrm{ub}, i \leq \iota \leq j\}.$$

Step

Let v_L and v_R be the left and right children of v.

- If v is a positive node of T_σ (*i.e.*, all elements in the interval associated to v_R are larger than all elements in the interval associated to v_L), then

$$\hat{\downarrow}_{v,i,j}[\text{ub}] := \max\{\hat{\downarrow}_{v_L,i,\iota-1}[\hat{\downarrow}_{v_R,\iota,j}[\text{ub}]] : i < \iota \leq j\}.$$

- If v is a negative node of T_σ (*i.e.*, all elements in the interval associated to v_R are smaller than all elements in the interval associated to v_L), then

$$\hat{\downarrow}_{v,i,j}[\text{ub}] := \max\{\hat{\downarrow}_{v_R,\iota,j}[\hat{\downarrow}_{v_L,i,\iota-1}[ub]] : i < \iota \leq j\}.$$

Key observation

For computing all the entries $\hat{\downarrow}_{v,\cdot,\cdot}[\cdot]$ for a node v with left and right children v_L and v_R, we only need the entries $\hat{\downarrow}_{v_L,\cdot,\cdot}[\cdot]$ and $\hat{\downarrow}_{v_R,\cdot,\cdot}[\cdot]$.

Policy

- All problems for a same node v are solved together.
- Their solution is maintained in memory until the problems for the parent of v have also been solved.
- At that point the memory used for node v is released.

Pattern matching for separable patterns

Reducing the memory consumption to $O(n^4 \log k)$

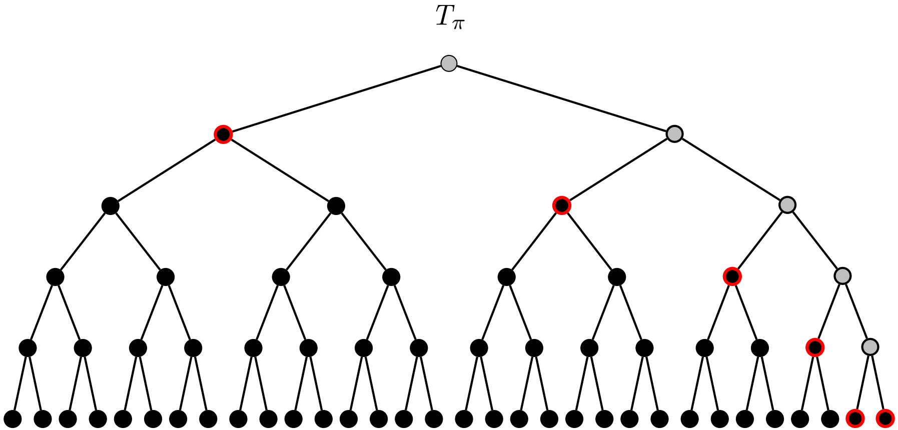
DFS Largest first

procedure DFS-LF(T)
  for every node u of T **do**
    color(u) $\leftarrow$ WHITE
  end for
  DFS-LF-Visit(T.root)
end procedure

procedure DFS-LF-VISIT(u)
  color[u] = GRAY
  for every child v of u in order of decreasing size **do**
    DFS-LF-Visit(v)
  end for
  color(u) $\leftarrow$ BLACK
end procedure

Pattern matching for separable patterns

DFS–Largest First for complete binary trees



Observation

If both π and σ are separable permutations, deciding whether σ is involved in π reduces to ordered and labelled tree inclusion (on the separating trees).

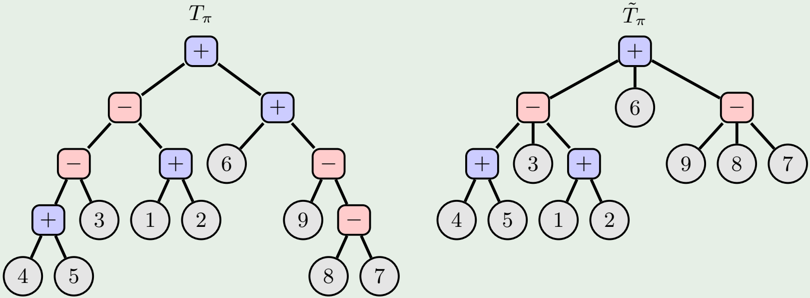
Remarks

- We cannot focus any longer on binary separating trees.
- Ordered and labelled tree inclusion is an important query primitive in XML databases.

Pattern matching for separable patterns

Both π and σ and separable permutations

Example



Pattern matching for separable patterns

Both π and σ and separable permutations

Theorem ([Bille, Gørtz. 11])

Let T and T' be two labelled ordered trees. Deciding whether T can be obtain from T' bu deleting nodes is solvable in $O(n_T)$ space and

$$O\left(\min\left\{\begin{array}{l} l_{T'}\, n_T \\ l_{T'}\, l_T \log\log n_T + n_T \\ \frac{n_T\, n_{T'}}{\log n_T} + n_T \log n_T \end{array}\right\}\right)$$

time, where n_T (resp. $n_{T'}$) denotes the number of node of T (resp. T') and l_T (resp. $l_{T'}$) denotes the number of leaves of T (resp. T').

Pattern matching for separable patterns

σ is a vincular separable pattern

Theorem

Let $\pi \in S_n$ and $\sigma \in S_m$, σ being a bivincular separable pattern. One can decide whether σ is involved in π in $O(mn^6)$ time and $O(mn^4)$ space.

Remarks

- We need to take care to both positional constraints and value constraints.
- HUGE dynamic programming.

Dynamic programming

For every node v of T_σ, for every two $i, j \in [n]$ with $i \leq j$, for every lower and upper bound $\mathrm{lb}, \mathrm{ub} \in [n]$ with $\mathrm{lb} \leq \mathrm{ub}$, and for every $Z \subseteq \{N, S, W, E\}$, where the semantic is the following

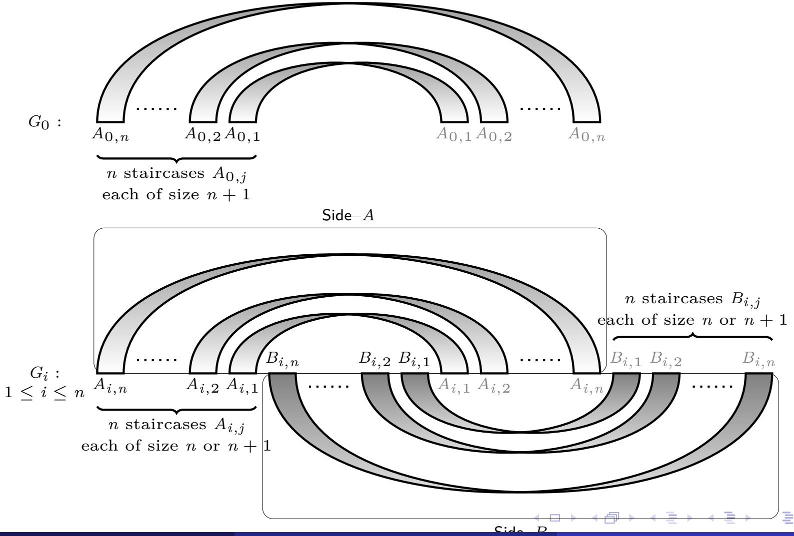
$$P^Z_{v,i,j,\mathrm{lb},\mathrm{ub}} \triangleq \begin{cases} \text{true} & \text{if is there exists a match of the bivincular pattern} \\ & (\sigma(v), X|\sigma(v), Y|\sigma(v)) \text{ in } \pi[i,j] \text{ with every element} \\ & \text{in the interval } [\mathrm{lb}, \mathrm{ub}], \text{ and} \\ & \quad - \text{ if } N \in Z \text{ then value ub occurs in the match,} \\ & \quad - \text{ if } S \in Z \text{ then value lb occurs in the match,} \\ & \quad - \text{ if } W \in Z \text{ then } \pi[i] \text{ is included in the match, and} \\ & \quad - \text{ if } E \in Z \text{ then } \pi[j] \text{ is included in the match.} \\ \text{false} & \text{otherwise.} \end{cases}$$

Known results

- $O(n^8)$ time algorithm for computing the largest common separable pattern that is involved in two permutations of size (at most) n, one of these two permutation being separable [Rossin, Bouvel. 06].
- $O(n^{6k+1})$ time and $O(n^{4k+1})$ space algorithm for computing the largest separable pattern that is involved in k permutations of size (at most) n [Bouvel, Rossin, V. 07].
- Computing the largest separable pattern that is involved in a collection of given separable permutations is **NP**-complete [Bouvel, Rossin, V. 07].

Pattern matching for separable patterns

Hardness of finding a largest common separable pattern



Pattern matching for separable patterns

Finding a largest separable pattern in a permutation: a simpler approach

Theorem ([Rizzi, V. 13])

Let $\pi \in S_n$. One can find the largest separable permutation that is involved in π in $O(n^6)$ time and $O(n^4)$ space.

Theorem ([Rizzi, V. 13])

Let $\pi_1, \pi_2 \in S_n$. One can find the largest separable permutation that is involved in π_1 and in π_2 in $O(n^{12})$ time and $O(n^8)$ space.

Theorem ([Rizzi, V. 13])

Let $\pi_1 \in S_n$ and $\pi_2 \in S_m$, π_2 being separable. One can find the largest separable permutation that is involved both in π_1 and in π_2 in $O(mn^6)$ time and $O(n^4 \log m)$ space.

Outline

Consecutive occurrences

Definition

A permutation π is said to consecutively contain another permutation σ if there exists a substring of entries of π that has the same relative order as σ, and in this case σ is said to be a consecutive pattern of π.

Example

$$\pi = 3 \quad 2 \quad 5 \quad \boxed{2 \quad 5 \quad 8 \quad 5 \quad 2} \quad 4 \quad 3 \quad 4$$

$$\sigma = 3 \quad 4 \quad 7 \quad 4 \quad 3$$

Consecutive patterns

Both π and σ are sequences

Lemma ([Kubica, Kulczyńskia, Radoszewskia, Ryttera, Waleń. 13])

*Let σ be a sequence of length m whose symbols can be sorted in $O(m)$ time. After $O(m)$ preprocessing time, for any sequence σ' one can answer queries of the form "**Assuming that** $\sigma[1 \ldots x] \approx \sigma'[1 \ldots x]$**, is** $\sigma[1 \ldots x+1] \approx \sigma'[1 \ldots x+1]$" in constant time.*

Theorem ([Kubica, Kulczyńskia, Radoszewskia, Ryttera, Waleń. 13])

Let π be a sequence of length n and σ be a sequence of length m. One can check in $O(n + m \log m)$ time whether π contains a substring which is order-isomorphic to σ.

The time complexity reduces to $O(n + m)$ if the symbols of σ can be sorted in $O(m)$ time.

Consecutive patterns

π is a permutation

Theorem ([Belazzougui, Pierrot, Raffinot, V. 13])

Let $\pi \in S_n$ and σ be a sequence of m distinct integers. Deciding whether σ is order-isomorphic to a substring of π can be done in $O(n + m \log\log m)$ time.

Remarks

- $O(m)$ space automaton.
- Forward automaton.
- Morris-Pratt automaton

Theorem ([Belazzougui, Pierrot, Raffinot, V. 13])

Let $\pi \in S_n$ and σ be a sequence of m distinct integers. Deciding whether σ is order-isomorphic to a substring of π can be done in $O(m \frac{\log m}{\log \log m} + \frac{n}{m} \frac{\log m}{\log \log m})$ *average time.*

Remarks

- Tree of all substrings of σ of length $3.5 \frac{\log m}{\log \log m}$.
- Algorithm is optimal on average.

Theorem ([Belazzougui, Pierrot, Raffinot, V. 13])

Let $\pi \in S_n$ *and* $\sigma_1, \sigma_2, \ldots, \sigma_d$ *be sequences of distinct integers of maximal length* r*. After* $O(m \log \log r)$ *preprocessing time, one can search for substrings of* π *that are order-isomorphic to* $\sigma_1, \sigma_2, \ldots, \sigma_d$ *in randomized* $O(nt)$ *time, where* $t = \min(\log \log n, \sqrt{\frac{\log r}{\log \log r}}, d)$.

Definition

Let $\pi = \pi_1 \; \pi_2 \; \ldots \; \pi_n$ be a sequence of length n over an integer alphabet (polynomially bounded in terms of n). Define:

$$\mathsf{prev}_<(\pi, i) = |\{j : j < i \text{ and } \pi_j < \pi_i\}|$$
$$\mathsf{prev}_=(\pi, i) = |\{j : j < i \text{ and } \pi_j = \pi_i\}|$$

Codes of positions and strings are defined by:

$$\phi(\pi, i) = (\mathsf{prev}_<(\pi, i), \mathsf{prev}_=(\pi, i))$$
$$\mathsf{code}(\pi) = (\phi(\pi, 1), \phi(\pi, i), \ldots, \phi(\pi, n))$$

Finally, define the family of sequences:

$$\mathsf{SuffCodes}(\pi) = \{\mathsf{code}(\mathsf{suff}_1(\pi))\,\#, \mathsf{code}(\mathsf{suff}_2(\pi))\,\#, \ldots, \mathsf{code}(\mathsf{suff}_n(\pi))\,\}$$

Example. $\pi = 6\,8\,2\,0\,7\,9\,3\,1\,4\,5$

Suffixes of π

6	8	2	0	7	9	3	1	4	5
	8	2	0	7	9	3	1	4	5
		2	0	7	9	3	1	4	5
			0	7	9	3	1	4	5
				7	9	3	1	4	5
					9	3	1	4	5
						3	1	4	5
							1	4	5
								4	5
									5

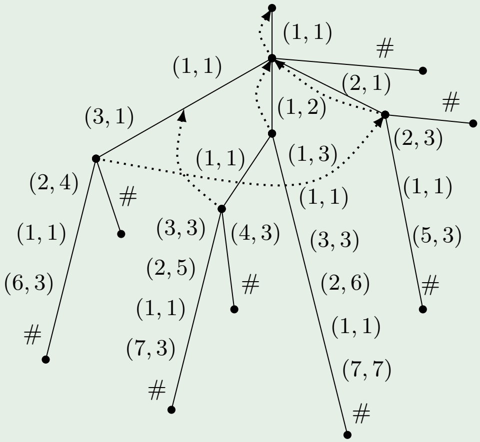
SuffCodes(π)

0	1	0	0	3	5	2	1	4	5	#
	0	0	0	2	4	2	1	4	5	#
		0	0	2	3	2	1	4	5	#
			0	1	2	1	1	3	4	#
				0	1	0	0	2	3	#
					0	0	0	2	3	#
						0	0	2	3	#
							0	1	2	#
								0	1	#
									0	#

Consecutive patterns

Order-preserving suffix trees

The uncompacted trie of $\pi = 6\ 8\ 2\ 0\ 7\ 9\ 3\ 1\ 4\ 5$



Theorem ([Crochemore, et al. 2013])

The order-preserving suffix tree of a sequence of length n can be constructed in $O(\frac{n \log n}{\log \log n})$ randomized time.

Theorem ([Crochemore, et al. 2013])

Assume we are given an order-preserving suffix tree for a sequence π of length n.

Given a pattern σ of length m, one can check if σ is a substring of π in $O(\frac{m \log n}{\log \log n})$ time and report in all occurrences in $O(\frac{m \log n}{\log \log n} + occ)$, where occ is the number of occurrences.

Definition

A sequence uv is called an **order-preserving square** (op-square) if $u \approx v$.

Lemma ([Crochemore, et al. 2013])

The sequence $\pi[i \ldots i+2k-1]$ *is an op-square if and only if the LCA of the leaves corresponding to* suff_i *and* suff_{i+k} *in the order-preserving suffix tree of* π *has depth at least* k*.*

Theorem ([Crochemore, et al. 2013])

All op-squares in a sequence π *of length* n *can be computed in* $O(n \log n + occ)$ *time, where occ is the total number of occurrences of op-squares.*

Outline

Confining the combinatorial explosion to σ

For $\pi \in S_n$ and $\sigma \in S_k$, can we decide whether σ is involved in π in $f(k)\ n^{O(1)}$ time, where f is an arbitrary function depending only on k?

If yes, how large has to be the associated kernel?

Remarks

- Deciding whether σ is involved in π is $\mathbf{W}[\mathbf{1}]$-complete for vincular patterns [Bruner, Lackner 11],
- Deciding whether σ is involved in π is $\mathbf{W}[\mathbf{1}]$-complete for 2-coloured σ and π [Guillemot, V. 09].

Some open problems (my point of view)

Approximate occurrences

Approximate order-preserving matching

What about “approximate” order-preserving matching?

Remarks

- Probably more suited for consecutive patterns!?
- Probably more suited for sequences!?
- But what is (should be) an approximate order-preserving matching?

Pattern involvement for $O(1)$ size pattern

What about the complexity of deciding whether σ is involved in π for $|\sigma| = 5, 6, \ldots$?

Remarks

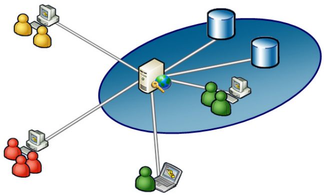
- Is there a generic approach for this task?
- What jump in complexity should we expect going from $|\sigma| = i$ to $|\sigma| = i + 1$?

Further lines of research

- Pattern matching for compressed permutations.
- Suffix arrays viewed as permutations, Burrows-Wheeler permutations, . . .
- Combinatorics on words.
- Comparative genomics.
- . . .

Open Combinatorial Structures (OCS)

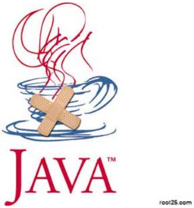
A database structured by subjects for storing combinatorial structures seen in everyday practices.



- Collaborative database.
- Automatic data acquisition.
- Open Database License (ODbL).
- Data indexing.

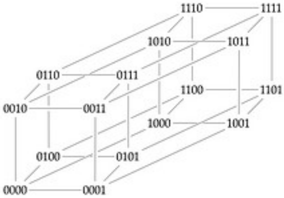
Funded by Université Paris-Est Marne-la-Vallée.

Open Combinatorial Structures (OCS)



Patched JVM

Storing and organizing data



Data visualization