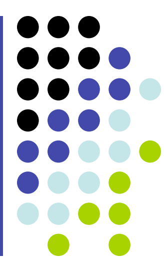
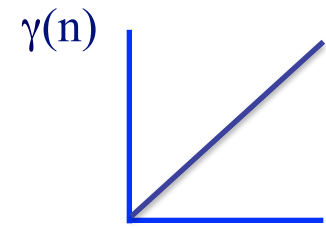
Scoring the gaps more accurately

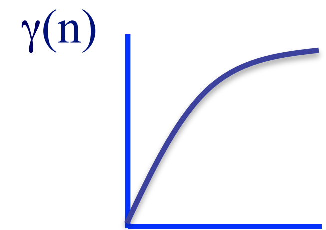




Current model:

Gap of length n
incurs penalty $n \times d$

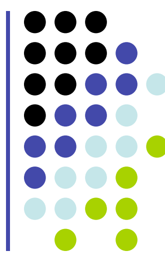
However, gaps usually occur in bunches



Concave gap penalty function $\gamma(n)$
(aka Convex $-\gamma(n)$):

$\gamma(n)$:
for all n, $\gamma(n + 1) - \gamma(n) \leq \gamma(n) - \gamma(n - 1)$

Convex gap dynamic programming



Initialization: same

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ \max_{k=0\ldots i-1} F(k, j) - \gamma(i-k) \\ \max_{k=0\ldots j-1} F(i, k) - \gamma(j-k) \end{cases}$$

Termination: same

Running Time: $O(N^2M)$ (assume N>M)

Space: $O(NM)$

Compromise: affine gaps

$\gamma(n) = d + (n - 1) \times e$

gap open (d) | gap extend (e)

To compute optimal alignment,

At position i, j, need to "remember" best score if gap is open
best score if gap is not open

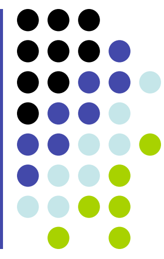
F(i, j): score of alignment $x_1 \ldots x_i$ to $y_1 \ldots y_j$
if x_i aligns to y_j

G(i, j): score **if** x_i aligns to a gap after y_j

H(i, j): score **if** y_j aligns to a gap after x_i

V(i, j) = best score of alignment $x_1 \ldots x_i$ to $y_1 \ldots y_j$

Needleman-Wunsch with affine gaps



Why do we need matrices F, G, H?

Because, perhaps

$\mathbf{G}(i, j) < \mathbf{V}(i, j)$

(it is best to align x_i to y_j if we were aligning only $x_1 \ldots x_i$ to $y_1 \ldots y_j$ and not the rest of x, y),

but on the contrary

$\mathbf{G}(i, j) - e > \mathbf{V}(i, j) - d$

(i.e., had we "fixed" our decision that x_i aligns to y_j, we could regret it at the next step when aligning $x_1 \ldots x_{i+1}$ to $y_1 \ldots y_j$)

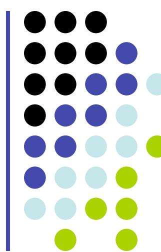
Add -d

$\mathbf{G}(i+1, j) = \mathbf{F}(i, j) - d$

Add -e

$\mathbf{G}(i+1, j) = \mathbf{G}(i, j) - e$

Needleman-Wunsch with affine gaps



Initialization: $V(i, 0) = d + (i - 1) \times e$
$V(0, j) = d + (j - 1) \times e$

Iteration:

$$F(i, j) = V(i - 1, j - 1) + s(x_i, y_j)$$

$$G(i, j) = \max \begin{cases} V(i - 1, j) - d \\ G(i - 1, j) - e \end{cases}$$

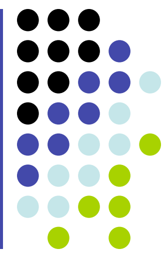
$$H(i, j) = \max \begin{cases} V(i, j - 1) - d \\ H(i, j - 1) - e \end{cases}$$

$$V(i, j) = \max\{ F(i, j), G(i, j), H(i, j) \}$$

Termination: V(i, j) has the best alignment

Time?
Space?

To generalize a bit…

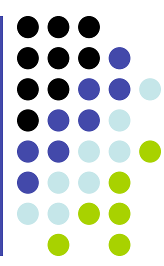


… think of how you would compute optimal alignment with this gap function

$\gamma(n)$

….in time O(MN)

Bounded Dynamic Programming



Assume we know that x and y are very similar

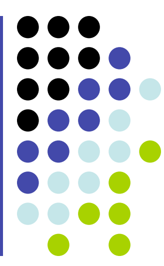
Assumption: # gaps(x, y) < k(N)

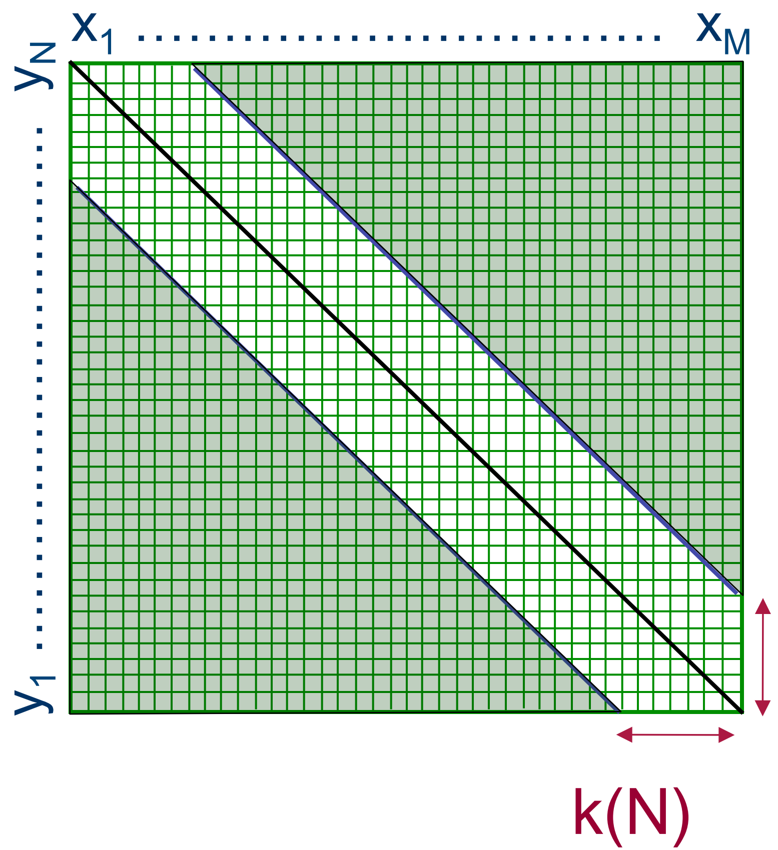
Then, x_i | y_j implies $| i - j | < k(N)$

We can align x and y more efficiently:

Time, Space: $O(N \times k(N)) \ll O(N^2)$

Bounded Dynamic Programming





Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

For i = 1…M

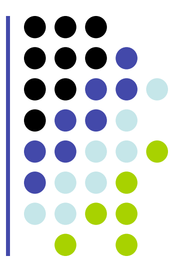
For j = max(1, i – k)…min(N, i+k)

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i - k(N) \\ F(i-1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

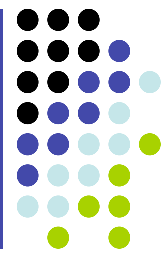
Termination: same

Easy to extend to the affine gap case

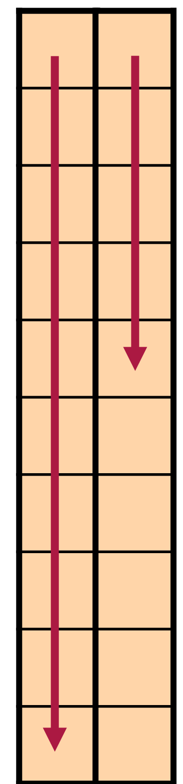
Outline



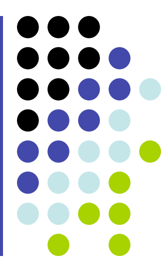
- Linear-Space Alignment
- BLAST – local alignment search
- Ultra-fast alignment for (human) genome resequencing



Linear-Space Alignment



Subsequences and Substrings



Definition A string x' is a ***substring*** of a string x,
if x = ux'v for some prefix string u and suffix string v

(similarly, $x' = x_i \ldots x_j$, for some $1 \le i \le j \le |x|$)

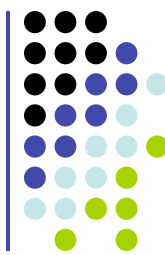
A string x' is a ***subsequence*** of a string x
if x' can be obtained from x by deleting 0 or more letters

($x' = x_{i1} \ldots x_{ik}$, for some $1 \le i_1 \le \ldots \le i_k \le |x|$)

Note: a substring is always a subsequence

Example: **x = abracadabra**
y = cadabr; ***substring***
z = brcdbr; ***subseqence, not substring***

Hirschberg's algortihm



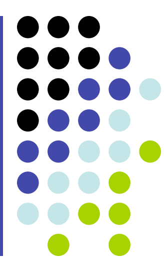
Given a set of strings x, y,…, a ***common subsequence*** is a string u that is a subsequence of all strings x, y, …

- Longest common subsequence
  - Given strings $x = x_1\ x_2\ \ldots\ x_M$, $y = y_1\ y_2\ \ldots\ y_N$,
  - Find longest common subsequence $u = u_1\ \ldots\ u_k$

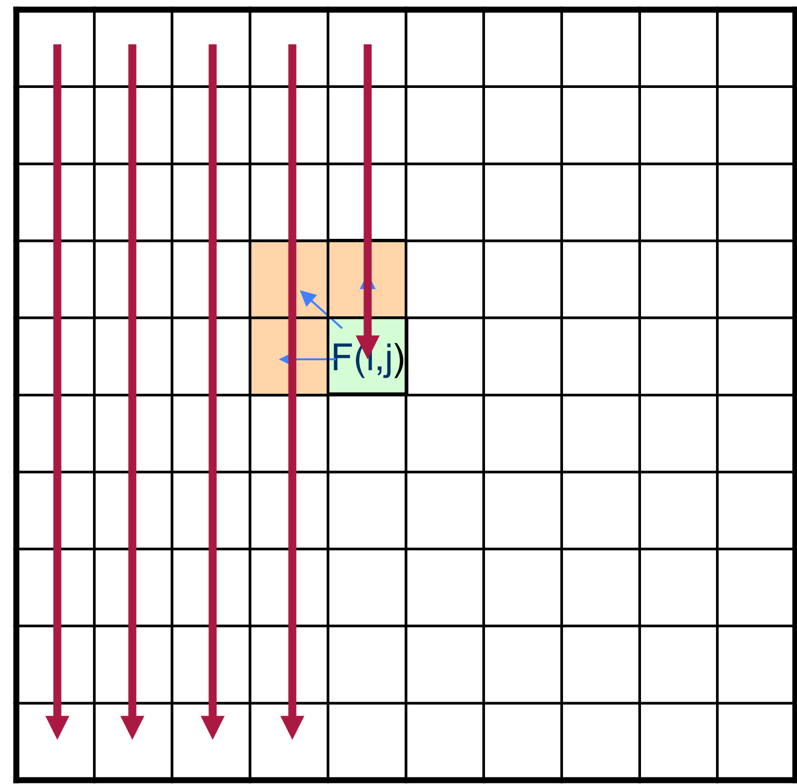
- Algorithm:
  - $$F(i, j) = \max \begin{cases} F(i-1, j) \\ F(i, j-1) \\ F(i-1, j-1) + [1, \text{ if } x_i = y_j;\ 0 \text{ otherwise}] \end{cases}$$
  - Ptr(i, j) = *(same as in N-W)*
  - Termination: trace back from Ptr(M, N), and prepend a letter to u whenever
    - Ptr(i, j) = DIAG **and** $F(i-1, j-1) < F(i, j)$

- Hirschberg's original algorithm solves this in linear space

Introduction: Compute optimal score



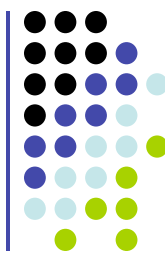
It is easy to compute F(M, N) in linear space



Allocate (column[1])
Allocate (column[2])

For i = 1….M
    If i > 1, then:
        Free(column[i – 2])
        Allocate(column[i])
    For j = 1…N
        F(i, j) = …

Linear-space alignment



To compute both the optimal score **and** the optimal alignment:

Divide & Conquer approach:

Notation:

x^r, y^r: reverse of x, y

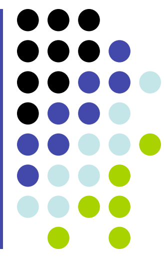
E.g. x = accgg;

x^r = ggcca

$F^r(i, j)$: optimal score of aligning $x^r_1 \ldots x^r_i$ & $y^r_1 \ldots y^r_j$

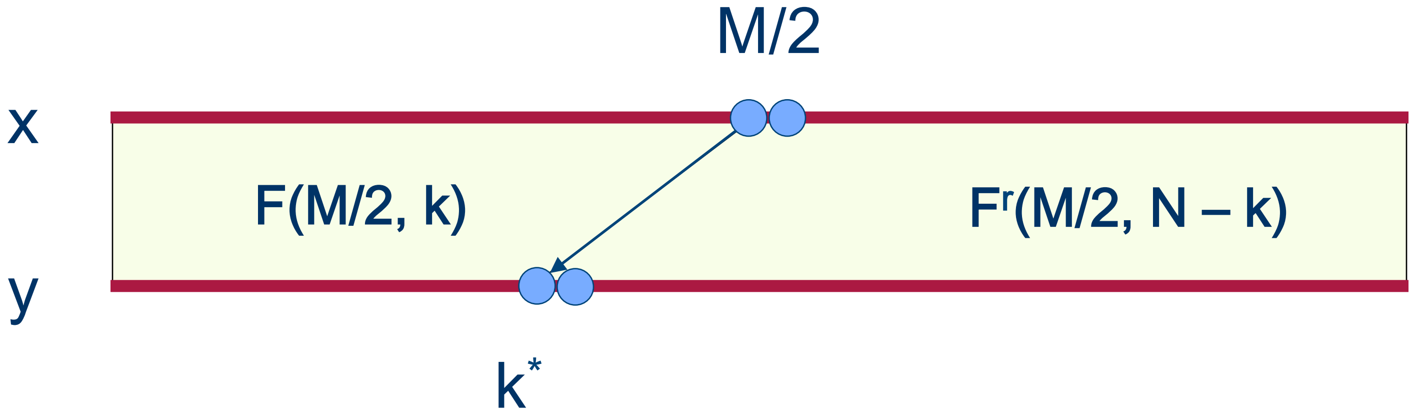
same as aligning $x_{M-i+1} \ldots x_M$ & $y_{N-j+1} \ldots y_N$

Linear-space alignment

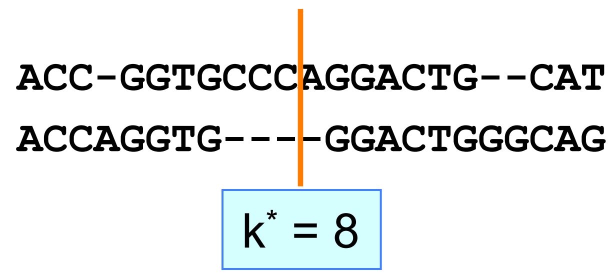


Lemma: (assume M is even)

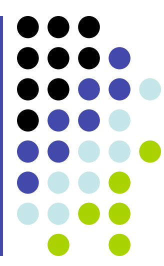
$$F(M, N) = \max_{k=0\ldots N}(F(M/2, k) + F^r(M/2, N - k))$$



Example:

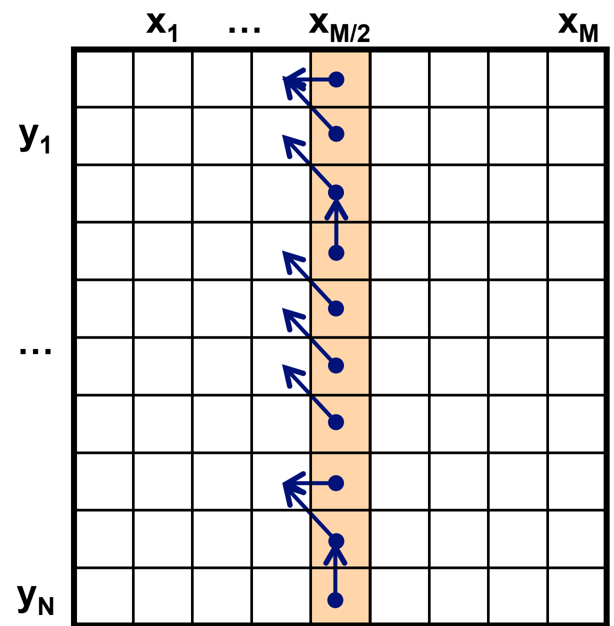


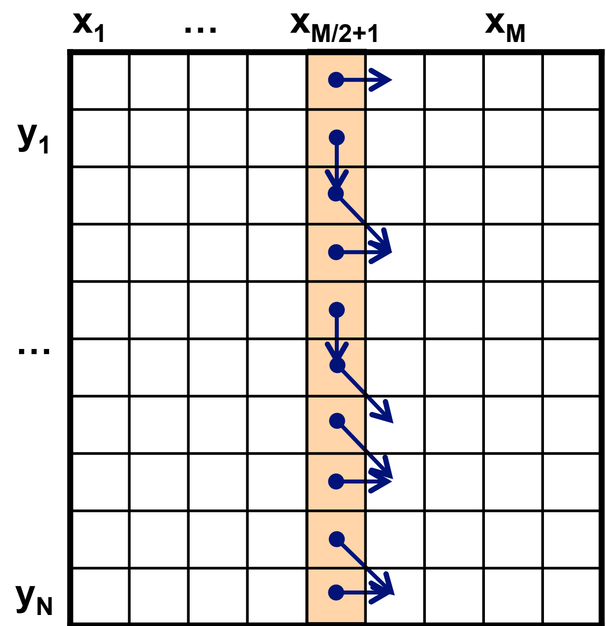
Linear-space alignment



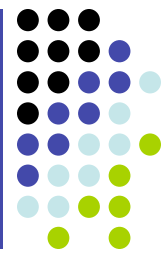
- Now, using 2 columns of space, we can compute for k = 1…M, F(M/2, k), F^r(M/2, N – k)

PLUS the backpointers

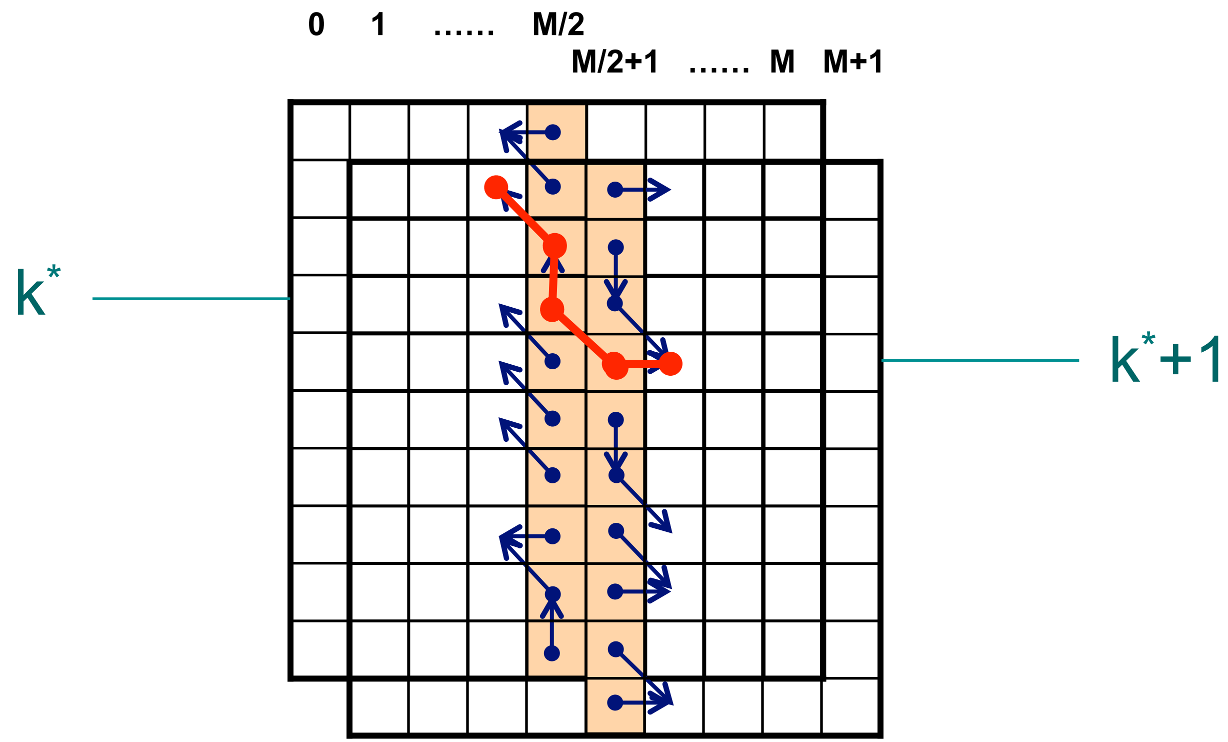




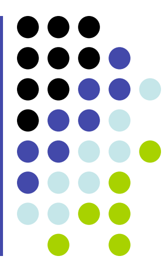
Linear-space alignment



- Now, we can find k^* maximizing $F(M/2, k) + F^r(M/2, N-k)$
- Also, we can trace the path exiting column M/2 from k^*

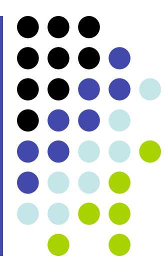


Linear-space alignment



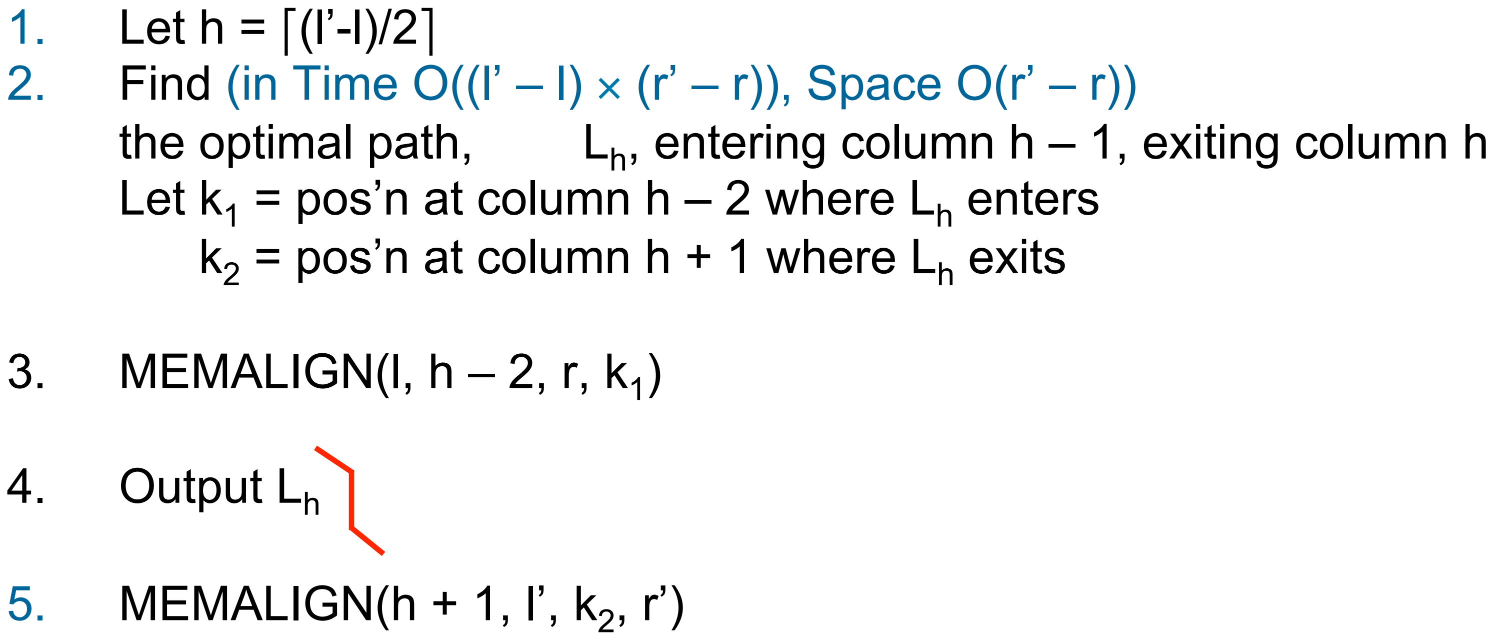
- Iterate this procedure to the left and right!

Linear-space alignment



Hirschberg's Linear-space algorithm:

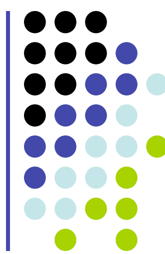
MEMALIGN(l, l', r, r'): (aligns $x_l \ldots x_{l'}$ with $y_r \ldots y_{r'}$)



1. Let $h = \lceil (l'-l)/2 \rceil$
2. Find (in Time $O((l' - l) \times (r' - r))$, Space $O(r' - r)$)
   the optimal path, L_h, entering column $h - 1$, exiting column h
   Let k_1 = pos'n at column $h - 2$ where L_h enters
   k_2 = pos'n at column $h + 1$ where L_h exits
3. MEMALIGN(l, $h - 2$, r, k_1)
4. Output L_h
5. MEMALIGN($h + 1$, l', k_2, r')

Top level call: MEMALIGN(1, M, 1, N)

Linear-space alignment



Time, Space analysis of Hirschberg's algorithm:

To compute optimal path at middle column,

For box of size $M \times N$,

Space: $2N$

Time: cMN, for some constant c

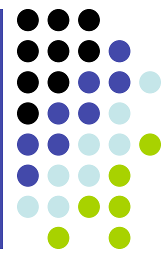
Then, left, right calls cost $c(M/2 \times k^* + M/2 \times (N - k^*)) = cMN/2$

All recursive calls cost

Total Time: $cMN + cMN/2 + cMN/4 + \ldots.. = 2cMN = O(MN)$

Total Space: $O(N)$ for computation,

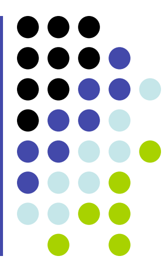
$O(N + M)$ to store the optimal alignment



Heuristic Local Alignerers

1. The basic indexing & extension technique
2. Indexing: techniques to improve sensitivity
   Pairs of Words, Patterns
3. Systems for local alignment

Indexing-based local alignment



Dictionary:

All words of length k (~10)

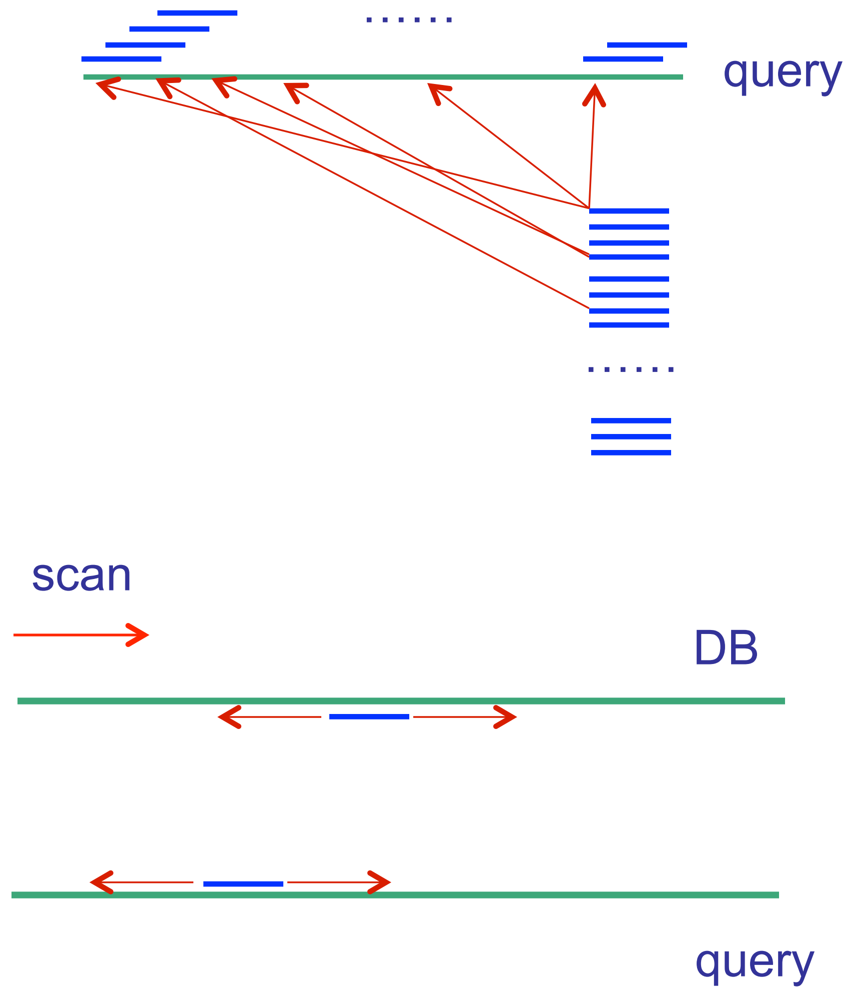
Alignment initiated between
words of alignment score ≥ T
(typically T = k)

Alignment:

Ungapped extensions until score
below statistical threshold

Output:

All local alignments with score
> statistical threshold



Indexing-based local alignment— Extensions

Gapped extensions until threshold

- Extensions with gaps until score < C below best score so far

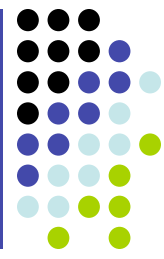
Output:

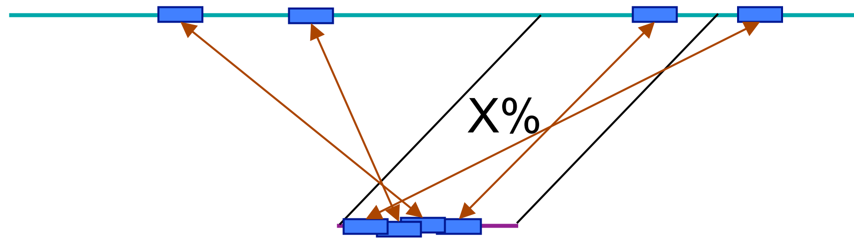
```
GTAAGGTCCAGT
GTTAGGTC-AGT
```

A C G A A G T A A G G T C C A G T

C T G A T C C T G G A T T G C G A

Sensitivity-Speed Tradeoff





	long words (k = 15)	short words (k = 7)
Sensitivity		✓
Speed	✓	

Table 3. Sensitivity and Specificity of Single Perfect Nucleotide K-mer Matches as a Search Criterion

Sens.

	7	8	9	10	11	12	13	14
A. 81%	0.974	0.915	0.833	0.726	0.607	0.486	0.373	0.314
83%	0.988	0.953	0.897	0.815	0.711	0.595	0.478	0.415
85%	0.996	0.978	0.945	0.888	0.808	0.707	0.594	0.532
87%	0.999	0.992	0.975	0.942	0.888	0.811	0.714	0.659
89%	1.000	0.998	0.991	0.976	0.946	0.897	0.824	0.782
91%	1.000	1.000	0.998	0.993	0.981	0.956	0.912	0.886
93%	1.000	1.000	1.000	0.999	0.995	0.987	0.968	0.957
95%	1.000	1.000	1.000	1.000	0.999	0.998	0.994	0.991
97%	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999

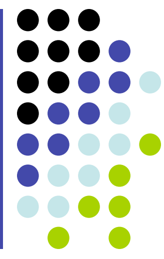
Speed

B. K	7	8	9	10	11	12	13	14
F	1.3e+07	2.9e+06	635783	143051	32512	7451	1719	399

(*A*) Columns are for K sizes of 7–14. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated from equation 3 assuming a homologous region of 100 bases. The larger the value of K, the fewer homologies are detected.
(*B*) K represents the size of the perfect match. F shows how many perfect matches of this size expected to occur by chance according to equation 4 in a genome of 3 billion bases using a query of 500 bases.

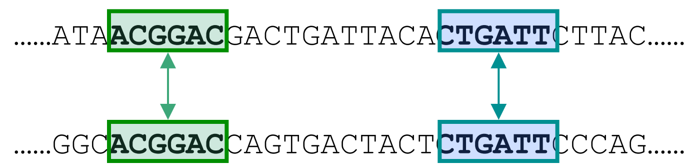
Kent WJ, Genome Research 2002

Sensitivity-Speed Tradeoff

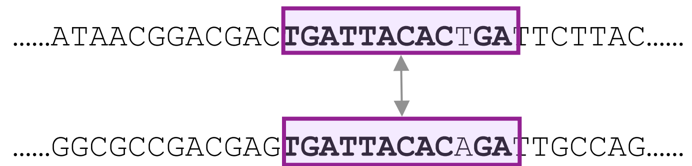


Methods to improve sensitivity/speed

1. Using pairs of words



2. Using inexact words



3. Patterns—non consecutive positions

Measured improvement

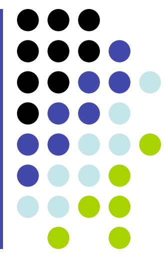


Table 7. Sensitivity and Specificity of Multiple (2 and 3) Perfect Nucleotide K-mer Matches as a Search Criterion

	2,8	2,9	2,10	2,11	2,12	3,8	3,9	3,10	3,11	3,12
A. 81%	0.681	0.508	0.348	0.220	0.129	0.389	0.221	0.112	0.051	0.021
83%	0.790	0.638	0.475	0.326	0.208	0.529	0.339	0.193	0.099	0.045
85%	0.879	0.762	0.615	0.460	0.318	0.676	0.487	0.313	0.180	0.093
87%	0.942	0.866	0.752	0.611	0.461	0.809	0.649	0.470	0.305	0.177
89%	0.978	0.940	0.868	0.761	0.625	0.910	0.801	0.648	0.476	0.314
91%	0.994	0.980	0.947	0.884	0.787	0.969	0.914	0.815	0.673	0.505
93%	0.999	0.996	0.986	0.962	0.912	0.993	0.976	0.933	0.851	0.722
95%	1.000	1.000	0.998	0.993	0.979	0.999	0.997	0.987	0.961	0.902
97%	1.000	1.000	1.000	1.000	0.999	1.000	1.000	0.999	0.997	0.987
B. N,K	2,8	2,9	2,10	2,11	2,12	3,8	3,9	3,10	3,11	3,12
F	524	27	1.4	0.1	0.0	0.1	0.0	0.0	0.0	0.0

(*A*) Columns are for N sizes of 2 and 3 and K sizes of 8–12. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated by equation 10. (*B*) N and K represent the number and size of the near-perfect matches, respectively. F shows how many perfect clustered matches expected to occur by chance according to equation 14 in a translated genome of 3 billion bases using a query of 167 amino acids.

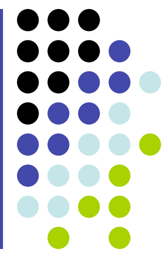
Table 5. Sensitivity and Specificity of Single Near-Perfect (One Mismatch Allowed) Nucleotide K-mer Matches as a Search Criterion

	12	13	14	15	16	17	18	19	20	21	22
A. 81%	0.945	0.880	0.831	0.721	0.657	0.526	0.465	0.408	0.356	0.255	0.218
83%	0.975	0.936	0.904	0.820	0.770	0.649	0.591	0.535	0.480	0.361	0.318
85%	0.991	0.971	0.954	0.900	0.865	0.767	0.719	0.669	0.619	0.490	0.445
87%	0.997	0.990	0.983	0.954	0.935	0.867	0.833	0.796	0.757	0.634	0.591
89%	1.000	0.997	0.995	0.984	0.976	0.939	0.920	0.897	0.872	0.775	0.741
91%	1.000	1.000	0.999	0.996	0.994	0.979	0.971	0.962	0.950	0.890	0.869
93%	1.000	1.000	1.000	0.999	0.999	0.996	0.994	0.991	0.988	0.963	0.954
95%	1.000	1.000	1.000	1.000	1.000	1.000	0.999	0.999	0.999	0.994	0.992
97%	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
B. K	12	13	14	15	16	17	18	19	20	21	22
F	275671	68775	17163	4284	1070	267	67	17	4.2	1.0	0.3

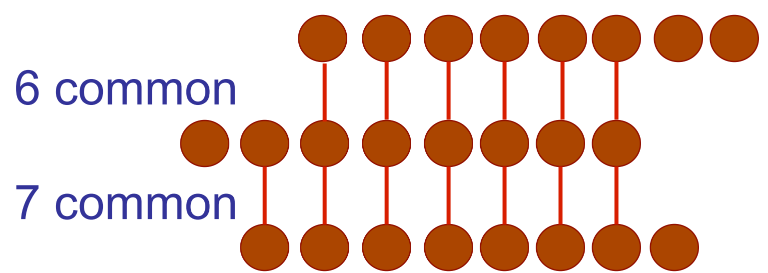
(*A*) Columns are for K sizes of 12–22. Rows represent various percentage identities between the homologous sequences. The table entries show the fraction of homologies detected as calculated by equation 6 assuming a homologous region of 100 bases. (*B*) K represents the size of the near-perfect match. F shows how many perfect matches of this size expected to occur by chance accordi… billion bases using a query of 500 bases.

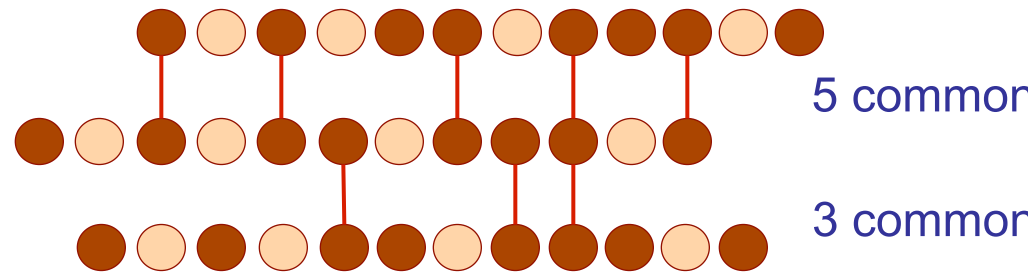
Kent WJ, Genome Research 2002

Non-consecutive words—Patterns



Patterns increase the likelihood of *at least one* match within a long conserved region

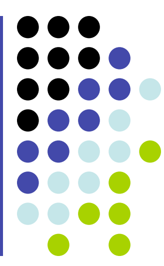




On a 100-long 70% conserved region:

	Consecutive	Non-consecutive
Expected # hits:	1.07	0.97
Prob[at least one hit]:	0.30	0.47

Advantage of Patterns



11 positions

11 positions

10 positions

110100110010101111

11111111111

1111111111

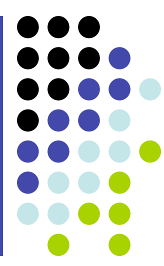
sensitivity

similarity

0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Multiple patterns



```
TTTGATTTACCACCAGAT
 T G TT CAC G
 T G T  C CAG
 TTGATT  A  G
```

How long does it take to search the query?

Seed	Pattern	Pr[detection]	Alignments Found	Time (s)
π_c	$\{0,1,2,3,4,5,6,7,8,9,10\}$	0.600	66419	15802
π_{c10}	$\{0,1,2,3,4,5,6,7,8,9\}$	0.707	73539	24129
π_{ph}	$\{0,1,2,4,7,9,12,13,15,16,17\}$	0.691	75518	16717
π_{N_0}	$\{0,1,2,4,7,8,11,13,16,17,18\}$	0.683	75231	16225
π_{N_5}	$\{0,1,2,3,5,6,7,10,12,13,14\}$	0.709	75547	16817
$\pi_1+\pi_2$	$\{0,1,2,4,5,9,14,16,17,18,19,20\}+$ $\{0,1,2,3,4,6,7,8,10,11,12,13\}$	0.744	77211	22033

Buhler et al. RECOMB 2003
Sun & Buhler RECOMB 2004