A Modification of the Landau-Vishkin Algorithm Computing Longest Common Extensions using Suffix Arrays

Rodrigo de Castro Miranda
Maurício Ayala-Rincón
Grupo de Teoria da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Topics

- Edit Distance
- Approximate String Matching with *K* Errors
- Landau-Vishkin Algorithm
- Modified Landau-Vishkin Algorithm
- Results and Further Work

Edit Distance - 1

- Given two strings *T* and *P* of length *n* and *m*, the *Edit Distance* between *T* and *P* is the minimum number of operations required to transform *T* into *P*
- Allowed operations are
  - Substitution
  - Insertion
  - Removal

Edit Distance - 2

- Can be solved by a *Dynamic Programming* algorithm
  - *O(mn)* time complexity
  - *O(m)* space
- The sequence of operations can be retrieved by using a set of traceback pointers
- The traceback pointers form a path in the *Dynamic Programming Table*

Edit Distance - 3

- *P* = *ontology* and *T* = *anthological*

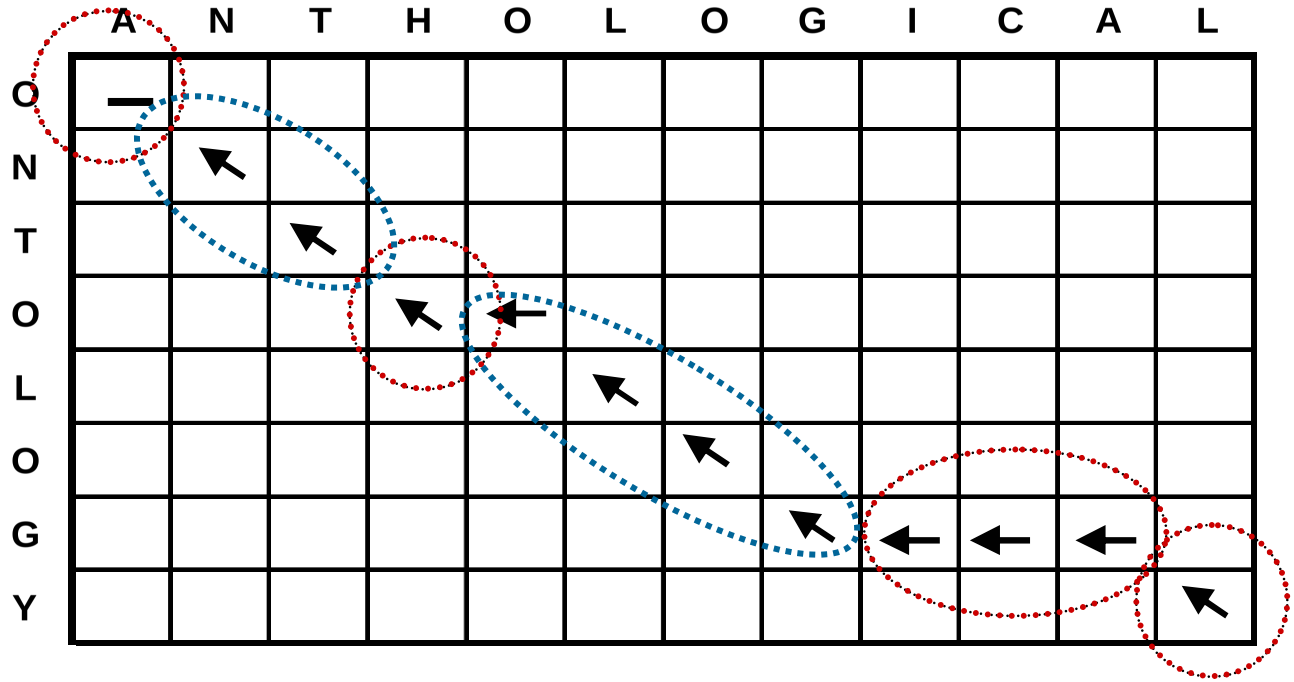
		A	N	T	H	O	L	O	G	I	C	A	L
		1	2	3	4	4	5	6	7	8	9	10	11
O		2	**1**	2	3	4	5	6	7	8	9	10	11
N		3	2	**1**	2	3	4	5	6	7	8	9	10
T		4	3	2	**2**	**2**	3	4	5	6	7	8	9
O		5	4	3	3	3	**2**	3	4	5	6	7	8
L		6	5	4	4	3	3	**2**	3	4	5	6	7
O		7	6	5	5	4	4	3	**2**	**3**	**4**	**5**	6
G		8	7	6	6	5	5	4	3	3	4	5	**6**
Y													

O N T – O L O G - - - Y

A N T H O L O G I C A L

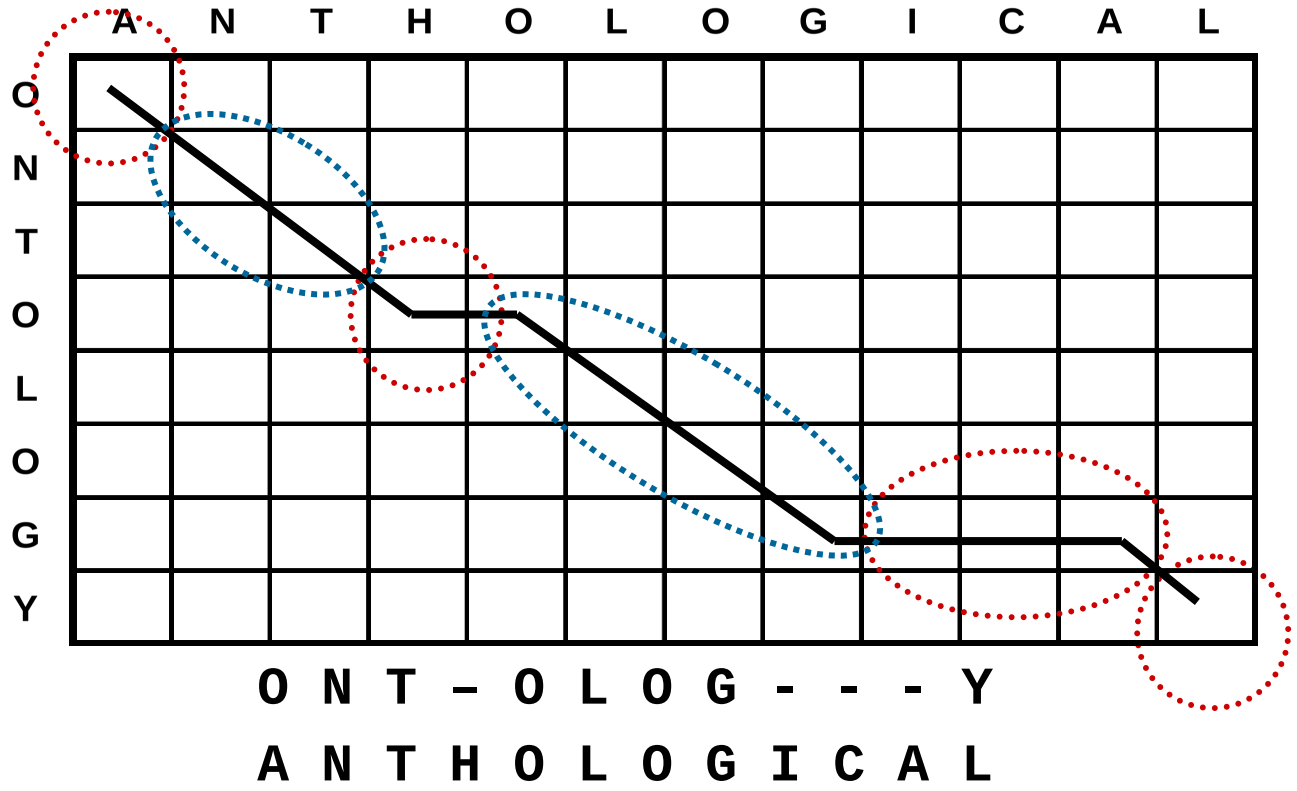
Edit Distance - 4

- *P = ontology* and *T = anthological*



Edit Distance - 5

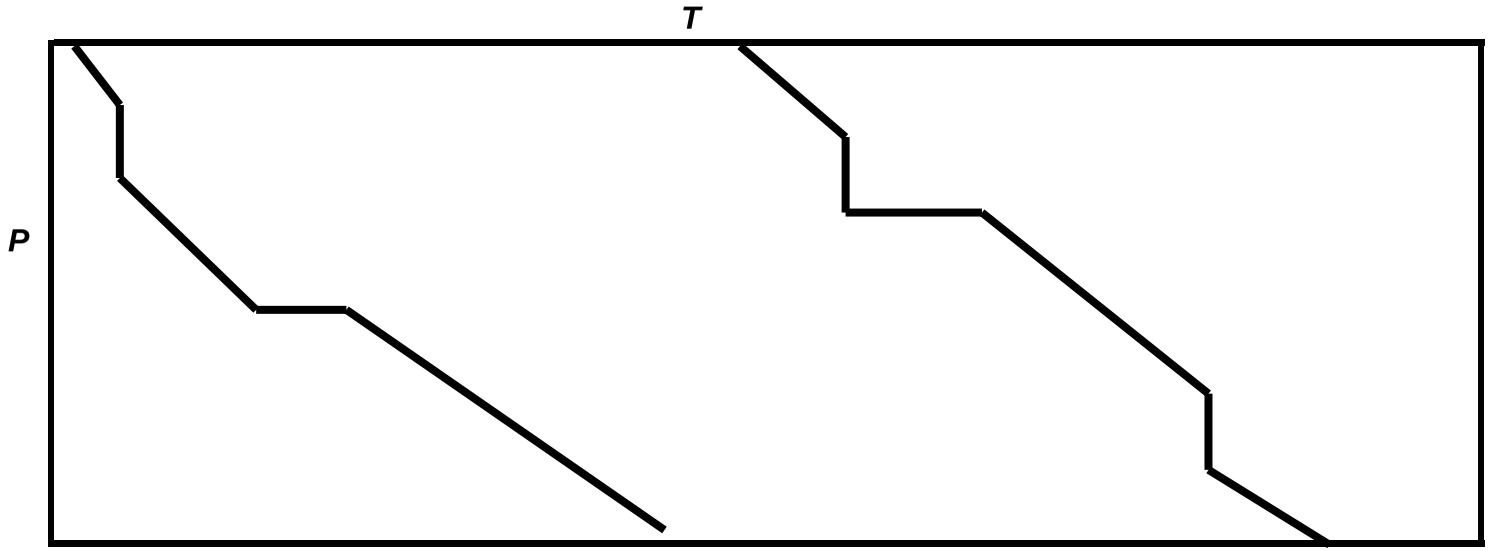
- *P* = *ontology* and *T* = *anthological*



Aproximate String Matching with *K* Errors - 1

- Given the pattern *P* and the text *T*, the *Approximate String Matching Problem with K Errors* is the problem of finding all positions of *T* where *P* occurs with at most *K* differences
- Solved by a variation of the Edit Distance Algorithm

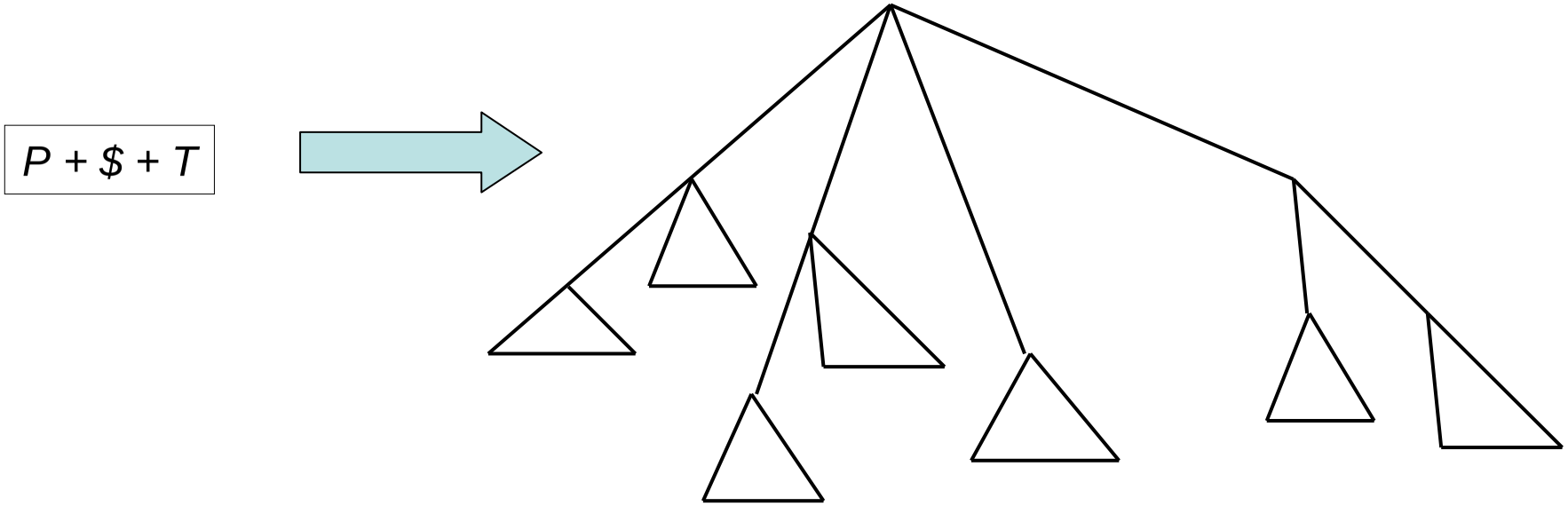
Aproximate String Matching with *K* Errors - 2



Landau-Vishkin Algorithm - 1

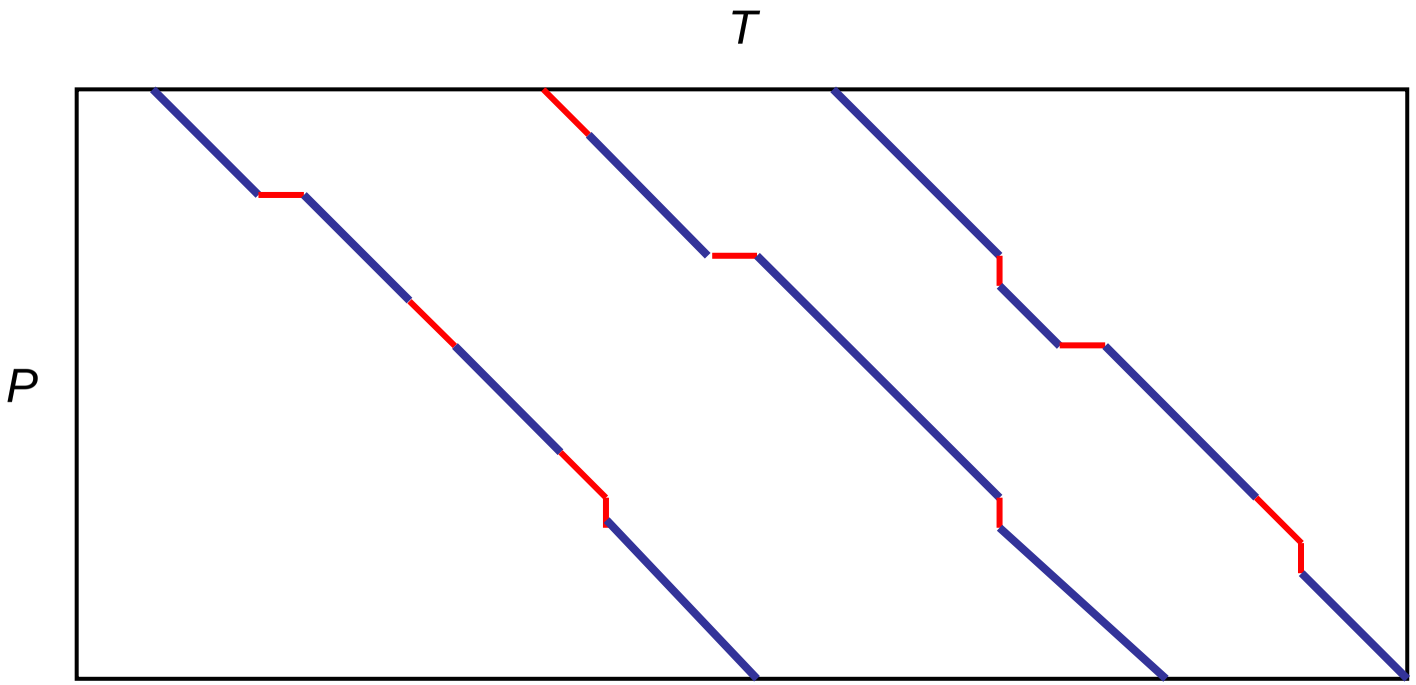
- Finds all matches with at most *K* errors of *P* in *T* in *O(kn)*
- Two phases
  - Pre-processing
  - Iteration
- Uses a *O(1)* Suffix Tree LCA query for computing the Longest Common Extension of any two suffixes

Landau-Vishkin Algorithm - 2



- Preprocessing Phase
  - Builds a Suffix Tree for the concatenation of *P* and *T*
  - Preprocesses the tree for answering *O(1)* LCA queries between suffixes of *P* and *T*

Landau-Vishkin Algorithm - 3



- Iteration Phase
  - Iterates through the diagonals building paths with at most *k* errors

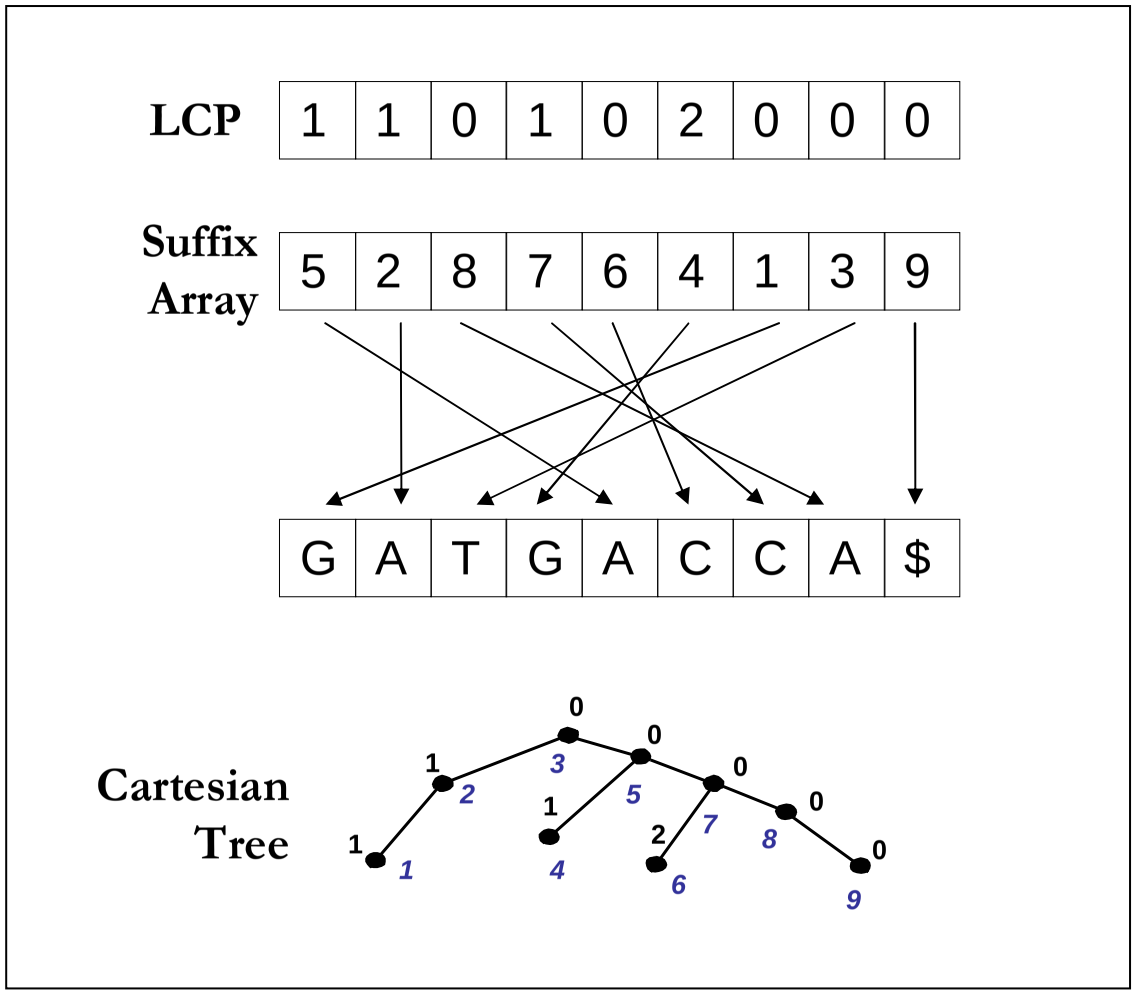
Landau-Vishkin Algorithm - 4

- Why use a suffix tree
  - *O(n)* time and space
  - The LCA of two leaves gives us the Longest Common Prefix of the corresponding suffixes
  - It can be preprocessed in O(n) for a O(1) LCA query
- Why ***not*** use a suffix tree
  - O(n) space → *12n – 28n*
  - Complex construction algorithm
  - Difficult to distribute

Modified Landau-Vishkin Algorithm - 1

- Changes only the preprocessing phase
- Keeps the complexity bounds while saving a little space
- Uses *Suffix Arrays* with a LCP table
- The Longest Common Extension will be the smallest value of the interval in the LCP table – given by a *Cartesian Tree*

Modified Landau-Vishkin Algorithm - 2



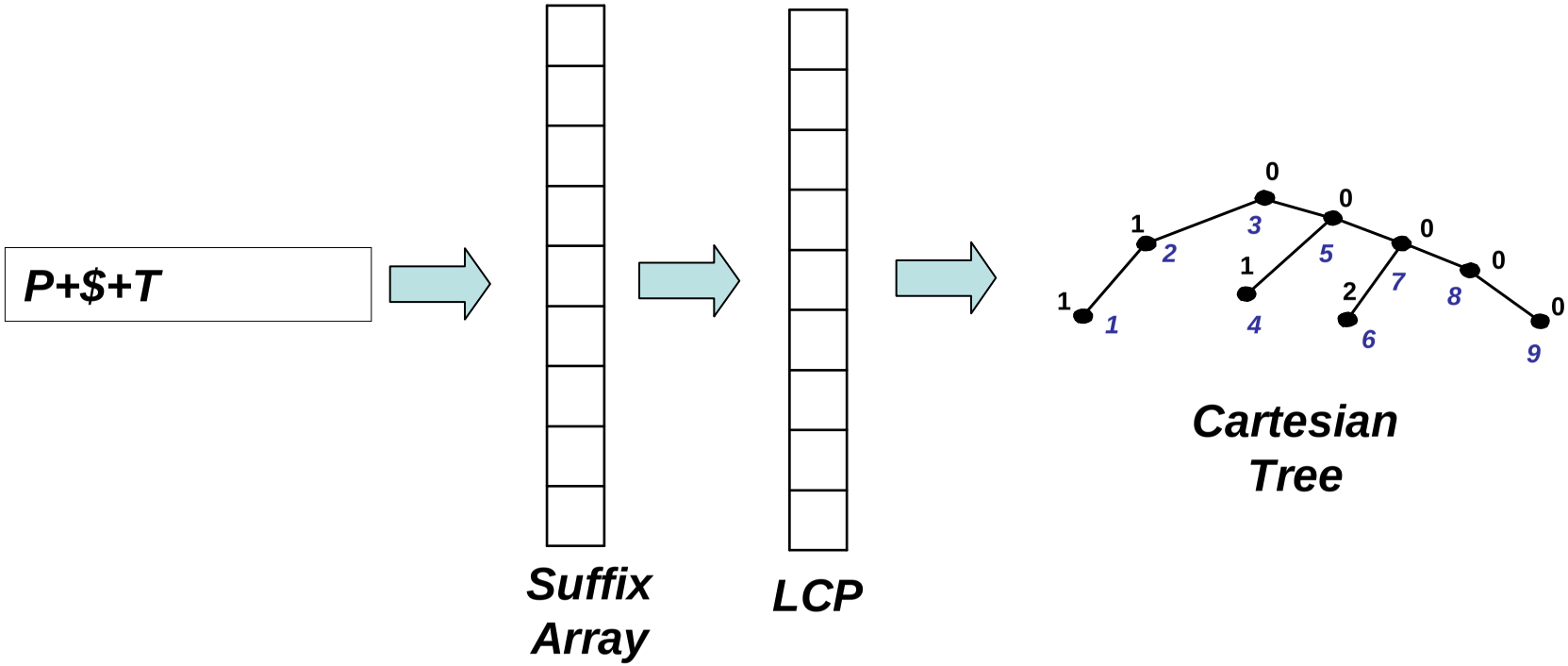
T_5	ACCA$
T_2	ATGACCA$
T_8	A$
T_7	CA$
T_6	CCA$
T_4	GACCA$
T_1	GATGACCA$
T_3	TGACCA$
T_9	$

A Modification of the Landau-Vishkin Algorithm using Suffix Arrays

Modified Landau-Vishkin Algorithm - 3

- Changes to the preprocessing phase:
  - Builds a suffix array for the concatenation of *P* and *T* with a LCP table
  - Builds a Cartesian Tree for the LCP table
  - Preprocesses the Cartesian Tree for *O(1)* LCA queries

Modified Landau-Vishkin Algorithm - 4



Results and Further Work - 1

- Theoretical results
  - Since the iteration phase is exactly the same, we are only reporting the preprocessing phase results
  - Economy of an average ***6n*** space over a optimized suffix tree for DNA (≥*18n* over **strmat** suffix tree construction)
  - Preprocessing is expected to be slower than with suffix tree (although it is also *O(n)* it takes more time to build the sufffix array)

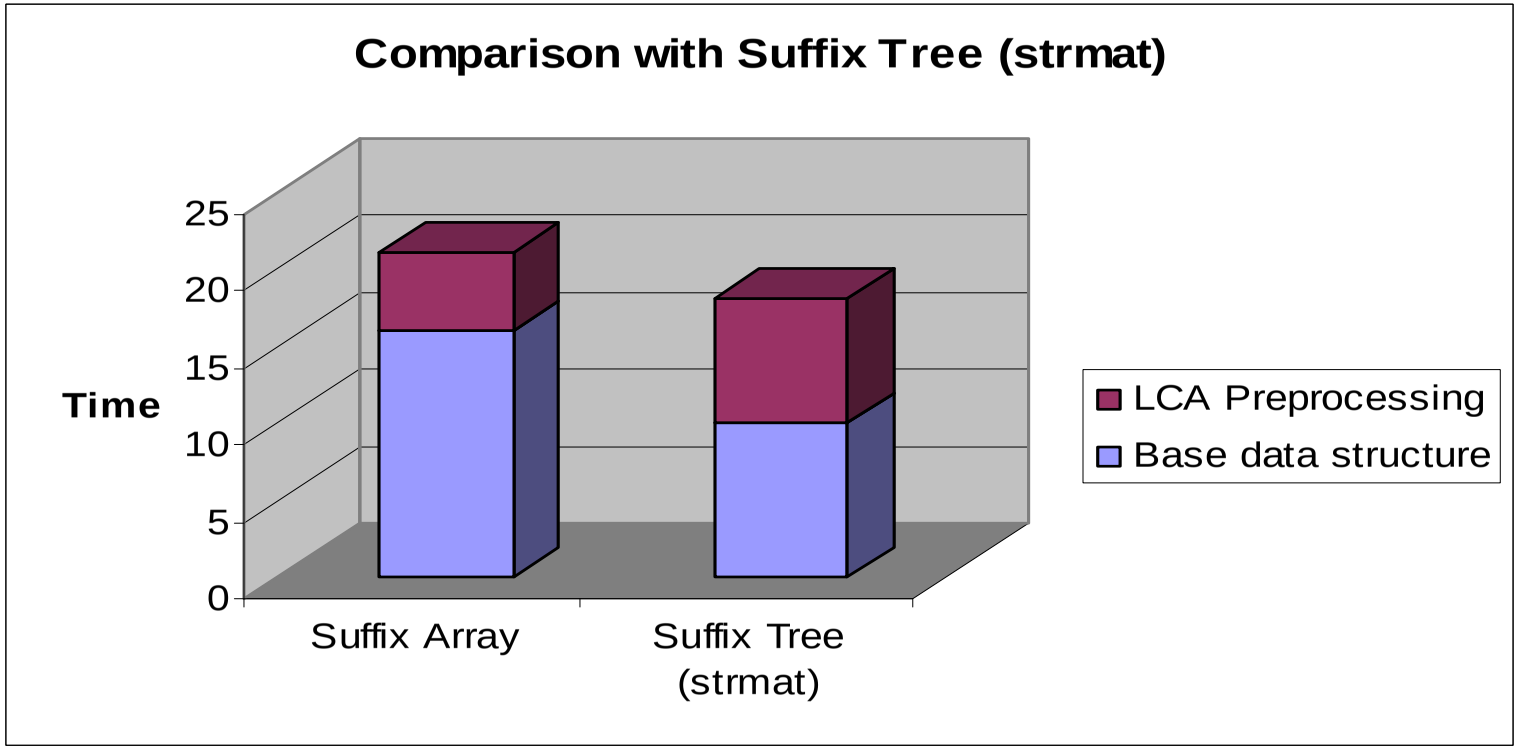
Results and Further Work - 2

- Experimental results - First Results
  - Comparison with **strmat**
  - Sequences from GenBank ~3M base pairs

	Runing Time (s)	***Memory Usage (MB)***
Suffix Array	**21**	73
Construction	**16**	40
LCA Preprocessing	8	33
Suffix Tree (strmat)	18	446
Construction	10	**390**
LCA Preprocessing	8	**56**

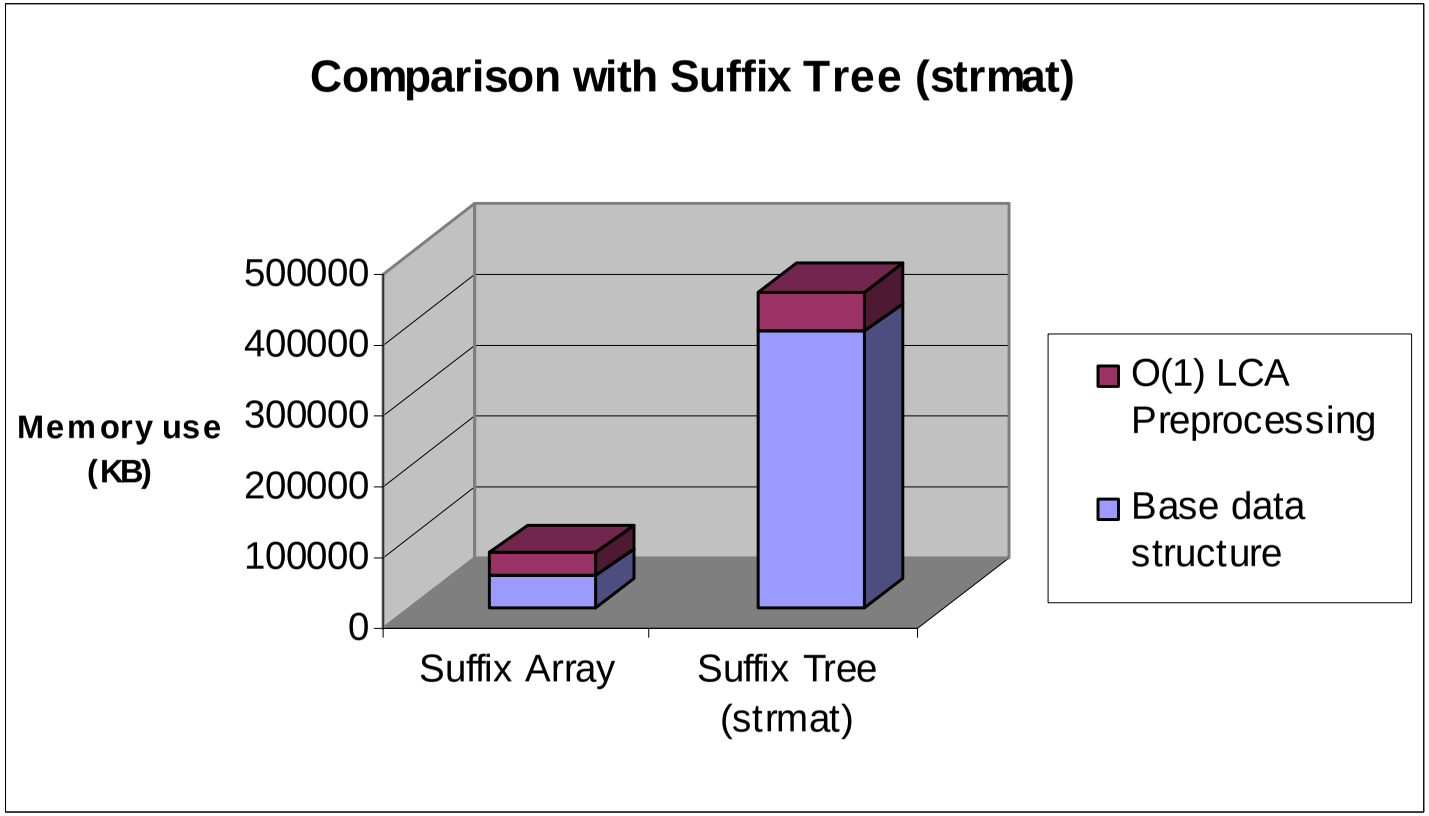
Results and Further Work - 3

- Experimental results – Running time



Results and Further Work - 4

- Experimental results – Memory time



Results and Further Work - 5

- Experimental results (**strmat**)
  - Preprocessing is generally 10-15% slower than with *suffix trees* because of the *suffix array* construction
  - 80-90% memory usage improvements for DNA strings
  - Used [Ko & Aluru 2001] Suffix Array Construction Algorithm

Results and Further Work - 6

- Future Work
  - Redo the memory use measurements for the suffix tree
  - Enhance the suffix array construction algorithm for improving the running time
  - Compare with more compact/faster suffix tree implementations
  - Compare with other string matching algorithms
  - Modify the algorithm for
    - Distribution
    - Handling maximization problems (similarity with scoring matrixes)

Gracias por su atención e interés

rodrigo.miranda@acm.org

ayala@mat.unb.br

Work supported by CNPq grant 471791/2004-0
Presentation supported by FINATEC