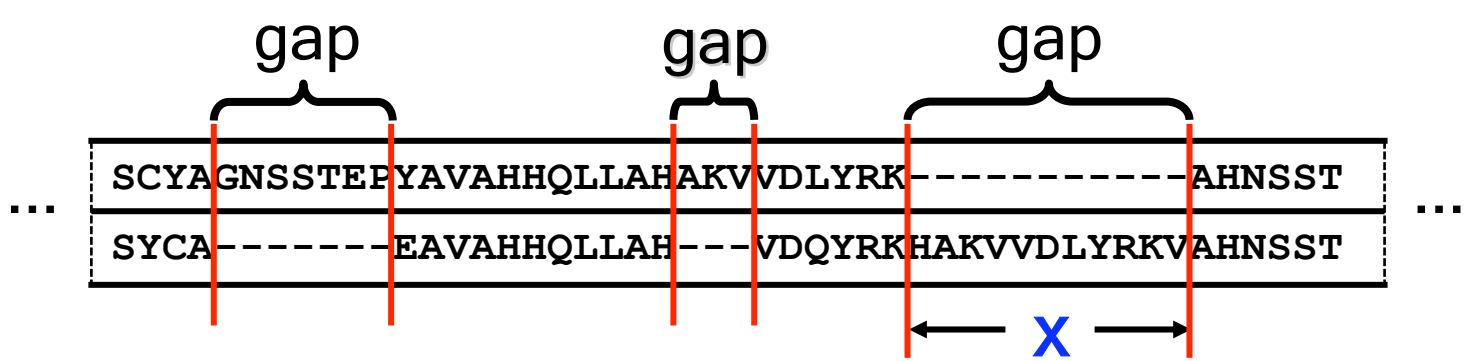
Aligning alignments exactly

John Kececioglu
Dean Starrett

Department of Computer Science
The University of Arizona
Tucson Arizona USA

Motivation

Linear gap-costs are necessary for biologically-correct alignment [Fitch, Smith 1981].



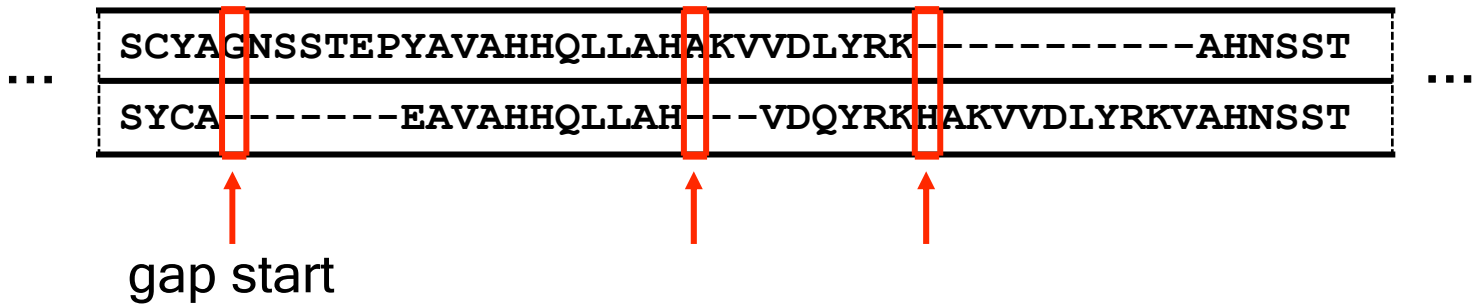
Gap cost $\gamma + \lambda x$

- γ, initiation cost
- λ, extension cost
- x, gap length

Motivation continued

A *two-sequence* alignment, with linear gap-costs, is scored,

$$\sum_{\text{columns}} \begin{pmatrix} \text{substitution} \\ \text{cost} \end{pmatrix} + \sum_{\text{columns}} \begin{pmatrix} \text{insertion} \\ \text{deletion} \\ \text{cost} \end{pmatrix} + \gamma \cdot \begin{pmatrix} \text{gap} \\ \text{count} \end{pmatrix}$$



Motivation continued

A *multiple-sequence* alignment, under the sum-of-pairs objective, is scored,

$$\sum_{\substack{\text{rows} \\ i,j}} \omega_{i,j} \cdot \left(\text{score for the 2-sequence alignment induced by rows } i \text{ and } j \right)$$

```
  SCYAGNSSTEPYAVA--QLLAHAKV--------
  --YAGNSSTEPYAVA---LLAHAKVVDSCYAGN
i SCYAGNSSTEPYAVA--QLLA-AKVVDSCY---
  -----NSSTEPYAVA--QLLAHAKVVDSCY---   ...
  SCYAGNSSTE----AHHQLLAHAKVVDSCY---
j SCYAGNSSTEPYAVAHHQLLA--KV--------
  -------STEPYAVAHHQLLAHAKVVDSCYAGN
```

Motivation continued

A *multiple-sequence* alignment, under the sum-of-pairs objective, is scored,

$$\sum_{\substack{\text{rows}\\ i,j}} \omega_{i,j} \cdot \left(\sum_{\text{cols}} \begin{pmatrix}\text{substitution}\\ \text{cost on } i,j\end{pmatrix} + \sum_{\text{cols}} \begin{pmatrix}\text{insertion}\\ \text{deletion}\\ \text{cost on } i,j\end{pmatrix} + \underbrace{\gamma \cdot \begin{pmatrix}\text{gap count}\\ \text{on } i,j\end{pmatrix}}_{\text{gap-count term}} \right)$$

```
SCYAGNSSTEPYAVA--QLLAHAKV--------
--YAGNSSTEPYAVA---LLAHAKVVDSCYAGN
SCYAGNSSTEPYAVA--QLLA-AKVVDSCY---
-----NSSTEPYAVA--QLLAHAKVVDSCY---
SCYAGNSSTE----AHHQLLAHAKVVDSCY---
SCYAGNSSTEPYAVAHHQLLA--KV--------
-------STEPYAVAHHQLLAHAKVVDSCYAGN
```

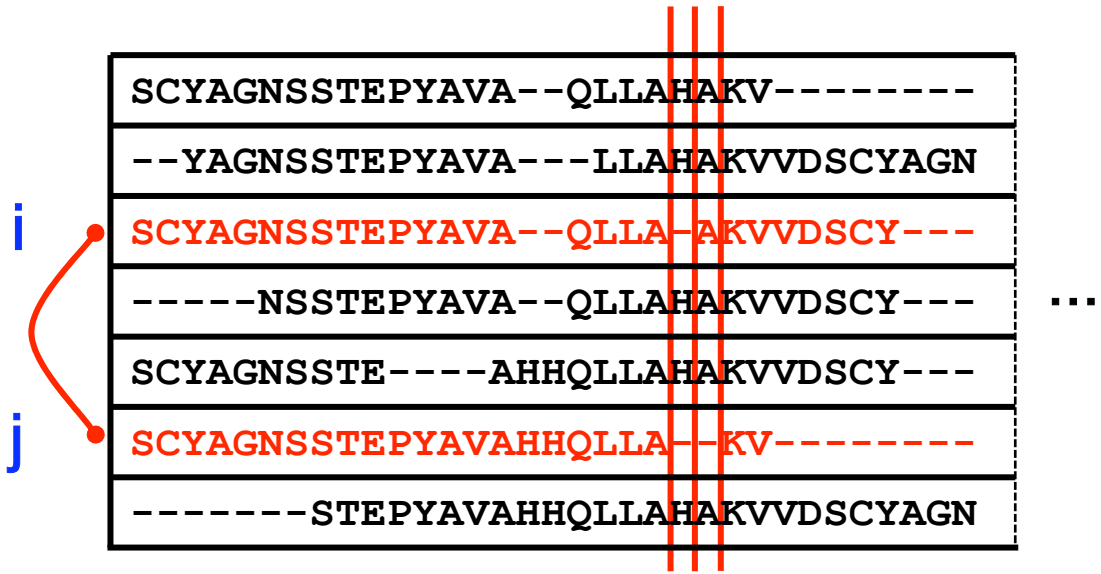
i

j

...

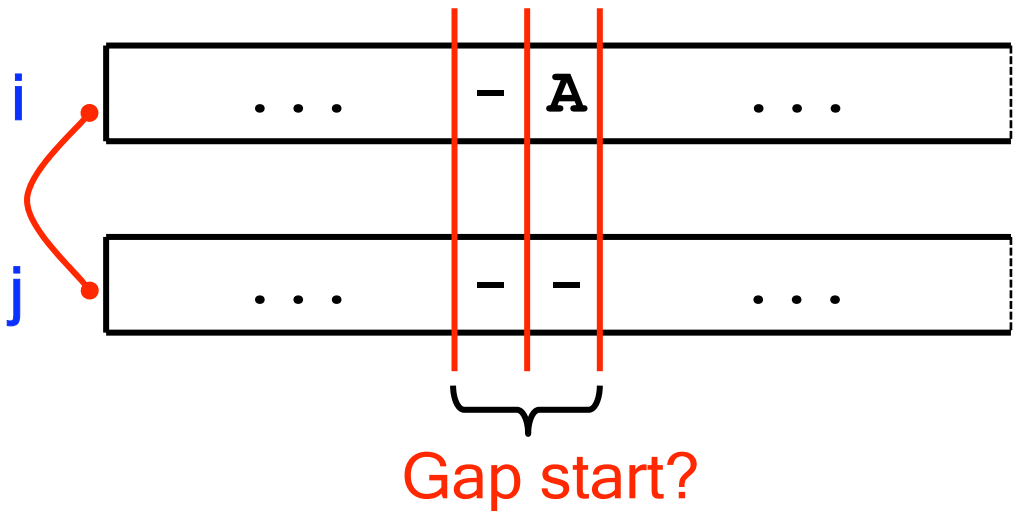
Motivation continued

- Computing the *gap-count* term is not easy.
- Known algorithms do not use *exact* gap-counts.



Motivation continued

- Computing the *gap-count* term is not easy.
- Known algorithms do not use *exact* gap-counts.



Motivation continued

The inherent *complexity* of gap-counts in multiple alignment has been a mystery.

- *Approximate* gap-counts [Altschul 1989] add exponential overhead [Gupta, Kececioglu, Schäffer 1995].
- *Without* gap-counts, multiple alignment is already NP-complete [Wang, Jiang 1994; Wareham 1995; Kececioglu 1993].

Motivation continued

We show that a form of multiple alignment, called *Aligning Alignments*, is,

(1) NP-complete *with* exact gap-counts,

(2) polynomial-time solvable *without* them, yet

(3) can be exactly-solved with gap-counts *in practice*.

Together (1) and (2) show exact gap-counts are *inherently hard*.

Problem

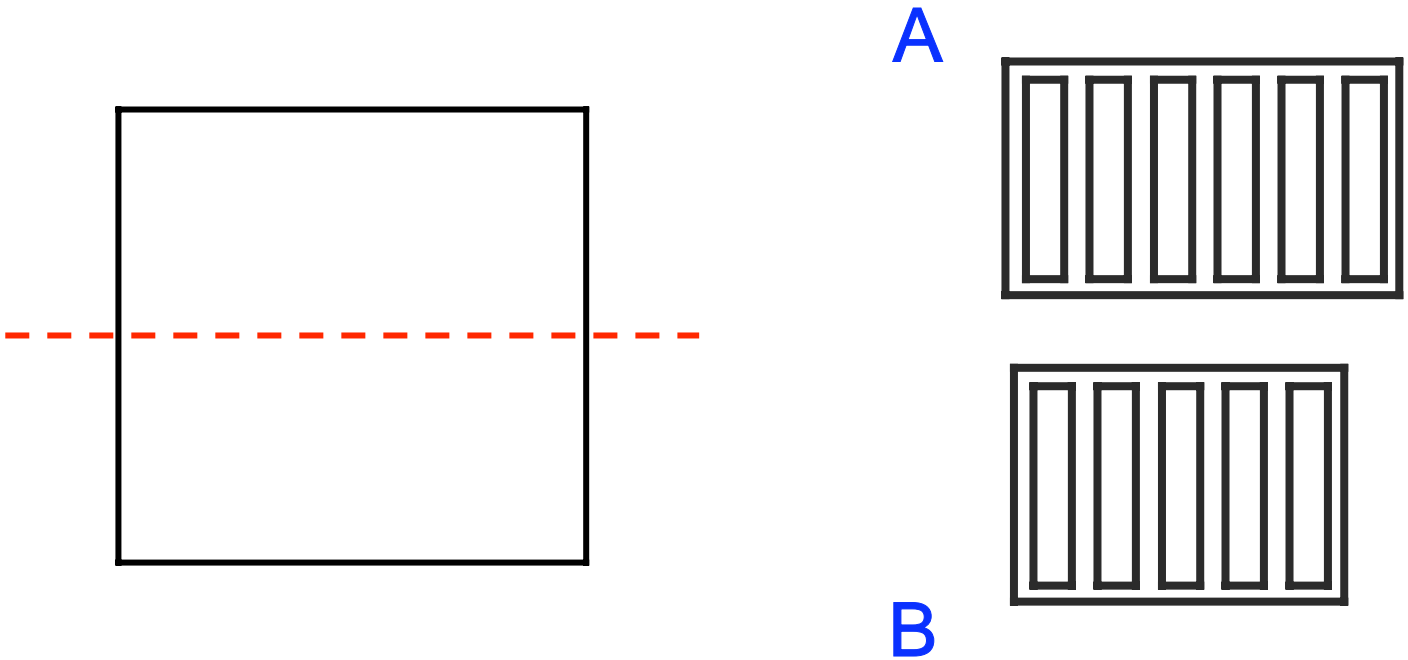
Aligning Alignments is the following problem.

Given multiple alignments A and B, find an optimal alignment of

- the *columns* of A versus the columns of B,
- under the sum-of-pairs objective,
- with linear *gap-costs*.

Problem continued

The alignment of A with B substitutes, inserts, and deletes *columns*.



This yields a multiple alignment containing A and B.

Results

For Aligning Alignments, we show the following.

- NP-completeness
- Exact algorithm
- Tight analysis
- Speedup techniques
  - *Biological alignments* in 1 second (25 sequences, length 900)
  - *Simulated alignments* in 3 minutes (200 sequences, length 1000)
- Linear space
- Ceiling phenomenon

Related work

Gotoh (1993,1994)

- First to consider Aligning Alignments.
- Presented four *procedures* (one finds an optimal solution).
- Gave complex criteria for eliminating *candidate solutions*.
- Showed how to *evaluate* gap-counts in linear time.

Related work continued

Kececioglu and Zhang (1998)

- Introduced *optimistic-pessimistic* gap-counts for aligning alignments, profiles, or both.
- Gave a polynomial-time exact-algorithm for aligning a *sequence* versus an alignment.
- Conjectured that Aligning Alignments is *NP-complete*.

Related work continued

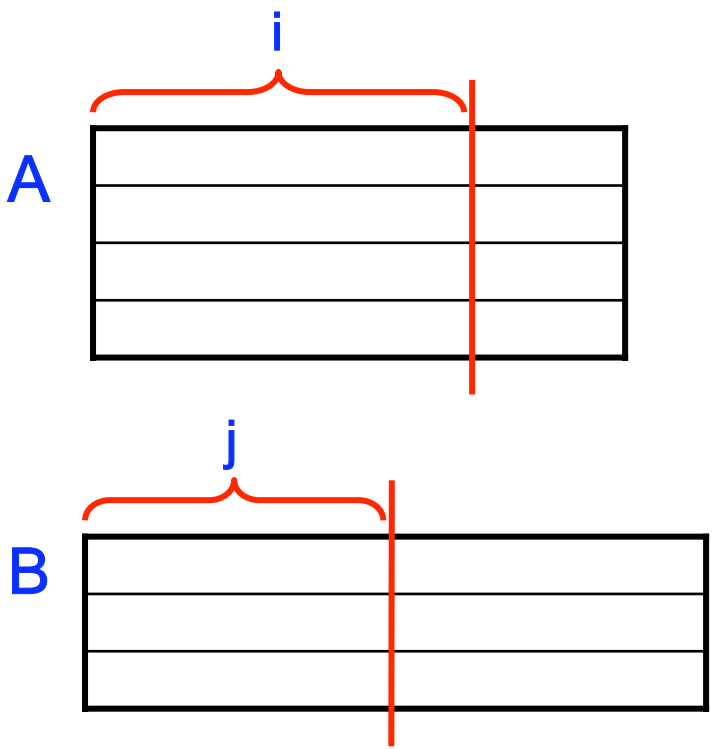
Ma, Wang and Zhang (2003)

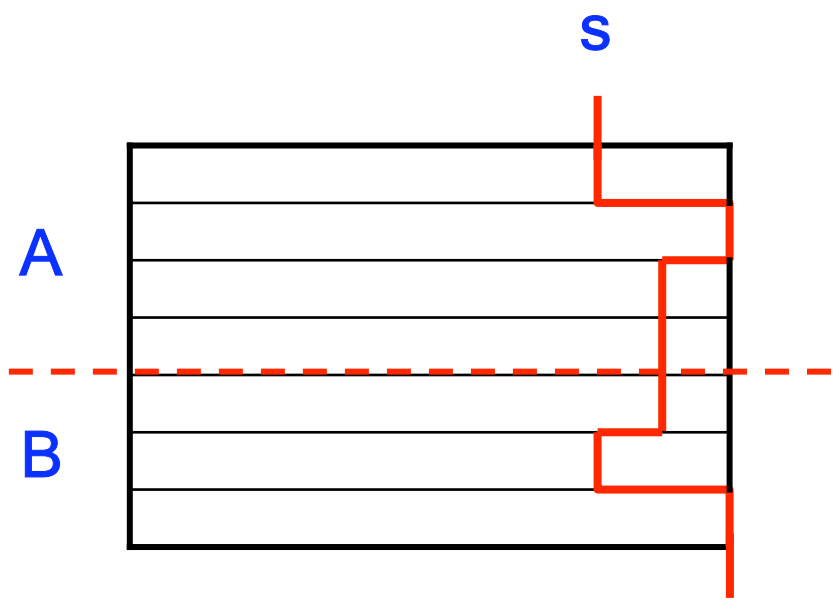
- Independently proved *NP-completeness*.
- Rediscovered Gotoh's *heuristic* and *speedup*.

Algorithm

We solve Aligning Alignments by *dynamic programming*.

- Inputs A and B viewed as *sequences* of columns.
- Subproblem consists of prefixes i and j, and a *shape* s.

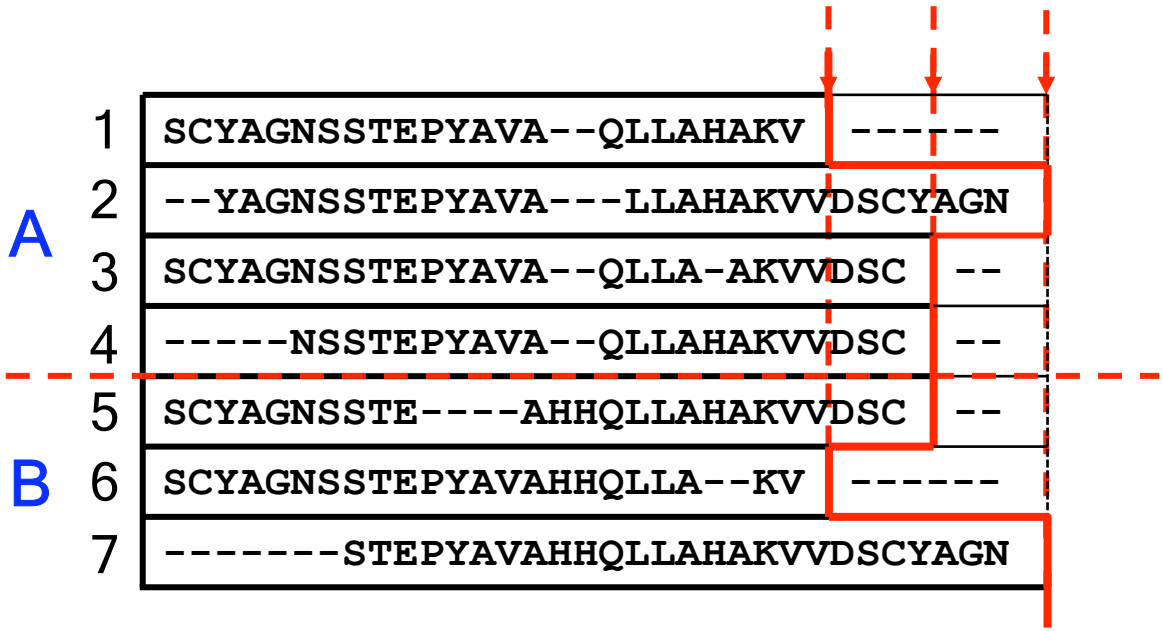




Algorithm continued

A shape is an *ordered partition* of the rows.

({1,6}, {3,4,5}, {2,7})



The *overhang* or *underhang* of a pair of rows is given by the order of the blocks.

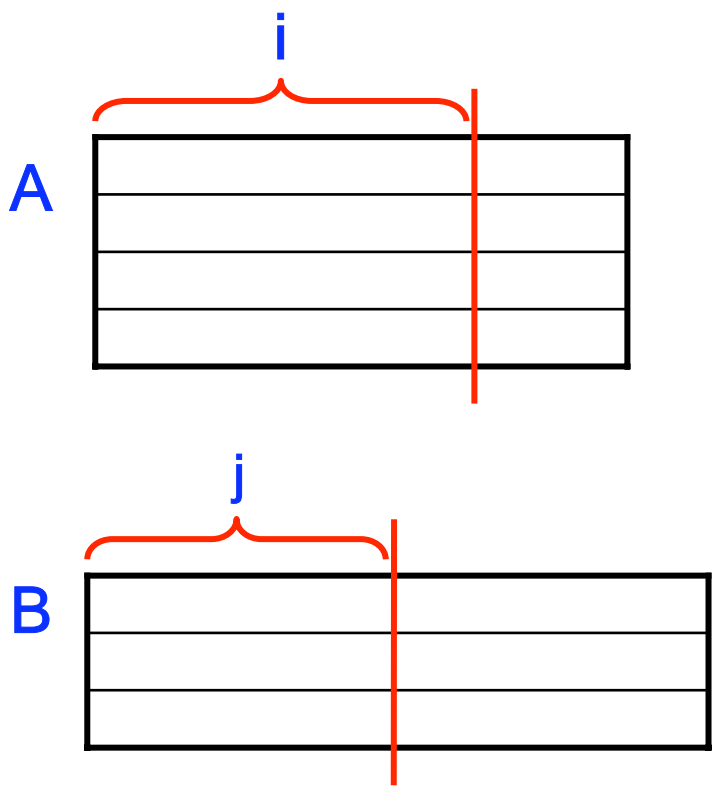
Algorithm continued

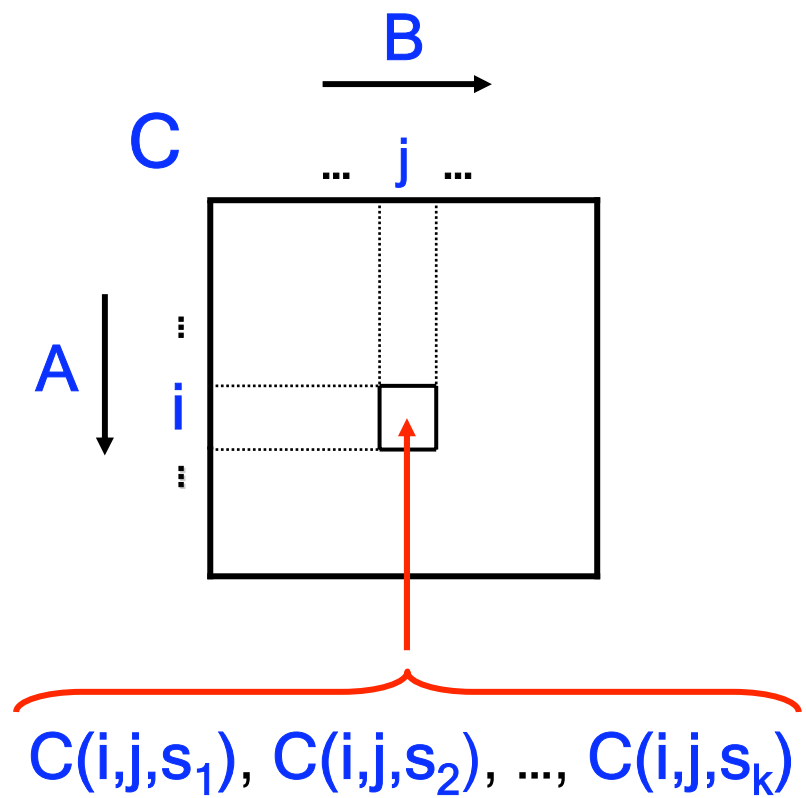
The *cost* of an optimal solution to subproblem (i,j,s) is *C(i,j,s)*.

$$C(i,j,s) \text{ is a function of } \begin{cases} C(i, j-1, t) & + \text{(column cost)}, \\ C(i-1, j, t') & + \text{(column cost)}, \\ C(i-1, j-1, t'') & + \text{(column cost)} \end{cases}$$

Algorithm continued

The costs C(i,j,s) are computed in a table.

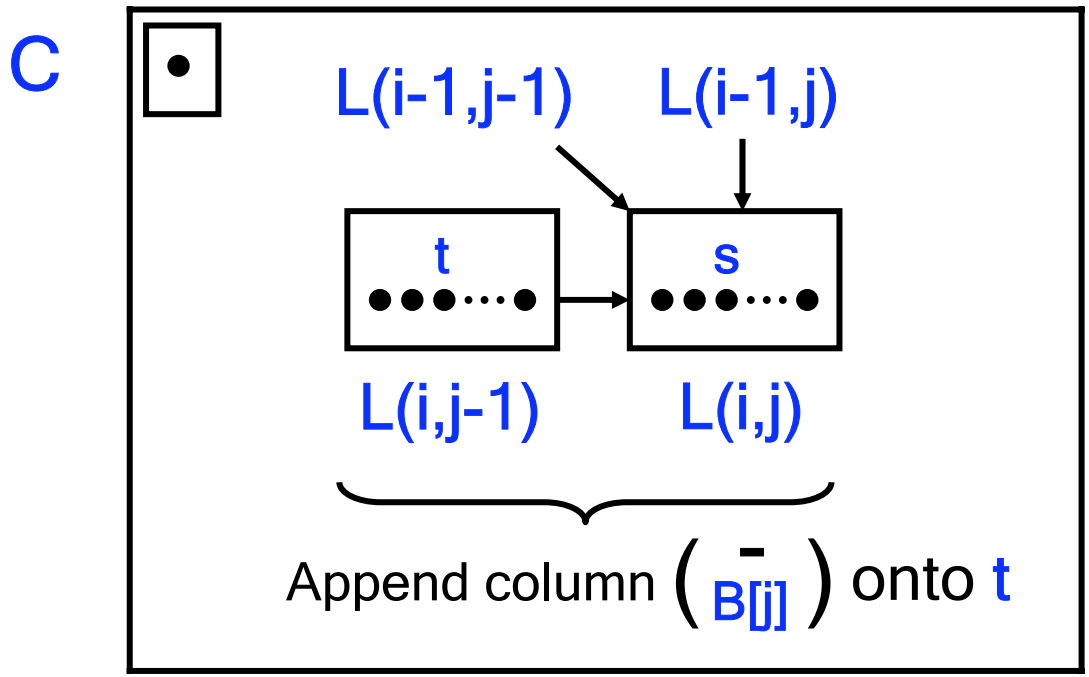




Algorithm continued

Entry (i,j) holds a list of *realizable shapes* L(i,j).

- L(0,0) holds the flat shape.
- L(i,j) is obtained from adjacent entries by appending columns.



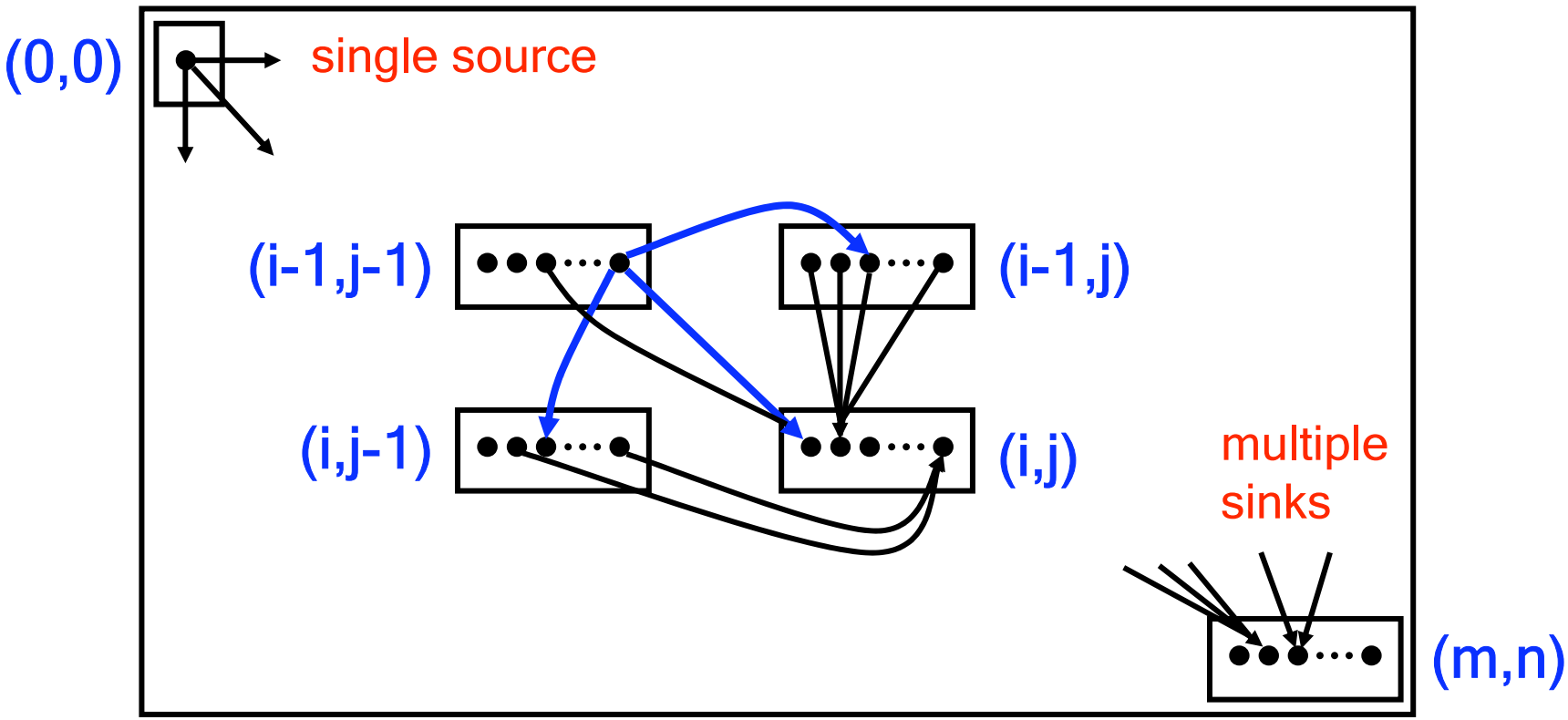
Algorithm continued

The dynamic program is viewed as a *shortest-path problem*.

- Vertex for each *subproblem* (i,j,s).
- Edge between subproblems that append a *column*.
- Weight of an edge is the column *cost*.
- Graph built on-the-fly in *lexicographic* order on (i,j).

Algorithm continued

We solve a *source-sink* shortest-path problem on the graph.



Time and space

The time and space for the exact algorithm depends on the *number of shapes* at entries.

- Number of shapes is a *complex function* of gap-structure.
- Let F(m,n) denote the *number of alignments* of two strings of lengths m and n.
- Let a and b denote the *number of sequences* in A and B.

Time and space continued

The *number of shapes* is a surprising function of F, a, and b.

Theorem The *worst-case* number of shapes at entry (i,j) is exactly,

$$\underbrace{F\big(\min\{a,i\}, \min\{b,j\}\big)}_{\leq\ F(a,b)}. \qquad \square$$

= number of alignments of two strings whose *lengths* are the number of *sequences* !

Time and space continued

We know the time and space when *both* inputs have k *sequences* and n *columns*.

Theorem The exact algorithm takes worst-case *time*,

$$\begin{cases} \theta\left((3+\sqrt{2})^k \; (n-k)^2 \; k^{3/2} \right), & k < n; \\ \theta\left((3+\sqrt{2})^n \; k^2 \; n^{-1/2} \right), & k \text{ ¿ } n. \end{cases}$$

The worst-case *space* is a factor k smaller. □

Time and space continued

The following bound is *simpler* but looser.

Corollary For k sequences and n columns, the exact algorithm runs in *time*,

$$O\left(5^{\min\{k,n\}} \cdot \max\{k,n\}^2\right). \quad \square$$

Speedup techniques

To reduce the time and space, we *prune* shapes at entries.

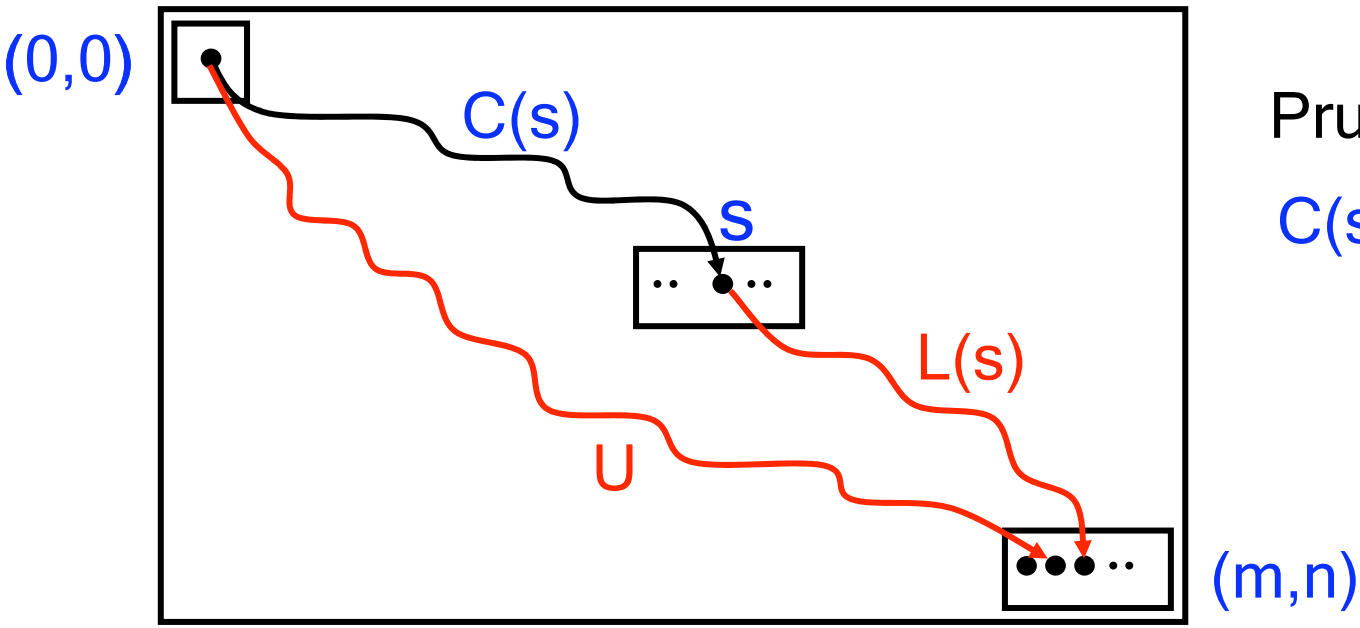
- *Bound pruning* uses upper- and lower-bounds on alignment costs.
- *Dominance pruning* uses a dominance relation on shapes.

Both preserve *correctness*.

Bound pruning

We prune shape s if, on every *extension*, it is no better than a feasible solution.

- Compute *lower-bound* L(s) on cost of extending s.
- Compute *upper-bound* U on cost of optimal solution.



Prune s if

$C(s) + L(s) \geq U$

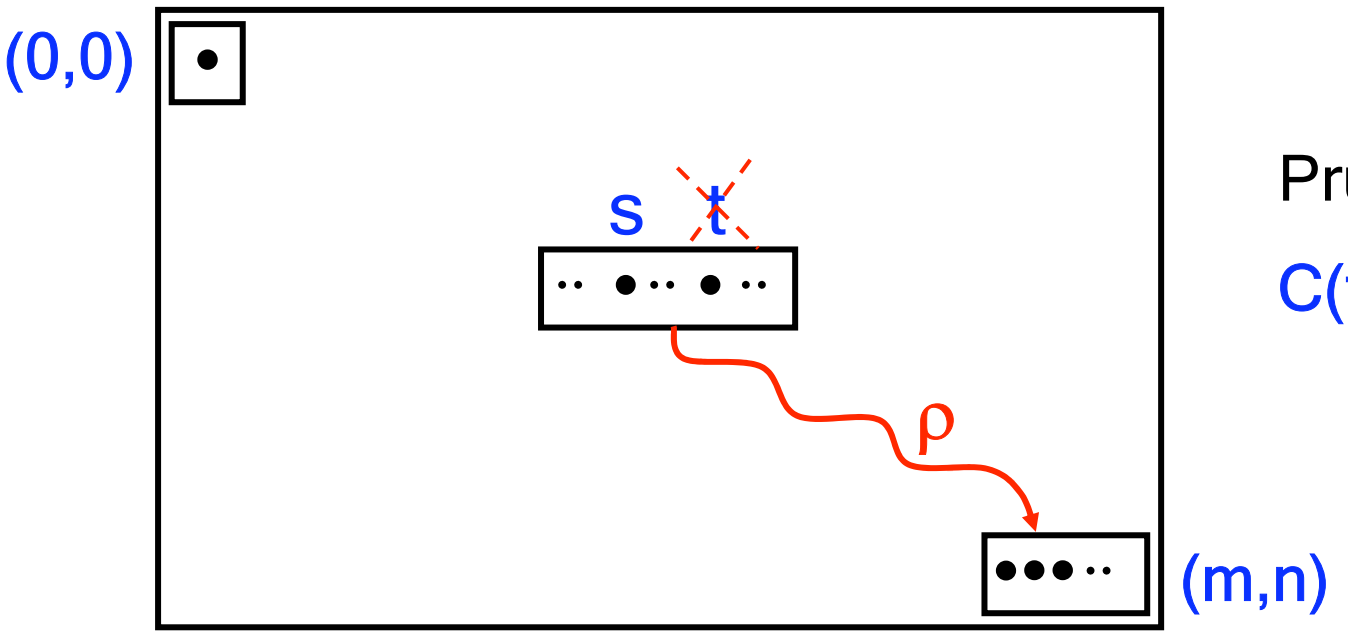
Bound pruning continued

The algorithm computes bounds using *approximate* gap-counts.

- Use *optimistic* gap-counts on suffixes for lower-bound L(s).
- Use *pessimistic* gap-counts for upper-bound U.
- Use a *lookup table* to efficiently evaluate L(s).

Dominance pruning

We prune shape t if it is no better than *some* shape s on *all* extensions.



Prune t if for all ρ,

$C(t \bullet \rho) \geq C(s \bullet \rho)$

Dominance pruning continued

We express this by a dominance *relation* on shapes.

Definition Shape s *dominates* shape t if,

$$C(t) \geq C(s) + \gamma \cdot \sum_{\substack{\text{rows} \\ p,q}} \begin{pmatrix} 1, & \text{if } t \text{ ends with a gap that } s \text{ does not;} \\ 0, & \text{otherwise} \end{pmatrix}$$

Dominance is an easily-tested *sufficient condition* for t to be no better than s.

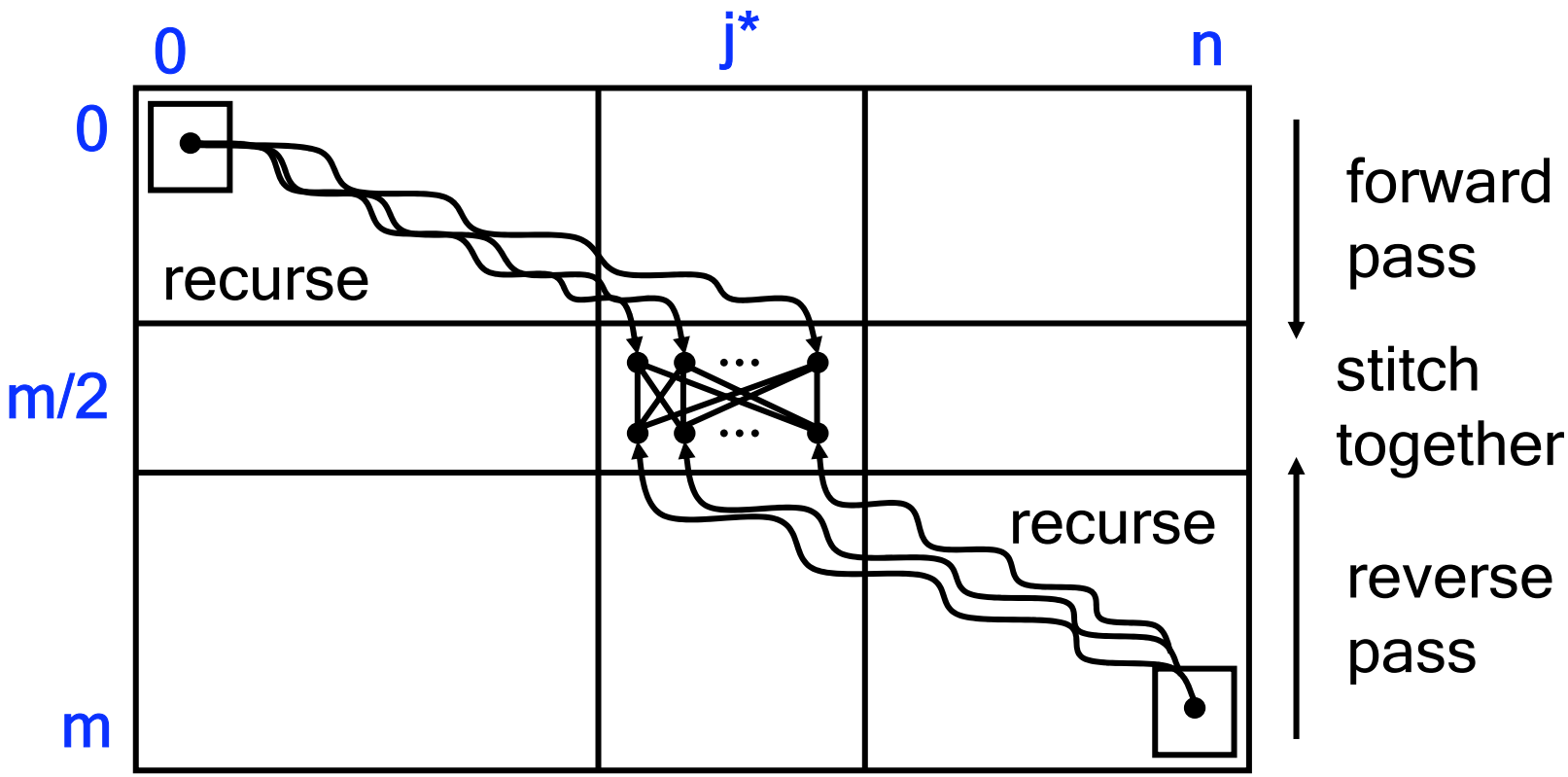
Linear space

We generalize the classic *linear-space* result on aligning two strings [Hirschberg 1975; Myers, Miller 1988].

- Space becomes linear in the number of *columns*.
- Subproblem decomposition is *complicated* by the presence of shapes.
- Technique is compatible with *dominance* pruning.
- Time with dominance pruning does *not increase*.

Linear space

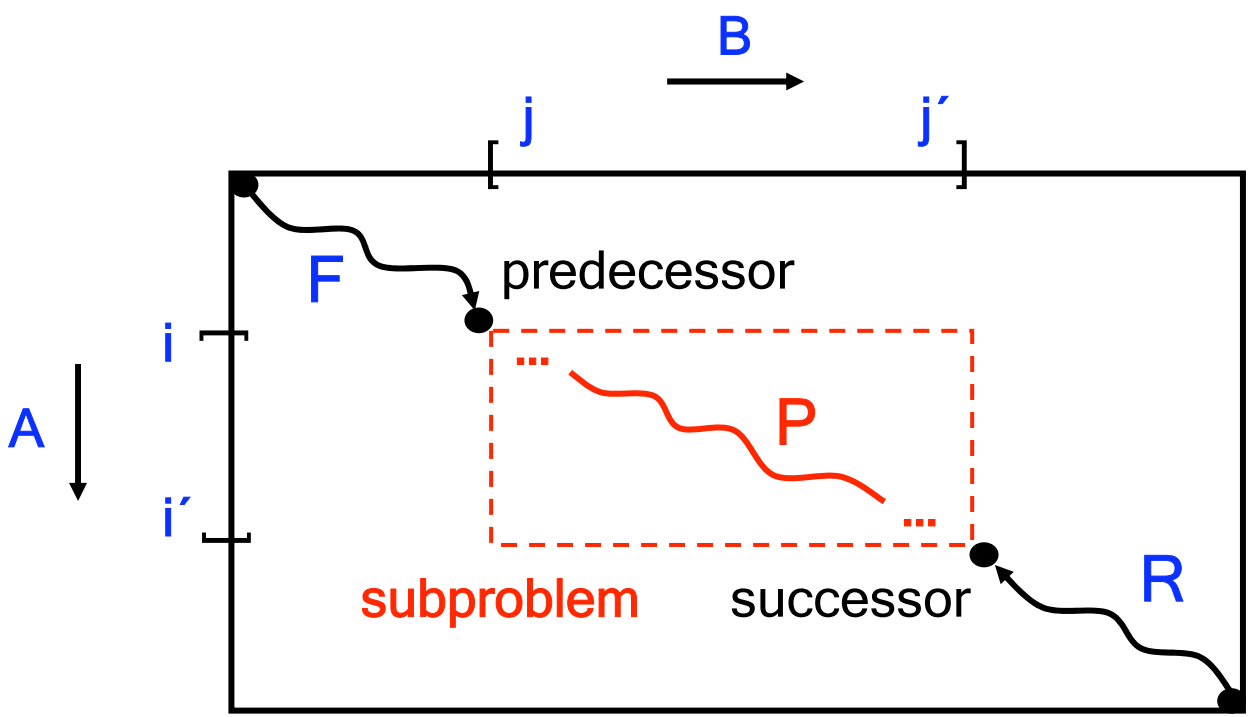
We find the cost, shape, and entry of an optimal path through the *middle row*.



Linear space continued

This leads to the *subproblem* of

- aligning A[i:i´] and B[j:j´], given
- *predecessor* and *successor* shapes.



Experimental results

To evaluate the feasibility of computing *optimal alignments*, we used both

- *biological data*, obtained from the **BAliBASE** [Thompson, Plewniak, Poch 1999] and **MVF** [McClure, Vassi, Fitch 1994] collections, and

- *simulated data*, generated by randomly aligning strings with parameterized gap-structure.

Experimental results continued

We implemented five *versions* of the exact algorithm:

- **Q**, no pruning, quadratic space,
- **BQ**, bound pruning, quadratic space,
- **DQ**, dominance pruning, quadratic space,
- **BDQ**, bound and dominance pruning, quadratic space,
- **DL**, dominance pruning, linear space.

Time and space continued

The **MVF** collection has 4 benchmarks with

- 12 *sequences*, and
- 150 - 400 *columns*.

The maxima, over 2047 *instances* for every benchmark, are:

	Version	Total shapes	Time (sec)	Space (Mb)
	Q	11,950,000	6,570.0	57.5
	BQ	3,315,000	980.0	28.8
most time efficient	**DQ**	199,000	0.8	3.3
most time efficient	**BDQ**	39,000	0.2	3.7
most space efficient	**DL**	208,000	1.9	0.1

Experimental results continued

The **BAliBASE** collection has 144 alignments with

- 3 - 30 *sequences*, and
- 60 - 1000 *columns*.

The maximum time and space, over 20 *instances* for every benchmark, is:

Version	Time (sec)	Space (Mb)
BDQ	1.0	23.2
DL	14.6	0.2

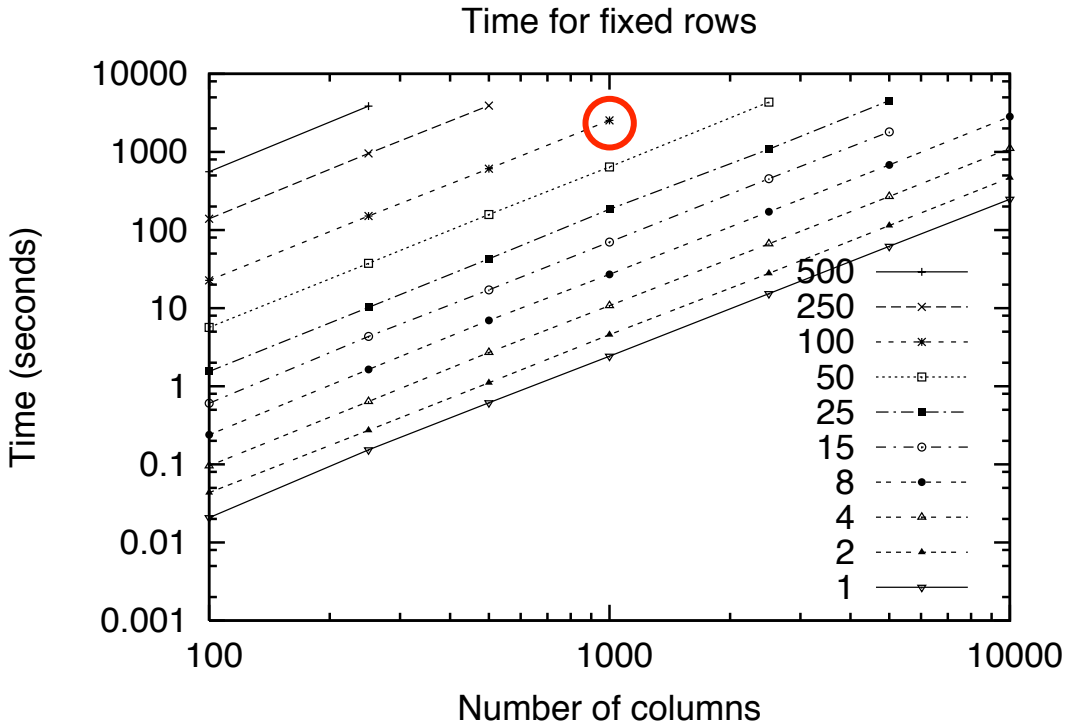
Experimental results continued

We used *simulated data* to study the observed growth in time and space.

- *Letter distribution* same as benchmarks.
- *Gap density* same as hardest benchmarks.
  - *Spacer density* , the percentage of entries that are spacers, fixed at 35%.
  - *Startup density* , the percentage of entries that start a gap, fixed at 10%.

Experimental results continued

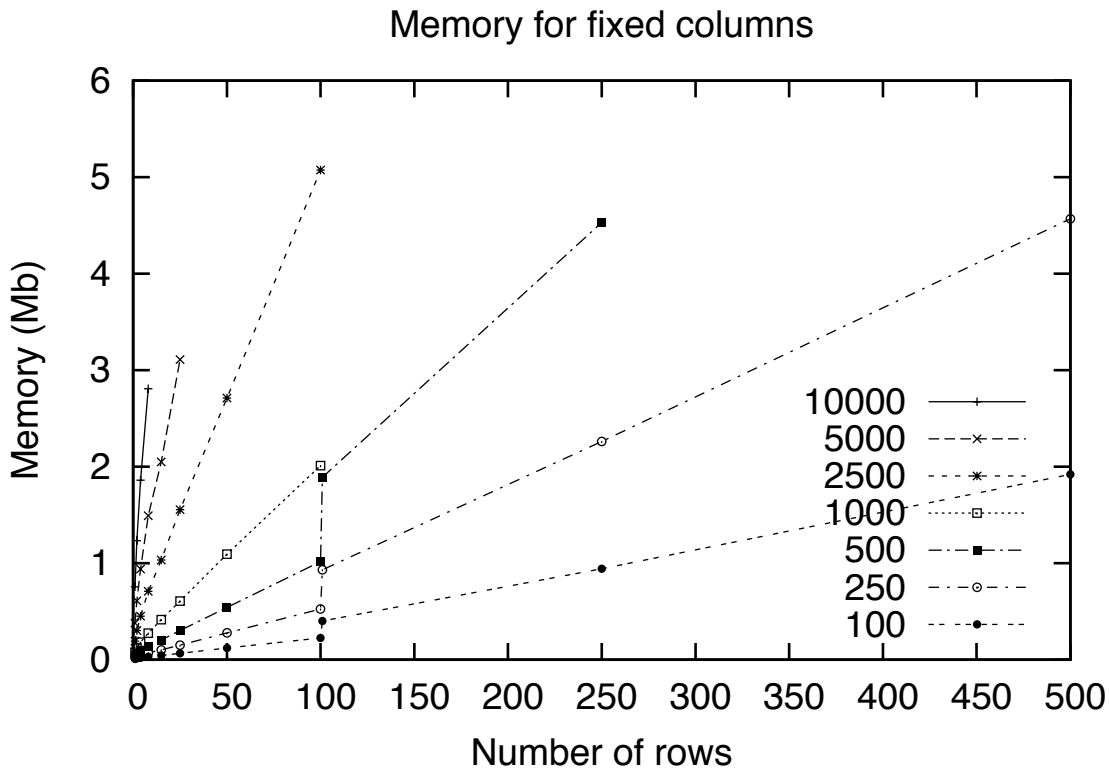
We studied time for version **DL** as a function of *columns*, n.



- *Time* is quadratic in n.
- Aligns 200 rows and 1000 columns in *3 minutes* and 2 Mb.

Experimental results continued

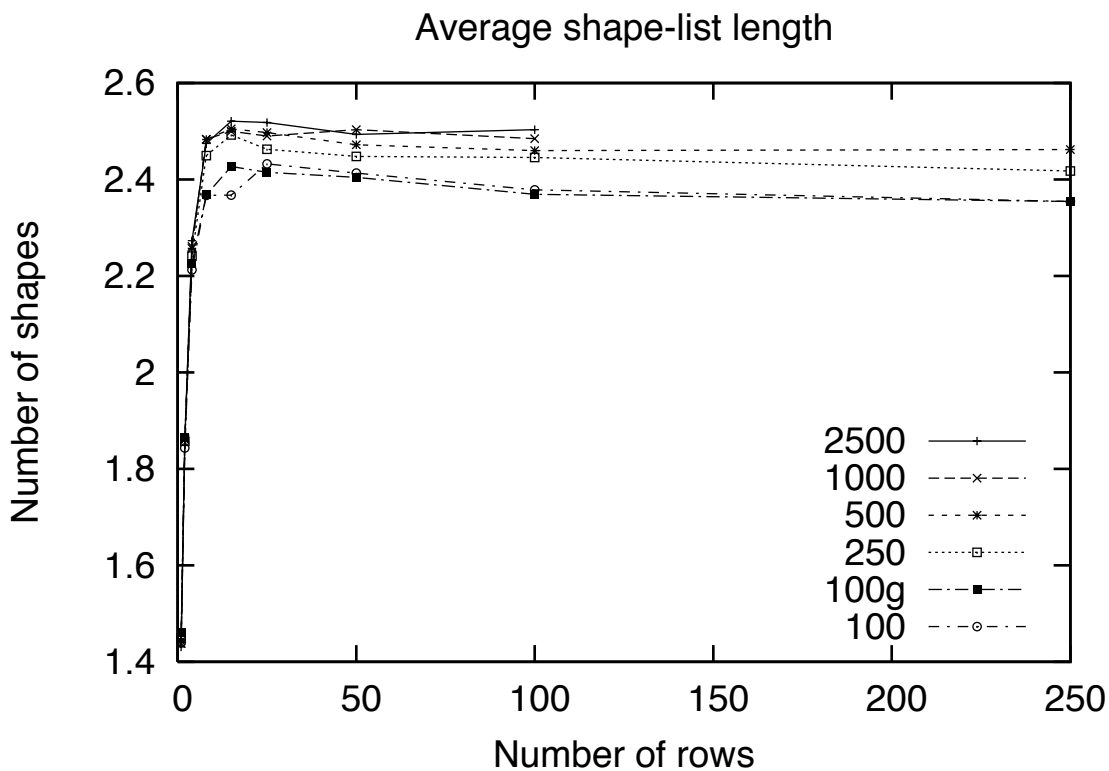
We studied space for version **DL** as a function of *rows*, k.



- *Space* is linear in k.
- This implies the number of *shapes* is constant in k!

Experimental results continued

A *ceiling phenomenon* in shape-growth causes this behavior.



- Less than *3 shapes* per entry, independent of k.
- Same behavior on *biological data*.

Conclusions

We can *solve* large, highly-gapped instances of Aligning Alignments in practice.

- *Aligns* 200 sequences, 1000 columns in 3 minutes, 2 Mb.
- *Fastest* version combines bound- and dominance-pruning.
- *Smallest* version does dominance-pruning in linear-space.
- *Ceiling phenomenon* explains tractability.

Future work

Many interesting questions remain *open*.

- Is Aligning Alignments *approximable*?
- How accurate is *gap placement* by heuristics, compared to the exact algorithm?
- Is the ceiling phenomenon explainable by a *probabilistic analysis*?