

Indexed Approximate String Searching

Sayyed Bashir Sadjad

School of Computer Science

University of Waterloo

bssadjad@uwaterloo.ca

Outline

- **Indexed and online approximate string matching**
- Word vs substring searching
- Previous work and references
- Indexed approximate searching ideas
- Searching in metric spaces

Approximate String Matching

- Suppose we are looking for a family name, say *Lipschitz* in Google, but we don't know the exact spelling.
- On the other hand there may be lots of misspelling for this word over the pages in Web.
- So we need the search engine to look for *similar* words also.
- First we need a precise definition of *similarity*. Similarity is defined by the counterpart concept of *distance* between strings.

Edit Distance

- The most popular distance function is edit (or Levenshtein) distance, which is shown to be useful in Information Retrieval, Computational Biology, Signal Processing,
- **Definition.** The edit distance between two strings A and B of characters, i.e. $ed(A, B)$ is the minimum number insertion, deletion and substitution that will change A to B . For example $ed(\text{"Lipschitz"}, \text{"Lipchits"}) = 2$.
- **Approximate String Matching.** Given a *large* text $S = s_1s_2 \dots s_n$ and a *small* pattern $P = p_1p_2 \dots p_m$, we want to find each **approximate** occurrence of P in S , i.e. substrings $S_{ij} = s_i s_{i+1} \dots s_j$ where $ed(P, S_{ij}) \leq d$ where d is a given constant.

Edit Distance Computation

- The edit distance between two strings A and B can be computed in time $O(|A||B|)$ by a Dynamic Programming approach.
- Let $C_{i,j}$ be the edit distance between $A_{1,i}$ and $B_{1,j}$. We know $C_{i,0} = i$, $C_{0,j} = j$, and for $1 \leq i \leq |a|$ and $1 \leq j \leq |b|$ we have:

$$C_{i,j} = C_{i-1,j-1} \quad \text{if } a_i = b_j$$

$$C_{i,j} = 1 + \min\{C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}\} \quad \text{o.w.}$$

Indexed vs Online Searching

- For some applications, *grep* like, online search tools are enough or even necessary, e.g. text size $|S|$ is not huge or the text is dynamic.
- When we have a very huge text (e.g. the Web) in advance, since online tools can not respond in a reasonable time, we should build some kind of index structures over the text to speed up search.
- There is a reasonable large amount of work for online approximate matching ([Nav01] is a good survey), but the indexed approximate searching is premature.

Outline

- Indexed and online approximate string matching
- **Word vs substring searching**
- Previous work and references
- Indexed approximate searching ideas
- Searching in metric spaces

Word vs Substring Searching

- For text indexing, it is important that we are looking for *each* substring or there is a concept of *word* as the smallest element.

- Suppose that we are searching Web by Google:

Searching Google for *Tarkhoon*

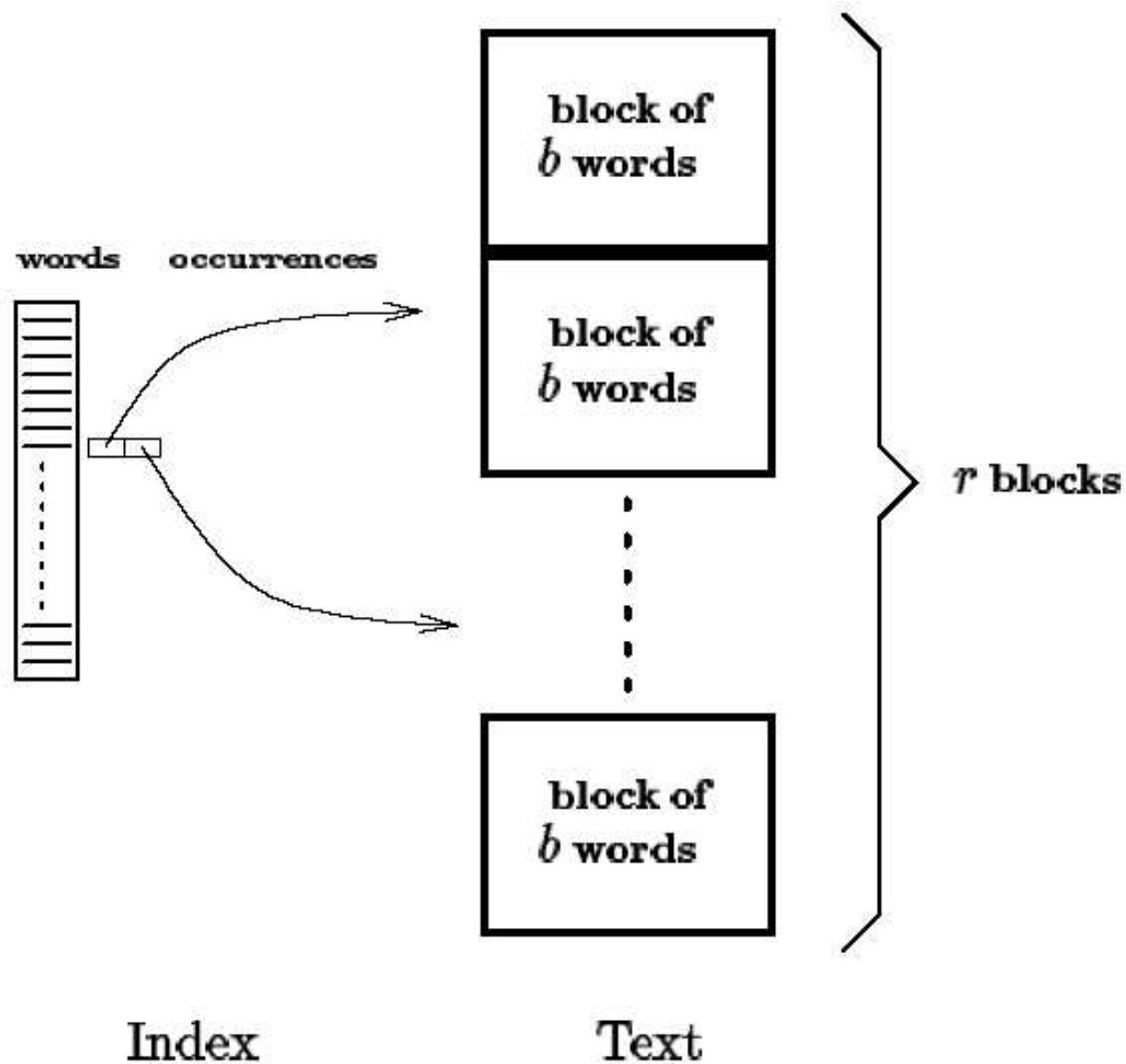
Searching Google for *Tarkhoo*

- One reason for using *words* is that vocabulary size grows slowly as text grows [BN97]. It is a well-known phenomenon in information retrieval called **Heaps' law** which says for large natural language text collections, vocabulary size is less than 1% of text size.

Word vs Substring Searching (cont.)

- One of the fastest and simplest way to indexing natural texts are *inverted files*, in which for each entry in vocabulary list there is a list of its occurrences in text.
- Since the number of occurrences may be large, **block addressing** is introduced [MW94].
- This idea has been changed a little to support approximate searching as well (e.g. GLIMPSE) [MW94, WM91].
- Why general substring searching? Applications like computational biology: searching in huge DNA or protein sequences allowing error.

GLIMPSE: GLObal IMPLICIT SEArch



Outline

- Indexed and online approximate string matching
- Word vs substring searching
- **Previous work and references**
- Indexed approximate searching ideas
- Searching in metric spaces

Previous Work and References

Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

Sun Wu and Udi Manber. agrep - A fast approximate pattern-matching tool. In *Proceedings of the Usenix Winter 1992 Technical Conference*, pages 153–162. Usenix Association, 1991.

Udi Manber and Sun Wu. GLIMPSE: A tool to search through entire file systems. In *Proceedings of the Winter 1994 USENIX Conference: 1994, San Francisco, CA*, pages 23–32, 1994.

E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, October 1994.

Edgar Chávez and Gonzalo Navarro. A metric index for approximate string matching. *Lecture Notes in Computer Science*, 2286:181–191, 2002.

Outline

- Indexed and online approximate string matching
- Word vs substring searching
- Previous work and references
- **Indexed approximate searching ideas**
- Searching in metric spaces

Indexing Ideas

- **Backtracking.** Some sort of *Branch and Bound* techniques on suffix tree of the text or suffix array, ...
- **Partitioning.** Split pattern to some smaller patterns to reduce search to a exact search (i.e. at least some of the smaller patterns should occur exactly).
- **Hybrid.** A combination of the previous methods.

Myers Indexing Structure

- In [CN02], it has been claimed that the best bound for general indexed searching (before this paper) is only sublinear in text size (n), i.e. it is n^λ in which $\lambda < 1$.
- One of these sublinear algorithms is proposed in [Mye94].
- The basic idea of Myers' technique is pretty simple: the pattern is partitioned to some smaller strings each with lengths logarithmic in n .
- For each pattern P of such size, all patterns P' for which $ed(P, P') \leq d$, are generated in advance and all occurrences of them in text are recorded in the index (similar to inverted files).

Outline

- Indexed and online approximate string matching
- Word vs substring searching
- Previous work and references
- Indexed approximate searching ideas
- **Searching in metric spaces**

What Is Searching in Metric Spaces?

There is a universe \mathbb{X} of *objects* and a *distance* function $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}$, defined on pairs of objects, satisfying:

- $d(a, b) \geq 0$ and $d(a, b) = 0 \Leftrightarrow a = b$ (strict positiveness)
- $d(a, b) = d(b, a)$ for all $a, b \in \mathbb{X}$ (symmetry)
- $d(a, b) \leq d(a, c) + d(c, b)$ for all $a, b, c \in \mathbb{X}$ (triangle inequality)

Our *database* is a subset \mathbb{U} of \mathbb{X} . For a *query* point $q \in \mathbb{X}$ we may perform *range* query or *k-nearest neighbor* query.

Example. Edit distance and the universe of all strings.

The Goal of Similarity Search in Metric Spaces

- The main goal is to index database by distance information between objects to reduce *the number of distance computations* in query time.
- It is supposed that the distance computations is a very expensive task (there are also some methods that consider I/O as well).
- There are lots of structures proposed for metric space indexing. One major category of these structures are pivot-based methods.

Pivot Based Indexing

1. Choose a set of pivots $\{p_1, p_2, \dots, p_k\}$ from the universe.
2. In preprocessing compute and store distance between each p_i and each object in our dataset.
3. Suppose the we are looking for objects o such that $d(q, o) \leq r$. We compute $d(q, p_i)$ and we discard an object o if:

$$|d(q, p_i) - d(p_i, o)| > r$$

Reason. By triangle inequality we have: $d(q, o) \geq |d(q, p_i) - d(p_i, o)|$.

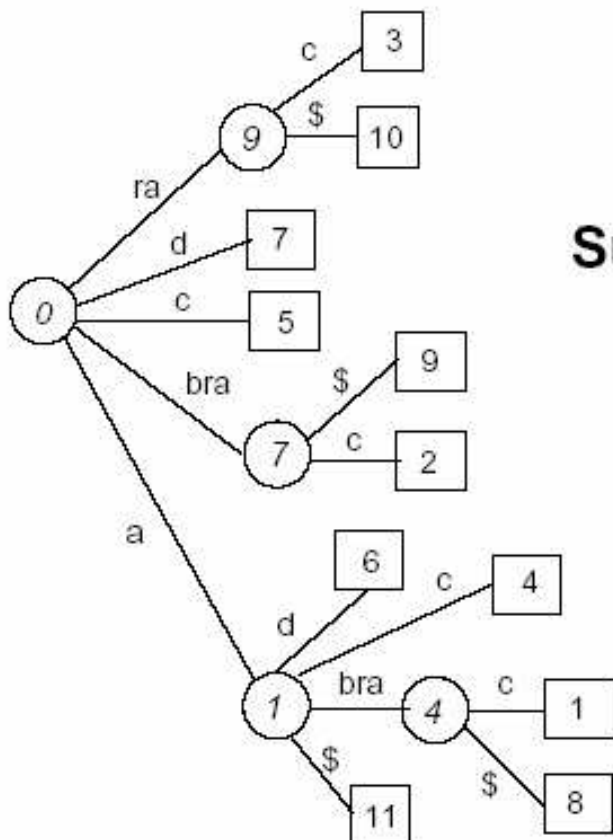
Notice. We discard o without direct distance computation.

How to Model Approximate Substring Searching

Problem. There are $O(n^2)$ sub-strings in a string of size n . So in the naive reduction there are $O(n^2)$ objects in the metric space. The following method is proposed in [CN02].

1 2 3 4 5 6 7 8 9 10 11
a b r a c a d a b r a

Text



Suffix Tree

Node	Suffix trie	Suffix tree	Node	Suffix trie/tree
$i(0)$	ϵ	ϵ	$e(1)$	abra[cadabra]
$i(1)$	a	[a]	$e(2)$	bra[cadabra]
$i(2)$	ab		$e(3)$	ra[cadabra]
$i(3)$	abr		$e(4)$	a[cadabra]
$i(4)$	abra	a[bra]	$e(5)$	[cadabra]
$i(5)$	b		$e(6)$	a[dabra]
$i(6)$	br		$e(7)$	[dabra]
$i(7)$	bra	[bra]	$e(8)$	abra
$i(8)$	r		$e(9)$	bra
$i(9)$	ra	[ra]	$e(10)$	ra
			$e(11)$	a

Project Plan

- In [CN02] it has been claimed that the running time of their pivot-based technique is $O(m \log^2 n + m^2 + R)$ in which R is the size of result.
- There are some issues about number of pivots and probability distribution assumptions of their method which need verification!
- The most effective method prior to this was sublinear in n , an exhaustive comparison between these types of approaches is needed. The claimed complexity of the new method is breakthrough over previous ones.
- Apply other distance based indexing methods to indexed substring searching problem.

This was generated by L^AT_EX!

```
\documentclass[landscape]{slides}
    % Package for slides
\usepackage{amsmath,amssymb,amsfonts}
    % Typical maths resource packages
\usepackage{graphics}
    % Packages to allow inclusion of graphics
\usepackage{color}
    % For creating coloured text and background
\usepackage{hyperref}
    % For creating hyperlinks in cross references
```