PETER

Fast similarity searches and similarity joins in Oracle DB

Astrid Rheinländer, Ulf Leser

Humboldt-Universität zu Berlin

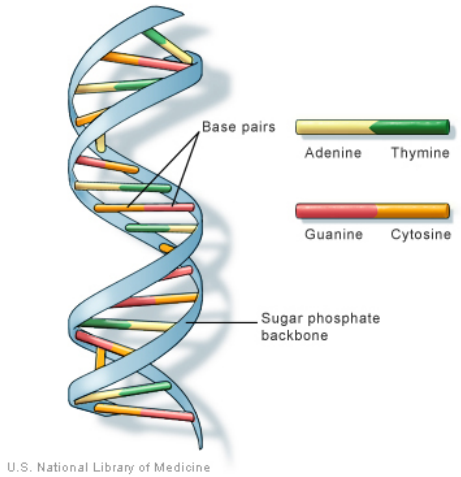
Department of Computer Science

Knowledge Management in Bioinformatics

Motivation

- Approximately searching DNA sequences is important many fields of modern genomics
  - 1,160,000 citations of BLAST in Google Scholar
- ESTs (Expressed Sequence Tags) are small portions of DNA
  - Find (near-)duplicates
  - Find homolog sequences

  → Similarity based search and join algorithms needed



- dbEST strings contains more than 60 million records

  → efficient execution is crucial

Our contribution - PETER

- Prefix tree based index structure
  - Combines many tricks for query speedup
- Exact and similarity based search and join queries
  - Similarity measures: Hamming and Edit distance
- Evaluated on real data from dbEST [NCBI, online, 1992]
- Real software
  - standalone Unix command line tool
  - Plugin for Oracle DB

Outline

- PETER-Index
- Pruning strategies
- Integration to Oracle DB
- Results + Conclusion

Idea of PETER

Idea: manage ESTs according to shared prefixes

→ prefix tree index [Shang et al, IEEE TKDE, 1996]

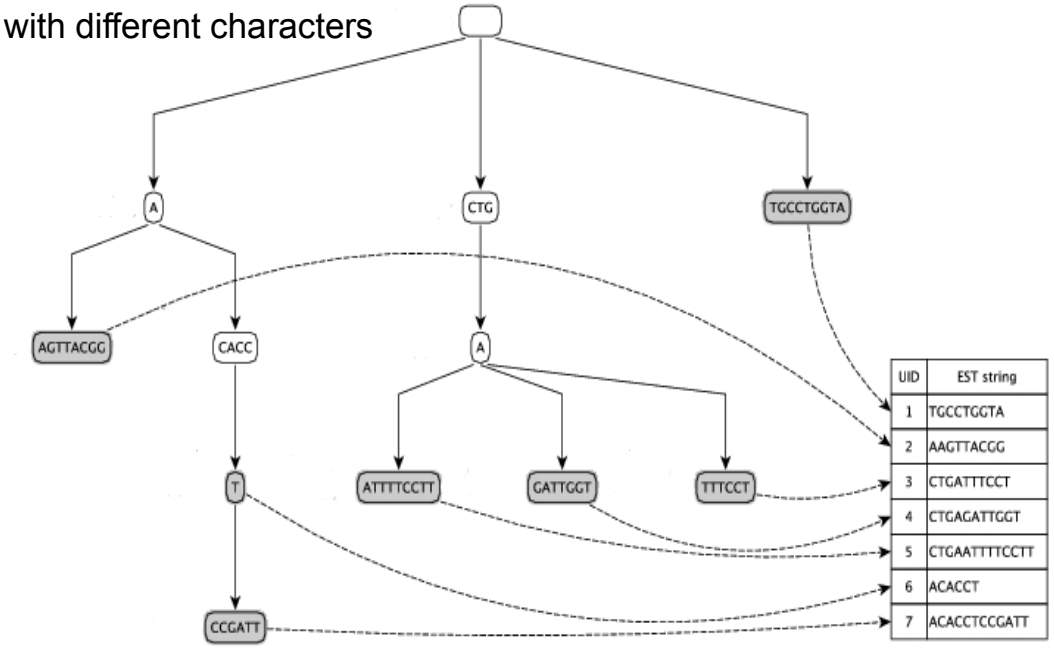
- strings with common prefixes occur in one subtree
- Nodes are labeled with some substring of the EST
- any two children of some node start with different characters

Goal: minimize query response time

→ exclude whole subtrees early from search space

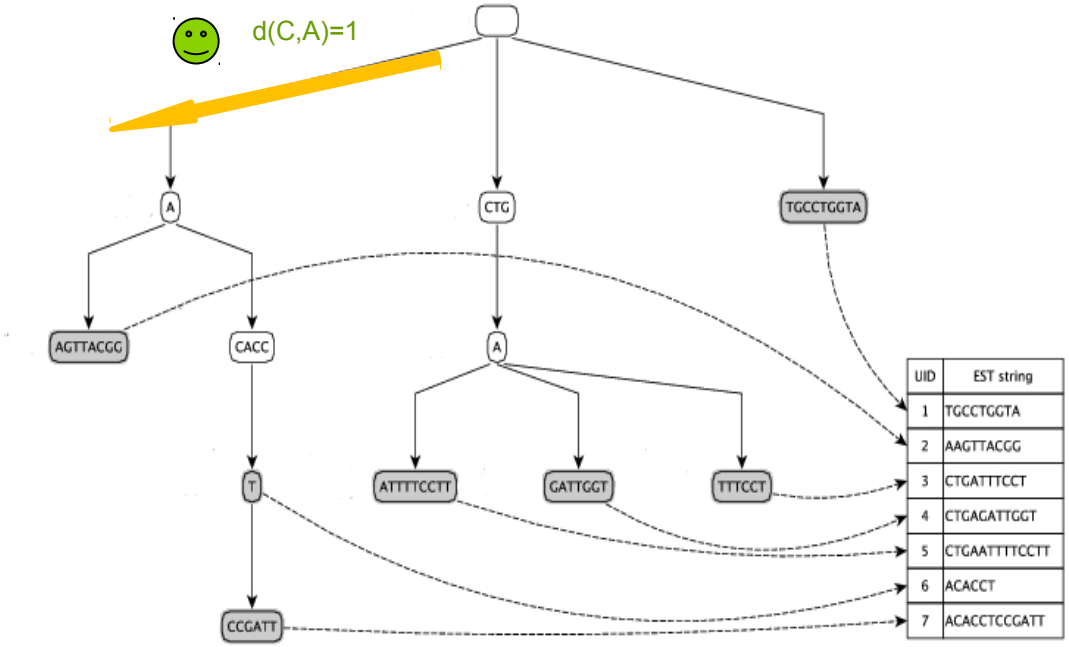
→ add information on length and character frequency for pruning

Operations are based on DFS traversal



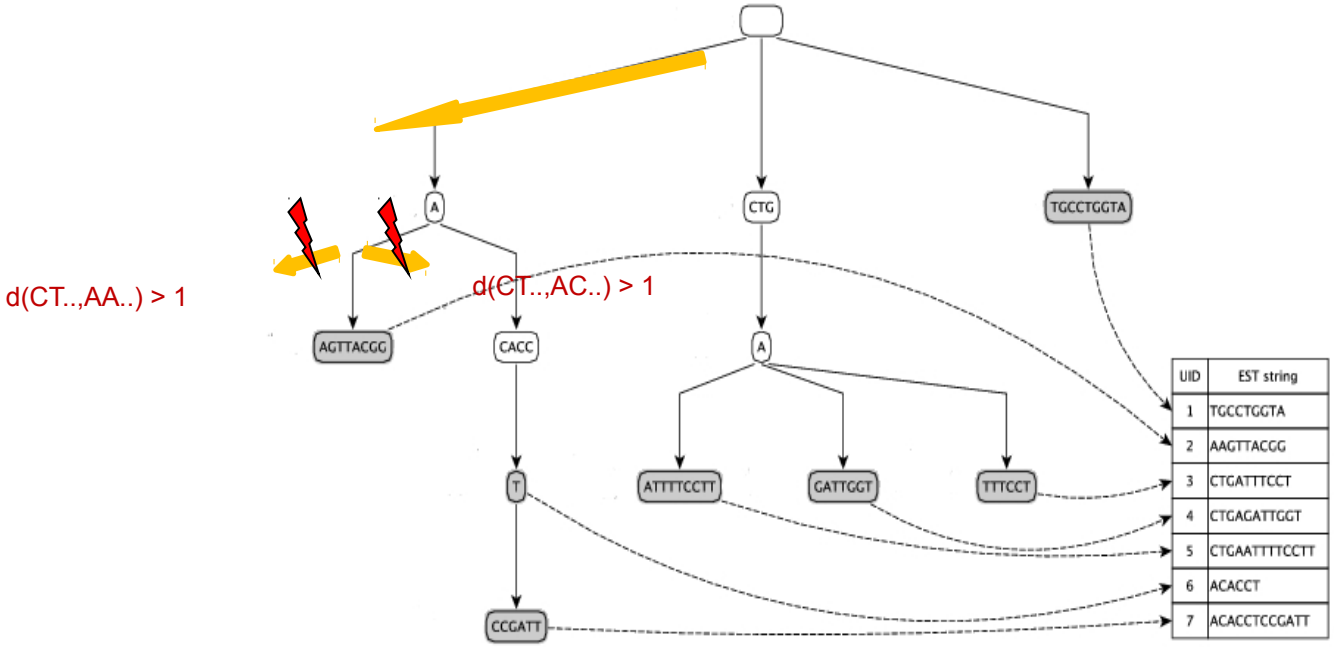
Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



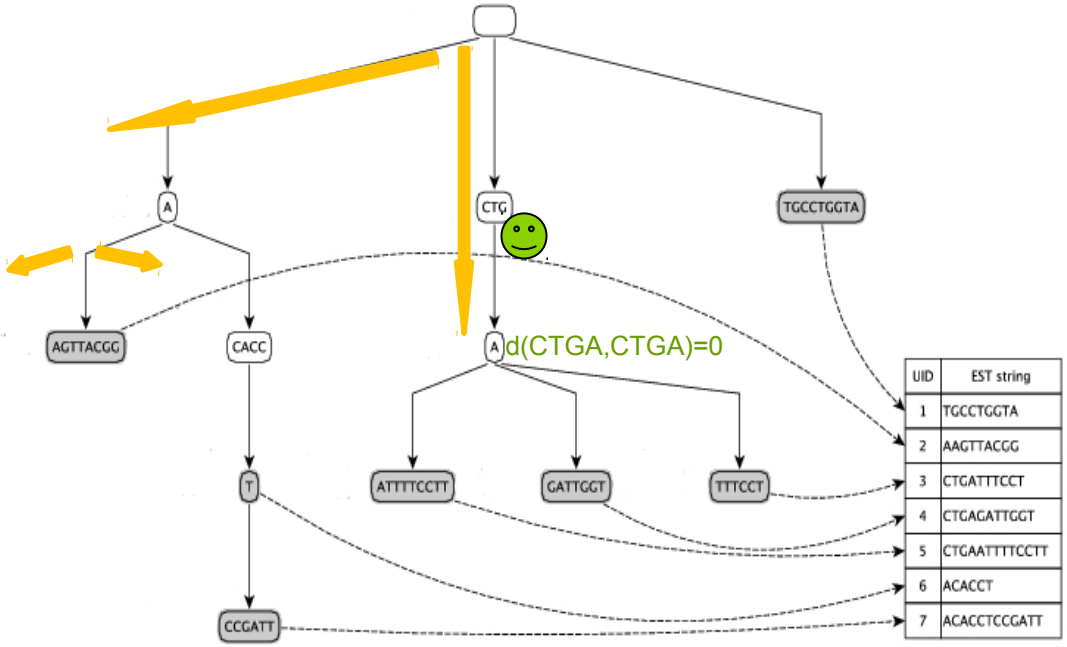
Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



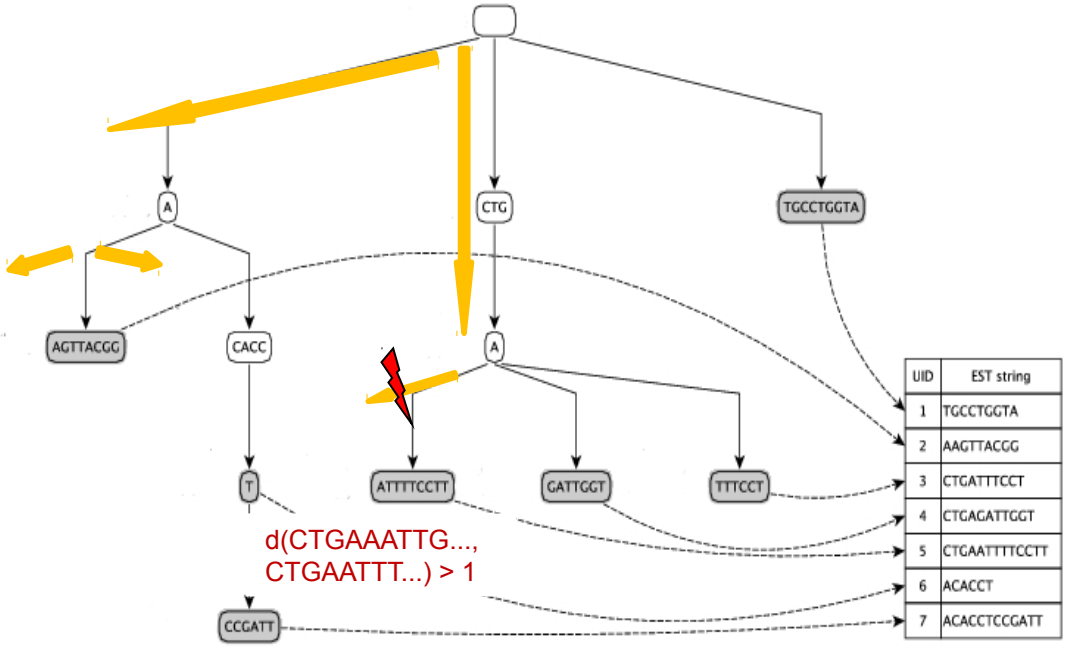
Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



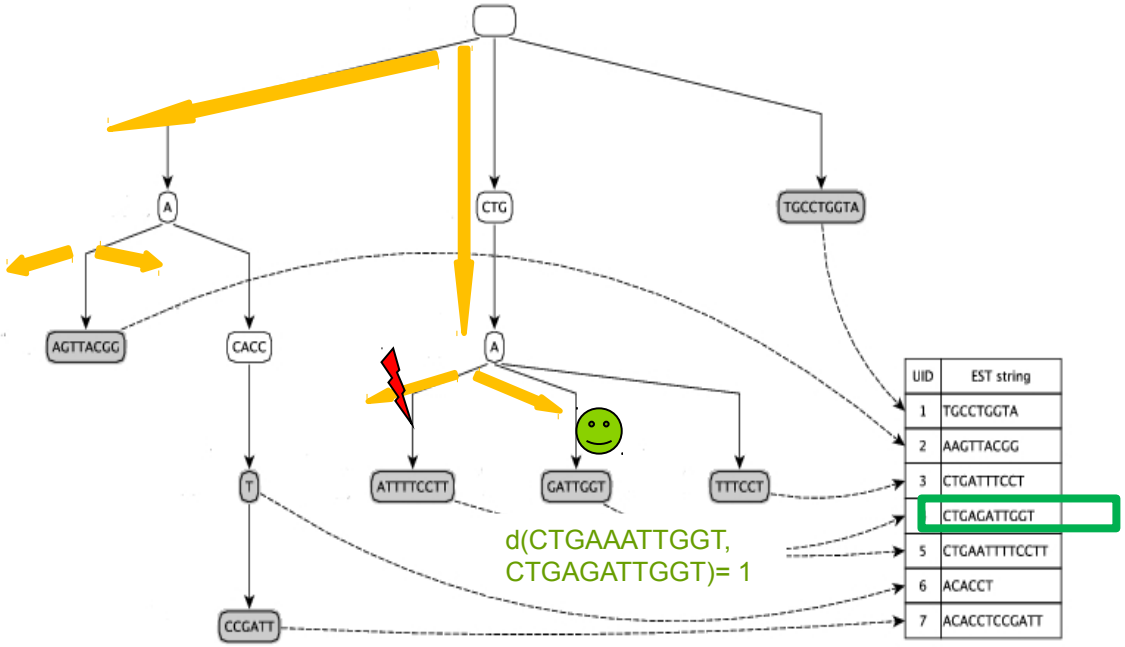
Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



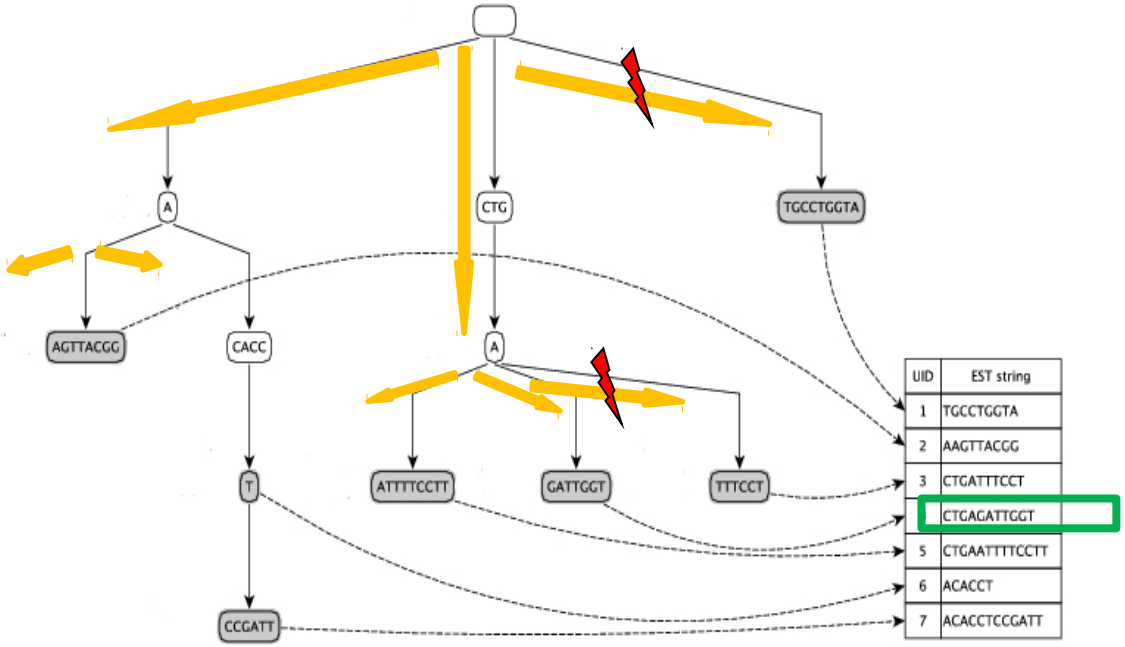
Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



Example: Search

Hamming distance search for p = CTGAAATTGGT, k=1



Filter by string length

Assumptions

- Hamming distance
  - two strings p,t are only worth examining if they are of equal length
- Edit distance
  - p and t are worth examining only if $|t|-|p| \leq k$.

- Idea
  - include min/max string lengths in the index
  - at each node x, we know the length of the longest and shortest string that starts with prefix t[1..x]

→ E.g., Hamming distance:

if ($|max(x)| < |p|$) ∨ ($|min(x)| > |p|$) then

prune subtree x

Filter by character frequency

- Basically evaluates the character frequencies in the strings
- Gives a lower bound for Hamming and edit distance
  [Aghili et al., String processing and Information Retrieval, 2003]
- Not very effective due to small alphabet size
- Details omitted

Filter by q-grams

- No. of mismatching q-grams gives lower bound to edit distance
  [Xiao et al., VLDB, 2008]

- computing q-grams is on avg. cheaper than computing edit distance directly

- used for suffix pre-selection
  - Computed on the fly when EST node is reached
  - ≥ 90% of our strings have long, unique suffixes
  - edit distance will probably exceed threshold in the suffix part

- Not evaluated for Hamming distance queries

Lessons learned

runtime improvements by filtering increases with threshold

→ we achieve a speed up of query response time up to 80 %

Best configuration for PETER:

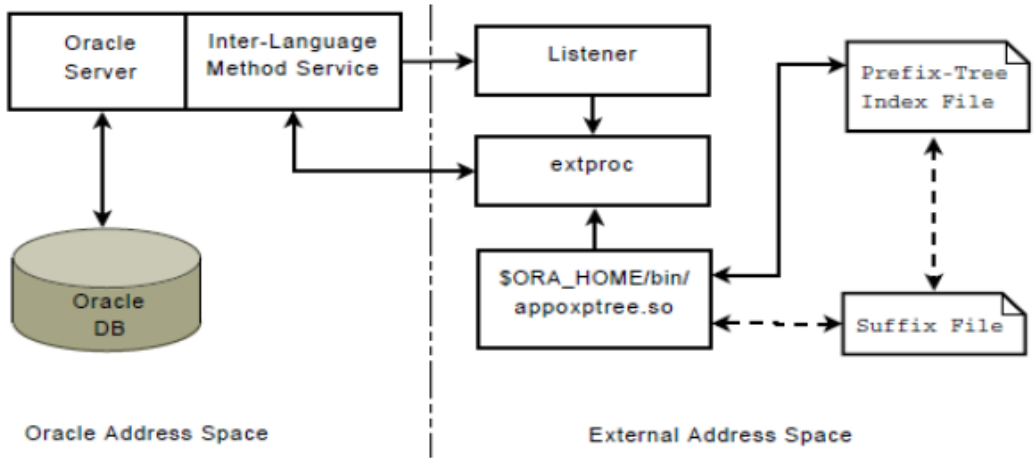
→ combination of length and q-gram filter

Integration into Oracle DB

- integrated as shared library using ODCI
  - in Oracle Express 10g
  - provides functionality for user-defined indexes and table-functions
- two steps
  - compile code as shared library and save it in $ORA_HOME/bin
  - PL/SQL scripts that defines and registers the index and functions in the DBS
  - straight-forward

Execution in Oracle

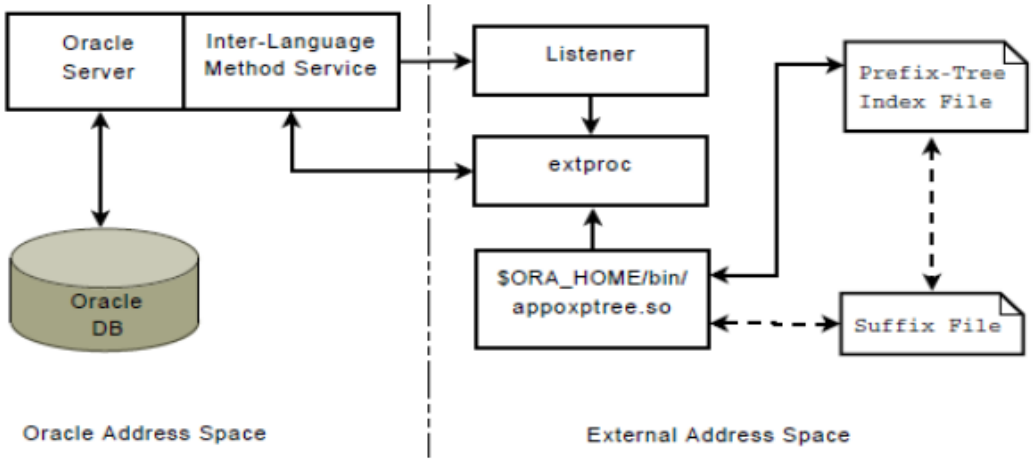
- first access in a session
  - PL/SQL locates PETER via Data dictionary
  - listener invokes extproc and passes procedure parameters
    - extproc remains alive during session
    - initialization costs emerge only once



Execution in Oracle

- extproc
  - loads PETER-lib and invokes function calls

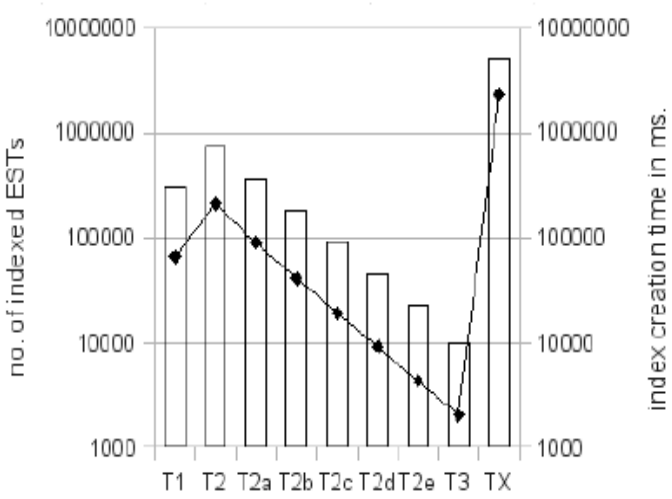
    → opens index and performs desired operation inside PETER
  - return values are passed back via extproc



Results - Setup

- Competitors for similarity operations:
  - Unix tools agrep, nrgrep and Flamingo library
  - build-in and UDFs inside the Oracle
- EST sets extracted from dbEST
- Index creation and optimization was done in advance, not included in the measured times
- time for index creation grows linear with the number of indexed strings

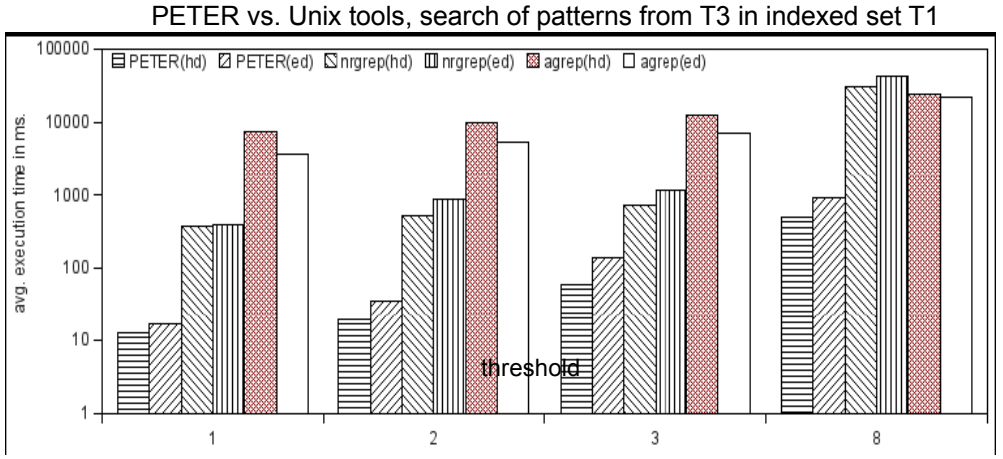
Set	# EST strings	avg. string length	min/max length	# tree nodes	# ext. suffixes
T_1	307,542	348	14/3,615	589,062	293,764
T_2	736,305	387	12/3,707	1,482,709	689,590
T_{2a}	368,152	382	12/2,774	711,632	352,872
T_{2b}	184,076	385	22/2,774	349,329	177,846
T_{2c}	92,038	383	25/2,774	171,964	89,198
T_{2d}	46,019	381	28/2,774	84,954	44,716
T_{2e}	23,009	373	31/ 878	42,375	22,366
T_3	10,000	536	16/3,707	16,310	8,774
TX	5,000,000	359	14/3,247	10,478,214	4,834,231



Results - Similarity Search

- Agrep: bounded with pattern length of 32 chars
- nrgrep: arbitrary patterns allowed

→ PETER is orders of magnitudes faster in both cases



- Comparison is a bit unfair
  - Agrep and nrgrep do not build an index
- Indexing amortizes fast
  - Takes 15 searches to outperform agrep, 105 searches for nrgrep

→ PETER more suited for searching the same set of strings multiple times

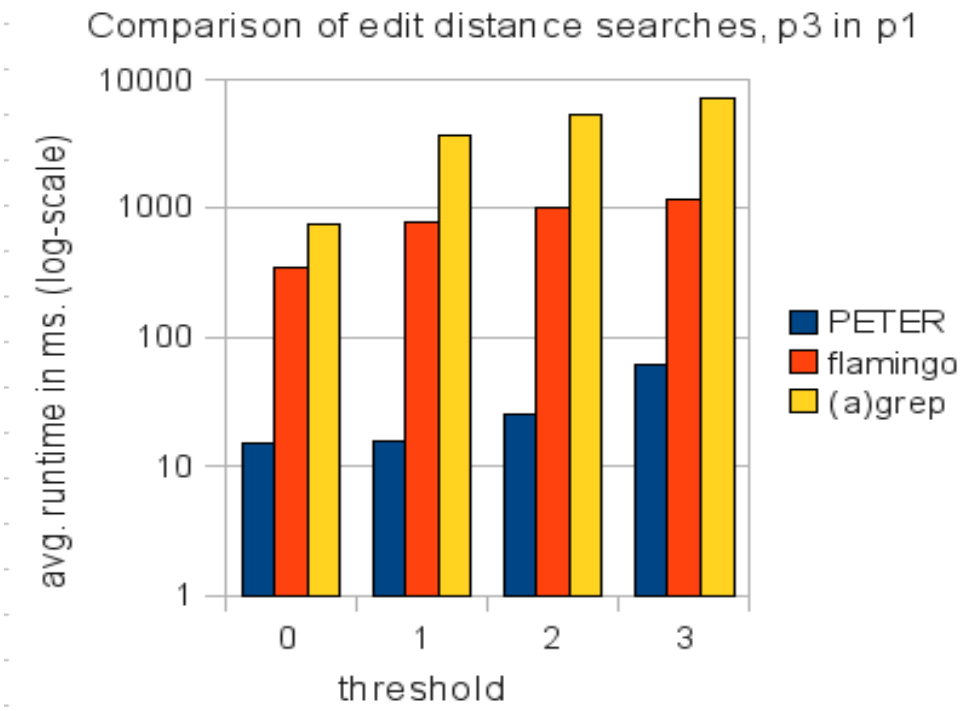
Results – Similarity Search

- Compared to Flamingo Package for Approximate String Matching
  [Li et al., online 2007]

- Flamingo
  - creates an inverted index on q-grams
  - Uses a length and a charsum filter

- PETER is orders of magnitudes faster

PETER vs. Flamingo, search of patterns from T3 in indexed set T1



Results – Similarity Join

- Application: Find common ESTs in human and mouse
- no Unix command line tool found for comparison
- Flamingo is currently not available with join option
- Generally
  - joins on Hamming distance always perform better than joins on Edit distance (30 to 60 %)
  - Join execution time grows exponentially with respect to the threshold
  - But the result sets don't grow exponentially

→ Reason: search space increases exponentially with growing k

Results – inside Oracle DB

- exact search:
  - built-in SELECT-operator on B*-indexed relation performs better than PETER
    - factors between 1.3 and 2 (depending on string length)
  - reasons
    - handling via extproc produces overhead for each call to PETER
    - no caching for user-defined indices possible in ORACLE

- exact join
  - compared to Hash-Join and Sort-Merge join
  - PETER always outperformed built-in operators
    - Hash-Join: Factors 1.5 to 4
    - Sort-Merge Join: Factors 3.8 to 10
  - caching is not severe here

    → both indexes need to be loaded only once

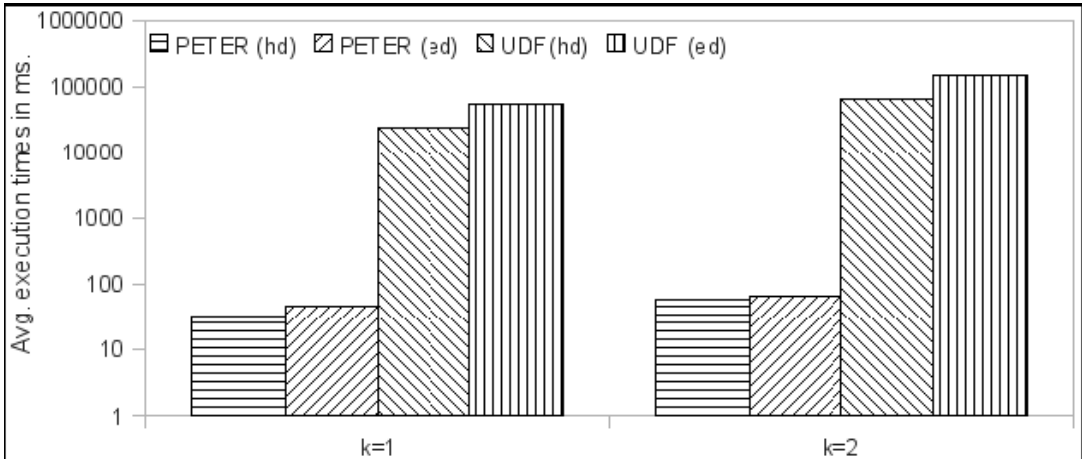
Results – inside Oracle DB

- No similarity operations included in Oracle

- Sim. Search:

  PETER outperforms UDFs with orders of magnitudes

Performance of similarity search inside Oracle Express



- Sim. Join: tried to perform join for k = 1 with UDFs on smallest test sets
  - Did not finish within a day, aborted
  - PETER returns result within a minute

Conclusion

PETER...

- is an efficient data structure for genomic strings
- is used for similarity search and similarity joins on Hamming or Edit distance
- Contains some tricks for query speedup
- outperforms all other methods we compared to
  - either inside or outside a RDBMS
  - in all performed similarity operations
- also outperforms exact joins inside a RDBMS

Thank you!

Questions?

PETER - Index

Only the shortest unambiguous prefix is stored in the tree

Suffixes are stored apart in an extra file

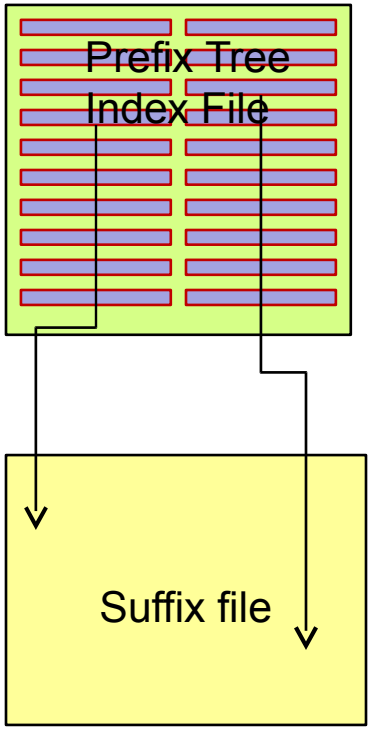
Suffixes are commonly quite large for ESTs

allows to keep the prefix tree itself in main memory
even for very large string collections

Example: Index file for 5,000,000 strings has a size

549 MB on disk

943 MB in main memory



Similarity Join – Result Sets

Set	# EST strings	avg. string length	min/max length	# tree nodes	# ext. suffixes
T_1	307,542	348	14/3,615	589,062	293,764
T_2	736,305	387	12/3,707	1,482,709	689,590
T_{2a}	368,152	382	12/2,774	711,632	352,872
T_{2b}	184,076	385	22/2,774	349,329	177,846
T_{2c}	92,038	383	25/2,774	171,964	89,198
T_{2d}	46,019	381	28/2,774	84,954	44,716
T_{2e}	23,009	373	31/ 878	42,375	22,366
T_3	10,000	536	16/3,707	16,310	8,774
TX	5,000,000	359	14/3,247	10,478,214	4,834,231

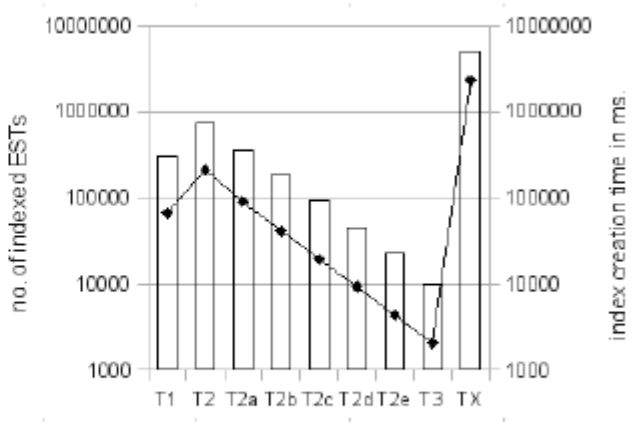


Fig. 2. Left: Properties of EST sets. Right: Index creation (line) wrt. set size (bar).

Join	$\lvert A \times B \rvert$	$\lvert A \bowtie B \rvert$ $k=0$	$\lvert A \bowtie_{hd} B \rvert$ $k=1$	$\lvert A \bowtie_{hd} B \rvert$ $k=3$	$\lvert A \bowtie_{ed} B \rvert$ $k=1$	$\lvert A \bowtie_{ed} B \rvert$ $k=3$
$T_1 \bowtie T_2$	226,444,712,310	299	514	841	941	2,552
$T_1 \bowtie T_{2a}$	113,222,202,384	187	326	374	239	463
$T_1 \bowtie T_{2b}$	56,611,101,192	128	214	236	152	301
$T_1 \bowtie T_{2c}$	28,305,550,596	78	127	178	75	232
$T_1 \bowtie T_{2d}$	14,152,775,298	42	71	91	46	165
$T_1 \bowtie T_{2e}$	7,076,233,878	28	41	55	33	106
$T_1 \bowtie T_3$	3,075,420,000	1,048	1,053	1,059	1,058	1,082
$T_{2e} \bowtie T_3$	230,090,000	54	70	106	94	258
$T_1 \bowtie TX$	1,537,710,000,000	37	115	372	354	992

Fig. 7. Join cardinalities for Hamming distance ($\bowtie_{hd}$) and edit distance ($\bowtie_{ed}$) join.