Optimal Enumeration of Regular Pattern Matches

Formal Language and Automata Theory Seminars

Florin Manea

Institut für Informatik
Georg-August-Universität Göttingen

8.4.2026

NOT one FLAT World

A Formal Languages and Automata Theory World?

- Formal Languages and Automata Theory (FL/AT) slipped out of their main role in Theoretical Computer Science (e.g., Track B of ICALP/STACS is much smaller than Track A, smaller presence at LICS, very seldom papers in STOC, FOCS; small DLT, CIAA, DCFS) but...

A Formal Languages and Automata Theory World?

- Formal Languages and Automata Theory (FL/AT) slipped out of their main role in Theoretical Computer Science (e.g., Track B of ICALP/STACS is much smaller than Track A, smaller presence at LICS, very seldom papers in STOC, FOCS; small DLT, CIAA, DCFS) but...
- ... they still remain a central topic, and not only in theory! E.g., some *highlight* FL/AT results in the main TCS conferences, and recent papers in POPL, CAV, NeurIPS, PODS.

A Formal Languages and Automata Theory World?

- Formal Languages and Automata Theory (FL/AT) slipped out of their main role in Theoretical Computer Science (e.g., Track B of ICALP/STACS is much smaller than Track A, smaller presence at LICS, very seldom papers in STOC, FOCS; small DLT, CIAA, DCFS) but...
- ... they still remain a central topic, and not only in theory! E.g., some *highlight* FL/AT results in the main TCS conferences, and recent papers in POPL, CAV, NeurIPS, PODS.
- So: research in FL/AT remains an interesting, important topic, especially when there is a good motivation behind it and a good way to communicate and integrate it in hotter research areas!

Today

This talk:

An attempt to transfer *cool* stringology (or, to a certain extent, FL/AT) results to database theory.

Enumerating Regular Pattern Matches

Optimal Enumeration of Regular Pattern Matches
Paweł Gawrychowski, Florin Manea, Paul Sarnighausen-Cahn, Stefan Siemer:
PODS 2026

Introduction

Enumeration Algorithms

Enumeration algorithms are designed to compute one-by-one the complete solutions space of a given computational problem, rather than: deciding whether a solution exists; computing a single solution; computing the entire solution set and outputting it at the end of the computation.

- **Input**: an instance of a problem (e.g., on strings/words, graphs, logical formulas).
- **Output**: the complete set of solutions (one-by-one, no repetition).
- Examples:
  - *Generating* substrings of a text.
  - *Enumerating* paths in a graph.
  - *Listing* satisfying assignments of a formula.
  - *Querying* a database (`SELECT * FROM table WHERE condition_holds`).

Enumeration Framework

- Maintain an object **Sol** that represents the current solution (e.g., path-segments, pointers, arrays).
- A procedure **report** signals that **Sol** contains fresh solution.
- We analyse the algorithm in two phases:
  1. Preprocessing is the time before the first call of **report**.
  2. Delay is the time between two calls of **report**.

Enumeration Framework – Motivation

[D. E. Knuth: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms]

We are interested in having the solutions to our combinatorial problem momentarily present in a data structure, allowing other programs to examine them.

Enumeration Framework – Motivation

[D. E. Knuth: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms]

We are interested in having the solutions to our combinatorial problem momentarily present in a data structure, allowing other programs to examine them.

- Control over exploration of solution space.
  Example: explore solution space until some property holds.
  (`SELECT * FROM table WHERE condition_holds LIMIT count`).
- Streamlining processes.
  Example: start new threads with freshly generated results.
- Pipelining results into various tasks.
  Example: ranking, filtering, or optimization procedures.

Enumeration in Python

We use the keywords `yield` and `next`.

```
1  def enumerate_suffix_matches(text, pattern):
2      i = 0
3      while i < len(text):
4          sol = lin_search(text[i:], pattern)
5          if sol == -1: break
6          sol += i
7          yield sol
8          i = sol+1
9
10 enum = enumerate_suffix_matches("bananananana", "nan")
11 start = next(enum)
12 print(f"[{start}:{start+len(pattern)-1}]")
13 for start in enum:
14     print(f"[{start}:{start+len(pattern)-1}]")
```

Enumerating Regular Pattern Matches

Matching Patterns with Variables

$\mathcal{X}$ – **variables**, e.g., $\mathcal{X} := \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$
$PAT_\Sigma := (\mathcal{X} \cup \Sigma)^+$ – **patterns**, e.g., $\mathtt{a}\mathtt{x}_1\mathtt{a}\mathtt{b}\mathtt{a}\mathtt{x}_1\mathtt{x}_2\mathtt{b}$.

Exact Matching Problem: `Match`
Input: Pattern $\alpha \in PAT_\Sigma$, text $T \in \Sigma^*$.
Task: Exists substitution h with $h(\alpha) = T$?

Example

$\alpha = \mathtt{x}_1\mathtt{x}_1\mathtt{b}\mathtt{a}\mathtt{b}\mathtt{x}_2\mathtt{x}_2$, $\quad T = \mathtt{aaaababbb}$

$h(\mathtt{x}_1) = \mathtt{aa}$
$h(\mathtt{x}_2) = \mathtt{b}$

$h(\alpha) = \mathtt{aaaa}\mathtt{b}\mathtt{a}\mathtt{b}\mathtt{bb}$
$T = \mathtt{aaaababbb}$

Matching Patterns with Variables

$\mathcal{X}$ – **variables**, e.g., $\mathcal{X} := \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$; Σ alphabet of terminals;
$PAT_\Sigma := (\mathcal{X} \cup \Sigma)^+$ – **patterns**, e.g., $\mathtt{a}\mathtt{x}_1\mathtt{a}\mathtt{b}\mathtt{a}\mathtt{x}_1\mathtt{x}_2\mathtt{b}$.

Exact Matching Problem: `Match`
Input: Pattern $\alpha \in PAT_\Sigma$, text $T \in \Sigma^*$.
Task: Exists substitution h with $h(\alpha) = T$?

Example (Classical Pattern Matching/ Subsequence Matching)

Find $P \in \Sigma^*$ in $T \in \Sigma^*$: match $\alpha = \mathtt{x}_1 P \mathtt{x}_2$ to $T = \mathtt{aaaababbb}$
Find subsequence v (where $|v| = k, v = v_1 \cdots v_k$) in T: match
$\alpha = \mathtt{x}_0 \mathtt{v}_1 \mathtt{x}_1 \mathtt{v}_2 \mathtt{x}_2 \ldots \mathtt{x}_{k-1} \mathtt{v}_k \mathtt{x}_k$ to T.

Complexity:

NP-hard in general. Parameterised complexity thoroughly investigated by Schmid and co-authors (see also our survey from WORDS 2019).

Matching Patterns with Variables: Information Extraction

- `Match`: task which requires compiling structured information from an input text document $\rightarrow$ textual *information extraction* task.
- Match between pattern α (query) and text T (document) maps every variable occurring in α to a substring of T $\rightarrow$ *extracts and stores in each variable* a substring (aka span) of T.

Matching Patterns with Variables: Information Extraction

- `Match`: task which requires compiling structured information from an input text document $\rightarrow$ textual *information extraction* task.
- Match between pattern α (query) and text T (document) maps every variable occurring in α to a substring of T $\rightarrow$ *extracts and stores in each variable* a substring (aka span) of T.

The general picture:

- Pattern: particular regular expression with capture variables (*variable regex*)
- `Match`: particular case of variable regex matching. Heavily investigated in relation to database theory (the study of document spanners).
- Document spanners: information extraction framework. Main idea: use regular languages to locate the data to be extracted, and variables to store this data.

See, e.g.:
Ronald Fagin, Benny Kimelfeld, Frederick Reiss, Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. J. ACM (2015)
Markus L. Schmid, Nicole Schweikardt: Document Spanners - A Brief Overview of Concepts, Results, and Recent Developments. PODS 2022.
Relevant other works by Antoine Amarilli, Liat Peterfreund, Cristian Riveros, Dominik Freydenberger, Markus L. Schmid, Nicole Schweikardt, etc.

Challenges

- `Match` is computationally hard in general.
- To fit the information extraction framework: we need to enumerate all matches of a pattern (more general: variable regex) to a document. This seems even harder!

Challenges

- `Match` is computationally hard in general.
- To fit the information extraction framework: we need to enumerate all matches of a pattern (more general: variable regex) to a document. This seems even harder!

Possible solution: work with restricted classes of patterns / variable regexes (queries)

- `Match` tractable for structurally restricted pattern-classes [survey, WORDS 2019].
- Enumerating the matches of a variable regex to a document can be done with delay constant in the size of the document and polynomial in the size of the query, after a preprocessing requiring time linear in the size of the document and polynomial in the size of the query, if the variable regex is representable by a sequential variable-set automaton. [Amarilli et al., ACM TODS 2021]

Challenges

- `Match` is computationally hard in general.
- To fit the information extraction framework: we need to enumerate all matches of a pattern (more general: variable regex) to a document. This seems even harder!

Possible solution: work with restricted classes of patterns / variable regexes (queries)

- `Match` tractable for structurally restricted pattern-classes [survey, WORDS 2019].
- Enumerating the matches of a variable regex to a document can be done with delay constant in the size of the document and polynomial in the size of the query, after a preprocessing requiring time linear in the size of the document and polynomial in the size of the query, if the variable regex is representable by a sequential variable-set automaton. [Amarilli et al., ACM TODS 2021]

Question: can we get *linear preprocessing + constant delay enumeration (combined complexity)* for some relevant/non-trivial class of variable regexes?
Reality check: we should not aim too high! Even regex matching is harder.

Regular Patterns

Definition

$$\alpha \in \mathtt{Reg} \text{ if } \alpha = (\prod_{i=0}^{k} w_i x_i) w_{k+1}, \text{ with } w_i \in \Sigma^\star.$$

Relevant, non-trivial:

- Appear in algorithmic learning theory, in relation to the computation of descriptive patterns for sets of words [Shinohara, 1982].
- Particular class of variable regex representable by a sequential variable-set automaton.

Regular Pattern

Definition

$$\alpha \in \texttt{Reg} \text{ if } \alpha = (\prod_{i=0}^{k} w_i x_i) w_{k+1}, \text{ with } w_i \in \Sigma^\star.$$

Pattern Matching for Regular Patterns ($\texttt{Match}_{\texttt{Reg}}$)

Input: Regular pattern $\alpha = (\prod_{i=0}^{k} w_i x_i) w_{k+1}$ and text T.

Task: Decide if substitution h with $h(\alpha) = T$ exists.

Example

$\alpha = \texttt{ab}\underline{x_1}\texttt{cb}\underline{x_2}\texttt{bab}\underline{x_3}\texttt{bab}$, $T = \texttt{abbcbcbbababababbab}$

$h(\texttt{x}_1) = \texttt{b}$

$h(\texttt{x}_2) = \texttt{cb}$

$h(\texttt{x}_3) = \texttt{abab}$

$h(\alpha) = \texttt{abbcbcbbababababbab}$

$T = \texttt{abbcbcbbababababbab}$

Valid Embedding

Representing valid substitution as valid embedding of $w_0, \ldots, w_{k+1}$ in T.

Valid Embedding

Representing valid substitution as valid embedding of $w_0, \ldots, w_{k+1}$ in T.

w_0 must match $T[1 : |w_0|]$, w_{k+1} must match $T[n - |w_{k+1}| + 1 : n]$:

Example

$\alpha = \underline{\texttt{ab}}\texttt{x}_1\texttt{cbx}_2\texttt{babx}_3\underline{\texttt{bab}}$, $T = \texttt{abbcbcbbabababbbab}$

$w_0 = T[1 : 2]$ $\qquad$ $h(\alpha) = \texttt{ab}\texttt{bcbcbbabababb}\texttt{bab}$

$w_3 = T[15 : 17]$ $\qquad$ $T = \texttt{abbcbcbbabababbbab}$

Valid Embedding

Representing valid substitution as valid embedding of $w_0, \ldots, w_{k+1}$ in T.

w_0 must match $T[1 : |w_0|]$, w_{k+1} must match $T[n - |w_{k+1}| + 1 : n]$:

Example

$\alpha = \underline{\texttt{ab}}\texttt{x}_1\texttt{cbx}_2\texttt{babx}_3\underline{\texttt{bab}}$, $T = \texttt{abbcbcbbababababbab}$

$$w_0 = T[1:2] \qquad h(\alpha) = \texttt{ab}\texttt{bcbcbbababab}\texttt{bab}$$
$$w_3 = T[15:17] \qquad T = \texttt{abbcbcbbababababbab}$$

All w_i must occur one after another (non-overlapping) in $T' = T[|w_0| + 1 : n - |w_{k+1}|]$:

Example

$\alpha = \texttt{x}_1\underline{\texttt{cb}}\texttt{x}_2\underline{\texttt{bab}}\texttt{x}_3$, $T' = \texttt{bcbcbbababab}$

$$w_1 = T'[2:3] \qquad h(\alpha) = \texttt{b}\texttt{cb}\texttt{cbbab}\texttt{abab}$$
$$w_2 = T'[6:8] \qquad T = \texttt{bcbcbbababab}$$

Valid Embedding

- Assume that α starts and ends with a variable,
- keep positions $(j_1, \ldots, j_k)$, such that $T[j_i : j_i + |w_i| - 1] = w_i$, and
- $j_i + |w_i| - 1 < j_{i+1}$.

Pattern Matching for Regular Patterns ($\texttt{Match}_{\texttt{Reg}}$)

Input: Regular pattern $\alpha = (x_0 \prod_{i=1}^{k} w_i x_i)$ and text T.

Task: Compute a valid embedding $(j_1, \ldots, j_k)$ for $w_1, \ldots, w_k$ in T.

This is solved in linear time $\mathcal{O}(|T| + |\alpha|)$ by a greedy strategy using linear substring search (e.g., KMP).

Enumerating Regular Pattern Matches

Enumeration of Regular Patterns Matches ($\texttt{EnumMatch}_{\texttt{Reg}}$)

Input: Regular pattern $\alpha = (x_0 \prod_{i=1}^{k} w_i x_i)$ and text T.

Task: Report all valid embeddings $(j_1, \ldots, j_k)$ for $w_1, \ldots, w_k$ in T.

In the following we denote by S the set of all valid embeddings.

`Sol` is an array containing the current embedding $(j_1, \ldots, j_k)$.

Example

$\alpha = \texttt{x}_1\underline{\texttt{cb}}\texttt{x}_2\underline{\texttt{bab}}\texttt{x}_3$, $T' = \texttt{bcbcbbababa}$

$\texttt{Sol} = [2, 8]$ $\quad h_1(\alpha) = \texttt{bcbcbbababa}$

$\texttt{Sol} = [2, 6]$ $\quad h_2(\alpha) = \texttt{bcbcbbababa}$

$\texttt{Sol} = [4, 8]$ $\quad h_3(\alpha) = \texttt{bcbcbbababa}$

$\texttt{Sol} = [4, 6]$ $\quad h_4(\alpha) = \texttt{bcbcbbababa}$

Left-/right canonical embeddings

- A *left-canonical embedding* $(\ell_1, \ldots, \ell_i, \ldots, \ell_k)$ is a valid embedding where each ℓ_i is the minimal possible among all valid embeddings.
- $(r_1, \ldots, r_i, \ldots, r_k)$ is a *right-canonical embedding* if each r_i is the maximal possible among all valid embeddings.
- They can be computed in $\mathcal{O}(n)$ time, greedily.
- For all valid embeddings $(j_1, \ldots, j_i, \ldots j_k)$, we have that $j_i \in [\ell_i : r_i]$, for all $i \in [k]$.
- Every occurrence of w_i starting in $T[\ell_i : r_i]$ belongs to a valid embedding.
- $O_i \coloneqq \{p \in [\ell_i : r_i] \mid T[p : p + |w_i| - 1] = w_i\}$ the set of valid occurrences of w_i.
- Useful observation: $\sum_{i=1}^{k}(|O_i| - 1) \leq \min\{kn, |S|\}$.
  Indeed: $(\ell_1, \ldots, \ell_{i-1}, j_i, r_{i+1}, \ldots, r_k)$, for all $j_i \in O_i$.

Enumerating all embeddings naïvely

Naïve, recursive computation of all solutions:

Lemma

Given sorted sets $O_1, \ldots, O_k$, all valid embeddings can be enumerated in $\mathcal{O}(|S|k)$ time.

Idea: Assume that $j_1, \ldots, j_{i-1}$ have already been fixed (and saved in `Sol`), call the algorithm recursively to set j_{i+1}.
In the call, for the current index i, we iterate in decreasing order over the sorted set $O_i^x := \{j_i \mid j_i \in O_i, j_i \geq x\}$ where $x = j_{i-1} + |w_{i-1}|$.
Whenever $i = k$, `report` the current array `Sol`.

Enumerating all embeddings

Lemma

Provided data structures allowing us to:

- *enumerate, for a given i and a number x, with constant delay, all elements of O_i greater or equal to x, starting with r_i,*
- *retrieve the largest two elements in O_i in $\mathcal{O}(1)$ time,*

all valid embeddings can be enumerated, after an additional preprocessing running in $\mathcal{O}(k)$ time, with constant delay. The space used, additionally to that allocated for the provided data structures, is $\mathcal{O}(k)$.

Enumerating all embeddings

Lemma

Provided data structures allowing us to:

- *enumerate, for a given i and a number x, with constant delay, all elements of O_i greater or equal to x, starting with r_i,*
- *retrieve the largest two elements in O_i in $\mathcal{O}(1)$ time,*

all valid embeddings can be enumerated, after an additional preprocessing running in $\mathcal{O}(k)$ time, with constant delay. The space used, additionally to that allocated for the provided data structures, is $\mathcal{O}(k)$.

Idea:

- Define a recursive function `enum`; as opposed to the previous approach, each recursive call will report a valid embedding.
- When the function is called, the prefix $\texttt{Sol[1]}, \ldots, \texttt{Sol[i} - \texttt{1]}$ has been already fixed, $\texttt{Sol[i]} = r_i, \ldots, \texttt{Sol[k]} = r_k$ holds, and there are at least two solutions with the respective prefix.

Enumerating all embeddings

```
1  for i ∈ [k] do
2      Sol[i] ← r_i;
3  report;
4  if jump(1) ≠ k + 1 then
5      enum(jump(1), false);
6  function enum(i, b):
7      for x ∈ O_i \ {r_i} s.t. x ≥ j_{i−1} + |w_{i−1}| do
8          Sol[i] ← x;
9          if not b then
10             report;
11         if the largest two elements of O_{i+1} are ≥ x + |w_i| then
12             enum(i + 1, not b);
13         else
14             if jump(i) ≠ k + 1 then
15                 enum(jump(i), not b);
16         if b then
17             report;
18     Sol[i] ← r_i;
19     if jump(i) = k + 1 then
20         return;
21     else
22         enum(jump(i), b);        ▷ tail recursion
```

Enumerating all embeddings

Thus, we need data structures allowing us to:

- enumerate, for a given i and a number x, with constant delay, all elements of O_i greater or equal to x, starting with r_i,
- retrieve the largest two elements in O_i in $\mathcal{O}(1)$ time.

Suffix Array

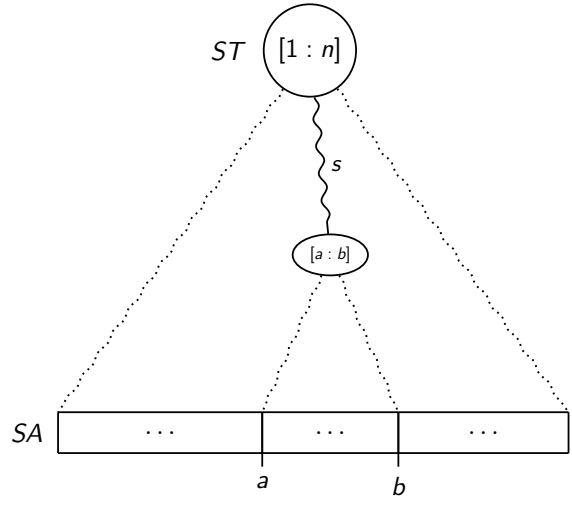
	bananaban								
Index	1	2	3	4	5	6	7	8	9
Char	b	a	n	a	n	a	b	a	n
SA	6	8	4	2	7	1	9	5	3
	aban	an	anaban	ananaban	ban	bananaban	n	naban	nanaban

Finding all occurrences of $w_1, \ldots, w_k$

Suffix Tree (ST) on top of a Suffix Array (SA).

Lemma

Let $T[1:n]$ be a text and let $w_1, \ldots, w_k$ be strings with $\sum_{i=1}^{k} |w_i| \leq n$. Then, in total $\mathcal{O}(n)$ time, we can compute, for every $i \in [k]$, the range $SA[a_{w_i} : b_{w_i}]$ containing exactly the positions j such that $T[j : j + |w_i| - 1] = w_i$.



Two sided range reporting

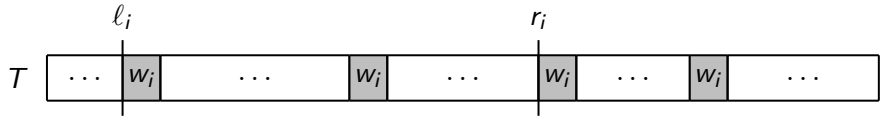


Figure: We want all occurrences of w_i starting in $T[\ell_i : r_i]$

We can not directly do this in constant time per occurrence.

Three sided range reporting

Let $T[1:n]$ be a text and let $w_1, \ldots, w_k$ be strings with $\sum_{i=1}^{k} |w_i| \leq n$. Then, after $\mathcal{O}(n)$ preprocessing, given an index i and a position x, we can enumerate all occurrences of w_i starting at position x or later with $\mathcal{O}(1)$ delay.

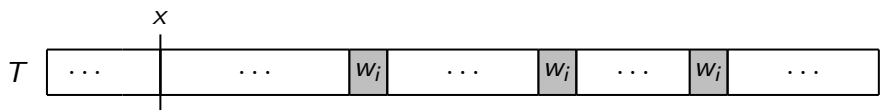


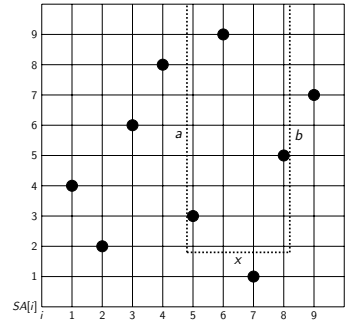
Figure: All occurrences of w in T starting at position x or later.

Three sided range reporting

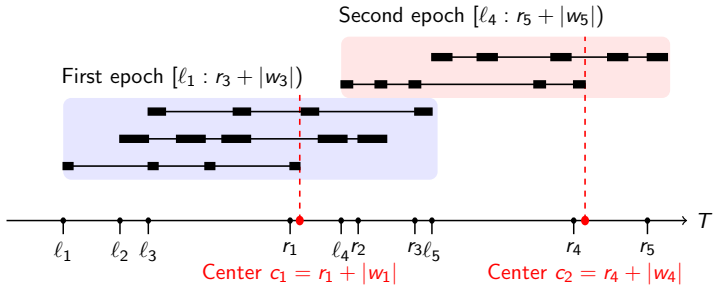
Lemma

A permutation of $[n]$ *stored in an array* $A[1:n]$ *can be preprocessed in* $\mathcal{O}(n)$ *time for the following queries: given integers* $a, b, x \in [n]$ *with* $a \leq b$*, enumerate all elements of the set* $\{i \in [a:b] \mid A[i] \geq x\}$ *with worst-case constant delay.*

- Report all elements in the range $A[a:b]$ that are bigger than x.
- $\texttt{RMaxQ}(a, b) = p \geq x$.
- Repeat:
  - $\texttt{RMaxQ}(a, p-1)$
  - $\texttt{RMaxQ}(p+1, b)$



Epochs



- $e_0 = \emptyset$. For $g \geq 1$, assume that $[\ell_1 : r_1 + |w_1|), \ldots, [\ell_{a-1} : r_{a-1} + |w_{a-1}|)$ are contained in epochs $e_1, \ldots, e_{g-1}$ and $[\ell_a : r_a + |w_a|)$ is not contained in any of these epochs. Then the next epoch e_g is defined as the interval $[\ell_a : r_b + |w_b|)$, such that $\ell_b < r_a + |w_a|$ and $\ell_{b+1} \geq r_a + |w_a|$ (or $b = k$). Thus, e_g contains $[\ell_a : r_a + |w_a|), \ldots, [\ell_b : r_b + |w_b|)$.
- For $g \geq 1$, let $\text{left}_g = \min e_g$ and $\text{right}_g = \max e_g$, and $c_g = r_a + |w_a|$. We call, respectively, $[\text{left}_g : c_g - 1]$ the left and $[c_g + 1 : \text{right}_g]$ the right part of e_g.

Epochs

Lemma
$\sum_{g=1}^{v}(\text{right}_g - \text{left}_g) = \mathcal{O}(n)$.

Use three sided range reporting for ℓ_i to center (L_i), and from center to r_i (R_i). For occurrences around the center use linear search (M_i). $O_i := L_i \cup M_i \cup R_i$.

Lemma
The epochs can be computed in $\mathcal{O}(k)$ time and can be preprocessed in $\mathcal{O}(n)$ time for the following queries:

1. *given an index $i \in [k]$, enumerate the elements of L_i and R_i with constant delay.*
2. *given an index $i \in [k]$, enumerate the elements of M_i with constant delay.*

Epochs

Lemma
$\sum_{g=1}^{v}(\text{right}_g - \text{left}_g) = \mathcal{O}(n)$.

Use three sided range reporting for ℓ_i to center (L_i), and from center to r_i (R_i). For occurrences around the center use linear search (M_i). $O_i := L_i \cup M_i \cup R_i$.

Lemma
The epochs can be computed in $\mathcal{O}(k)$ time and can be preprocessed in $\mathcal{O}(n)$ time for the following queries:

1. *given an index $i \in [k]$, enumerate the elements of L_i and R_i with constant delay.*
2. *given an index $i \in [k]$, enumerate the elements of M_i with constant delay.*

Lemma
We can compute all sorted $O_1, \ldots, O_k$ in $\mathcal{O}(|S| + n)$ time and $\mathcal{O}(n + \sum_{i=1}^{k} |O_i|)$ space.

First result

Theorem

We can enumerate the set S of all valid embeddings of α in T with constant delay, after a preprocessing running in $\mathcal{O}(n + |S|)$ time. The used space is $\mathcal{O}(n + \min\{|S|, nk\})$.

Towards an improvement

A valid embedding $(j_1, \ldots, j_k)$ is *simple* when, for some $i \in [k]$, we have:

- $j_1 = \ell_1, \ldots, j_{i-1} = \ell_{i-1}$,
- $\ell_i < j_i \leq r_i$, and
- $j_{i+1} = r_{i+1}, \ldots, j_k = r_k$.

Lemma

After a preprocessing running in $\mathcal{O}(n)$ time, all simple embeddings can be enumerated with constant delay. After all simple embeddings have been enumerated, we have the unsorted list O_i, for every $i \in [k]$.

Second result

- Preprocessing: enumerate up to n simple embeddings, do not report them.
- If in fact there are no more simple embeddings than we have already obtained, for every i, the unsorted list O_i.
  – We can sort all of those lists together with radix sort in additional $\mathcal{O}(n)$ time to obtain, for every i, the sorted list O_i. We can then proceed as in the first result.

Second result

- Preprocessing: enumerate up to n simple embeddings, do not report them.
- If in fact there are no more simple embeddings than we have already obtained, for every i, the unsorted list O_i.
  – We can sort all of those lists together with radix sort in additional $\mathcal{O}(n)$ time to obtain, for every i, the sorted list O_i. We can then proceed as in the first result.
- If there are more than n simple embeddings.
  – Enumerate and do report the simple embeddings.
  – After having reported all the $\Omega(n)$ simple embeddings we have the unsorted list O_i, and we can sort all of them together with radix sort in additional $\mathcal{O}(n + \sum_i |O_i|)$ time to obtain, for every i, the sorted list O_i.
  – Observe that $\mathcal{O}(n + \sum_i |O_i|)$ is only amortized constant time per every valid embedding reported so far.
  – Having the sorted lists O_i, we now proceed as in the first result.

Second result

Theorem

We can enumerate the set S of all valid embeddings of α in T with amortized constant delay, after a preprocessing running in $\mathcal{O}(n)$ time. The used space is $\mathcal{O}(n + \min\{|S|, nk\})$.

Third result

Theorem

We can enumerate the set S of all valid embeddings of α in T with worst-case constant delay, after a preprocessing running in $\mathcal{O}(n)$ time. The used space is $\mathcal{O}(n \min\{|S|, nk\})$.

- Avoid generating the sorted lists O_i altogether (we actually do not need this in our structures!!), but rather improve the way we can access the elements of L_i, R_i, and M_i.
- This requires a much deeper understanding of how the elements of these sets are retrieved during the calls to `enum` and the three-sided reporting procedure.

Indexing

Index for the Enumeration of Regular Patterns Matches ($\texttt{EnumMatch}_{\texttt{Reg}}$)

Input: Text $T[1:n]$.

Task: Preprocess T for the following queries: given a regular pattern $\alpha = (\prod_{i=0}^{k} w_i x_i) w_{k+1}$, report all valid substitutions (as valid embeddings) for α and T.

Indexing

Index for the Enumeration of Regular Patterns Matches ($\texttt{EnumMatch}_{\texttt{Reg}}$)

Input: Text $T[1:n]$.

Task: Preprocess T for the following queries: given a regular pattern $\alpha = (\prod_{i=0}^{k} w_i x_i) w_{k+1}$, report all valid substitutions (as valid embeddings) for α and T.

Theorem

We can preprocess the text T in $\mathcal{O}(n \log n)$ time, so that when given a pattern α with k variables, after an additional processing running in $\mathcal{O}(|\alpha| + k \log n)$ time, the set S of all valid embeddings of α in T can be enumerated with constant delay.

Overview and Outlook

Theorem

We can enumerate the set S of all valid embeddings of α in T with worst-case constant delay, after a preprocessing running in $\mathcal{O}(n)$ time.

Theorem

We can preprocess the text T in $\mathcal{O}(n \log n)$ time, so that when given a pattern α with k variables, after an additional processing running in $\mathcal{O}(|\alpha| + k \log n)$ time, the set S of all valid embeddings of α in T can be enumerated with $\mathcal{O}(1)$ delay.

Future Work

- Enumeration algorithms for other tractable classes of pattern. Include various types of constraints on the variables (e.g., length). Weighted enumeration.

Overview and Outlook

Theorem

We can enumerate the set S of all valid embeddings of α in T with worst-case constant delay, after a preprocessing running in $\mathcal{O}(n)$ time.

Theorem

We can preprocess the text T in $\mathcal{O}(n \log n)$ time, so that when given a pattern α with k variables, after an additional processing running in $\mathcal{O}(|\alpha| + k \log n)$ time, the set S of all valid embeddings of α in T can be enumerated with $\mathcal{O}(1)$ delay.

Future Work

- Enumeration algorithms for other tractable classes of pattern. Include various types of constraints on the variables (e.g., length). Weighted enumeration.
- Further work in transferring methods from FL/AT/stringology and data structures to information extraction.

Overview and Outlook

Theorem

We can enumerate the set S of all valid embeddings of α in T with worst-case constant delay, after a preprocessing running in $\mathcal{O}(n)$ time.

Theorem

We can preprocess the text T in $\mathcal{O}(n \log n)$ time, so that when given a pattern α with k variables, after an additional processing running in $\mathcal{O}(|\alpha| + k \log n)$ time, the set S of all valid embeddings of α in T can be enumerated with $\mathcal{O}(1)$ delay.

Future Work

- Enumeration algorithms for other tractable classes of pattern. Include various types of constraints on the variables (e.g., length). Weighted enumeration.
- Further work in transferring methods from FL/AT/stringology and data structures to information extraction.

Thank you for your attention