Semi-local string comparison

Alexander Tiskin
http://www.dcs.warwick.ac.uk/~tiskin

Department of Computer Science
University of Warwick

1. The problem
2. Efficient output representation
3. The algorithms
4. Conclusions and future work

The problem

String matching: find an *exact* pattern in a string

String comparison: find *similar* patterns in two strings

- *global:* compare whole string against whole string
- *local:* compare substrings against substrings
- *semi-local:* compare whole string against substrings (will extend this definition later)

Often called "approximate string matching" (no relation to approximation algorithms!)

Applications: computational biology, image recognition, ...

The problem

Consider *strings* (= *sequences*) over finite or infinite alphabet

Distinguish contiguous *substrings* and not necessarily contiguous *subsequences*

Special cases of substring: *prefix*, *suffix*

Usual notation: strings a, b of length m, n respectively

The problem

Longest common subsequence (LCS) problem: determine the LCS length for string a against string b

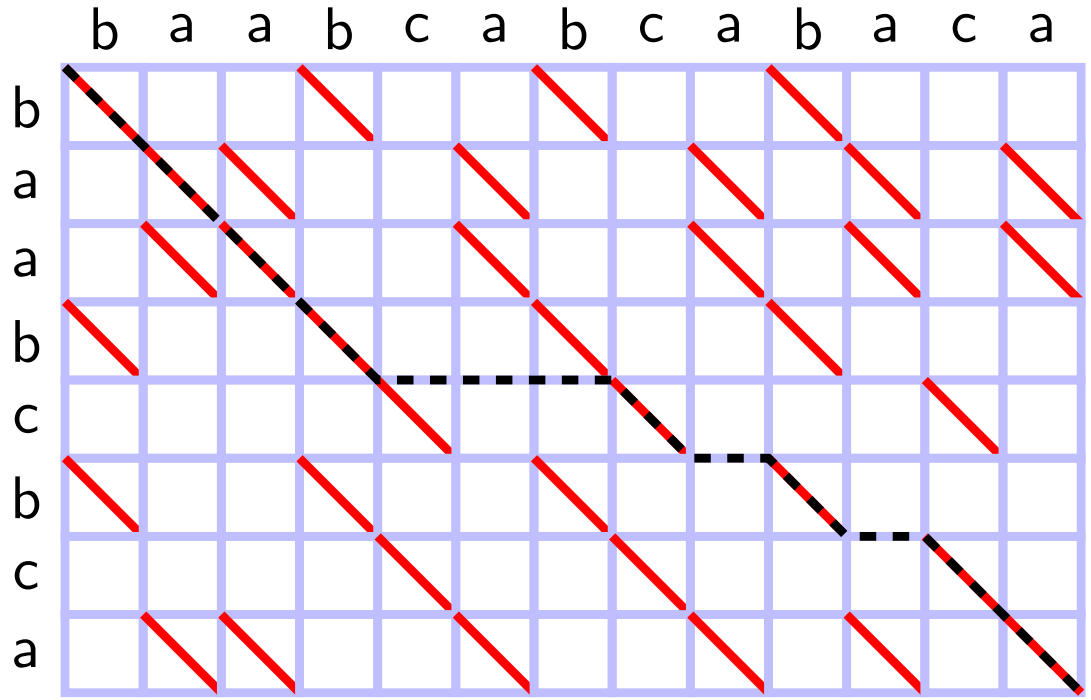
A variant of the *edit distance problem*

$O(mn)$ by dynamic programming [Needleman, Wunsch, 1970]

$O\left(\frac{mn}{\log(m+n)}\right)$ [Masek, Paterson, 1980]
extended by [Crochemore+, 2003]

The problem

$LCS(\texttt{"baabcbca"}, \texttt{"baabcabcabaca"}) = \texttt{"baabcbca"}$ $\quad m \leq n$



$blue = 0$

$red = 1$

Alignment graph

Longest common subsequence $\sim$ longest source-to-sink path

The problem

All string-substring LCS problem: determine the maximum LCS length for string a against all substrings of b

Naive approach: $O\left(\frac{mn^3}{\log(m+n)}\right)$ to compute $\Theta(n^2)$ outputs

$O(mn)$ [Schmidt, 1998]
extended by [Alves+, 2005]

New result: $O\left(\frac{mn}{\log^{0.5}(m+n)}\right)$, with outputs represented implicitly

The problem

LCS problem on cyclic strings: determine the maximum LCS length for string a against all cyclic rotations of b

Naive approach: $O\left(\frac{mn^2}{\log(m+n)}\right)$ to compute $\Theta(n)$ partial results

$O(mn \log m)$ [Maes, 1990]

$O(mn)$ [Bunke, Bühler, 1993]
extended by [Landau+, 1998], [Schmidt, 1998]

New result: $O\left(\frac{mn}{\log^{0.5}(m+n)}\right)$

The problem

Semi-local subsequence recognition problem on compressed text (against plain pattern): given a straight-line program of size M generating a, determine substrings of a containing b as a subsequence

Covers various compression paradigms, e.g. Lempel-Ziv

Note m can be $O(c^M)$. Assume address arithmetic on a still $O(1)$.

Naive approach: uncompress a. Can be $O(c^M)$.

$O(Mn^2 \log n)$ [Cegielski+, 2006]

New result: $O(Mn^{1.5})$

The problem

Longest increasing subsequence (LIS) problem: determine the LCS length for permutation a of length n against $id = (1, 2, \ldots, n)$

Naive approach: $O\left(\frac{n^2}{\log n}\right)$ by generic LCS

$O(n \log n)$ implicit in [Erdös, Szekeres, 1935]
also [Robinson, 1938]
extended by [Knuth, 1970]
also [Dijkstra, 1980]

In a stronger model: $O(n \log \log n)$ [Hunt, Szymanski, 1977]
also [Chang, Wang, 1992]
extended by [Bespamyatnikh, Segal, 2000]

The problem

Window LIS problem: given a permutation a, determine the LIS length for every substring of a of fixed length r

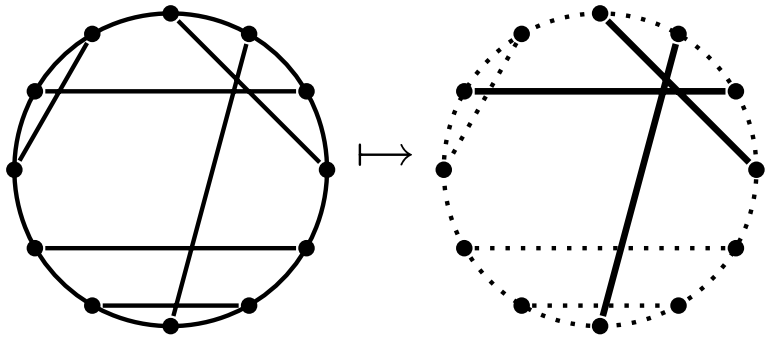
Naive approach: $O(n^2 \log n)$ to compute $\Theta(n)$ outputs

$O(n^2)$, computes every LIS completely [Albert+, 2004]
extended by [Chen+, 2005]

New result: $O(n^{1.5})$, even for variable r

The problem

Max-clique problem in a circle graph: given a circle and n chords, determine the maximum-size subset of pairwise intersecting chords



$O(n^3)$ [Gavril, 1973]

$O(n^2)$ [Rotem, Urrutia, 1981]
extended by [Hsu, 1985]
also [Masuda+, 1990], [Apostolico+, 1992]

New result: $O(n^{1.5})$, even to find (implicitly) all maximal subsets

The problem

All of the above can be captured by a single problem statement!

The problem

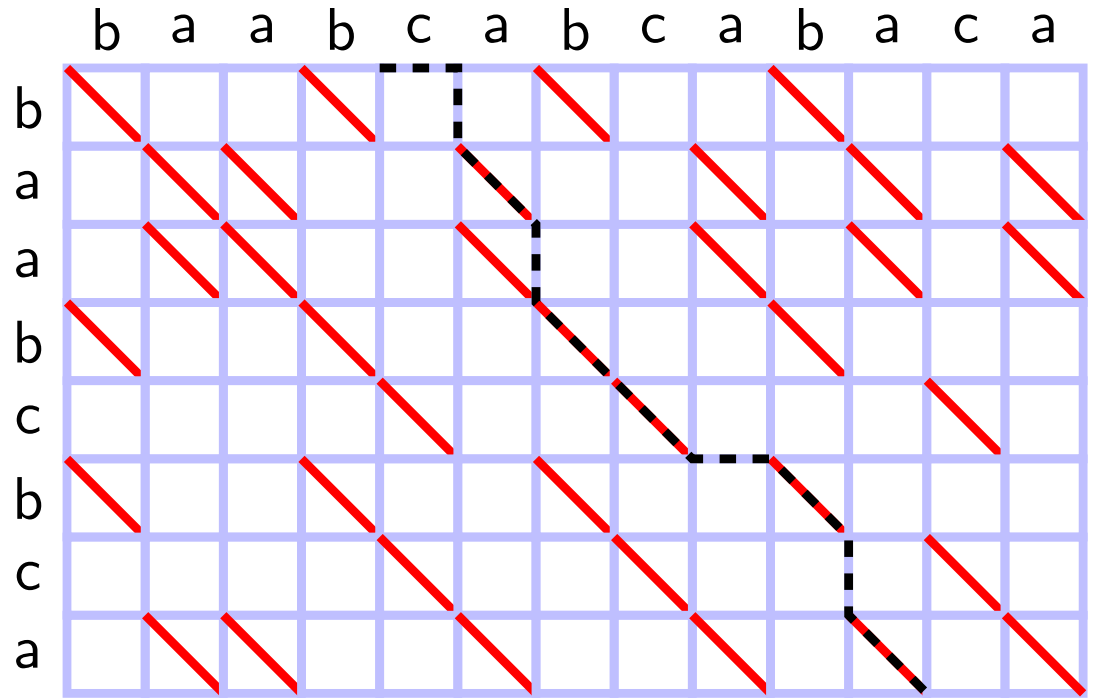
All semi-local longest common subsequences (LCS) problem:
determine the LCS length for

- string a against every substring of b
- every prefix of a against every suffix of b
- as above but with a and b swapped

Total $\Theta(n^2)$ outputs, allowed to be represented implicitly

The problem

$LCS(\texttt{"baabcbca"}, \texttt{"...cabcaba..."}) = \texttt{"abcba"}$ $m \leq n$



$blue = 0$

$red = 1$

All semi-local LCS $\sim$ all longest border-to-border paths

(string-substring $\sim$ top-to-bottom, etc.)

The problem

All semi-local LCS problem: implicit output representation

Trivial: size $O(n^2)$ query time $O(1)$

size $O(m^{1/2}n)$ query time $O(\log n)$ [Alves+, 2003]

size $O(n)$ query time $O(n)$ [Alves+, 2005]

New result: size $O(n \log n)$ query time $O(\log^2 n)$

In a stronger model: size $O(n)$ query time $O(\frac{\log n}{\log \log n})$

The problem

All semi-local LCS problem: computation time

$O(mn)$ for all string-substring LCS [Alves+, 2005]

New result: $O\left(\frac{mn}{\log^{0.5}(m+n)}\right)$ on strings over constant-size alphabet

Implies LCS on cyclic strings, all string-substring LCS

New result: $O(n^{1.5})$ on permutations

Implies Window LIS, Max-clique in circle graph

Efficient output representation

Notation

Integers $0, 1, 2, \ldots$ Odd half-integers $\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots$

$x \lhd y \Leftrightarrow y - x = 1 \quad x \lhd\!\!\lhd y \Leftrightarrow y - x = \frac{1}{2}$

Definition

Point (i_0, j_0) *dominates* point (i, j), if $i_0 < i$ and $j < j_0$

Efficient output representation

Definition

Point (i, j) is *A-critical*, if

$$A(i^-, j^-) \lhd A(i^-, j^+) = A(i^+, j^-) = A(i^+, j^+)$$

where $i^- \trianglelefteq i \trianglelefteq i^+ \quad j^- \trianglelefteq j \trianglelefteq j^+$

Notation

$d_A(i_0, j_0)$ is the number of A-critical points dominated by (i_0, j_0)

Efficient output representation

Lemma

$$A(i_0, j_0) = j_0 - i_0 - d_A(i_0, j_0)$$

$j_0 - i_0$: input substring length $\quad d_A(i_0, j_0)$: unmatched characters

Proof: simple induction

More generally:

- A is a *Monge matrix*
- its *density matrix* happens to be the permutation matrix of critical points

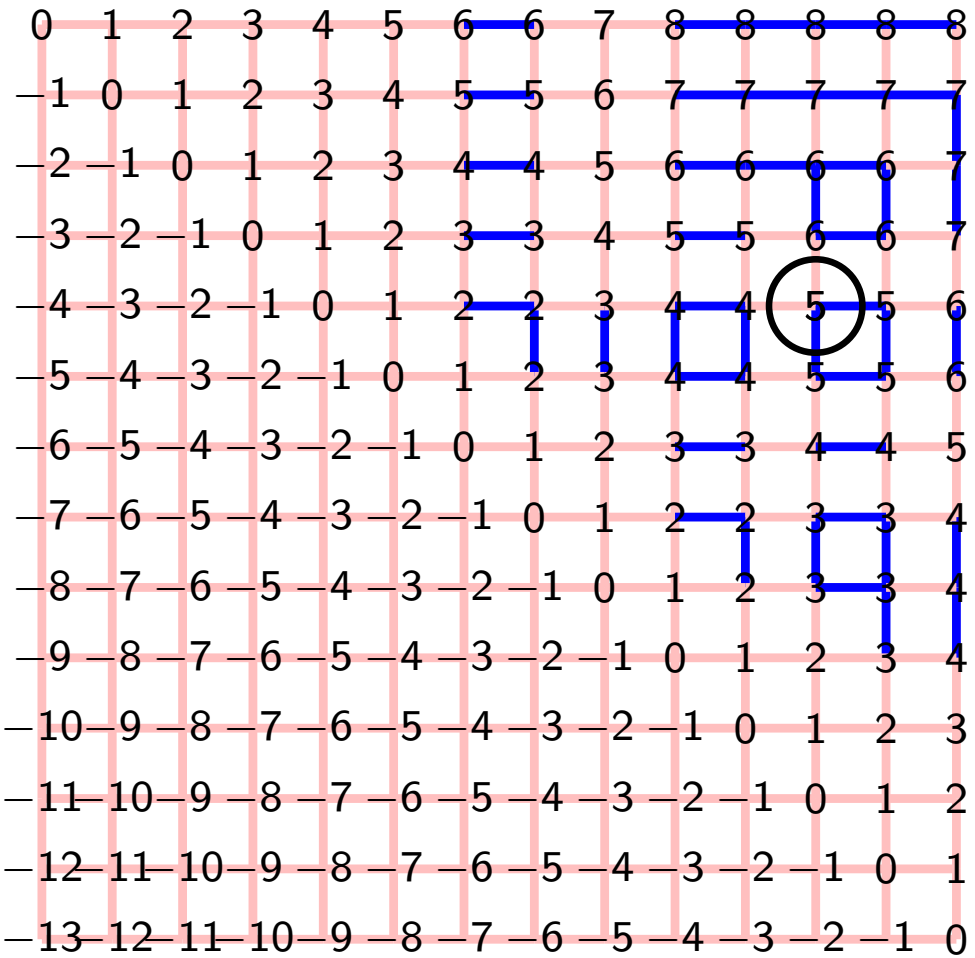
Efficient output representation

Full top-to-bottom score matrix: $A(i,j)$ $\quad 0 \leq i,j \leq n$

0	1	2	3	4	5	6	6	7	8	8	8	8	8
−1	0	1	2	3	4	5	5	6	7	7	7	7	7
−2	−1	0	1	2	3	4	4	5	6	6	6	6	7
−3	−2	−1	0	1	2	3	3	4	5	5	6	6	7
−4	−3	−2	−1	0	1	2	2	3	4	4	5	5	6
−5	−4	−3	−2	−1	0	1	2	3	4	4	5	5	6
−6	−5	−4	−3	−2	−1	0	1	2	3	3	4	4	5
−7	−6	−5	−4	−3	−2	−1	0	1	2	2	3	3	4
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	3	4
−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4
−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0

Efficient output representation

Full top-to-bottom score matrix: $A(i,j)$ $\quad 0 \le i,j \le n$



$A(i^+,j) \trianglelefteq A(i^-,j)$

$A(i,j^-) \trianglelefteq A(i,j^+)$

A totally monotone:

$A(i^+,j^+) \triangleleft A(i^-,j^+) \Rightarrow$
$A(i^+,j^-) \triangleleft A(i^-,j^-)$

A^T totally monotone:

$A(i^-,j^-) \triangleleft A(i^-,j^+) \Rightarrow$
$A(i^+,j^-) \triangleleft A(i^+,j^+)$

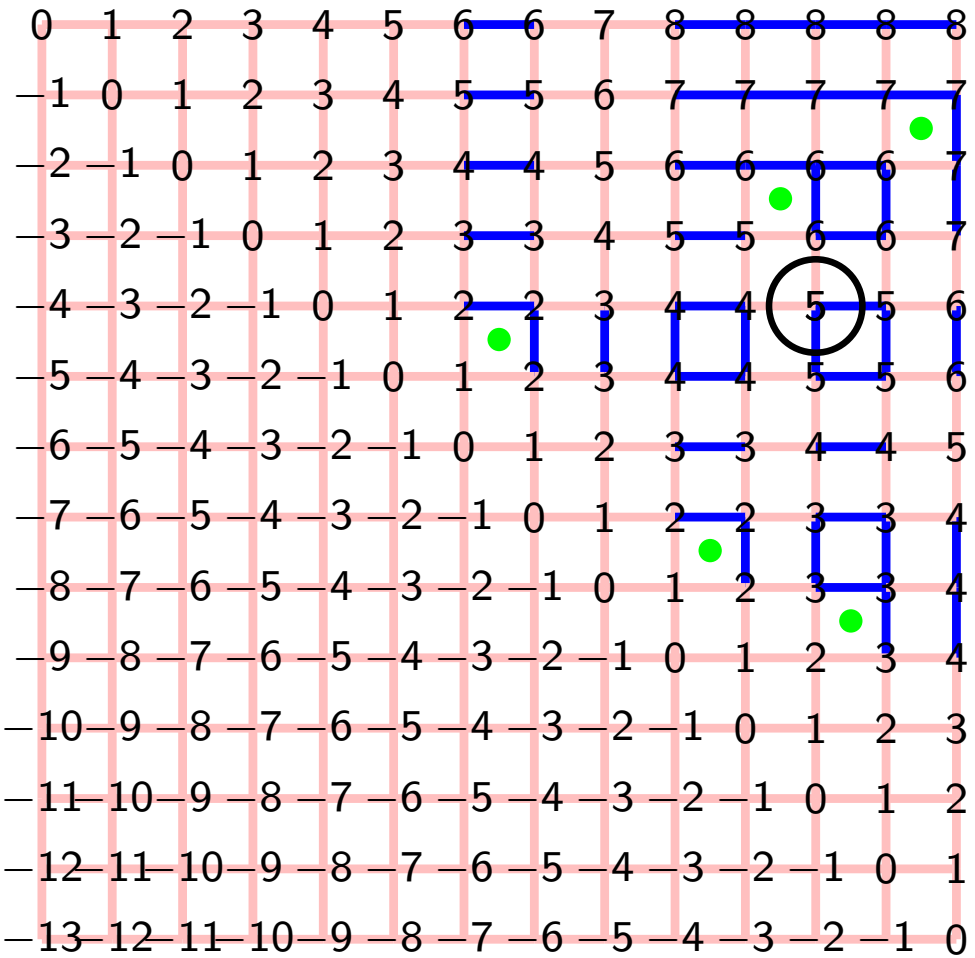
where $i^- \triangleleft i^+$, $j^- \triangleleft j^+$

blue $= 0$ *red* $= 1$

Efficient output representation

Full top-to-bottom score matrix: $A(i,j)$

$0 \le i,j \le n$



$$A(i^+,j) \trianglelefteq A(i^-,j)$$

$$A(i,j^-) \trianglelefteq A(i,j^+)$$

A totally monotone:

$$A(i^+,j^+) \triangleleft A(i^-,j^+) \Rightarrow A(i^+,j^-) \triangleleft A(i^-,j^-)$$

A^T totally monotone:

$$A(i^-,j^-) \triangleleft A(i^-,j^+) \Rightarrow A(i^+,j^-) \triangleleft A(i^+,j^+)$$

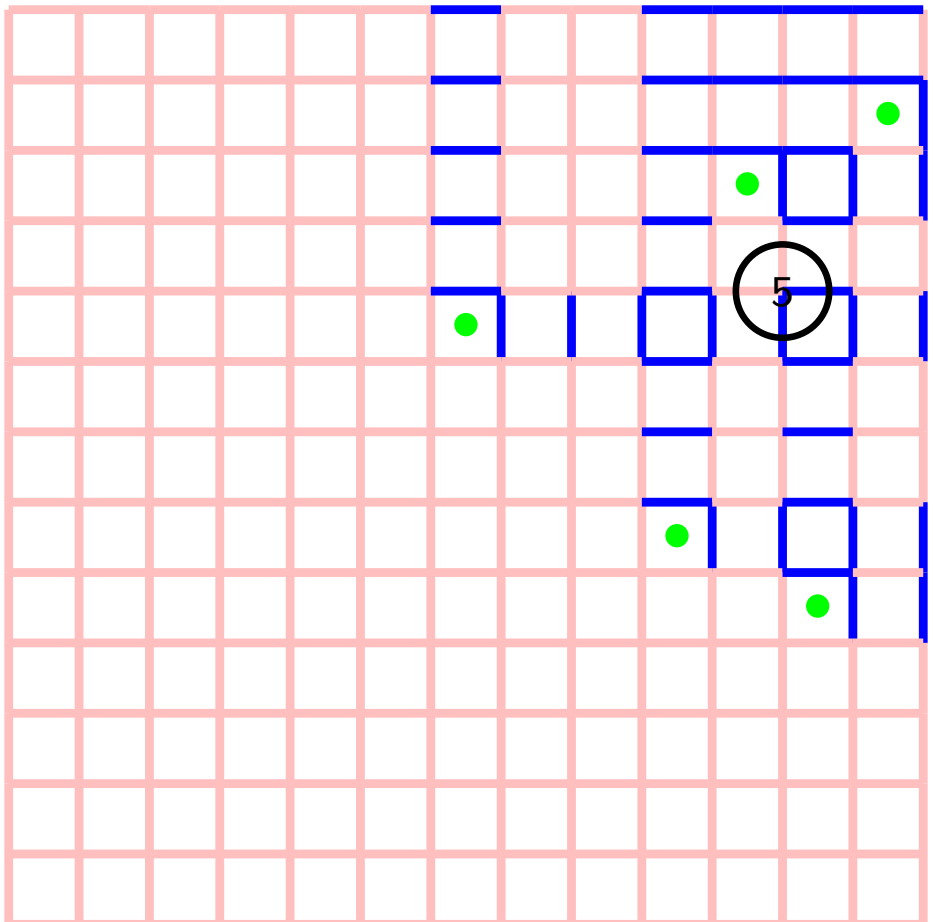
where $i^- \triangleleft i^+$, $j^- \triangleleft j^+$

blue $= 0$ *red* $= 1$ *green* $=$ *critical*

Efficient output representation

Full top-to-bottom score matrix: $A(i,j)$

$0 \leq i,j \leq n$



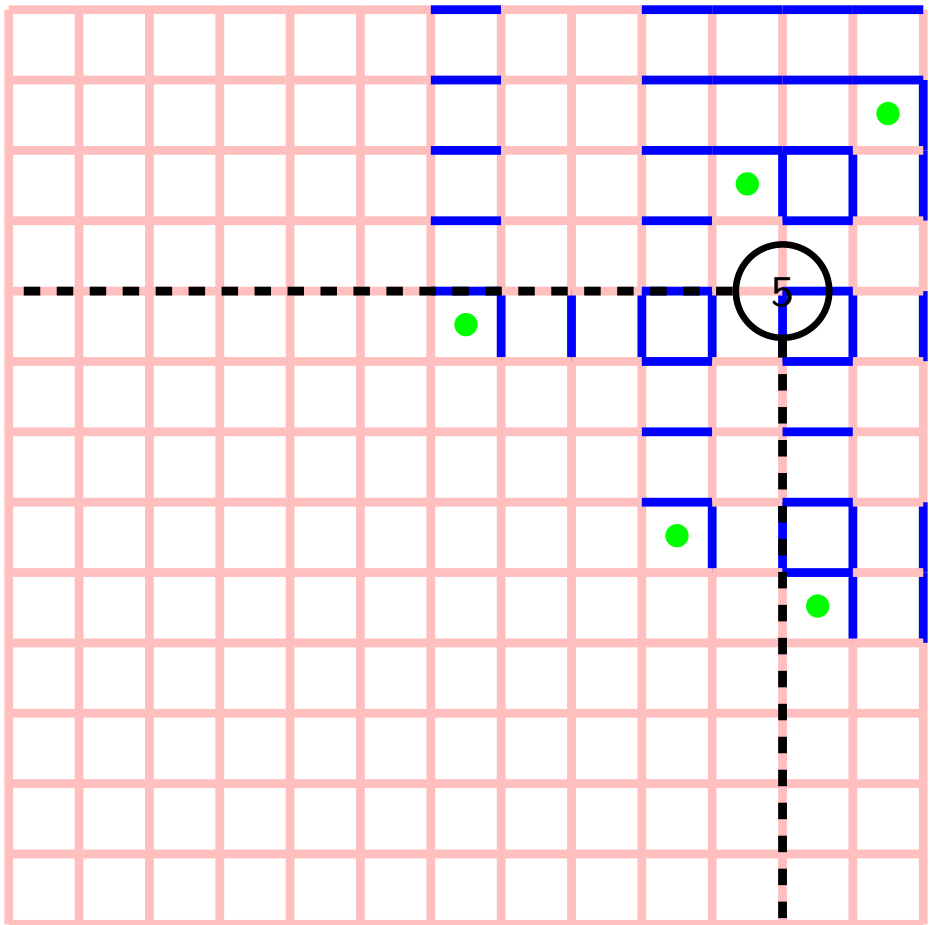
$j_0 - i_0 - d_A(i_0, j_0) =$
$11 - 4 - 2 = 5$

blue $= 0$ *red* $= 1$ *green* $=$ *critical*

Efficient output representation

Full top-to-bottom score matrix: $A(i,j)$

$0 \leq i,j \leq n$

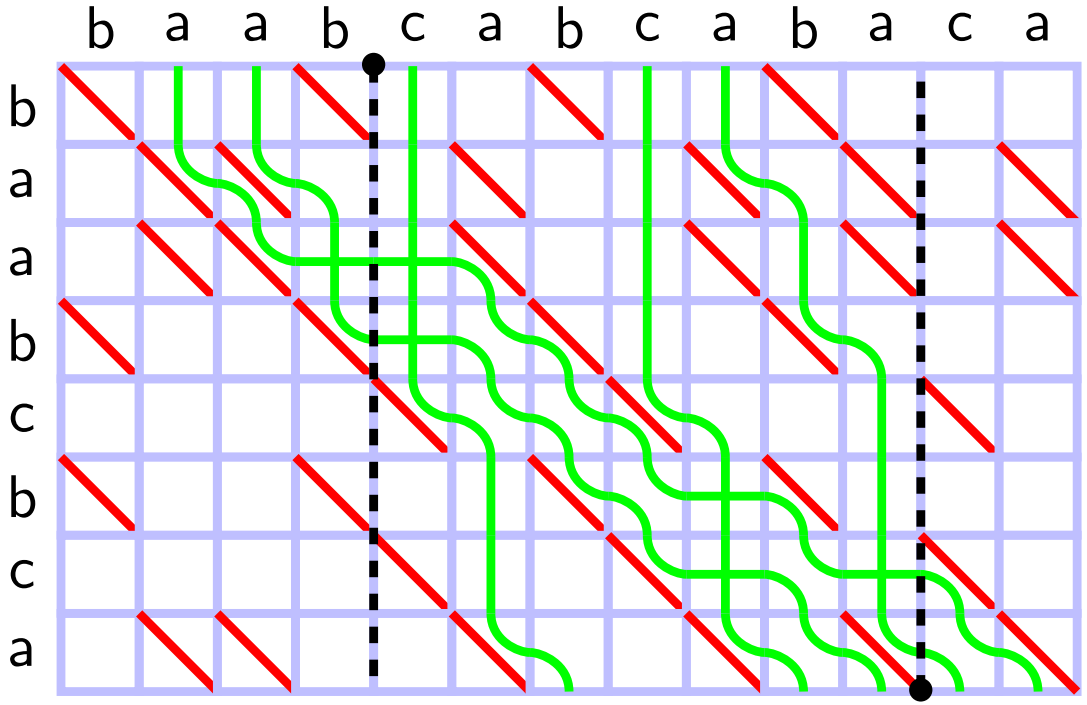


$j_0 - i_0 - d_A(i_0, j_0) =$
$11 - 4 - 2 = 5$

blue $= 0$ *red* $= 1$ *green* $=$ *critical*

Efficient output representation

Critical point (i, j) in score matrix gives *critical pair* $(top, i) \rightsquigarrow (bottom, j)$ in alignment graph



Also define *top* $\rightsquigarrow$ *right*, *left* $\rightsquigarrow$ *right*, *left* $\rightsquigarrow$ *bottom* critical pairs

Gives complete border-to-border graph-theoretic matching

Efficient output representation

Gaudi's seaweeds

Efficient output representation

Establishing $d_A(i_0, j_0)$: *dominance counting*

Range tree: [Bentley, 1980]

- binary search tree by i-coordinate for all nodes
- rooted at its every node, binary search tree by j-coordinate for relevant nodes

Every node represents a *canonical range* (rectangular region), and stores its point count

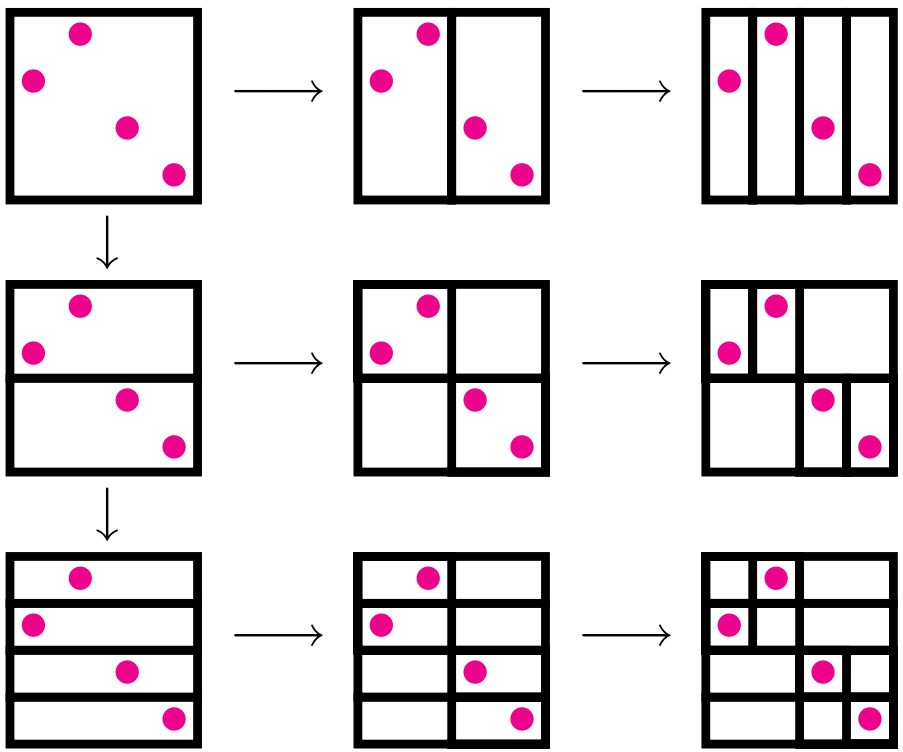
Theorem (Bentley, 1980)

A range tree on n points has

- *size* $O(n \log n)$
- *dominance counting query time* $O(\log^2 n)$

Efficient output representation

Range tree:



Every range can be decomposed into $\leq \log^2 n$ canonical ranges

Overall, $\leq n \log n$ canonical ranges are non-empty

Efficient output representation

Advanced dominance counting [JaJa+, 2004]

Requires RAM model

Theorem (JaJa+,2004)

There exists a data structure on n points with

- *size* $O(n)$
- *dominance counting query time* $O(\frac{\log n}{\log \log n})$

Corollary (new)

All semi-local LCS lengths can be represented in

- *size* $O(n \log n)$ *query time* $O(\log^2 n)$
- *size* $O(n)$ *query time* $O(\frac{\log n}{\log \log n})$

The algorithms

Main subroutine: score matrix multiplication $AB = C$

General matrices: time $O(n^3)$

Monge (= planar distance) matrices: time $O(n^2)$

Score matrices: time $O(n^{1.5})$ — proof later

The algorithms

Partition alignment graph into horizontal strips of width n

Each strip

- is initially of height 1
- represented by $O(n)$ critical points

Main pattern: merge strips $A, B \longrightarrow C$ by score matrix multiplication

A-critical points, B-critical points $\longrightarrow$ C-critical points

Eventually obtain critical points for the whole alignment graph

Post-process to efficient output representation (e.g. range tree)

The algorithms

Lemma (Alves+, 2005, based on Schmidt)

A strip can be merged with a strip of height 1 *in time* $O(n)$.

Proof: preliminary merge all points $(i, j), (j, k) \mapsto (i, k)$

Resulting triples (i, j, k) must not cross twice (cf. longest paths)

Scan the narrower strip left-to-right, removing double crossings □

Theorem (Alves+, based on Schmidt)

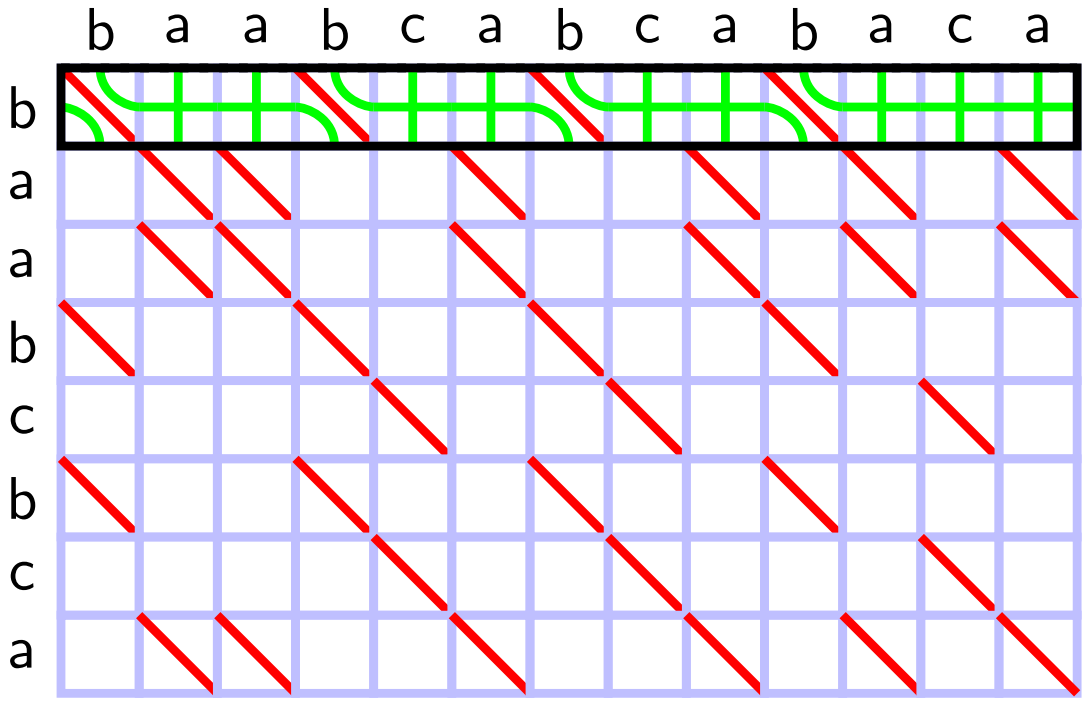
Algorithm for all semi-local LCS:

- *time* $O(mn)$ *memory* $O(n)$
- *outputs* $O(n)$ *critical points*

Proof: successive application of the lemma □

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



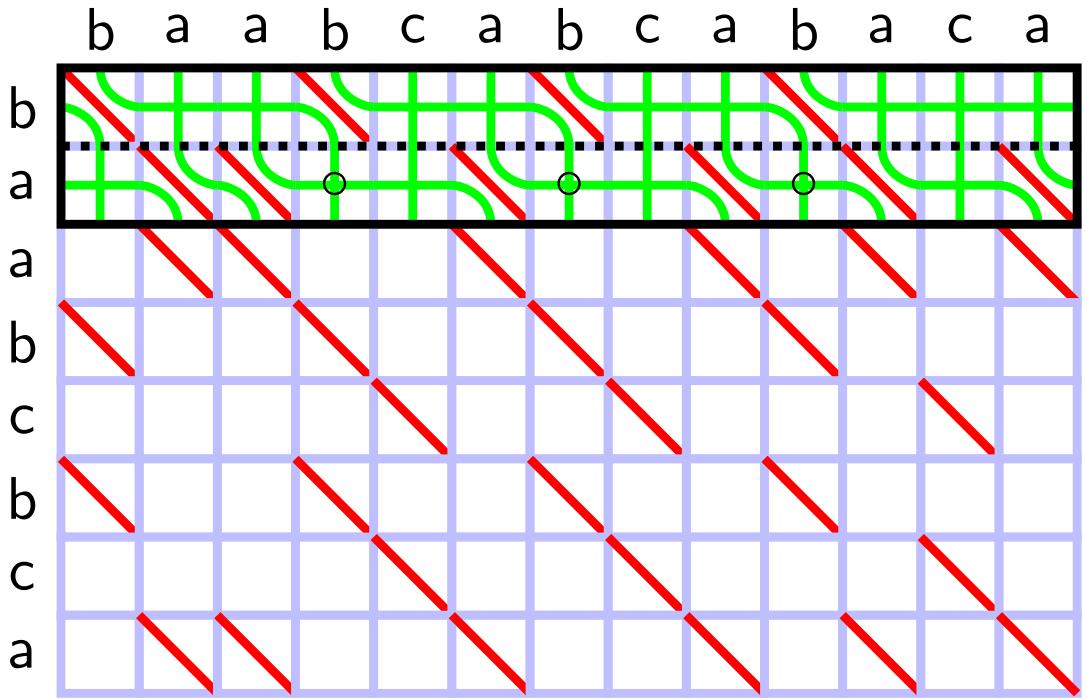
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



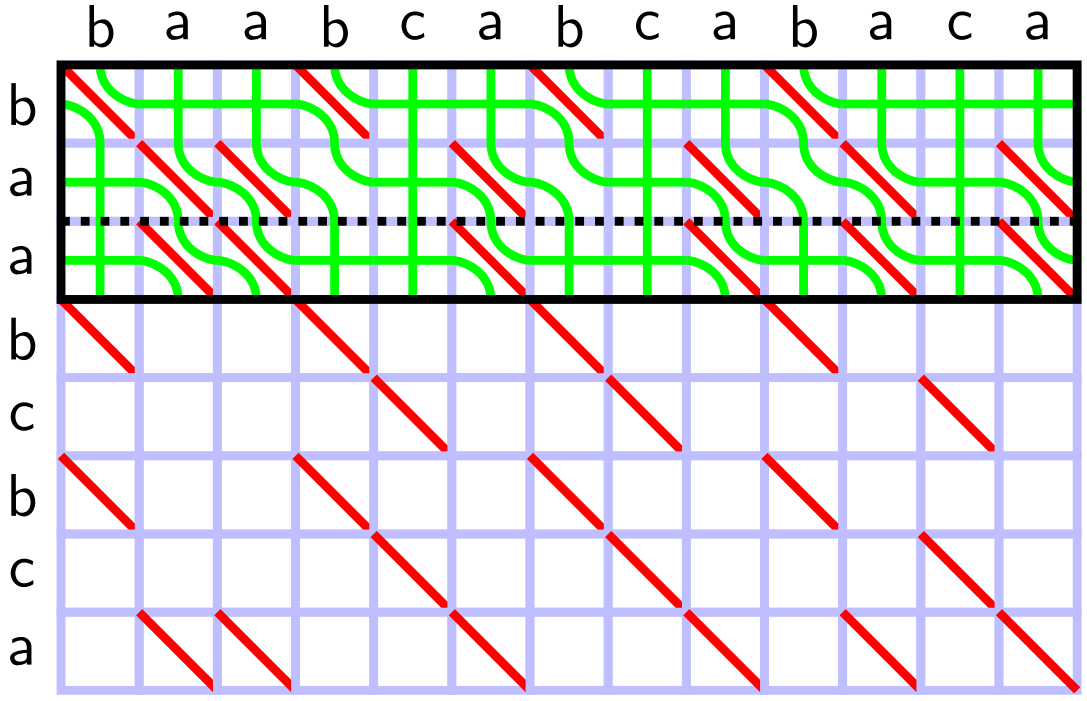
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



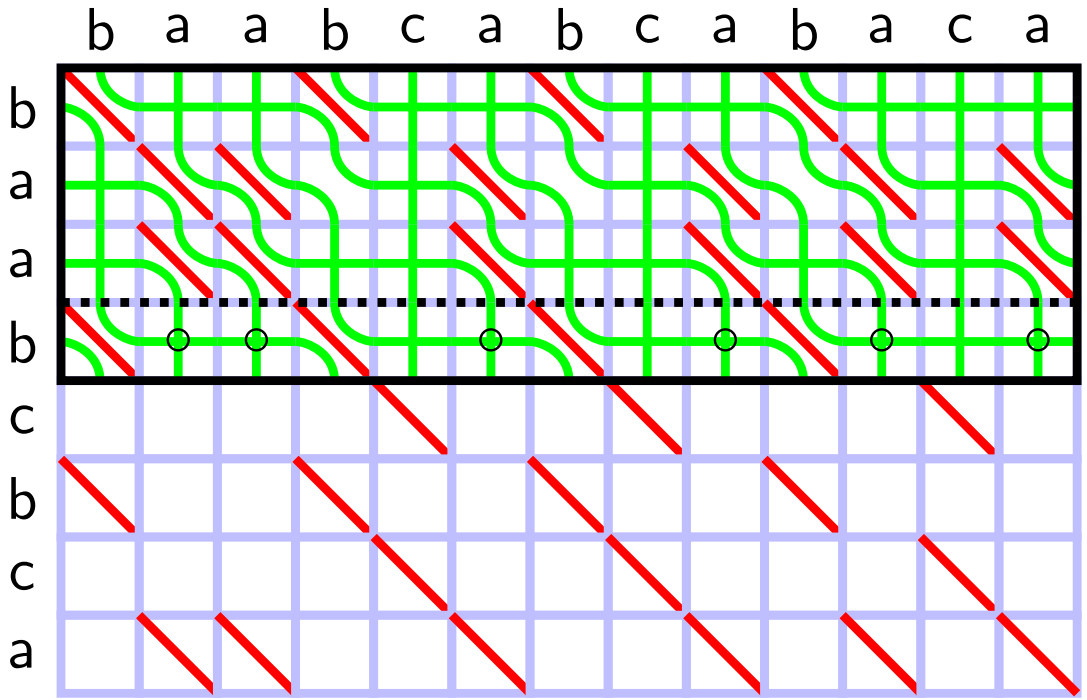
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



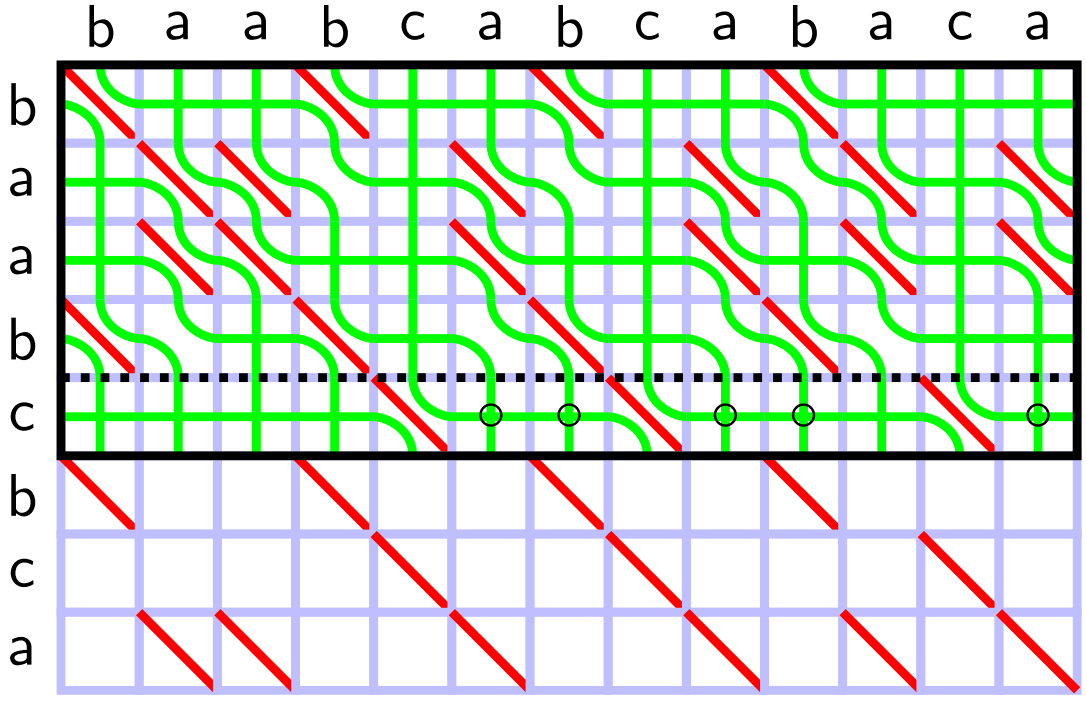
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



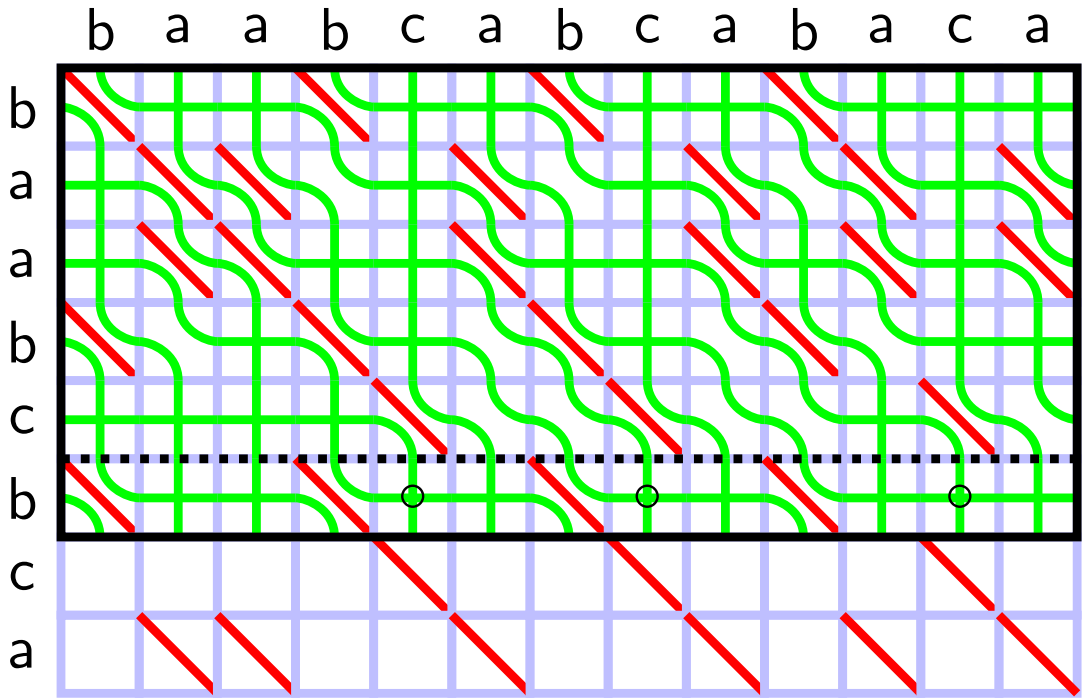
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



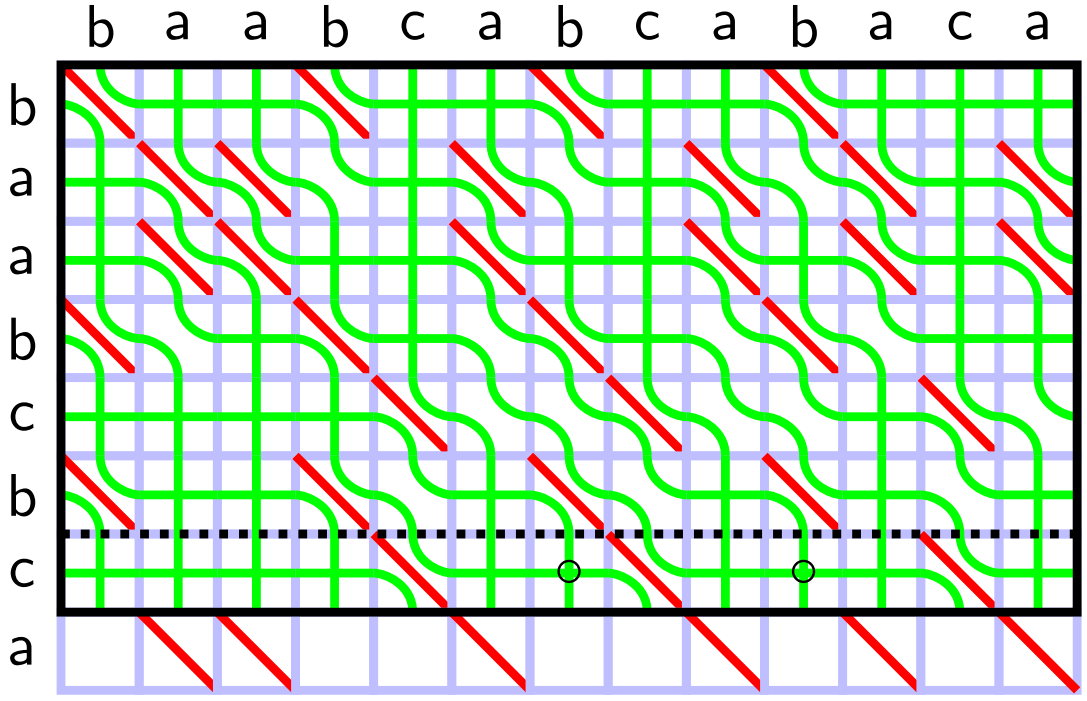
blue = 0

red = 1

green = *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



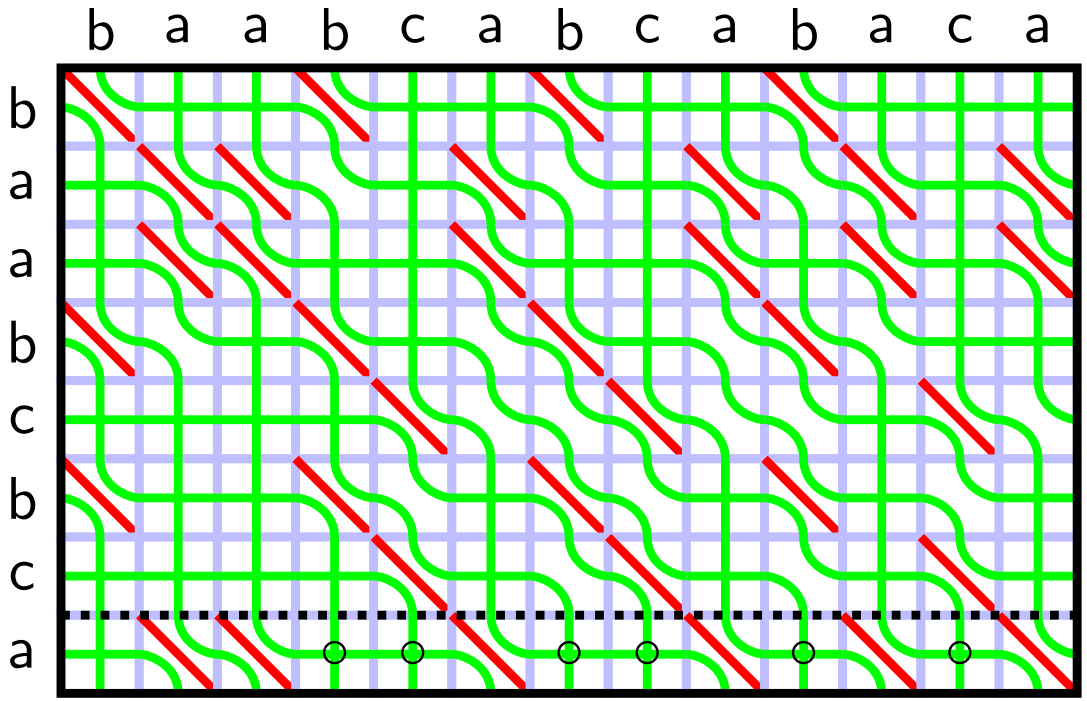
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



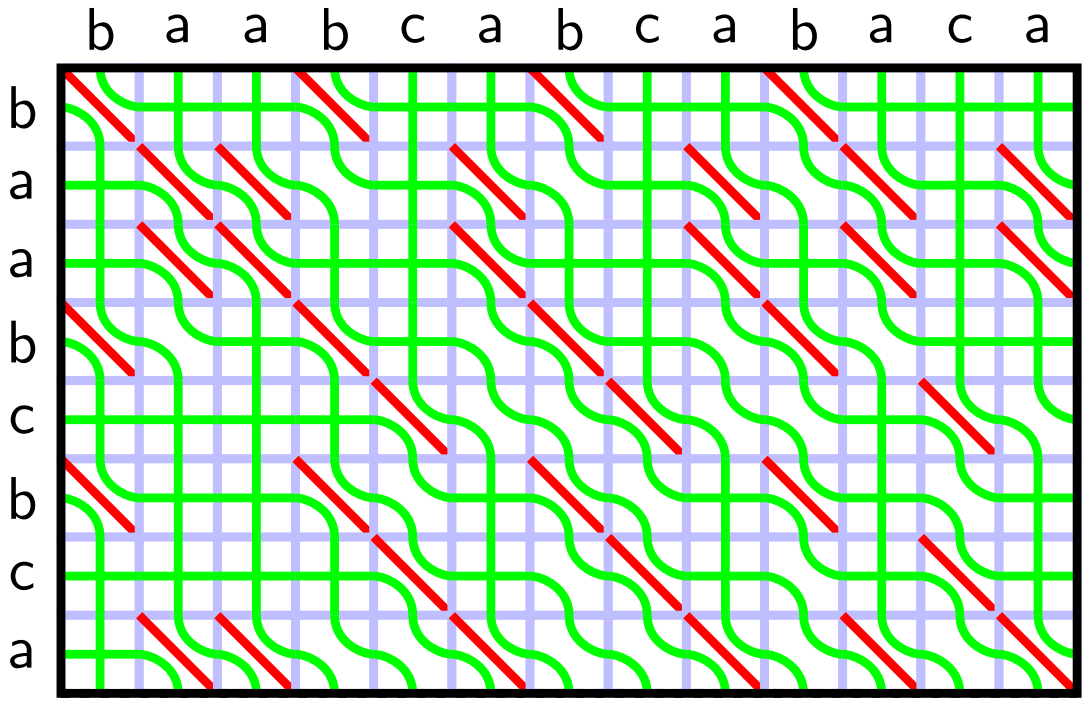
blue = 0

red = 1

green = *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



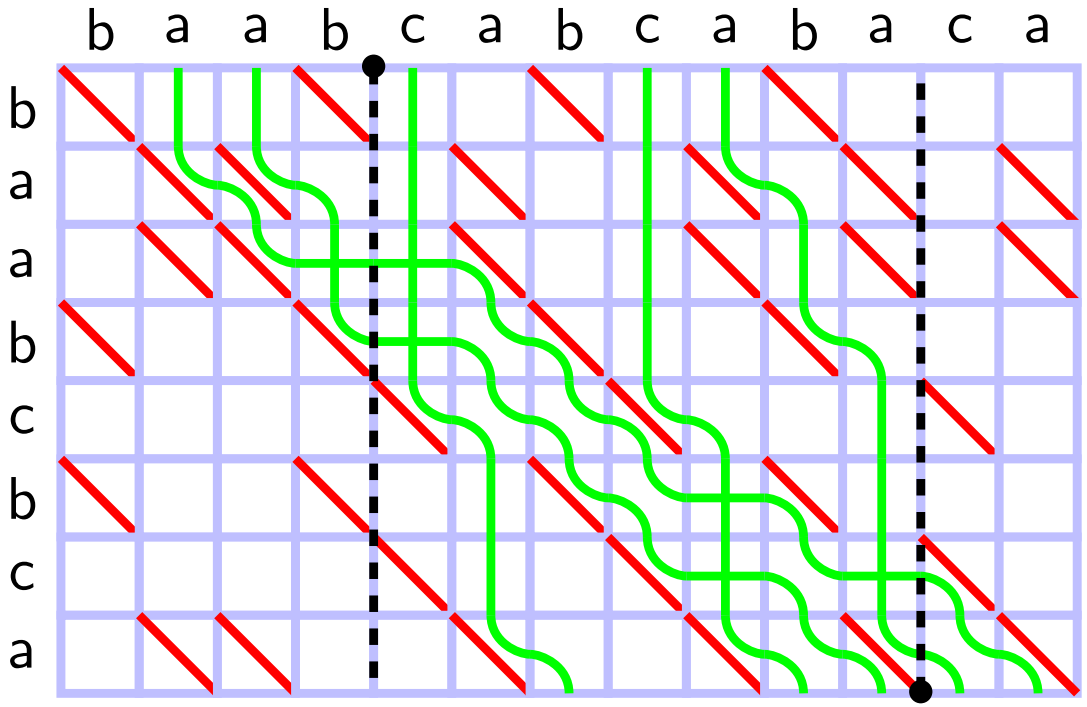
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Algorithm by Schmidt/Alves+: incremental merging of strips



blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Lemma (new)

Two strips of any height can be merged in time $O(n^{1.5})$.

Proof: divide-and-conquer on score matrix C

Count C-critical points in square blocks, beginning with full matrix

For block of size $r \times r$, only r A- and B-critical points are relevant

If the block has > 0 C-critical points, recurse into half-sized blocks

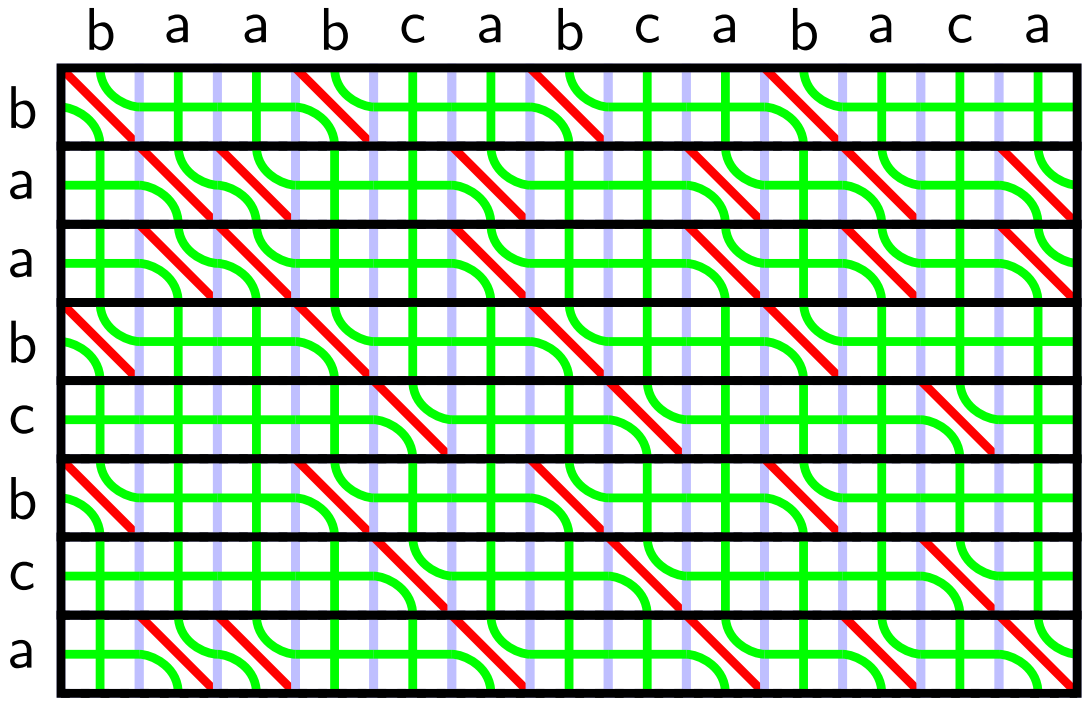
Worst case:

- n blocks of size $n^{0.5} \times n^{0.5}$
- for each need to consider at most $n^{0.5}$ A- and B-critical points

Overall time $n \cdot n^{0.5} = n^{1.5}$ □

The algorithms

New algorithm: divide-and-conquer on strips



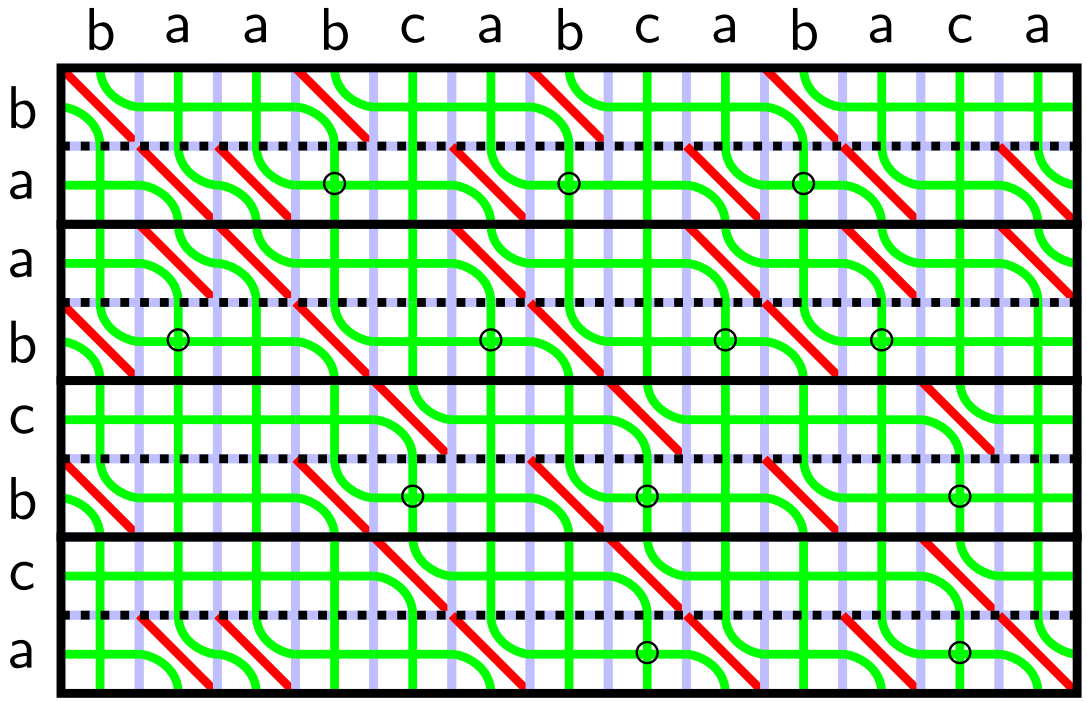
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

New algorithm: divide-and-conquer on strips



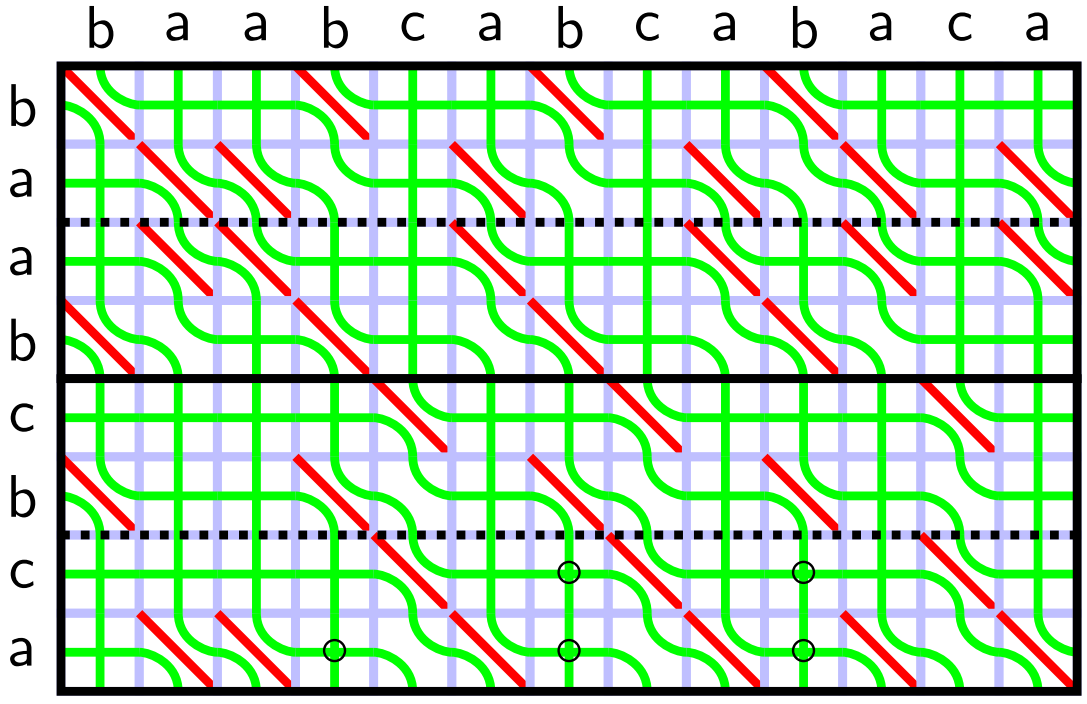
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

New algorithm: divide-and-conquer on strips



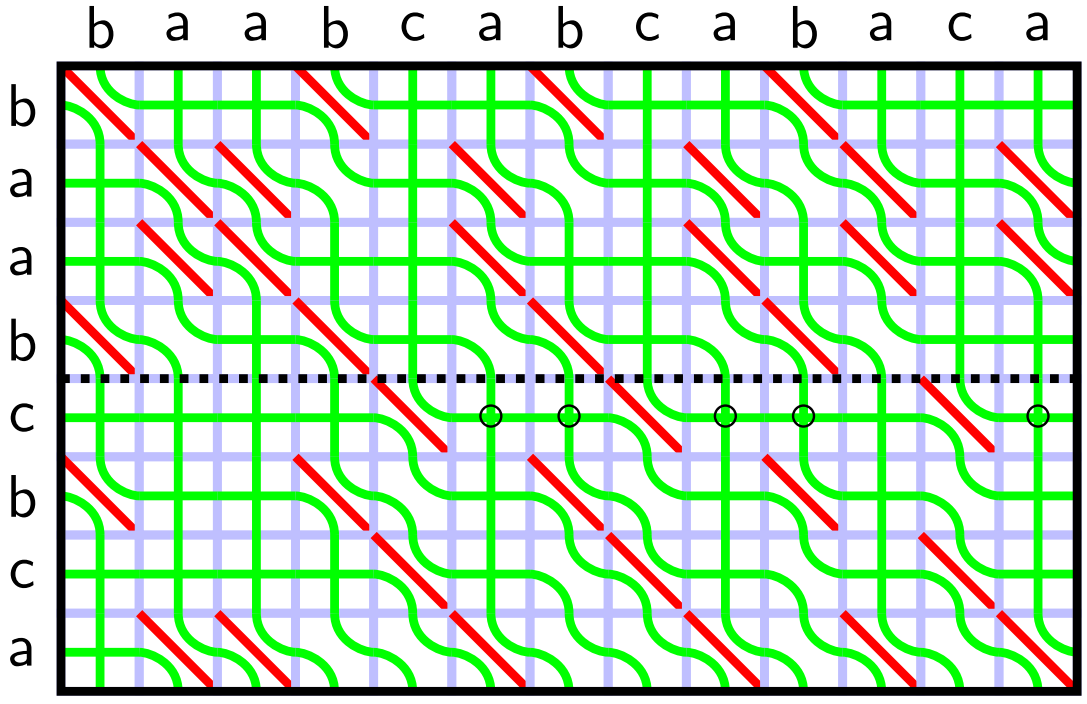
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

New algorithm: divide-and-conquer on strips



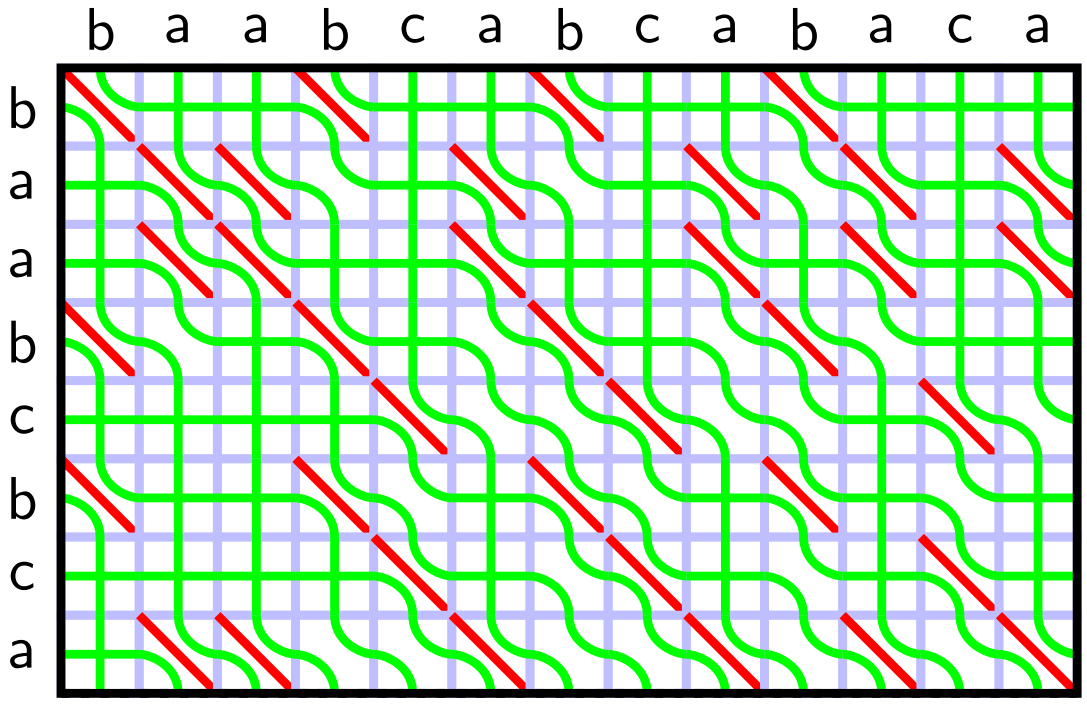
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

New algorithm: divide-and-conquer on strips



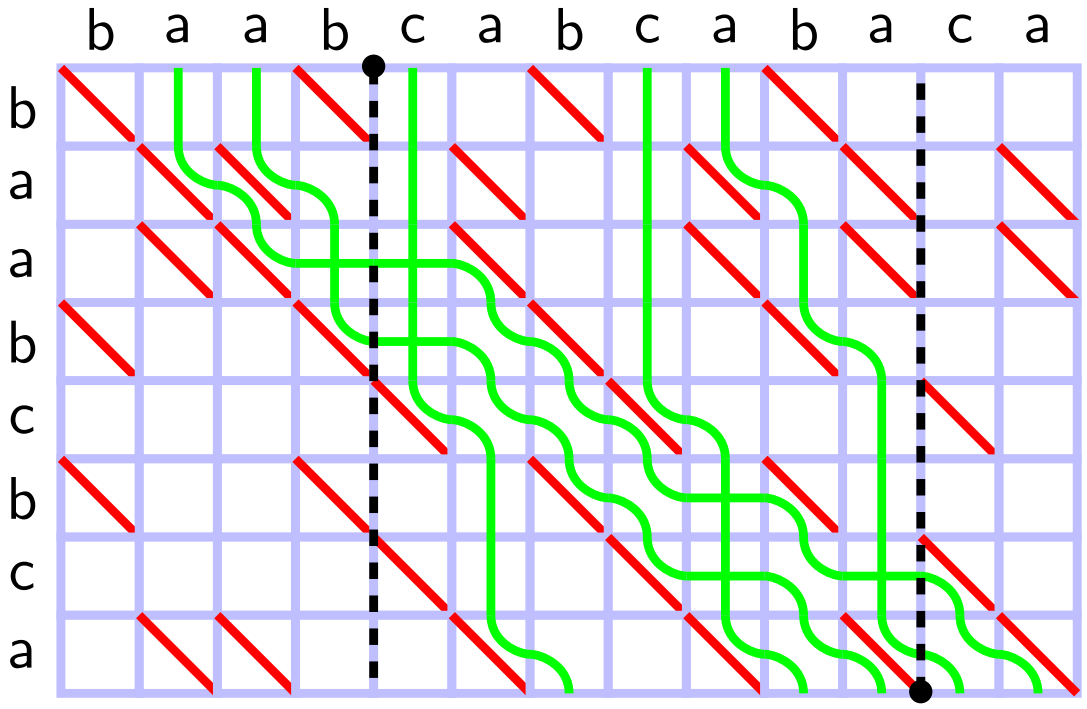
blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

New algorithm: divide-and-conquer on strips



blue $= 0$

red $= 1$

green $=$ *critical*

The algorithms

Theorem (new)

Algorithm for all semi-local LCS on permutations of length n:

- *time $O(n^{1.5})$ memory $O(n)$*
- *outputs $O(n)$ critical points*

Proof: in a strip of height k, at most k critical points non-trivial

Two such strips can be merged in time $O(k^{1.5})$

In every recursion level:

- number of subproblems up by a factor of 2
- time per subproblem down by a factor of $2^{1.5}$

Hence, top level dominates with time $O(n^{1.5})$ □

The algorithms

Application: Max-clique in circle graph with n chords

Standard reduction to strings: circle $\mapsto$ line, chords $\mapsto$ segments

Segment endpoints represented by permutation a of size $2n$

Chords intersect $\Leftrightarrow$ segments intersect without containment

Helly property: subset of segments intersect pairwise $\Rightarrow$ they all intersect at a common point

The algorithms

Theorem (new)

Algorithm for Max-clique in circle graph of size n:

- *time* $O(n^{1.5})$ *memory* $O(n)$

Proof: check all $2n + 1$ possible common intersection points

For each candidate intersection point, need to find the largest subset of covering segments without pairwise containment

Equivalent to prefix-suffix LCS on permutations a, id

Run semi-local LCS on a, id and build range tree: time $O(n^{1.5})$

Query prefix-suffix LCS for each candidate intersection point: time $(2n + 1) \cdot O(\log^2 n) = O(n \log^2 n)$

Overall time $O(n^{1.5})$ □

The algorithms

Theorem (new)

Algorithm for all semi-local LCS (over constant-size alphabet Σ):

- *time* $O\left(\frac{mn}{\log^{0.5}(m+n)}\right)$ *memory* $O(n)$
- *outputs* $O(m+n)$ *full-border critical points*

Must assume m and n "reasonably close", e.g. $(\log m)^{2.5} \leq n \leq m$

Proof: divide-and-conquer in alternate directions

Strips replaced by nearly-square blocks

Classical technique by Arlazarov+: when blocks sufficiently small, easier to precompute *all possible* blocks in advance

Threshold block size $0.5 \cdot \log_{|\Sigma|} m$

recursion time = precomputation time = $O\left(\frac{mn}{\log^{0.5}(m+n)}\right)$ $\square$

Conclusions and future work

Current state of the art for all semi-local LCS

Output representation:

- size $O(n \log n)$ query time $O(\log^2 n)$
- size $O(n)$ query time $O(\frac{\log n}{\log \log n})$ in a stronger model

Computation time:

- $O(\frac{mn}{\log^{0.5}(m+n)})$ on strings over constant-size alphabet
- $O(n^{1.5})$ on permutations

Memory: $O(m+n)$

Implies improvements for several existing problems

Conclusions and future work

Easy improvements (undergraduate exercises):

- extension to rational-weighted edit distance
- communication-efficient parallelisation

More sophisticated improvements (graduate projects):

- generalisation to sparse string comparison
- removing assumption of constant alphabet size

Potential further improvements (open problems):

- efficient recovery of full optimal subsequences
- extension to real-weighted edit distance
- new interesting special cases and applications
- better dominance counting
- local string comparison: alternative to Smith–Waterman?

References I


C. E. R. Alves, E. N. Cáceres, and S. W. Song.
An all-substrings common subsequence algorithm.
Electronic Notes in Discrete Mathematics, 19:133–139, 2005.


J. L. Bentley.
Multidimensional divide-and-conquer.
Communications of the ACM, 23(4):214–229, 1980.


J. JaJa, C. Mortensen, and Q. Shi.
Space-efficient and fast algorithms for multidimensional dominance reporting and counting.
In *Proceedings of the 15th ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 558–568, 2004.

References II

A. Tiskin.
All semi-local longest common subsequences in subquadratic time.
In *Proceedings of CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 352–363, 2006.

A. Tiskin.
Longest common subsequences in permutations and maximum cliques in circle graphs.
In *Proceedings of CPM*, 2006.
To appear.