More on Approximate String Matching

Giulia Bernardini
giulia.bernardini@units.it

Fundamentals of algorithms
a.y. 2021/2022

The k-difference global alignment problem

Given two strings S and T, the k-difference global alignment problem, is to find the best edit transcript between S and T that contains at most k errors (insertions, deletions and replacements), for some input k, if it exists.

Suppose, without loss of generality, that $|S| \leq |T|$. The goal is to reduce the time bound for the solution from $O(|S||T|)$ (obtained by applying the standard dynamic programming algorithm) to $O(k|T|)$.

The k-difference global alignment problem

Reference: Chapter 12.2.3 of: Gusfield, D. *Algorithms on Strings, Trees and Sequences.*

The k-difference global alignment problem

The idea is to fill in only an O(k|T|)-size portion of the dynamic programming table (assuming that |S|≤|T|)

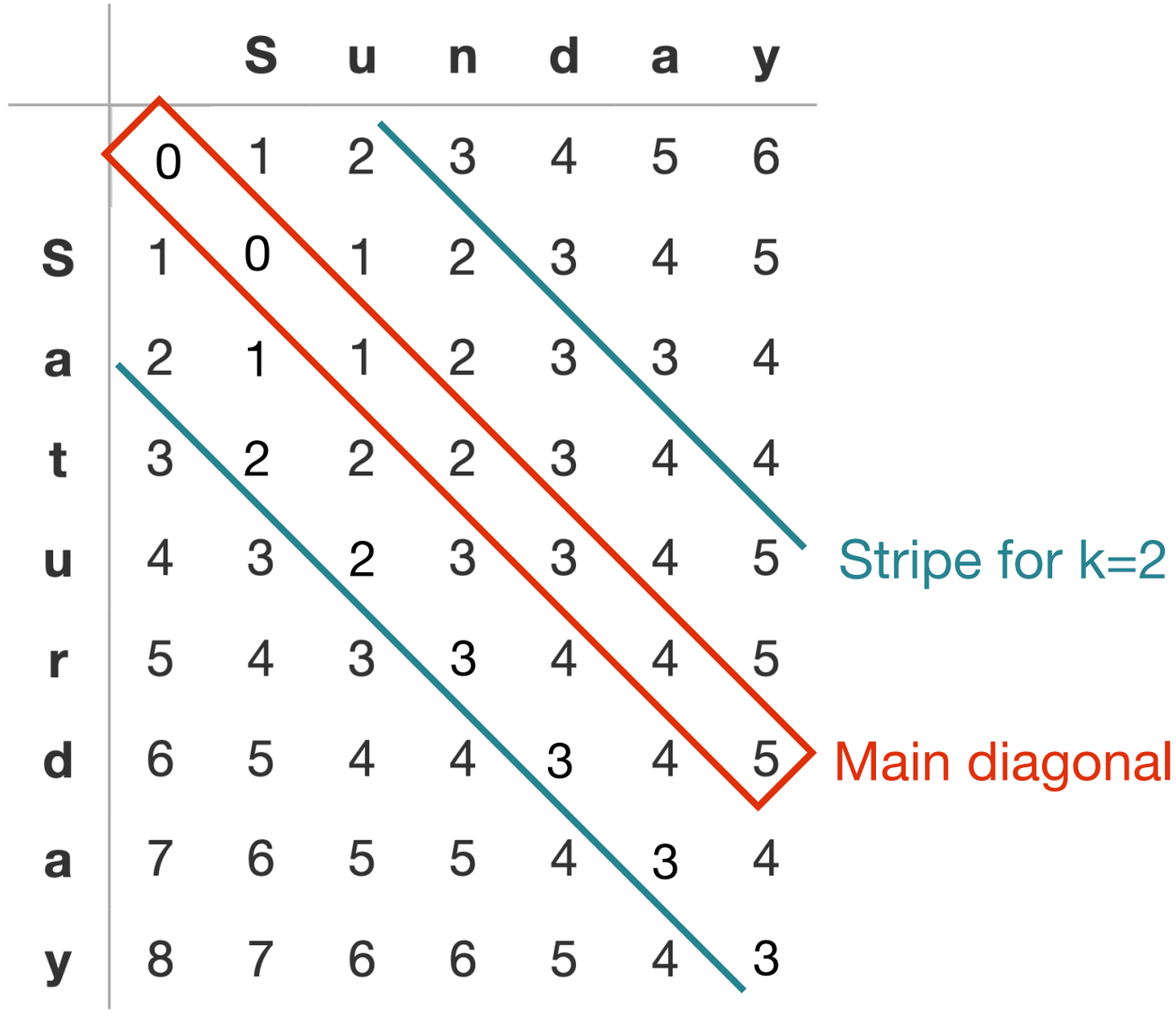
We make use of the following key observation. Let call cells (i,i), for all i≤|S|≤|T|, the main diagonal of the DP table.

Lemma 1. Any path in the DP table that defines a k-difference global alignment cannot contain any cell (i,i+z) or (i,i-z) with z>k.

Proof. Any path defining a global alignment must begin in cell (0,0) and end in cell (|T|,|S|). Therefore, the path must introduce one space in the alignment for every horizontal (or vertical) move that the path makes off the main diagonal. Thus, only the paths that are never more than k horizontal (or vertical) cells from the main diagonal are candidate solutions.

The k-difference global alignment problem

To find any k-difference global alignment, it suffices to fill in a stripe of the DP table consisting of 2k+1 cells in each row, centered on the main diagonal. The stripe contains O(k|S|)=O(k|T|) values.



The k-difference global alignment problem

If the bottom-right cell of the table contains a value d>k, then there does not exists a k-difference alignment: in this case, d is not necessarily the edit distance between S and T. If d≤k, then d is the correct edit distance between S and T, that gives a k-difference alignment.

		S	u	n	d	a	y
	0	1	2	3	4	5	6
S	1	0	1	2	3	4	5
a	2	1	1	2	3	3	4
t	3	2	2	2	3	4	4
u	4	3	2	3	3	4	5
r	5	4	3	3	4	4	5
d	6	5	4	4	3	4	5
a	7	6	5	5	4	3	4
y	8	7	6	6	5	4	3

Stripe for k=2

Main diagonal

A parameterised solution to edit distance

Suppose that the edit distance between S and T is d, which is clearly unknown a priori. The same idea used for the k-difference alignment problem can be used to compute d in $O(d|T|)$ time.

Indeed, one can fill in the DP table by progressively enlarging the considered strip around the main diagonal. The idea is to run the k-difference alignment algorithm starting with k=1, then k=2, then k=4…progressively doubling the parameter k, until the bottom-right cell contains a value smaller than the current k - which then will be equal to the edit distance d.

Theorem 1. The edit distance d between S and T, with $|S| \leq |T|$, can be computed in $O(d|T|)$ time and space.

Proof. Let k' be the largest value considered in the procedure described above: clearly, $k' \leq 2d$. So the total work of the method is $O(k'|T|+k'|T|/2+k'|T|/4+\ldots+|T|)=O(k'|T|)=O(d|T|)$.

Pattern matching under edit distance

Reference: Chapter 12.2.4 of: Gusfield, D. *Algorithms on Strings, Trees and Sequences.*

k-difference pattern matching

Consider the problem of searching for the all occurrences of a pattern P in a text T with edit distance at most k, for some fixed $1<k<|P|$.

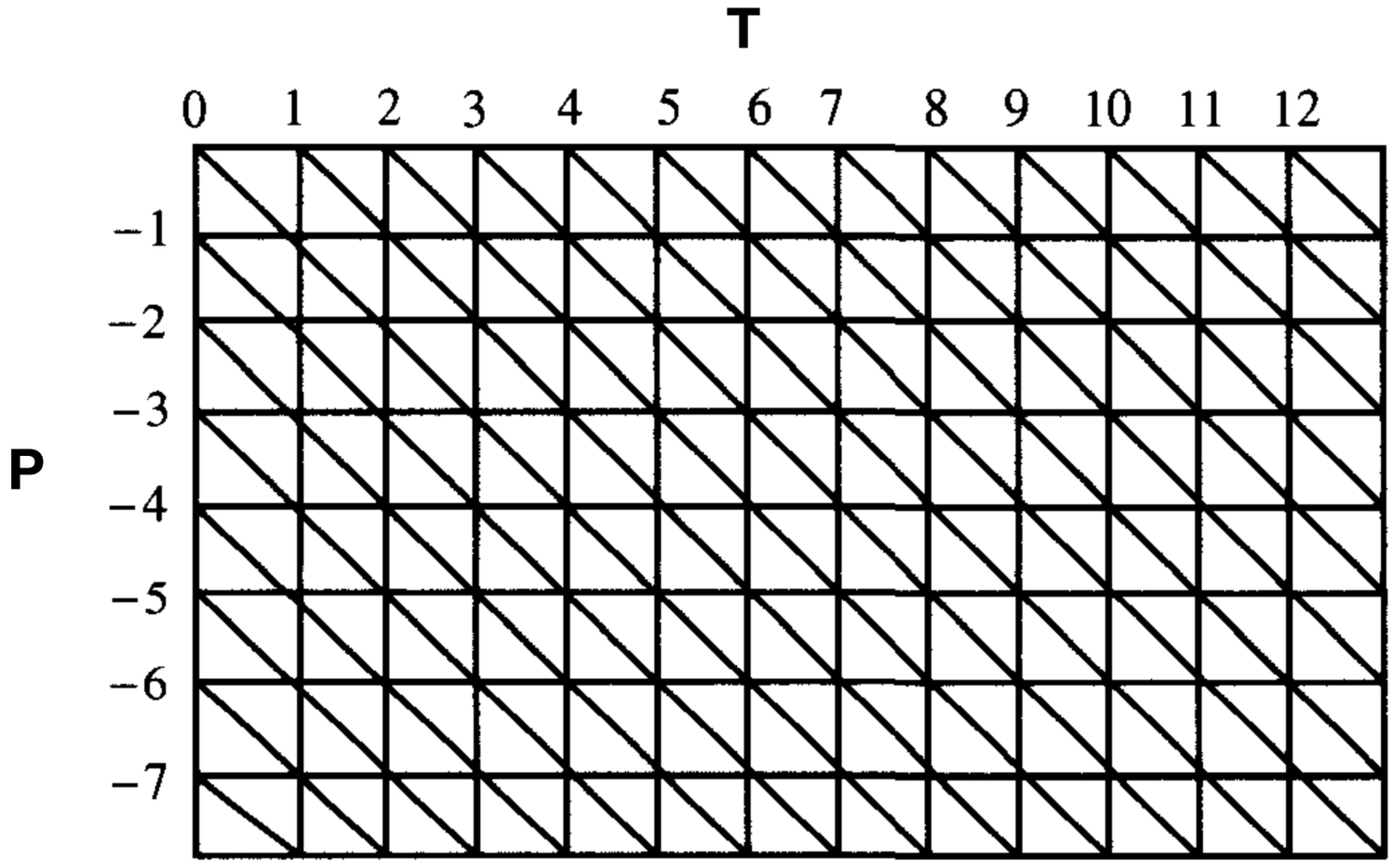
This problem is more difficult than both the k-mismatch problem, for which we have an $O(k|T|)$ solution using the suffix tree, and the k-difference global alignment problem, which can be solved in $O(k|T|)$ time using dynamic programming.

Nevertheless, the k-difference pattern matching problem can be solved combining the suffix tree with dynamic programming.

Theorem 2. The k-difference pattern matching problem can be solved in $O(k|T|)$ time after an $O(|T|+|P|)$-time preprocessing.

k-difference pattern matching

Let the main diagonal of the DP table be diagonal number 0. The diagonals above the main diagonal are numbered 1 through |T|: the diagonal starting in cell (0,i) is diagonal i. The diagonals below the main one are numbered -1 through -|P|: the diagonal starting at (j,0) is diagonal -j.



k-difference pattern matching

We initialise each cell in the first row of the DP table (row zero) to zero. This allows the pattern not to match the text up to the end, without paying for the remaining characters of T.

A d-path in the DP table is a path that starts at row zero and does exactly d edit errors.

A d-path is farthest-reaching in diagonal i if it is a d-path that ends in diagonal i, and the index of it's ending column (on diagonal i) is $\geq$ than the ending column of any other d-path ending in diagonal i.

The main idea of the O(k|T|)-time solution is to do k steps, each taking O(|T|) time, considering only d diagonals above and d diagonals below the main diagonal at each step $1 \leq d \leq k$.

k-difference pattern matching

At each step, a farthest-reaching d-path is found for each of the considered diagonals h, starting from the farthest-reaching (d-1)-paths on diagonals h-1, h and h+1.

Any farthest-reaching d-path reaching the last row at column c corresponds to an occurrence of P with edit distance exactly d ending at position c in T. The farthest reaching d-path on diagonal h is one of the three:

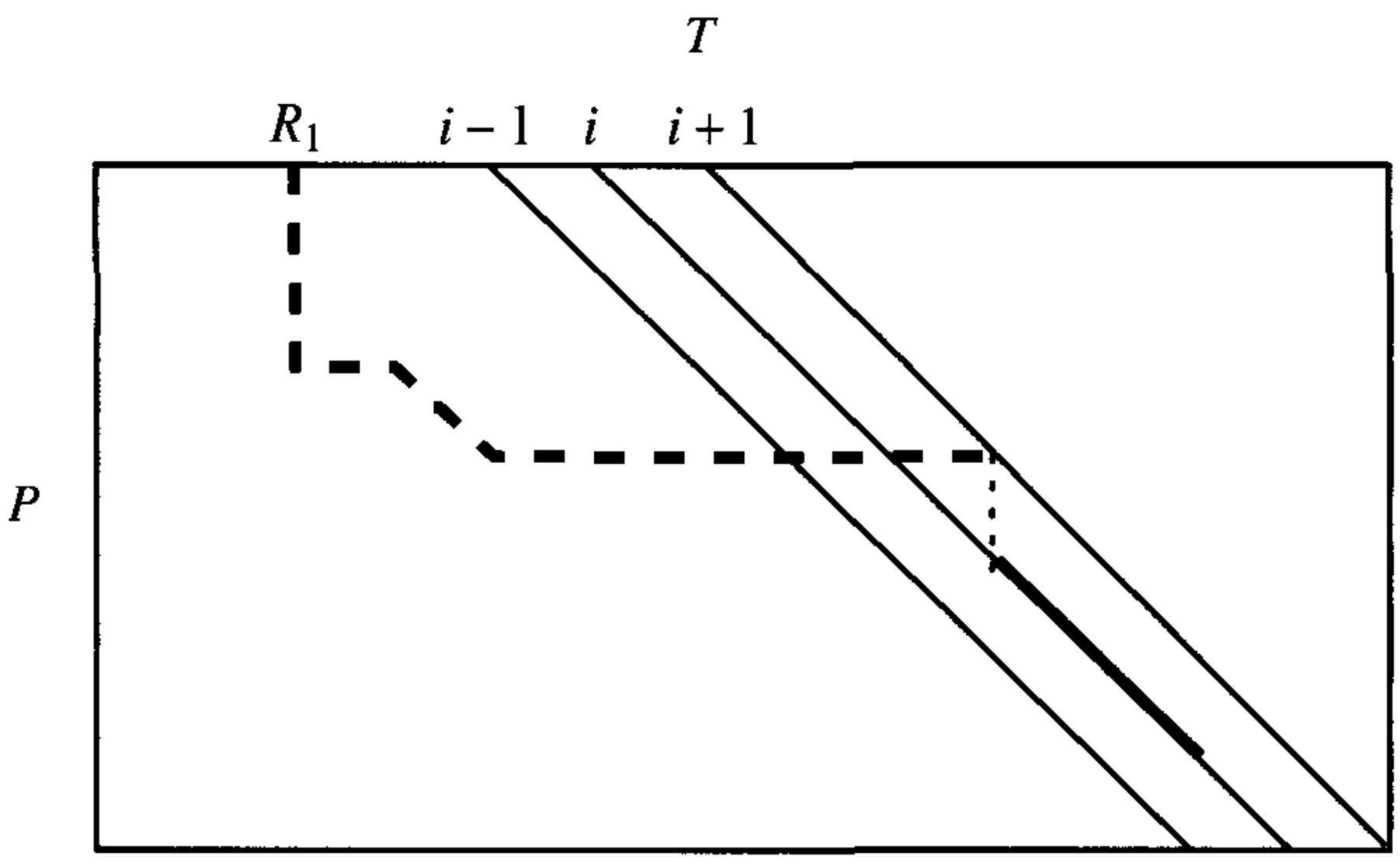
R_1: consists of the farthest reaching (d–1)-path on diagonal h+1, followed by a vertical edge to diagonal h, and then by the LCE along diagonal h. This path represents a deletion in T.

R_2: consists of the farthest reaching (d – 1)-path on diagonal h–1, followed by a horizontal edge to diagonal h, and then by the LCE along diagonal h. This path represents an insertion in T.

R_3: consists of the farthest reaching (d – 1)-path on diagonal h followed by a diagonal edge, and then by the LCE along diagonal h. This path represents a character replacement.

k-difference pattern matching

R_1: consists of the farthest reaching (d–1)-path on diagonal h+1, followed by a vertical edge to diagonal h, and then by the LCE along diagonal h. This path represents a deletion in T.



k-difference pattern matching

```
kDiff(T,P,k)
    FRP+←zeroes(|T|+1); \\stores the column of the farthest-reaching path on pos diagonals
    FRP-←zeroes(|P|); \\stores the column of the farthest-reaching path on negative diagonals
    sol ← ∅;
    for i=1…|T|
        ext←LCE_{T,P}(i,1);
        FRP+[i]←ext;
    for d=0…k
        for i=-d…|T|
            if i<0
                FRP-[i]←max_col{R_1^i,R_2^i,R_3^i};
                if FRP-[i]+i=|P| \\if the path on diagonal -i reaches the last row
                    sol.append{FRP-[i]};
            else
                FRP+[i]←max_col{R_1^i,R_2^i,R_3^i};
                if FRP+[i]-i=|P| \\if the path on diagonal i reaches the last row
                    sol.append{FRP+[i]};
```