PRIN 2017 Kick-off meeting

DiSIT, Università Piemonte Orientale

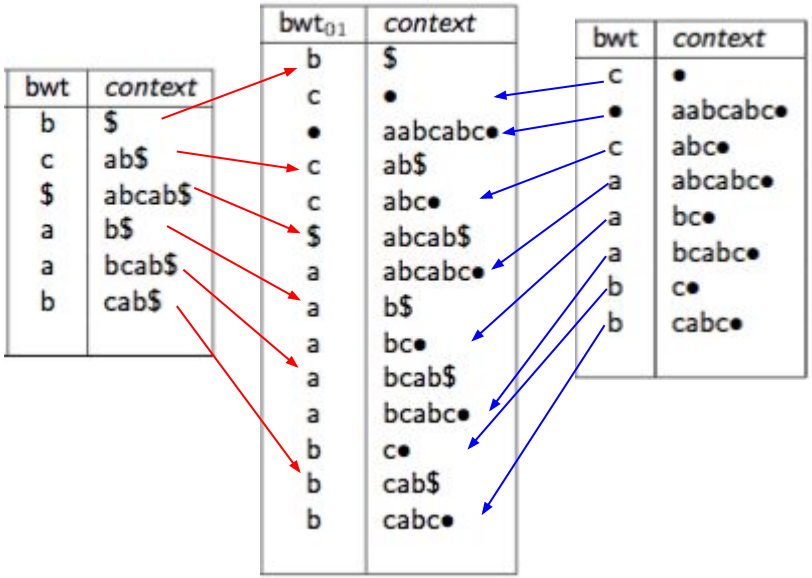
Research activities in the last 2 years

- Construction of compressed indices for very large data sets
- Merging of compressed indices
- External memory algorithms related to NGS reads
- Burrows-Wheeler Transform variants
- Indices for order preserving pattern matching

Holt&MacMillan technique for BWT merging (2014)

Given bwt_0 and bwt_1, obtain bwt_{01} without reconstructing the strings

- sort contexts by increasing length prefixes
- keep track of not ordered contexts
- time O(n $avelcp_{01}$)



Z
0
1
1
0
1
0
1
0
1
0
1
1
0
1

BWT and LCP merge/computation

Problem: given bwt_0/lcp_0 and bwt_1/lcp_1, obtain bwt_{01}/lcp_{01} without reconstructing the strings

(Egidi, Manzini, Spire 2017)

- technique: ignore blocks from the same BWT (irrelevant), keep track of length of prefix by which two contexts become different
- time $O(n\ avelcp_{01})$
- space $2n+O(\log n)$ + space to deal with irrelevant blocks

BWT and LCP computation in external memory

Problem: given a large collection of strings S and limited RAM, compute BWTs and LCPs.

(Egidi, Louza, Manzini, Telles, WABI 2018 and AMB 2019)

- technique: compute BWTs and LCPs for subcollections S_i of S using gSACA-K with S_i size constrained by RAM available, then merge the BWTs and LCPs (possibly in multiple rounds, according to memory availability)
- in addition: exploit sequentiality of read access and write access to move data structures to disk, LCP values written to files as soon as found
- flexible memory usage
- time: between O(n avelcp) and O(n maxlcp) I/Os (experiments: much faster than BCR)

Computation of maximal repeats (external memory)

Problem: find α that appears more than once

- Type 1: extensions have less occurrences
- Type 2: extensions occur only once

(Egidi, Louza, Manzini, Telles, WABI 2018 and AMB 2019)

- technique: keep track of LCP variation
- given bwt and lcp, single scan of each
- time O(n)
- space O(1) RAM

All pairs suffix-prefix overlaps (external memory)

Problem: for each pair of sequences *s* and *t*, find the longest overlap of the suffix of *s* and the prefix of *t*, of length at least τ

(Egidi, Louza, Manzini, Telles, WABI 2018 and AMB 2019)

- required: *bwt*, *lcp*, document array *da* and *xlcp*
  - xlcp[i] = 1 if suffix starting at sa[i] is prefix of suffix starting at sa[i+1]
  - da and lcp computed while computing/merging BWTs/LCPs
- technique: inspired by Ohlebusch&Gog, 2010, for each sequence *s* maintain a stack with longest common prefix with other sequences
- single scan of bwt, lcp, da and xlcp
- time $O(n+E_\tau)$, (E_τ number of repeated overlaps),
- space O(m + maxlcp) RAM (m input sequences),

Succinct de Bruijn graphs (external memory)

Problem: construct the succinct BOSS representation of the de Bruijn graph

(Egidi, Louza, Manzini, Telles, WABI 2018 and AMB 2019)

- technique: suffix array range [refixed by the same k-mer is identified by a maximal range with LCP values at least k
- single scan of bwt and lcp
- time O(n)
- space O(1) RAM

Merge of BOSS representations of de Bruijn graphs

Problem: given BOSS representations of de Bruijn graphs of collections S_1 and S_2, obtain BOSS representation of de Bruijn graph of $S_1 \cup S_2$ (without reconstructing S_1 and S_2)

(Egidi, Louza, Manzini, SPIRE 2019)

- technique: extension of BWT/LCP merge technique
- single scan of bwt and lcp
- time O(mk), with m = number of edges, k = order of the graph
- space 4n+O(σ) RAM, with n = number of nodes, σ = alphabet size

Merging other compressed indices (1)

Problem: given two compressed indices of collections S_1 and S_2, obtain the compressed index for the union collection. Cases: compressed labelled trees, compressed indices of circular patterns and compressed permuterm indices.

(Egidi, Manzini,TCS 2019, to appear)

- Compressed labelled trees (tries)
- Need to detect identical suffixes in the two collections: we compute here a sort of LCP array to this end
- time O(|T| hgt(T)), with T the merged trie and hgt(T) its height
- space 4n+ O(log n) bits, with *n* the total number of nodes in the input tries

Merging other compressed indices (2)

t=abaa s = aba
t^∞ = abaaabaaabaa.. s^∞ = abaabaaba…
$t^\infty \leqslant^\infty s^\infty$

(Egidi, Manzini,TCS 2019, to appear)

- Compressed indices of circular patterns
  - only primitive strings; in each collection, no string is a rotation of another string
- data structures $length_0$ and $length_1$ give the lengths of the rotations; required to tell when a string of one collection is a rotation of a string of another collection
- time O(n avelcp)
- space 2n+ O(log n) bits, plus space to handle irrelevant blocks

Merging other compressed indices (3)

t=abaa# s = aba#
t^∞ = abaa#abaa#... s^∞ = aba#aba#…
look for a#ab:
find all strings prefixed by ab, suffixed by a

(Egidi, Manzini,TCS 2019, to appear)

- Compressed permuterm indices (for prefix/suffix query)
  - only primitive strings; in each collection, no two strings are the same
- strings terminated by an end-of-string symbol (equal for all strings): count occurrences of # to determine that two rotations from two collections are identical
- time O(n avelcp)
- space 6n+ O(log n) bits, plus space to handle irrelevant blocks

Prefix-free parsing (1)

Technique for parsing a sequence T[1,n] in words

$$T = p_1\ p_2\ \ldots\ p_k$$

such that if T contains many repetitions many words p_i are repeated. Define the dictionary D of distinct words p_i and the parsing of T in terms of dictionary words.
Example:

T = baaaaabaaaabaaaabbccbccbaaaaabbaaaaabca

D = {aaaab, baaaaab, bcc, ca} P = 2 1 1 3 3 2 2 4

Prefix-free parsing (2)

We can solve many problems on T working on the much smaller sets D and P.

- BWT computation for genome collections [WABI ‘18]
- Applications to the r-index [Recomb 19 e J. Compu. Biology ‘19]
- Grammar compression [Spire ‘19]

BWT variants based on context adaptive orderings

In [DLT 18, CPM 19] we introduced and analyzed new BWT variants computable and invertible in linear time and supporting compression and indexing like the original BWT.

These variants forma a parametric family of BWT alternatives where the user can choose the one more suitable for his/her task.

Order Preserving Pattern Matching

Given a sequence of integers $T = a_1\ a_2\ a_3\ \ldots\ a_n$ we want to create an index for the order preserving pattern matching problem: given a pattern $P = x_1\ x_2\ \ldots\ x_m$ find all positions where T's elements appear in the same relative order of those in P.

Example:

T = 10 11 13 15 08 09 10 100 110 20 21 50 60 51 52

P = 1 2 3 5 4

In [SPE '19] we proposed a family of indices offering trade-offs between storage space and search speed. Recent results [SODA '17, Spire 19] suggest further improvements are within reach.

Ongoing and future activities related to the project

- Learned FM-index
- Compressed Linear Algebra
- Probabilistic Suffix Tree
- PPM* vs DeepZip
- ML Security

Compressed Linear Algebra

A recent paper by Elgohary et al. “Compressed linear algebra for large-scale machine learning” has shown the advantages of matrix compression to speedup some machine learning computations. This paper was best paper at VLDB and featured in CACM

The key idea: data compression improves speed by reducing access costs. This is something our community has used for years. Elgohary et al. mainly used heuristics, we plan to achieve better results in a more principled way

Probabilistic Suffix Tree

We plan to consider learning problems solved using Probabilistic Suffix Trees that makes predictions based on the current context.

We plan to improve PSTs performance using compressed indices and to compare them with NN approaches.

PPM* vs DeepZip

We plan to compare traditional prediction based compressors with new approaches based on NN.

The problem is related to Probabilistic Suffix Trees but now we have to maintain an adaptive model.

ML security

attacks to

- training (data manipulation)
- inference
  - attacks to privacy (of training data, of model)
  - attacks to integrity
    - meaningful for classifiers (alter input so that it is misclassified)
    - or for reinforced learning (alter environment to have the system misbehave)

(Papernot et al. SoK: Security and Privacy in Machine Learning. Euro S&P, 2018)