Online Approximate Matching with Non-local Distances

Raphaël Clifford and Benjamin Sach

{clifford,sach}@cs.bris.ac.uk

Department of Computer Science,
University of Bristol

Outline

Introduction

Related work on (local) online pattern matching

Results

Methods for non-local online pattern matching

An Open Problem

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

$$T: \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline a & b & c & ? & ? & ? & ? & ? & ? & ? \\ \hline \end{array}$$

$$P: \begin{array}{|c|c|c|} \hline a & b & a \\ \hline \end{array}$$

(dist = 1)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	?	?	?	?	?	?
P :		a	b	a						

(dist = 2)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	?	?	?	?	?
P :			a	b	a					

(dist = 2)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	b	?	?	?	?
P :				a	b	a				

(dist = 2)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	b	a	?	?	?
P :					a	b	a			

(dist = 0)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	b	a	b	?	?
P :						a	b	a		

(dist = 3)

- Find the **distance**, $d(i)$,between $T[i, i + m - 1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	b	a	b	a	?	
P :							a	b	a		(dist = 0)

- Find the **distance**, $d(i)$,between $T[i, i + m - 1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

(which we term Pseudo-Realtime)

Introduction: Online pattern matching

- Consider a text, T (length n) and a pattern P (length m)
- We assume we have P in advance but T arrives online...

T :	a	b	c	a	a	b	a	b	a	c	
P :								a	b	a	(dist = 3)

- Find the **distance**, $d(i)$,between $T[i, i+m-1]$ and P for all i.

(Hamming distance shown)

- We are concerned with worst-case time per text character.

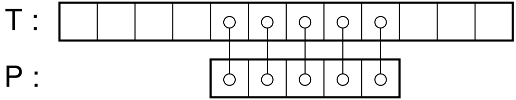
(which we term Pseudo-Realtime)

Local and non-local pattern matching

A distance is **local** if it can be written as:

$$d(i) = \sum_{j=0}^{m-1} \Delta(P[j], T[i+j])$$

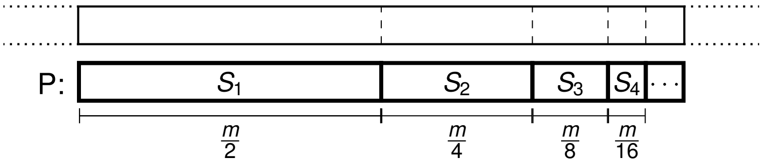
(where Δ is some function acting on alphabet symbols)



- Hamming, L_1, L_2, less-than and k-mismatch...are all local.
- Edit distance, k-difference, rearangement distance etc. are non-local.

Local online pattern matching (CEPP, 2008)[1]

- Split the pattern into $O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.



- Compute distances by summing the distance from each S_j to T.

[1] Clifford, Efremenko, Porat and Porat. CPM 2008

Local online pattern matching (CEPP,2008)

Example: (using Hamming distance)

T:	b	a	c	b	b	a	b	a	a	c	a	c	a	b	a	a	b	c	b	c
P:					a	b	b	a	b	c	a	b								

(dist = 4)

2 + 1 + 0+1

Plan:

- Compute distances from each subpattern, S_j, to T using an offline algorithm as a black box.
- Split T into overlapping partitions so that each distance is computed before it is needed.

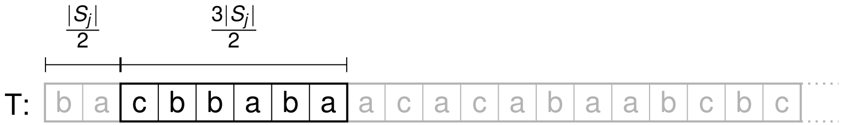
Local online pattern matching (CEPP,2008)

$\overbrace{\hphantom{bacbba}}^{\frac{3|S_j|}{2}}$

T:	b	a	c	b	b	a	b	a	a	c	a	c	a	b	a	a	b	c	b	c	…

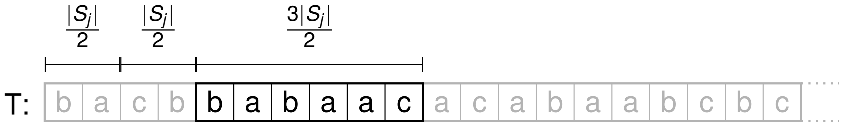
- Text partitions are different for each subpattern, S_j.

Local online pattern matching (CEPP,2008)



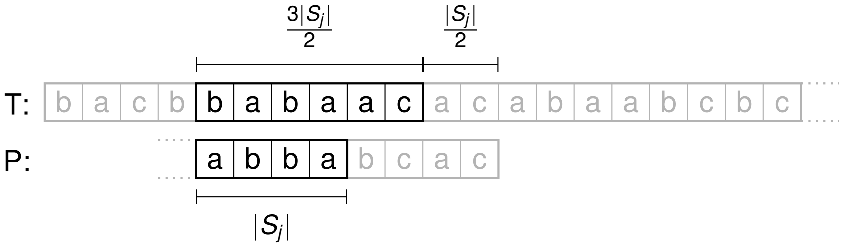
- Text partitions are different for each subpattern, S_j.

Local online pattern matching (CEPP,2008)



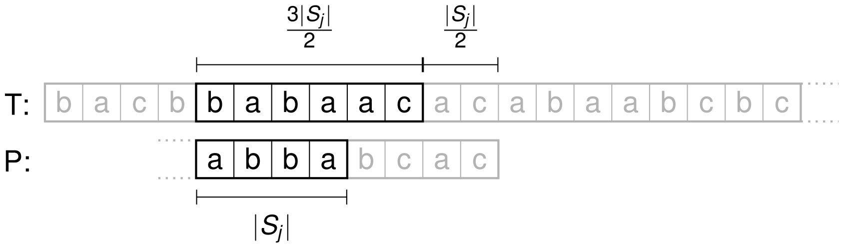
- Text partitions are different for each subpattern, S_j.

Local online pattern matching (CEPP,2008)



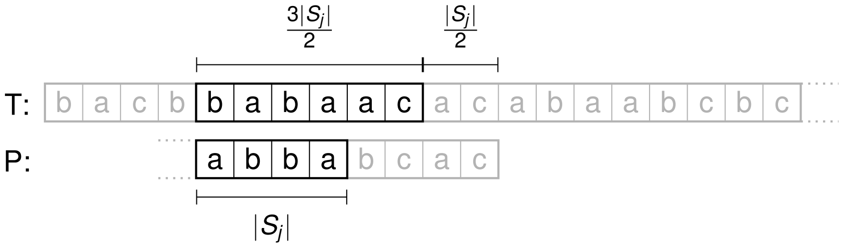
- Text partitions are different for each subpattern, S_j.

Local online pattern matching (CEPP,2008)



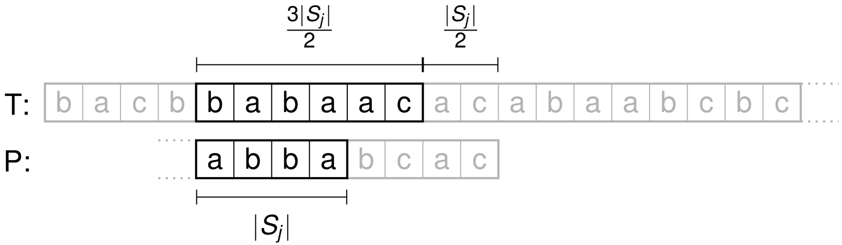
- Text partitions are different for each subpattern, S_j.
- Compute matches in a text partition using an offline algorithm.

Local online pattern matching (CEPP,2008)



- Text partitions are different for each subpattern, S_j.
- Compute matches in a text partition using an offline algorithm.
- Distribute the work across the next $|S_j|/2$ characters.

Local online pattern matching (CEPP,2008)



- Text partitions are different for each subpattern, S_j.
- Compute matches in a text partition using an offline algorithm.
- Distribute the work across the next $|S_j|/2$ characters.
- Time complexity increases by at worst multiplicative $O(\log m)$.

Results

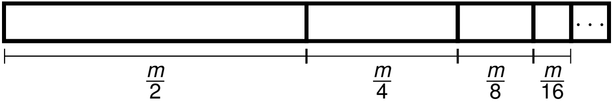
Problem	Offline per char time	Online/PsR penalty	**Method**
local matching	various	$O(\log m)$	Splitting (CEPP,2008)
function	various	$O(\log m)$	PsR Cross-correlations
parameterised	$O(\log \lvert\Sigma\rvert)$	$O(1)$	Realtime KMP
edit distance/LCS	$O(m)$	$O(1)$	Immediate
k-differences	$O(k)$	$O(\log m)$	Split & Feed
swap-mismatch	$O(\sqrt{m \log m})$	$O(1)$	Split & Correct
swap	$O(\log m \log \lvert\Sigma\rvert)$	$O(\log m)$	Split & Correct
overlap	$O(\log m)$	$O(\log m)$	Split & Correct
k-diff with transpositions	$O(k)$	$O(\log m)$	Split & Feed
self normalised	$O(\log m)$	$O(\log m)$	PsR Cross-correlations
faulty bits	$O(m \log m)$	$O(1)$	Immediate
flipped bits	$O(\log m)$	$O(\log m)$	PsR Cross-correlations
L_1 rearrangement	$O(m)$	$O(1)$	Immediate
L_2 rearrangement	$O(\log m)$	$O(\log m)$	PsR Cross-correlations

Methods for non-local problems

PsR cross-correlations: Replace cross-correlations with pseudo-realtime cross-correlations.

Split and Correct: Split the pattern into subpatterns and correct for non-local effects at the boundaries between subpatterns.

Split and Feed: Split the pattern into subpatterns and ‘feed’ the distances from one subpattern into the input of the next.



Method: PsR Cross-correlations

The cross-correlation between X and Y is defined by:

$$(X \otimes Y)[i] = \sum_{j=0}^{|Y|-1} X[i+j]Y[j]$$

- Offline: $O(|X| \log |Y|)$ total time (via FFTs).
- Pseudo-realtime: $O(\log^2 |Y|)$ time per character.
  - The problem is local so apply the method of (CEPP,2008).

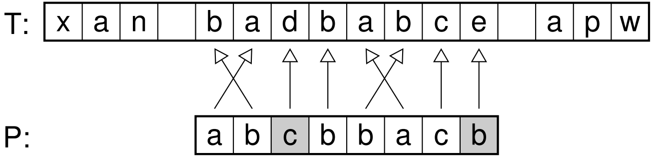
Method: Peel apart your favourite pattern matching algorithm and replace the cross-correlation step.

Method: Split and Correct

Example Problem: (Swap-Mismatch)

For each i, find the minimum number of moves to transform P into $T[i, i+m-1]$. No two moves can be applied to the same character. The valid moves are:

- *swap* (exchange two adjacent characters)
- *mismatch* (replace a character).

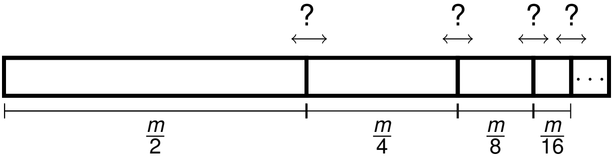


(AEP,2006)[2] solved this problem offline in $O(n\sqrt{m \log m})$ time.

[2] Amir, Eisenberg and Porat, Algorithmica 2006

Method: Split and Correct

- Consider splitting the pattern into $O(\log m)$ subpatterns...
- What about the swaps at the boundaries?
  - Only four possible cases



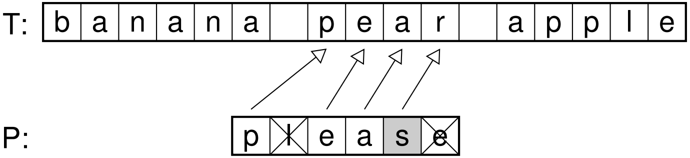
1. Compute distances for all transformed subpatterns using the black box method.
2. Stitch the solutions for the subpatterns together by calculating the optimal swaps at each boundary.

Method: Split and Feed

Example Problem: (k-differences)

For each i, find the minimum number of moves, $d(i)$, to transform P into a suffix of $T[1,i]$. We only output if $d(i)$ is $\leq k$.The valid moves are:

- *mismatch* (replace a character).
- *insert* (add a character).
- *delete* (remove a character).

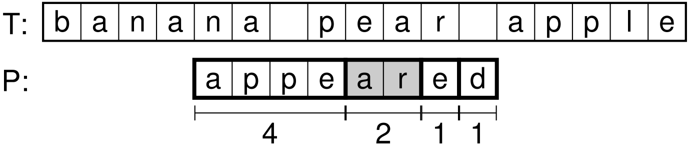


(LV,1988)[3] give an offline solution in $O(nk)$ time.

[3] Landau and Viskin, Comput. Syst. Sci. 2006

Method: Split and Feed - a naive approach

- Consider splitting the pattern into $O(\log m)$ subpatterns...
- What about alignment of subpatterns?
    - Inserts and deletes change subpattern alignment.

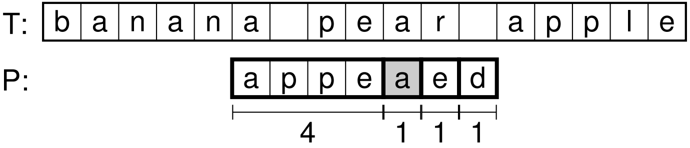


- Worse, the optimal transformation of P may contain suboptimal transformations of subpatterns.

Plan: ‘Charge’ subpatterns for starting at each alignment.

Method: Split and Feed - a naive approach

- Consider splitting the pattern into $O(\log m)$ subpatterns...
- What about alignment of subpatterns?
  - Inserts and deletes change subpattern alignment.



- Worse, the optimal transformation of P may contain suboptimal transformations of subpatterns.

Plan: ‘Charge’ subpatterns for starting at each alignment.

Method: Split and Feed - charging subpatterns

- Modify (LV,1988) to include starting costs as part of the input.
  - Requires $O(\ell k)$ time if the input has length $O(\ell)$.

C:	0	1	2	1	1	0	1	2	3	2	1	2	2	1	0	1	0
T:	b	a	n	a	n	a		p	e	a	r		a	p	p	l	e
P:								p	a	r	t						

- Starting costs reflect the moves used by previous subpatterns.
- Subpatterns now require the output of previous subpatterns before computation can begin.

Method: Split and Feed - charging subpatterns

- Modify (LV,1988) to include starting costs as part of the input.
  - Requires $O(\ell k)$ time if the input has length $O(\ell)$.

C:	0	1	2	1	1	0	1	2	3	2	1	2	2	1	0	1	0
T:	b	a	n	a	n	a		p	e	a	r		a	p	p	l	e
P:									p	a	r						

- Starting costs reflect the moves used by previous subpatterns.
- Subpatterns now require the output of previous subpatterns before computation can begin.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

$$\frac{7|S_j|}{4}$$

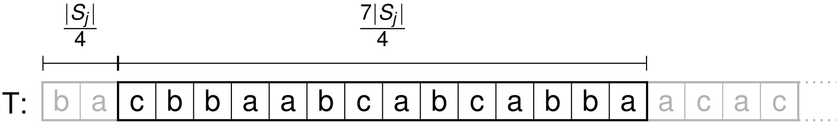
T:	b	a	c	b	b	a	a	b	c	a	b	c	a	b	b	a	a	c	a	c	…

- Again, text partitions are different for each S_j.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

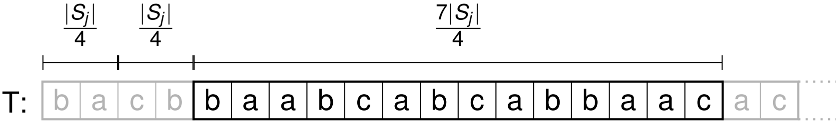


- Again, text partitions are different for each S_j.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

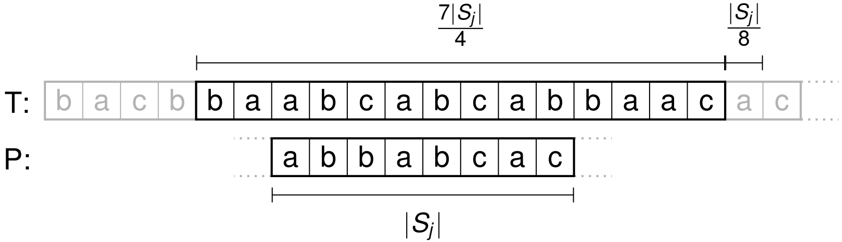


- Again, text partitions are different for each S_j.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \le |S_f| \le 8k.$$

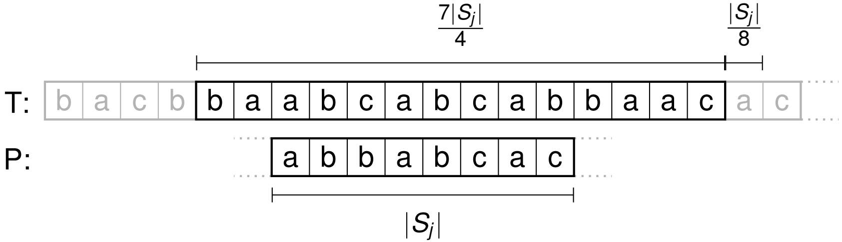


- Again, text partitions are different for each S_j.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

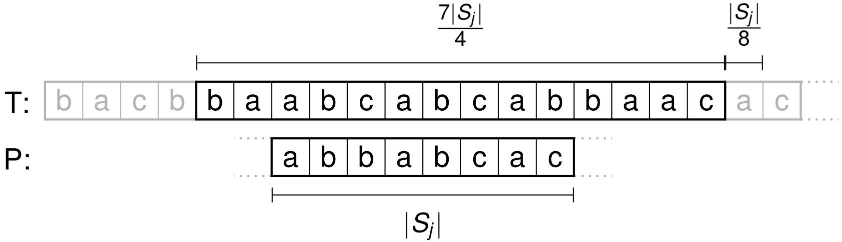


- Again, text partitions are different for each S_j.
- For all $j < f$, compute distances using modified (LV,1988).

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

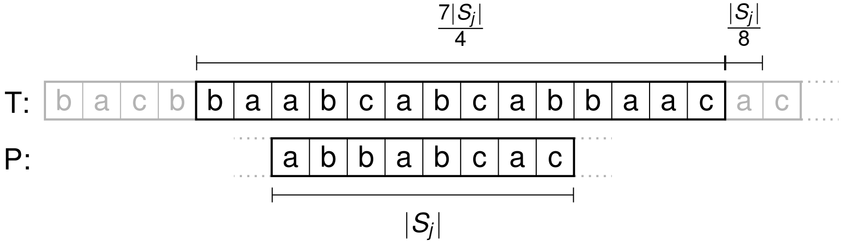


- Again, text partitions are different for each S_j.
- For all $j < f$, compute distances using modified (LV,1988).
- Distribute the work across the next $|S_j|/8$ characters.

Method: Split and Feed - modified partitioning

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

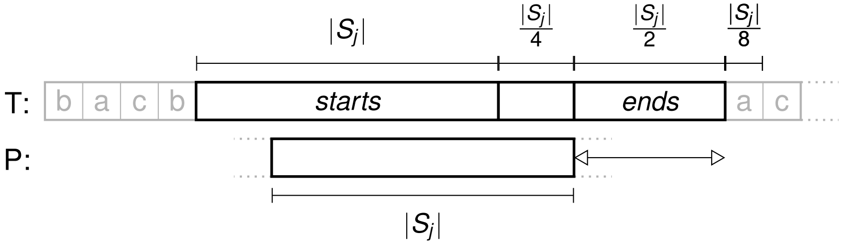


- Again, text partitions are different for each S_j.
- For all $j < f$, compute distances using modified (LV,1988).
- Distribute the work across the next $|S_j|/8$ characters.
- *Are the starting costs available in time? What about S_f?*

Method: Split and Feed - a closer look

- Split the pattern into halving subpatterns,

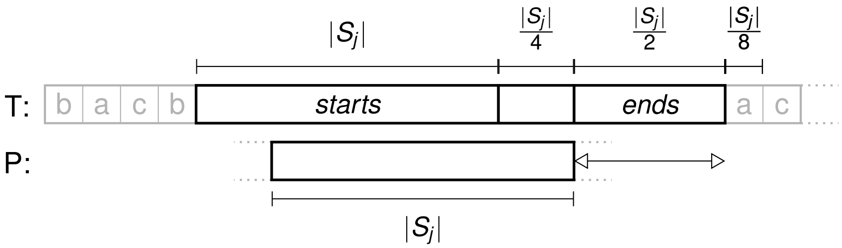
$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$



Method: Split and Feed - a closer look

- Split the pattern into halving subpatterns,

$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

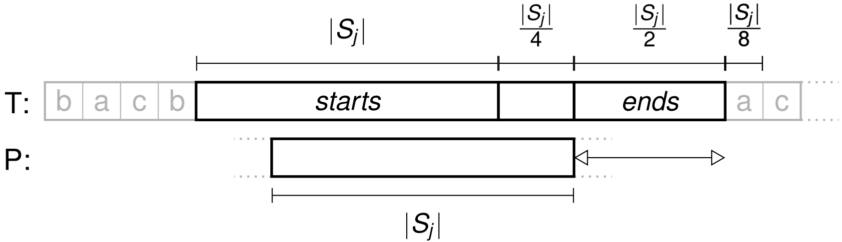


- Consider any alignment of the S_j ending in the *ends* section.
  - As $k \leq |S_j|/4$, any match must begin in the *starts* section.

Method: Split and Feed - a closer look

- Split the pattern into halving subpatterns,

$$S_1, S_2 \dots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$

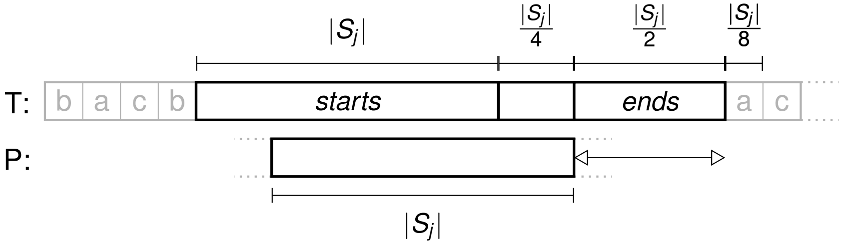


- Consider any alignment of the S_j ending in the *ends* section.
  - As $k \leq |S_j|/4$, any match must begin in the *starts* section.
- Therefore we only need S_{j-1} output from the *starts* section.
  - And S_{j-1} text partitions occur with twice the frequency ...

Method: Split and Feed - a closer look

- Split the pattern into halving subpatterns,

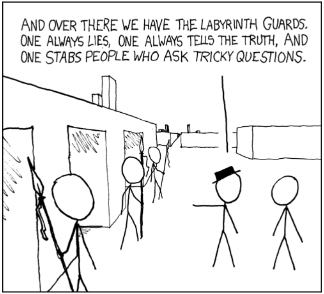
$$S_1, S_2 \ldots S_f \text{ where } 4k \leq |S_f| \leq 8k.$$



- Consider any alignment of the S_j ending in the *ends* section.
  - As $k \leq |S_j|/4$, any match must begin in the *starts* section.
- Therefore we only need S_{j-1} output from the *starts* section.
  - And S_{j-1} text partitions occur with twice the frequency ...
- *What about S_f?* It's not very big...use dynamic programming.

An Open Problem

- The black box method of (CEPP,2008) generalises well to 2D local pattern matching problems but naively requires $O(nm)$ space. Can this be reduced to $O(m^2)$?



(xkcd.com)

Thank you for listening.