Efficient Algorithm for δ - Approximate Jumbled Pattern Matching

Iván Castellanos[1] Yoan Pinzón[2]

[12]Departament of Computer and Industrial Engineering
National University of Colombia

Prague Stringology Conference, 2015

Outline

Outline

1. Preliminaries
   - Jumbled Pattern Matching
   - Approximate Version
2. Algorithm
   - ESR Algorithm
   - Implementation
3. Related problems
   - All Matchings
   - Min-Err Matching

Parikh Vector.

Definition

Let Σ be an alphabet with σ elements and $s \in \Sigma^*$, the Parikh vector of s denoted $p(s) = (p_i(s) \,|\, i = 1, 2, ..., \sigma)$ where $p_{a_i}(s) = |\{j \,|\, s_j = a_i\}|$ here s_j is the $j-th$ character of s and a_i is the $i-th$ element of Σ.

Example

Let $\Sigma = \{a, b, c\}$ and $s = abab$
Then $p_a(s) = 2,\ p_b(s) = 2, p_c(s) = 0$ and $p(s) = (2, 2, 0)$.

Parikh Vector.

Definition

Let Σ be an alphabet with σ elements and $s \in \Sigma^*$, the Parikh vector of s denoted $p(s) = (p_i(s) \,|\, i = 1, 2, ..., \sigma)$ where $p_{a_i}(s) = |\{j \,|\, s_j = a_i\}|$ here s_j is the $j - th$ character of s and a_i is the $i - th$ element of Σ.

Example

Let $\Sigma = \{a, b, c\}$ and $s = abab$
Then $p_a(s) = 2,\ p_b(s) = 2, p_c(s) = 0$ and $p(s) = (2, 2, 0)$.

Parikh Vector.

Definitions

Let q and r Parikh vectors of 2 strings over the same alphabet, we define $+$ and $<$ as follows:

- $q < r$ if $q_i < r_i$, for $i = 1, ..., \sigma$.
- $p = q + r$ where $p_i = q_i + r_i$, for $i = 1, ..., \sigma$.

Similarly the operations $-$, $>$, $\leq$ and $\geq$ are defined.

Parikh Vector.

Definitions

Let q and r Parikh vectors of 2 strings over the same alphabet, we define $+$ and $<$ as follows:

- $q < r$ if $q_i < r_i$, for $i = 1, ..., \sigma$.
- $p = q + r$ where $p_i = q_i + r_i$, for $i = 1, ..., \sigma$.

Similarly the operations $-$, $>$, $\leq$ and $\geq$ are defined.

Parikh Vector.

Definitions

Let q and r Parikh vectors of 2 strings over the same alphabet, we define $+$ and $<$ as follows:

- $q < r$ if $q_i < r_i$, for $i = 1, ..., \sigma$.
- $p = q + r$ where $p_i = q_i + r_i$, for $i = 1, ..., \sigma$.

Similarly the operations $-$, $>$, $\leq$ and $\geq$ are defined.

Parikh Vector.

Example

$p = (4,3,2,2)$, $q = (4,3,2,1)$, $r = (1,1,3,1)$
Then $q \not< p$, $q \leq p$ and $r \not\leq q$. Also $p + r = (5,4,5,3)$ and $p - q = (0,0,0,1)$.

- Clearly $r \not\leq p$ because $r \not\leq q$ and $q \leq p$.
- We use the operation $p - q$ only when $p \leq q$.
- If $p \leq q$ then p is called a sub-Parikh vector of q and q is called a super-Parikh vector of p.

Parikh Vector.

Example

$p = (4,3,2,2)$, $q = (4,3,2,1)$, $r = (1,1,3,1)$
Then $q \not< p$, $q \leq p$ and $r \nleq q$. Also $p + r = (5,4,5,3)$ and $p - q = (0,0,0,1)$.

- Clearly $r \nleq p$ because $r \nleq q$ and $q \leq p$.
- We use the operation $p - q$ only when $p \leq q$.
- If $p \leq q$ then p is called a sub-Parikh vector of q and q is called a super-Parikh vector of p.

Notation.

- $s[i,j]$ denotes the substring starting at position i and finishing at position j.
- $prv(s,i)$ denotes the Parikh Vector of the prefix at position i of string s, if s is clear we use just $prv(i)$.
- $prv(0) = (0, ..., 0)$
- $p(s[i,j])=$ $prv(s,j)$ - $prv(s,i-1)$.

Jumbled Pattern Matching.

Definitions

Let $s, t \in \Sigma^*$, and $s[i, j]$ the substring of s starting at position i and finishing at position j.
If $p(s[i, j]) = p(t)$ then we say that $s[i, j]$ is an occurrence of t in s.

Jumbled Pattern Matching problem: given s and t find if there are occurrences of t in s (decision problem) or find all the occurrences of t in s (occurrence problem).

Jumbled Pattern Matching.

Definitions

Let $s, t \in \Sigma^*$, and $s[i, j]$ the substring of s starting at position i and finishing at position j.
If $p(s[i, j]) = p(t)$ then we say that $s[i, j]$ is an occurrence of t in s.

Jumbled Pattern Matching problem: given s and t find if there are occurrences of t in s (decision problem) or find all the occurrences of t in s (occurrence problem).

Jumbled Pattern Matching.

- if $p(t) = p(t')$ then all ocurrences of t in s are equal to all the ocurrences of t' in s. It means we do not need t at all, just $q \in \mathbb{N}^\sigma$ such that $q = p(t)$.

Example

Let $\Sigma = \{a, b, c\}$ and $s = bacbbacbacbac$, $q_1 = (2,2,1)$ and $q_2 = (3,1,1)$.
There are 2 ocurrences of q_1 in s, $s[2, 6] = acbba$ and $s[5, 9] = bacba$.
There are no ocurrences of q_2 in s.

Jumbled Pattern Matching.

- if $p(t) = p(t')$ then all ocurrences of t in s are equal to all the ocurrences of t' in s. It means we do not need t at all, just $q \in \mathbb{N}^\sigma$ such that $q = p(t)$.

Example

Let $\Sigma = \{a, b, c\}$ and $s = bacbbacbacbac$, $q_1 = (2,2,1)$ and $q_2 = (3,1,1)$. There are 2 ocurrences of q_1 in s, $s[2,6] = acbba$ and $s[5,9] = bacba$. There are no ocurrences of q_2 in s.

Outline

1. Preliminaries
   - Jumbled Pattern Matching
   - Approximate Version
2. Algorithm
   - ESR Algorithm
   - Implementation
3. Related problems
   - All Matchings
   - Min-Err Matching

δ - Approximate Jumbled Pattern Matching.

Definition

We call $\delta \in \mathbb{N}^{\sigma}$ the vector of errors, the bound queries allowing the error for a Parikh vector q are $q+\delta$ and $q-\delta$ if $q_i < \delta_i$ for some i, $(q-\delta)_i = 0$.

Example

Let $q = (4, 0, 5)$ a Parikh Vector and $\delta = (1, 1, 0)$
Then $q+\delta = (5, 1, 5)$ and $q-\delta = (3, 0, 5)$

δ - Approximate Jumbled Pattern Matching.

Definition

We call $\delta \in \mathbb{N}^{\sigma}$ the vector of errors, the bound queries allowing the error for a Parikh vector q are $q+\delta$ and $q-\delta$ if $q_i < \delta_i$ for some i, $(q-\delta)_i = 0$.

Example

Let $q = (4, 0, 5)$ a Parikh Vector and $\delta = (1, 1, 0)$
Then $q+\delta = (5, 1, 5)$ and $q-\delta = (3, 0, 5)$

δ - Approximate Jumbled Pattern Matching.

Definition

Given a string s, a Parikh vector q and a vector of errors, (i,j) is an (approximate) occurrence of q in s if $|p_k(s[i,j]) - q_k| \leq \delta_k$ for $k = 1, ..., \sigma$.

(i,j) is a maximal occurrence if (i,j) is an occurrence and neither $(i-1,j)$ nor $(i,j+1)$ are occurrences.

δ - Approximate Jumbled Pattern Matching: finding all the maximal occurrences of q in s with an error δ is one version of the occurrence problem of δ - Approximate Jumbled Pattern Matching, similarly finding if there are occurrences of q in s with an error δ is the decision problem of δ - Approximate Jumbled Pattern Matching.

δ - Approximate Jumbled Pattern Matching.

Definition

Given a string s, a Parikh vector q and a vector of errors, (i,j) is an (approximate) occurrence of q in s if $|p_k(s[i,j]) - q_k| \leq \delta_k$ for $k = 1, ..., \sigma$.

(i,j) is a maximal occurrence if (i,j) is an occurrence and neither $(i-1,j)$ nor $(i,j+1)$ are occurrences.

δ - Approximate Jumbled Pattern Matching: finding all the maximal occurrences of q in s with an error δ is one version of the occurrence problem of δ - Approximate Jumbled Pattern Matching, similarly finding if there are occurrences of q in s with an error δ is the decision problem of δ - Approximate Jumbled Pattern Matching.

δ - Approximate Jumbled Pattern Matching.

Definition

Given a string s, a Parikh vector q and a vector of errors, (i,j) is an (approximate) occurrence of q in s if $|p_k(s[i,j]) - q_k| \leq \delta_k$ for $k = 1, ..., \sigma$.

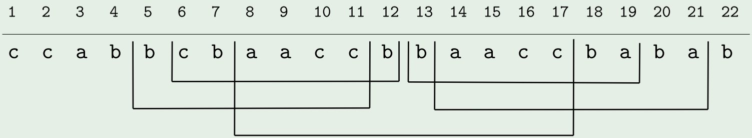
(i,j) is a maximal occurrence if (i,j) is an occurrence and neither $(i-1,j)$ nor $(i,j+1)$ are occurrences.

δ - Approximate Jumbled Pattern Matching: finding all the maximal occurrences of q in s with an error δ is one version of the occurrence problem of δ - Approximate Jumbled Pattern Matching, similarly finding if there are occurrences of q in s with an error δ is the decision problem of δ - Approximate Jumbled Pattern Matching.

δ - Approximate Jumbled Pattern Matching.

Example

Maximal occurrences of the query $q = (3, 1, 3)$ with $\delta = (1, 1, 1)$ for the string *ccabbcbaaccbbaaccbabab*.



Outline

ESR Algorithm.

Using a window approach to solve this problem works, however it is not efficient in time.

The main problem using a window approach is that many positions are useless, the idea is to skip them.

ESR Algorithm.

Using a window approach to solve this problem works, however it is not efficient in time.

The main problem using a window approach is that many positions are useless, the idea is to skip them.

ESR Algorithm.

We start having two pointers L and R that represents te window of the substring that is been taken into examination.

The ESR algorithm uses 3 phases (Expansion, Shrinking and Refine) in order to move L and R skipping as many positions as possible to mantein the correctness of the solution.

ESR Algorithm.

We start having two pointers L and R that represents te window of the substring that is been taken into examination.

The ESR algorithm uses 3 phases (Expansion, Shrinking and Refine) in order to move L and R skipping as many positions as possible to mantein the correctness of the solution.

Expansion.

Move the right border of the window (R) to the right until its corresponding Parikh vector is a super-Parikh of the lower bound.

After this, the Parikh vector of the window is not necessarily a sub-Parikh of the upper bound, if it is we have a match.

Expansion.

Move the right border of the window (R) to the right until its corresponding Parikh vector is a super-Parikh of the lower bound.

After this, the Parikh vector of the window is not necessarily a sub-Parikh of the upper bound, if it is we have a match.

Shrinkage.

Move the left border of the window (L) to the right until its corresponding Parikh vector is a sub-Parikh of the upper bound.

After this, the Parikh vector of the window is not necessarily a sub-Parikh of the upper bound, if it is we have a match.

Shrinkage.

Move the left border of the window (L) to the right until its corresponding Parikh vector is a sub-Parikh of the upper bound.

After this, the Parikh vector of the window is not necessarily a sub-Parikh of the upper bound, if it is we have a match.

Refine.

After we have a match we use refine to find a maximal one.

The right border of the window is extended to the right while the window is a sub-Parikh of the upper bound.

Refine.

After we have a match we use refine to find a maximal one.

The right border of the window is extended to the right while the window is a sub-Parikh of the upper bound.

Phases.

Example

First phases of the algorithm for the string *ccabbcbaaccbba* for the query $q = (3, 1, 3)$ and $\delta = (1, 1, 1)$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`Expand(0,{2,0,2})`	c	c	a	b	b	c	b	a	a	c	c	b	b	a
`Shrink(8,{4,2,4})`	c	c	a	b	b	c	b	a	a	c	c	b	b	a
`Expand(5,{2,0,2})`	c	c	a	b	b	c	b	a	a	c	c	b	b	a
`Refine(5,{4,2,4})`	c	c	a	b	b	c	b	a	a	c	c	b	b	a

Formulas.

The phases of the ESR Algorithm can be summarized as:

- $Expand(k, p) = min\{j \,|\, prv(j) \geq prv(k) + p\}$
- $Shrink(k, p) = min\{j \,|\, prv(k) - prv(j) \leq p\}$
- $Refine(k, p) = max\{j \,|\, prv(j) - prv(k) \leq p\}$

Formulas.

The phases of the ESR Algorithm can be summarized as:

- $Expand(k, p) = min\{j \,|\, prv(j) \geq prv(k) + p\}$
- $Shrink(k, p) = min\{j \,|\, prv(k) - prv(j) \leq p\}$
- $Refine(k, p) = max\{j \,|\, prv(j) - prv(k) \leq p\}$

Outline

1. Preliminaries
   - Jumbled Pattern Matching
   - Approximate Version
2. Algorithm
   - ESR Algorithm
   - Implementation
3. Related problems
   - All Matchings
   - Min-Err Matching

Bit Vectors.

The Parikh vectors are in practice vectors of integer wich are not too big and have just few operations

We can represent the Parikh vectors as bit vectors, storing space and time.

Bit Vectors.

The Parikh vectors are in practice vectors of integer wich are not too big and have just few operations

We can represent the Parikh vectors as bit vectors, storing space and time.

Operations on Bit Vectors.

For each element in the Parikh vector we use the number of bits that the integer represent and one more to use as a carry.

- Operations $+$ and $-$ stay equal as the sum of bit vectors an integers is equivalent, however we have to have control of the crry element.
- Comparison $\leq(a, b) \equiv (((b \,|\, carries) - a) \,\&\, carries) \neq carries)$

Operations on Bit Vectors.

For each element in the Parikh vector we use the number of bits that the integer represent and one more to use as a carry.

- Operations $+$ and $-$ stay equal as the sum of bit vectors an integers is equivalent, however we have to have control of the crry element.
- Comparison $\leq (a, b) \equiv (((b \,|\, carries) - a) \,\&\, carries) \neq carries)$

Operations on Bit Vectors.

Example

Parikh Vectors				Bit Vectors															
				c =	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
a =	3	1	3	a' =	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1
b =	2	2	3	b' =	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1
a + b	5	3	6	a' + b'	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0
				b''= b' | c	1	0	0	1	0	1	0	0	1	0	1	0	0	1	1
				a''=b'' - a'	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0
$a_i \leq b_i$	F	T	T	a'' & c	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
$a \leq b$	False			$a' \leq b'$	False														

Prv.

For the implementation we store the $prv(i)$ for every $i = 0, 1, ..., n$, this can be calculated quickly even when reading the string.

Clearly having this stored is a great advantage to get the queries faster.

Prv.

For the implementation we store the $prv(i)$ for every $i = 0, 1, ..., n$, this can be calculated quickly even when reading the string.

Clearly having this stored is a great advantage to get the queries faster.

ESR.

Theorem

Given a Parikh vector p and a position k we have that for the text s:

$Expand(k,p) = max\{i_j \mid i_j \, pos \, of \, the \, (p_j + prv_j(k)) - th \, occurrence \, of \, j\}$

$Shrink(k,p) = max\{i_j \mid i_j \, pos \, of \, the \, (prv_j(k) - p_j) - th \, occurrence \, of \, j\}$

$Refine(k,p) =$
$min\{i_j \mid i_j \, pos \, of \, the \, (p_j + prv_j(k) + 1) - th \, occurrence \, of \, j\} - 1$

As a consequence of this theorem we can calculate the functions Expand, Shrink and Refine just using an inverted index table of the string *s*.

ESR.

Theorem

Given a Parikh vector p and a position k we have that for the text s:

$Expand(k,p) = max\{i_j \mid i_j\, pos\, of\, the\, (p_j + prv_j(k)) - th\, occurrence\, of\, j\}$

$Shrink(k,p) = max\{i_j \mid i_j\, pos\, of\, the\, (prv_j(k) - p_j) - th\, occurrence\, of\, j\}$

$Refine(k,p) =$
$min\{i_j \mid i_j\, pos\, of\, the\, (p_j + prv_j(k) + 1) - th\, occurrence\, of\, j\} - 1$

As a consequence of this theorem we can calculate the functions Expand, Shrink and Refine just using an inverted index table of the string s.

ESR.

Proof.

Let $A = min\{j \,|\, prv(j) \geq prv(k) + p\}$
$B = max\{i_j \,|\, i_j\, pos\, of\, the\, (p_j + prv_j(k)) - th\, occurrence\, of\, j\}$
$B \geq A$ because for every $j = 1, ..., \sigma$ we have that $prv_j(B) \geq p + prv_j(k)$ due to the definition of B.
Because of the definition of B, $B = i_m$ for some m between 1 and σ, now if we take any $B' < B = i_m$ then $prv_m(B') \nleq p_m + prv_m(k)$, so we conclude $A \leq B$.

□

ESR.

Proof.

Let $A = max\{prv(j) - prv(k) \leq p\}$
$B = min\{i_j \,|\, i_j \, pos \, of \, the \, (p_j + prv_j(k) + 1) - th \, occurrence \, of \, j\}$
For the definition of B exists some m for which $B = i_m$, so we have $prv_m(B) - prv_m(k) = p_m + 1$ and $prv_j(B) - prv_j(k) < p_j + 1$ for $j \neq m$.
Clearly $prv(B) - prv(k) \nleq p$ but if we take $B - 1$ then $prv_j(B) - prv_j(k) < p_j + 1 \equiv prv_j(B) - prv_j(k) \leq p_j$, so we conclude that $B = A$.

□

Outline

1. Preliminaries
   - Jumbled Pattern Matching
   - Approximate Version
2. Algorithm
   - ESR Algorithm
   - Implementation
3. Related problems
   - All Matchings
   - Min-Err Matching

All Matchings.

So far has been showed how to calculate the maximal occurences of a query in a text, however maybe we ant to look for all the possible occurrences of a query in the text.

- Functions *Expand* and *Shrink* are enough to find matches.
- Function *Refine* return the longest possible match starting at a fixed position.
- We can keep track of the shortest and the longest possible match for a fixed position.

All Matchings.

So far has been showed how to calculate the maximal occurences of a query in a text, however maybe we ant to look for all the possible occurrences of a query in the text.

- Functions *Expand* and *Shrink* are enough to find matches.
- Function *Refine* return the longest possible match starting at a fixed position.
- We can keep track of the shortest and the longest possible match for a fixed position.

All Matchings.

Require: A Parikh vector q a vector of possible errors δ

Ensure: A set *Matches* with all the ocurrences of q in s allowing the error δ

$L \leftarrow 0,\ R \leftarrow 0,\ R' \leftarrow 0,\ Matches \leftarrow \emptyset$
while $L < n - |q - \delta|$ **and** $R < n$ **do**
  if $q - \delta \nleq p(s[L+1, R])$ **then**
    $R \leftarrow Expand(L, q - \delta)$
  $L \leftarrow Shrink(R, q + \delta)$
  if $p(s[L+1, R] \geq q - \delta)$ **then**
    $R' \leftarrow Refine(L, q + \delta)$
    for $j = R$ **to** R' **do**
      add $(L+1, j)$ **to** *Matches*
    $L \leftarrow L + 1$
return *Matches*

Outline

1. Preliminaries
   - Jumbled Pattern Matching
   - Approximate Version
2. Algorithm
   - ESR Algorithm
   - Implementation
3. Related problems
   - All Matchings
   - Min-Err Matching

Min-Err Matching.

In some cases can be of interest to find neither the maximal ocurrences nor all the occurrences, but the occurrences which are closer to the query.

Definition

An occurrence (i,j) is said to have minimal error, if neither $(i-1,j)$,$(i+1,j)$,$(i,j-1)$ nor $(i,j+1)$ is a match or has a bigger error.

This can be solved using the all matchings solution and looking for the minimal error, this naive solution have a $O(n\sigma|\delta|)$ time complexity.

Min-Err Matching.

In some cases can be of interest to find neither the maximal ocurrences nor all the occurrences, but the occurrences which are closer to the query.

Definition

An occurrence (i,j) is said to have minimal error, if neither $(i-1,j)$,$(i+1,j)$,$(i,j-1)$ nor $(i,j+1)$ is a match or has a bigger error.

This can be solved using the all matchings solution and looking for the minimal error, this naive solution have a $O(n\sigma|\delta|)$ time complexity.

Min-Err Matching.

In some cases can be of interest to find neither the maximal ocurrences nor all the occurrences, but the occurrences which are closer to the query.

Definition

An occurrence (i,j) is said to have minimal error, if neither $(i-1,j)$,$(i+1,j)$,$(i,j-1)$ nor $(i,j+1)$ is a match or has a bigger error.

This can be solved using the all matchings solution and looking for the minimal error, this naive solution have a $O(n\sigma|\delta|)$ time complexity.

Min-Err Matching.

Example

Maximal and Min-Err Ocurrences for the query $q = (3, 1, 3)$ and $\delta = (1, 1, 1)$ in the text *ccabbcbaaccbbaaccbabab*.

Max Match	Parikh Vector	Error
(5, 11)	{2, 2, 3}	2
(6, 12)	{2, 2, 3}	2
(8, 17)	{4, 2, 4}	3
(13, 19)	{3, 2, 2}	2
(14, 21)	{4, 2, 2}	3

Min-Err Match	Parikh Vector	Error
(6, 11)	{2, 1, 3}	1
(8, 12)	{2, 1, 2}	2
(8, 14)	{3, 2, 2}	2
(9, 16)	{3, 2, 3}	1
(11, 17)	{2, 2, 3}	2
(14, 19)	{3, 1, 2}	1

Conclusions

- We presented a new implementation of an algorithm to solve the δ - Approximate Jumbled Pattern Matching.
- The implementation has very good time performance in practice due to the bit vectors.
- Some related problems to δ - Approximate Jumbled Pattern Matching were explained.
- For binary alphabets more improvements can be done as here there are more properties.