



Definition of Minimum Edit Distance



How similar are two strings?

- Spell correction

- The user typed “graffe”

Which is closest?

- graf
 - graft
 - grail
 - giraffe

- Computational Biology

- Align two sequences of nucleotides

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC  
TAGCTATCACGACCGCGGTTCGATTGCCCCGAC
```

- Resulting alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---  
TAG-CTATCAC--GACCGC--GGTCGATTGCCCCGAC
```

- Also for Machine Translation, Information Extraction, Speech Recognition



Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
 - Insertion
 - Deletion
 - Substitution
- Needed to transform one into the other



Minimum Edit Distance

- Two strings and their **alignment**:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N



Minimum Edit Distance

I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s i s

- If each operation has cost of 1
 - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
 - Distance between them is 8



Alignment in Computational Biology

- Given a sequence of bases

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTGCCCCGAC

- An alignment:

–AGGCTATCACCTGACCTCCAGGCCGA–TGCCC–
TAG–CTATCAC–GACCGC–GGTCGATTGCCCCGAC

- Given two sequences, align each letter to a letter or gap



Other uses of Edit Distance in NLP

- Evaluating Machine Translation and speech recognition

R Spokesman confirms senior government adviser was shot

H Spokesman said the senior adviser was shot dead

S

I

D

I

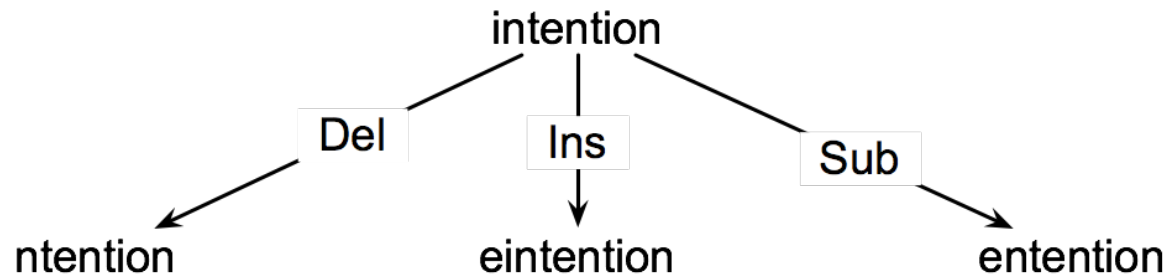
- Named Entity Extraction and Entity Coreference

- **IBM Inc.** announced today
- **IBM** profits
- **Stanford President John Hennessy** announced yesterday
- for **Stanford University President John Hennessy**



How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
 - **Initial state:** the word we're transforming
 - **Operators:** insert, delete, substitute
 - **Goal state:** the word we're trying to get to
 - **Path cost:** what we want to minimize: the number of edits





Minimum Edit as Search

- But the space of all edit sequences is huge!
 - We can't afford to navigate naïvely
 - Lots of distinct paths wind up at the same state.
 - We don't have to keep track of all of them
 - Just the shortest path to each of those revisited states.



Defining Min Edit Distance

- For two strings
 - X of length n
 - Y of length m
- We define $D(i,j)$
 - the edit distance between $X[1..i]$ and $Y[1..j]$
 - i.e., the first i characters of X and the first j characters of Y
 - The edit distance between X and Y is thus $D(n,m)$



Minimum Edit Distance

Computing Minimum Edit Distance



Dynamic Programming for Minimum Edit Distance

- **Dynamic programming:** A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems.
- Bottom-up
 - We compute $D(i,j)$ for small i,j
 - And compute larger $D(i,j)$ based on previously computed smaller values
 - i.e., compute $D(i,j)$ for all i ($0 < i < n$) and j ($0 < j < m$)



Defining Min Edit Distance (Levenshtein)

- Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

$D(N, M)$ is distance



The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



Backtrace for Computing Alignments



Computing alignments

- Edit distance isn't sufficient
 - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,
 - Trace back the path from the upper right corner to read off the alignment



Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N



MinEdit with Backtrace

n	9	↓ 8	↙↖↓ 9	↙↖↓ 10	↙↖↓ 11	↙↖↓ 12	↓ 11	↓ 10	↓ 9	↙ 8	
o	8	↓ 7	↙↖↓ 8	↙↖↓ 9	↙↖↓ 10	↙↖↓ 11	↓ 10	↓ 9	↙ 8	← 9	
i	7	↓ 6	↙↖↓ 7	↙↖↓ 8	↙↖↓ 9	↙↖↓ 10	↓ 9	↙ 8	← 9	← 10	
t	6	↓ 5	↙↖↓ 6	↙↖↓ 7	↙↖↓ 8	↙↖↓ 9	↙ 8	← 9	← 10	↖↓ 11	
n	5	↓ 4	↙↖↓ 5	↙↖↓ 6	↙↖↓ 7	↙↖↓ 8	↙↖↓ 9	↙↖↓ 10	↙↖↓ 11	↙↓ 10	
e	4	↙ 3	← 4	↙↖ 5	← 6	← 7	↖↓ 8	↙↖↓ 9	↙↖↓ 10	↓ 9	
t	3	↙↖↓ 4	↙↖↓ 5	↙↖↓ 6	↙↖↓ 7	↙↖↓ 8	↙ 7	↖↓ 8	↙↖↓ 9	↓ 8	
n	2	↙↖↓ 3	↙↖↓ 4	↙↖↓ 5	↙↖↓ 6	↙↖↓ 7	↙↖↓ 8	↓ 7	↙↖↓ 8	↙ 7	
i	1	↙↖↓ 2	↙↖↓ 3	↙↖↓ 4	↙↖↓ 5	↙↖↓ 6	↙↖↓ 7	↙ 6	← 7	← 8	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	



Adding Backtrace to Minimum Edit Distance

- Base conditions:

$$D(i, 0) = i$$

$$D(0, j) = j$$

Termination:

$$D(N, M) \text{ is distance}$$

- Recurrence Relation:

For each $i = 1 \dots M$

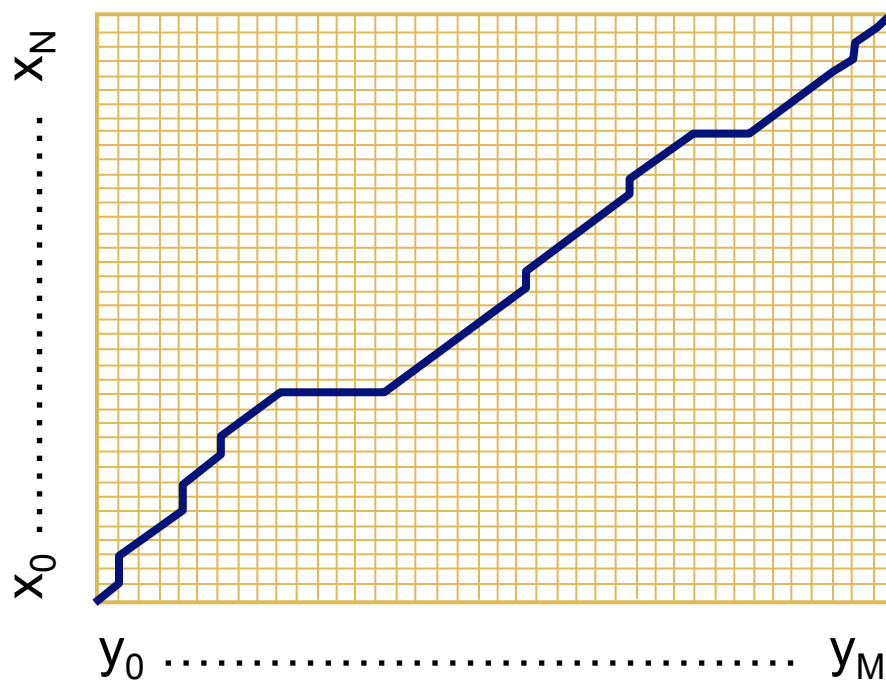
For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$



The Distance Matrix



Every non-decreasing path

from $(0,0)$ to (M, N)

corresponds to
an alignment
of the two sequences

An optimal alignment is composed
of optimal subalignments



Result of Backtrace

- Two strings and their **alignment**:

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N



Performance

- Time:
 $O(nm)$
- Space:
 $O(nm)$
- Backtrace
 $O(n+m)$

[illegible][illegible]

Dan Jurafsky



Weighted Edit Distance

- Why would we add weights to the computation?
 - Spell Correction: some letters are more likely to be mistyped than others
 - Biology: certain kinds of deletions or insertions are more likely than others



Confusion matrix for spelling errors

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

Dan Jurafsky





Weighted Min Edit Distance

- Initialization:

$$D(0,0) = 0$$

$$D(i,0) = D(i-1,0) + \text{del}[x(i)]; \quad 1 < i \leq N$$

$$D(0,j) = D(0,j-1) + \text{ins}[y(j)]; \quad 1 < j \leq M$$

- Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) + \text{del}[x(i)] \\ D(i,j-1) + \text{ins}[y(j)] \\ D(i-1,j-1) + \text{sub}[x(i),y(j)] \end{cases}$$

- Termination:

$D(N,M)$ is distance

Dan Jurafsky



Where did the name, dynamic programming, come from?

...The 1950s were not good years for mathematical research. [the] Secretary of Defense ...had a pathological fear and hatred of the word, research...

I decided therefore to use the word, “**programming**”.

I wanted to get across the idea that this was dynamic, this was multistage... I thought, let's ... take a word that has an absolutely precise meaning, namely **dynamic**... it's impossible to use the word, **dynamic**, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible.

Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to.”

Richard Bellman, “Eye of the Hurricane: an autobiography” 1984.



Sequence Alignment

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGGTCGATTGCCCCGAC

–AGGCTATCACCTGACCTCCAGGCCGA–TGCCC–
TAG–CTATCAC–GACCGC–GGTCGATTGCCCCGAC



Why sequence alignment?

- Comparing genes or regions from different species
 - to find important regions
 - determine function
 - uncover evolutionary forces
- Assembling fragments to sequence DNA
- Compare individuals to looking for mutations



Alignments in two fields

- In Natural Language Processing
 - We generally talk about **distance** (minimized)
 - And **weights**
- In Computational Biology
 - We generally talk about **similarity** (maximized)
 - And **scores**



The Needleman-Wunsch Algorithm

- Initialization:

$$D(i, 0) = -i * d$$

$$D(0, j) = -j * d$$

- Recurrence Relation:

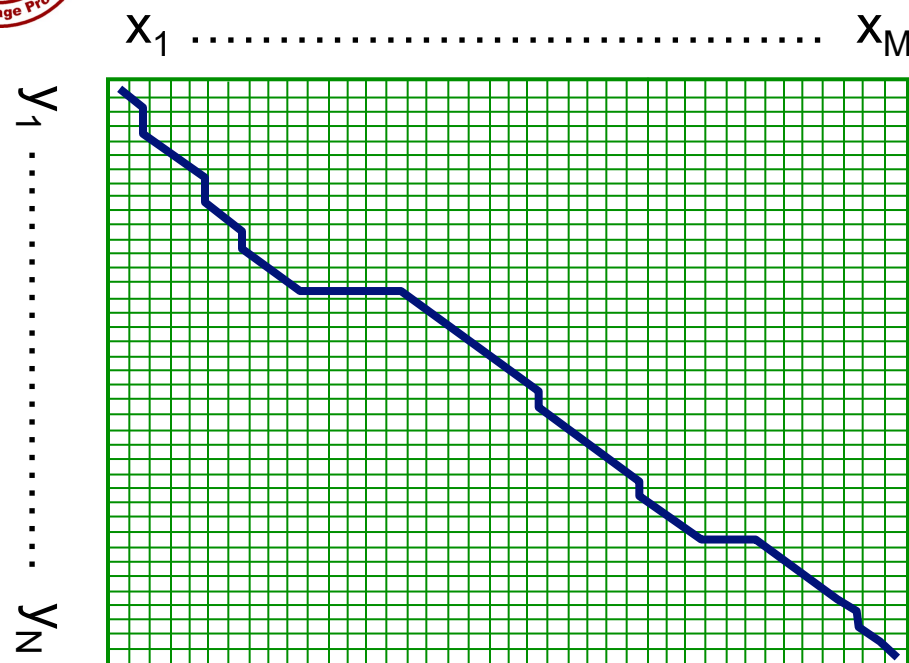
$$D(i, j) = \min \begin{cases} D(i-1, j) & - d \\ D(i, j-1) & - d \\ D(i-1, j-1) & + s[x(i), y(j)] \end{cases}$$

- Termination:

$D(N, M)$ is distance



The Needleman-Wunsch Matrix



(Note that the origin is at the upper left.)



A variant of the basic algorithm:

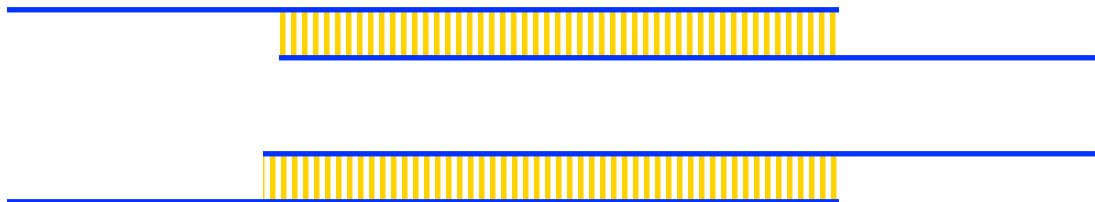
- Maybe it is OK to have an unlimited # of gaps in the beginning and end:

-----CTATCACCTGACCTCCAGGCCGATGCCCCCTTCCGGC
GCGAGTTCATCTATCAC--GACCGC--GGTCG-----

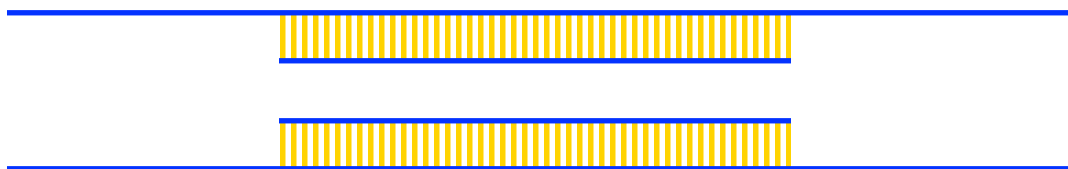
- If so, we don't want to penalize gaps at the ends



Different types of overlaps



Example:
2 overlapping "*reads*" from a
sequencing project



Example:
Search for a mouse gene
within a human chromosome



The Overlap Detection variant



Changes:

1. Initialization

For all i, j ,

$$F(i, 0) = 0$$

$$F(0, j) = 0$$

2. Termination

$$F_{\text{OPT}} = \max \left\{ \begin{array}{l} \max_i F(i, N) \\ \max_j F(M, j) \end{array} \right.$$



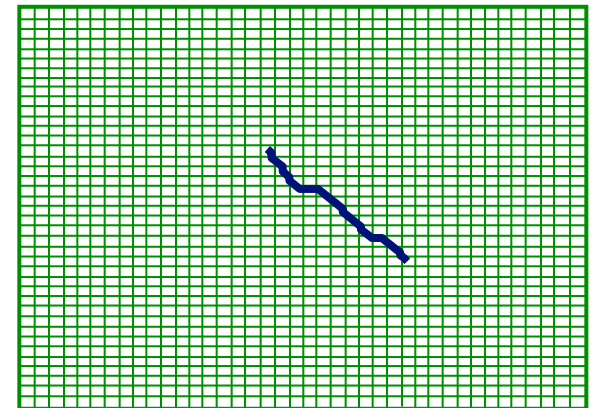
The Local Alignment Problem

Given two strings

$$x = x_1 \dots x_M,$$

$$y = y_1 \dots y_N$$

Find substrings x' , y' whose similarity
(optimal global alignment value)
is maximum



$x = \text{aaaacc}\text{cccggg}\text{gtta}$

$y = \text{ttcccggg}\text{aaccaacc}$



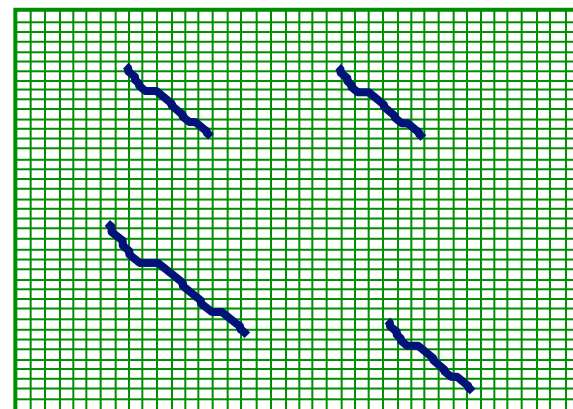
The Smith-Waterman algorithm

Idea: Ignore badly aligning regions

Modifications to Needleman-Wunsch:

Initialization: $F(0, j) = 0$
 $F(i, 0) = 0$

Iteration: $F(i, j) = \max \begin{cases} 0 \\ F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + s(x_i, y_j) \end{cases}$





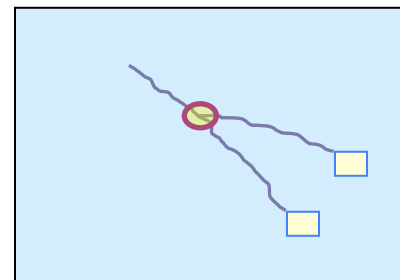
The Smith-Waterman algorithm

Termination:

1. If we want the **best** local alignment...

$$F_{OPT} = \max_{i,j} F(i, j)$$

Find F_{OPT} and trace back



2. If we want **all** local alignments **scoring** $> t$

?? For all i, j find $F(i, j) > t$, and trace back?

Complicated by overlapping local alignments

Slide from Serafim Batzoglou



Local alignment example

X = ATCAT

Y = ATTATC

Let:

$m = 1$ (1 point for match)

$d = 1$ (-1 point for del/ins/sub)

		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0						
T	0						
C	0						
A	0						
T	0						



Local alignment example

X = ATCAT

Y = ATTATC

		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0	1	0	0	1	0	0
T	0	0	2	1	0	2	0
C	0	0	1	1	0	1	3
A	0	1	0	0	2	1	2
T	0	0	2	0	1	3	2



Local alignment example

X = **ATCAT**

Y = **ATTAT**C

		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0	1	0	0	1	0	0
T	0	0	2	1	0	2	0
C	0	0	1	1	0	1	3
A	0	1	0	0	2	1	2
T	0	0	2	0	1	3	2



Local alignment example

X = **ATC**AT

Y = ATT**ATC**

		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0	1	0	0	1	0	0
T	0	0	2	1	0	2	0
C	0	0	1	1	0	1	3
A	0	1	0	0	2	1	2
T	0	0	2	0	1	3	2