Sequence alignment in XSLT 3.0

David J. Birnbaum
djbpitt@gmail.com
http://www.obdurodon.org

XML Prague, 2020-02-14

Outline

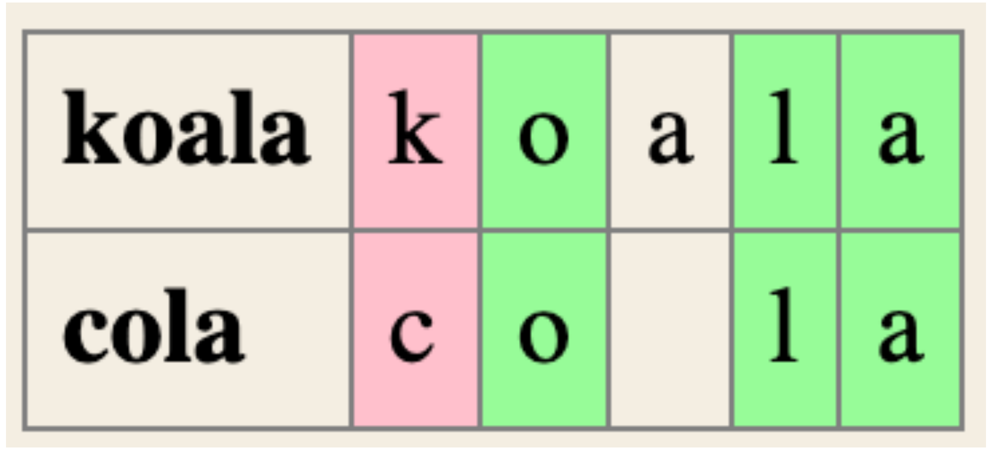
- The task
  - About sequence alignment
  - Alignment and scoring
- The method
  - Dynamic programming
  - The Needleman Wunsch algorithm
- Dynamic programming and XSLT
  - Recursion and iteration
  - Processing the anti-diagonal
  - Save yourself a trip … and some space
  - Performance
- Conclusions

About sequence alignment

- Biomedical (nucleotides), philological (words)
- Global and local
  - *Global* (collation; Gothenburg model pipeline)
  - *Local* (text reuse)
- How many sequences
  - *Pairwise*
  - *Multiple-witness* alignment
  - Progressive, iterative: order effects

Alignment and scoring

- Match
- Mismatch
- Gap (indel)



koala	k	o	a	l	a
cola	c	o		l	a

Dynamic programming: history

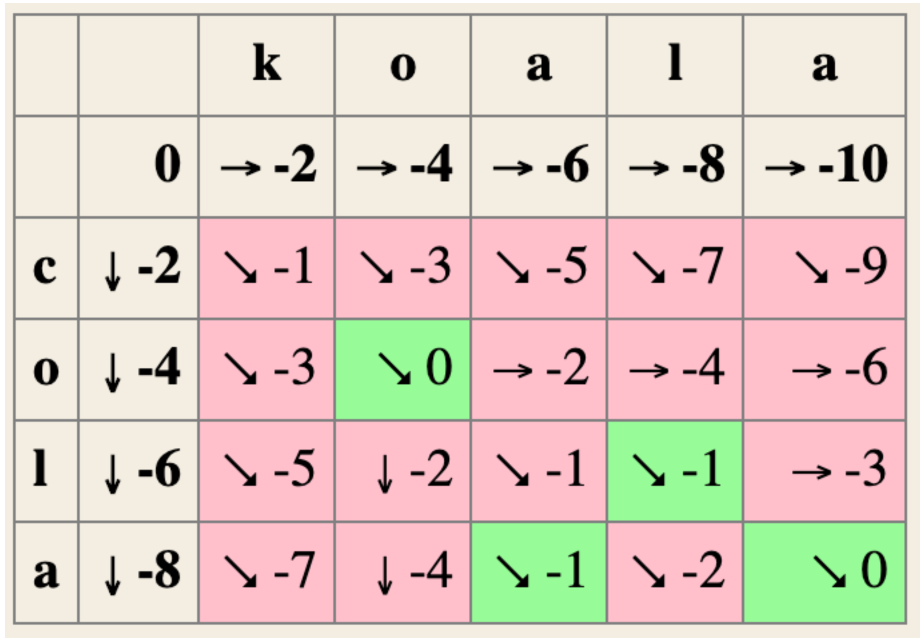
- Richard Bellman (Rand, 1950s)
- Express complex computational tasks as a combination of smaller, more tractable, overlapping ones
- Requirements for dynamic programming
    - *Optimal substructure:* optimal solution to a problem can be reached by determining optimal solutions to its subproblems
    - *Overlapping subproblems:* same subproblems recur repeatedly

Dynamic programming: example

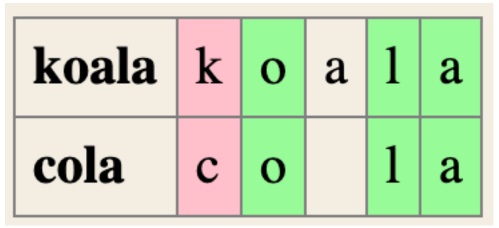
- Fibonacci series ([0 1] 1 2 3 5 8 13 21 34 …)
- Processing order
  - Tabulation: bottom up
  - Memoization: top down

The Needleman Wunsch algorithm

- Grid, scores, sequences, initialize
- Traverse LR, TB
  - Record
    - Best score of three neighbors
    - Source(s) of best score
- Similar to Levenshtein distance
- Backward traversal(s)
  - Only source matters
  - Align from end of strings



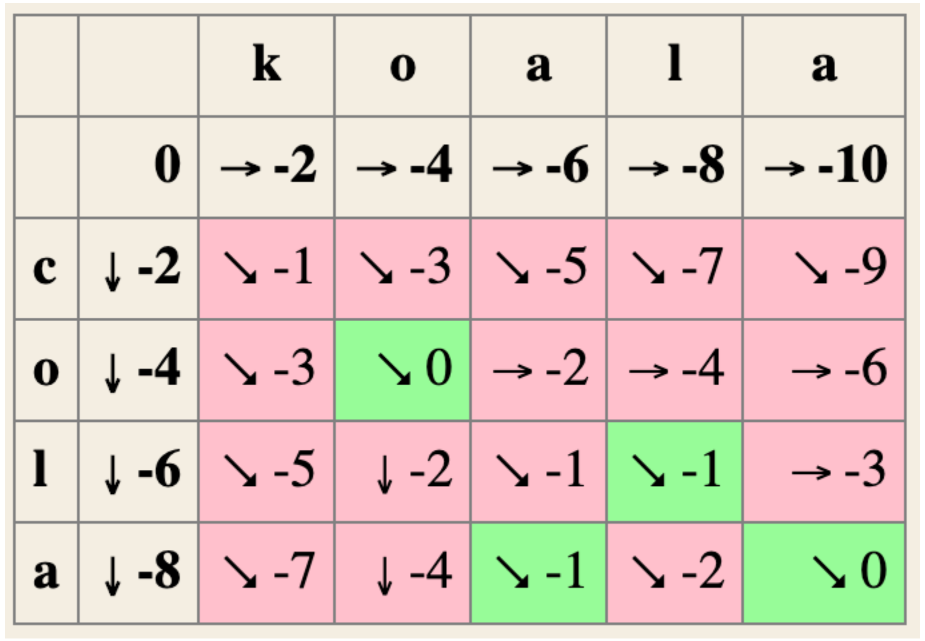
		k	**o**	**a**	**l**	**a**
	0	**→ -2**	**→ -4**	**→ -6**	**→ -8**	**→ -10**
c	**↓ -2**	↘ -1	↘ -3	↘ -5	↘ -7	↘ -9
o	**↓ -4**	↘ -3	↘ 0	→ -2	→ -4	→ -6
l	**↓ -6**	↘ -5	↓ -2	↘ -1	↘ -1	→ -3
a	**↓ -8**	↘ -7	↓ -4	↘ -1	↘ -2	↘ 0



koala	k	o	a	l	a
cola	c	o		l	a

Dynamic programming and XSLT

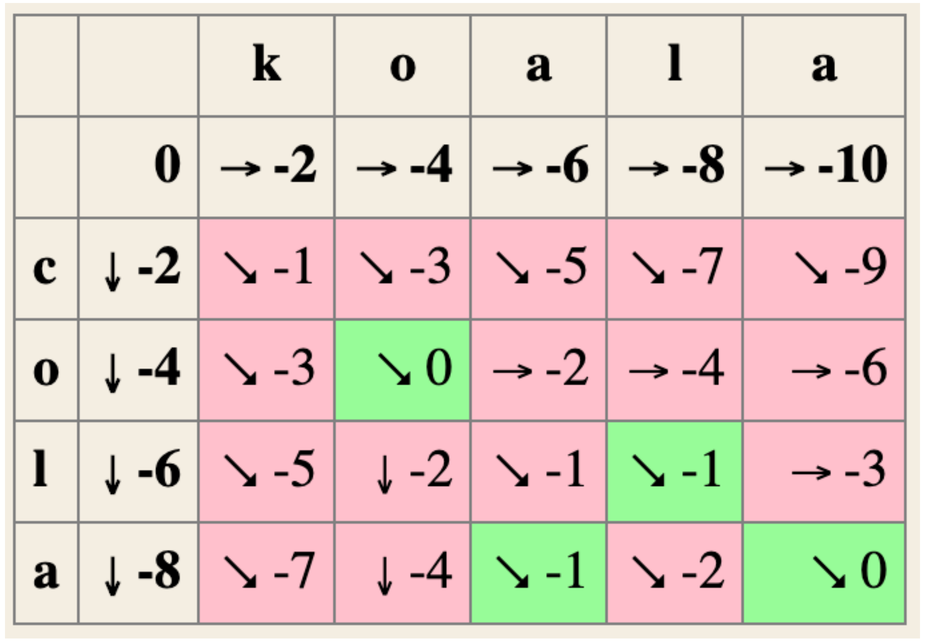
- **<xsl:for-each>** is functional, not iterative
  - Order of output, not of execution
- Cannot update cells inside
  - All cells have initial (null) value



		k	o	a	l	a
	0	→ -2	→ -4	→ -6	→ -8	→ -10
c	↓ -2	↘ -1	↘ -3	↘ -5	↘ -7	↘ -9
o	↓ -4	↘ -3	↘ 0	→ -2	→ -4	→ -6
l	↓ -6	↘ -5	↓ -2	↘ -1	↘ -1	→ -3
a	↓ -8	↘ -7	↓ -4	↘ -1	↘ -2	↘ 0

Recursion and iteration

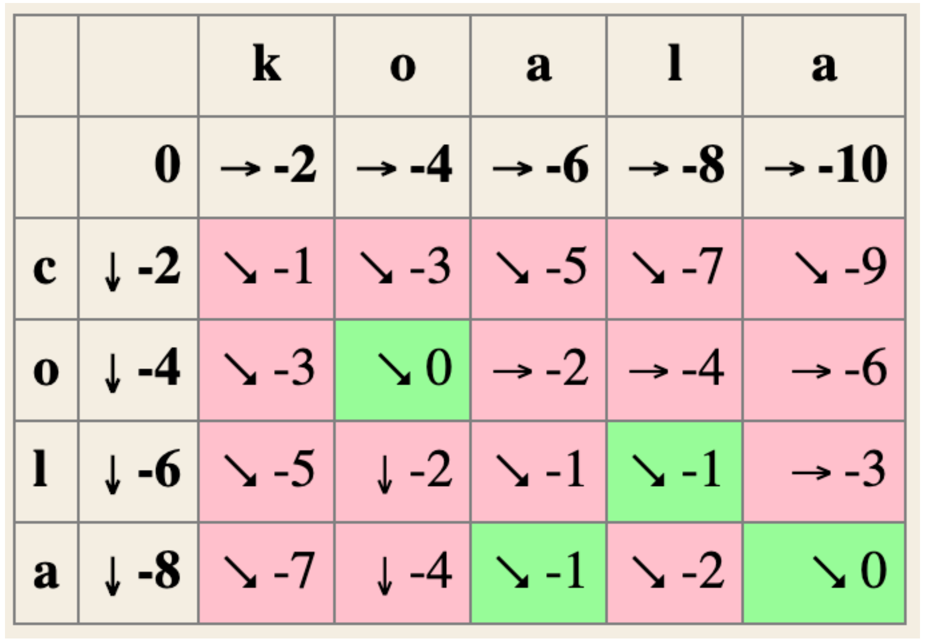
- XSLT 2.0: mimic loop with recursion
  - Quadratic
    - Eek! Stack overflow!
  - Mitigate with *tail call optimization*
    - Brittle
- XSLT 3.0: **<xsl:iterate>**



		k	**o**	**a**	**l**	**a**
	0	**→ -2**	**→ -4**	**→ -6**	**→ -8**	**→ -10**
c	**↓ -2**	↘ -1	↘ -3	↘ -5	↘ -7	↘ -9
o	**↓ -4**	↘ -3	↘ 0	→ -2	→ -4	→ -6
l	**↓ -6**	↘ -5	↓ -2	↘ -1	↘ -1	→ -3
a	**↓ -8**	↘ -7	↓ -4	↘ -1	↘ -2	↘ 0

Processing the anti-diagonal

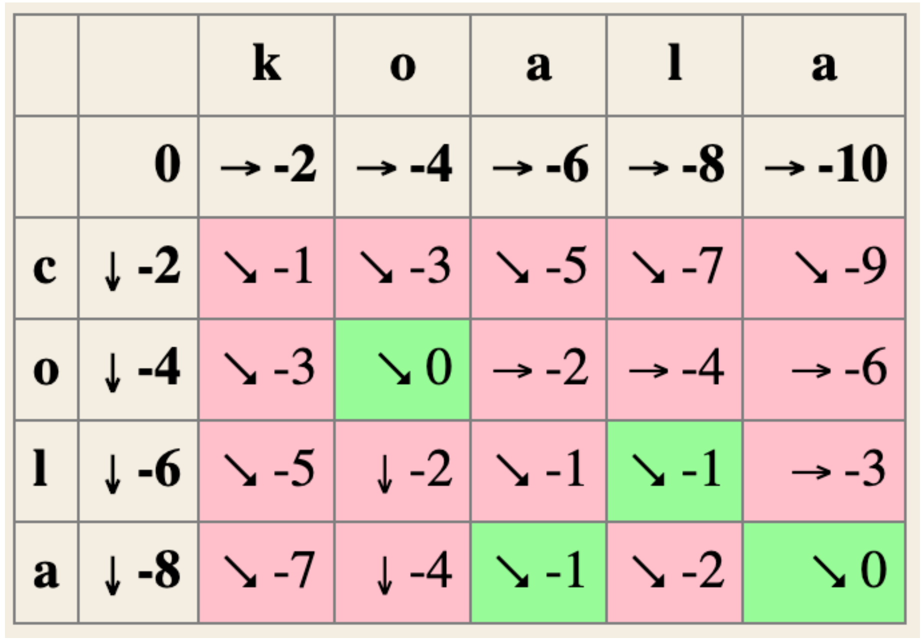
- Muraoka 1971 (*wave front*)
- Tennison 2007 (Levenshtein)
- **<xsl:for-each>** within anti-diagonal
  - No internal dependencies
- Recur only on new anti-diagonal
  - Linear



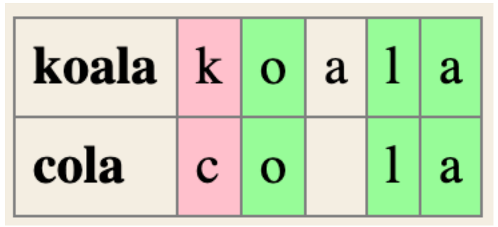
		k	o	a	l	a
	0	**→ -2**	**→ -4**	**→ -6**	**→ -8**	**→ -10**
c	**↓ -2**	↘ -1	↘ -3	↘ -5	↘ -7	↘ -9
o	**↓ -4**	↘ -3	↘ 0	→ -2	→ -4	→ -6
l	**↓ -6**	↘ -5	↓ -2	↘ -1	↘ -1	→ -3
a	**↓ -8**	↘ -7	↓ -4	↘ -1	↘ -2	↘ 0

Saving time and space

- Grid
  - Requires quadratic storage
  - Needed for backward traversal
    - (Unlike Levenshtein)
- Store full path in the cell
  - **<cell row="4" col="5" score="0" path="ddldd"/>**
  - Depends only on two preceding anti-diagonals
  - No need to pass entire grid
  - Three-anti-diagonal lifecycle
  - Last cell holds all alignment information



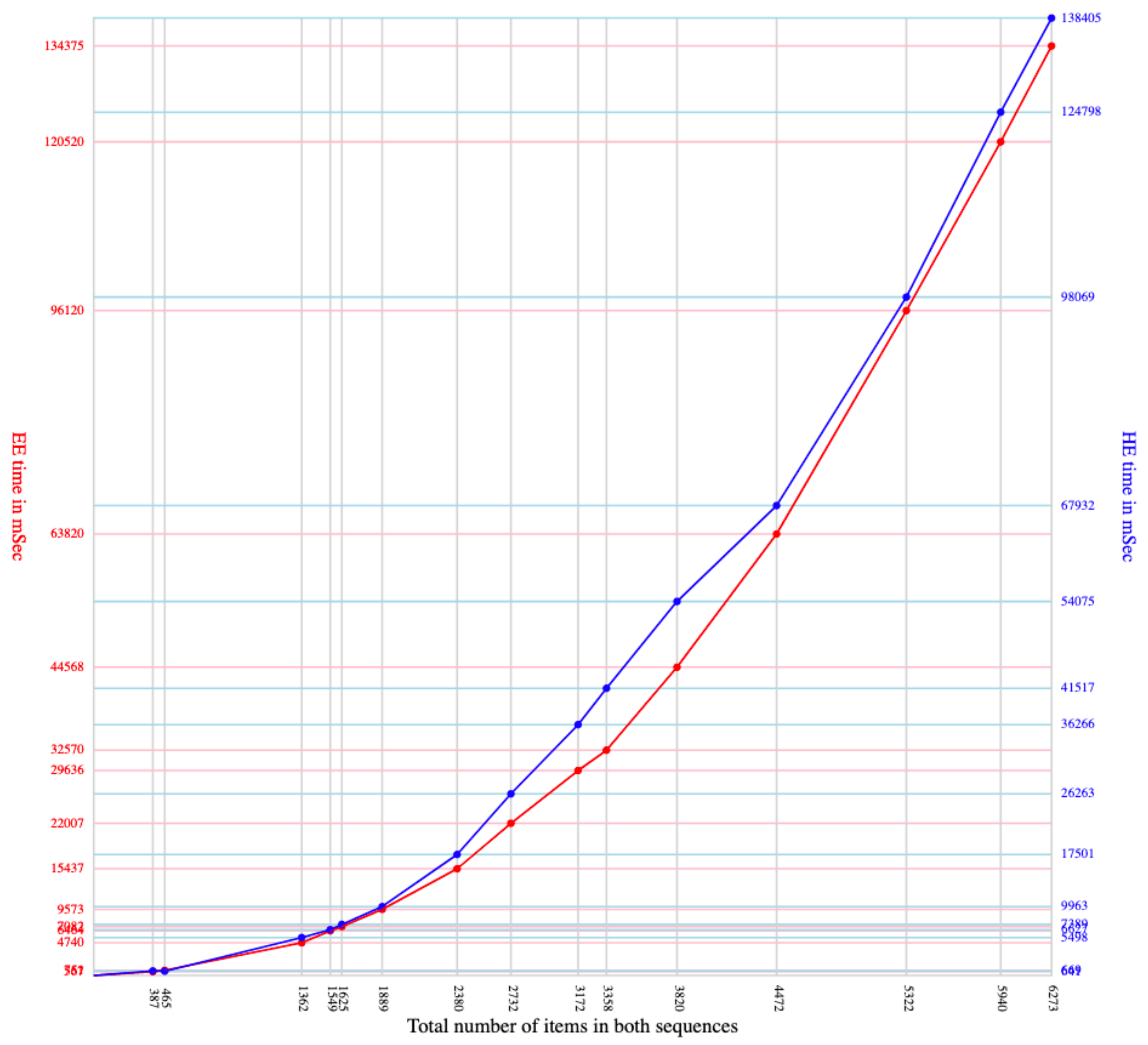
		k	o	a	l	a
	0	→ -2	→ -4	→ -6	→ -8	→ -10
c	↓ -2	↘ -1	↘ -3	↘ -5	↘ -7	↘ -9
o	↓ -4	↘ -3	↘ 0	→ -2	→ -4	→ -6
l	↓ -6	↘ -5	↓ -2	↘ -1	↘ -1	→ -3
a	↓ -8	↘ -7	↓ -4	↘ -1	↘ -2	↘ 0



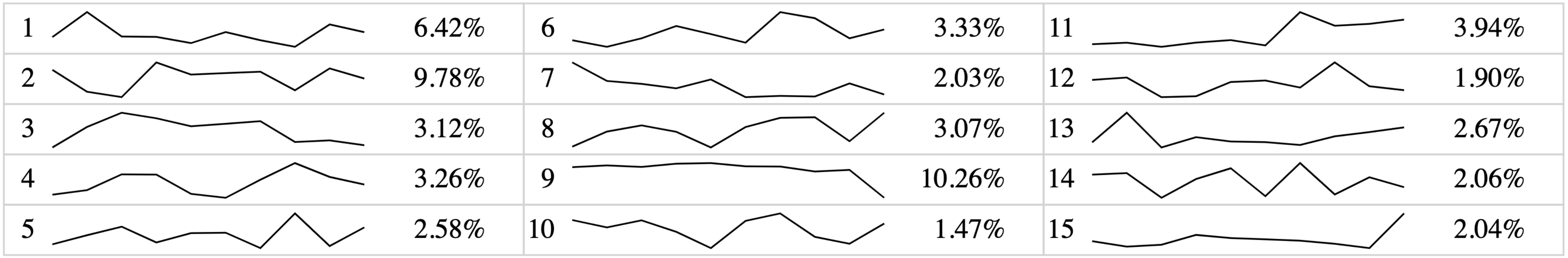
koala	k	o	a	l	a
cola	c	o		l	a

Performance

- *On the origin of species* 1859, 1860
  - 1 paragraph (193 + 194 = 387)
  - 15 paragraphs (3147 + 3126 = 6273)
  - Paragraphs are natural alignment units
- Cell count is quadratic



Parallelization: **@saxon:threads**



1	6.42%	6	3.33%	11	3.94%
2	9.78%	7	2.03%	12	1.90%
3	3.12%	8	3.07%	13	2.67%
4	3.26%	9	10.26%	14	2.06%
5	2.58%	10	1.47%	15	2.04%

- Multi-threading … works best when the body of the **<xsl:for-each>** instruction performs a *large amount of computation* but produces a *small amount of output* (Saxon documentation)
- Small output
- Small (not large) computation
- Memo function?

Conclusions

- “I guess the take-home messages are: (a) try to iterate rather than recurse whenever you can and (b) don’t blindly adapt algorithms designed for procedural programming languages to XSLT” (Tennison 2007)
- **<xsl:iterate>**
- Anti-diagonal traversal
- Store full path on each cell = reduce storage from quadratic to linear

Thank you!

- David J. Birnbaum
- djbpitt@gmail.com
- http://www.obdurodon.org
- https://github.com/djbpitt/xstuff/tree/master/nw