SIT330-770: Natural Language Processing

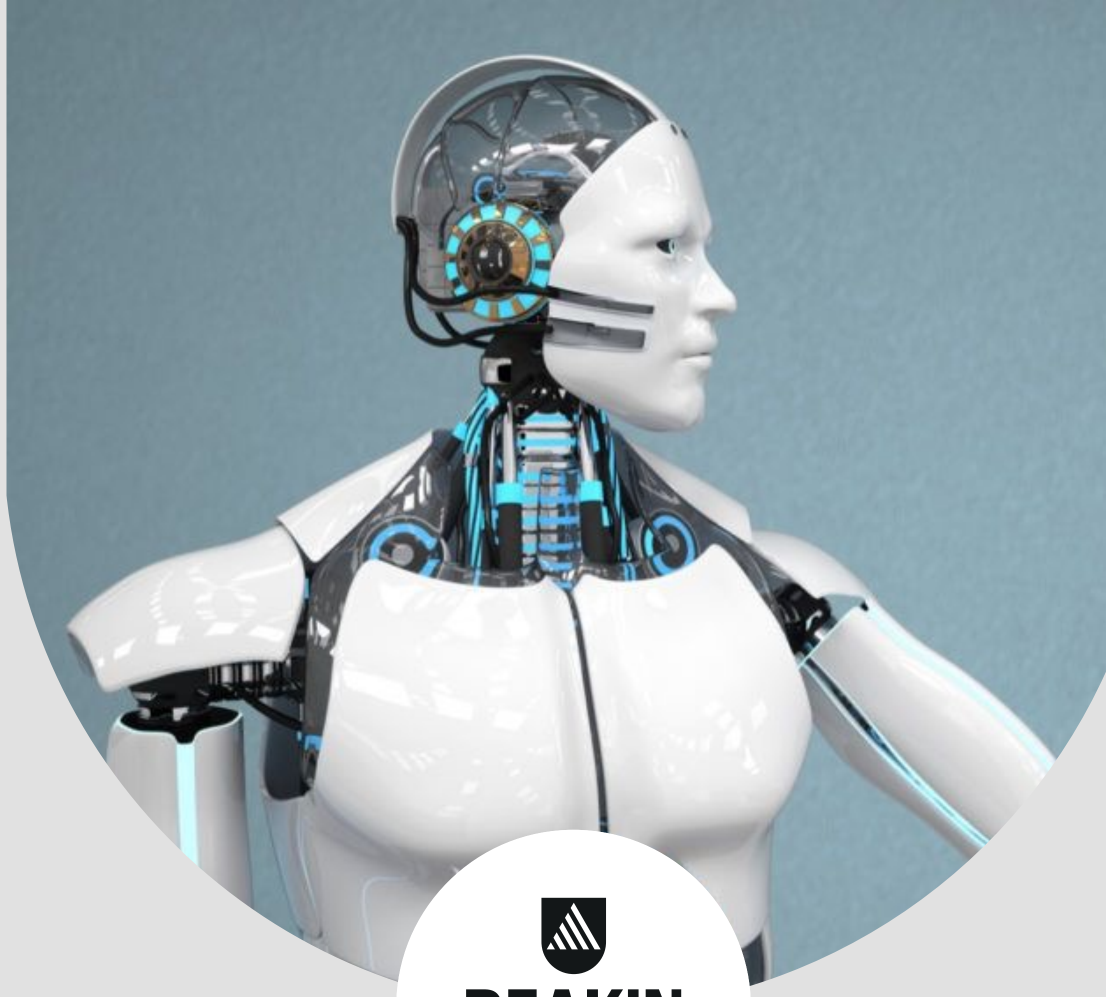
Week 3 - Text processing

Regular Expressions, Text Normalization, Edit Distance

Dr. Mohamed Reda Bouadjenek

School of Information Technology, Faculty of Sci Eng & Built Env

reda.bouadjenek@deakin.edu.au



DEAKIN UNIVERSITY

SIT330-770: Natural Language Processing

Week 3.1 - Regular Expressions

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

DEAKIN UNIVERSITY

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Regular Expressions: Negation in Disjunction

- Negations `[^Ss]`
  - Carat means negation only when first in []

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`I have no exquisite reason”`
`[^e^]`	Neither e nor ^	`Look here`
`a^b`	The pattern a carat b	`Look up a^b now`

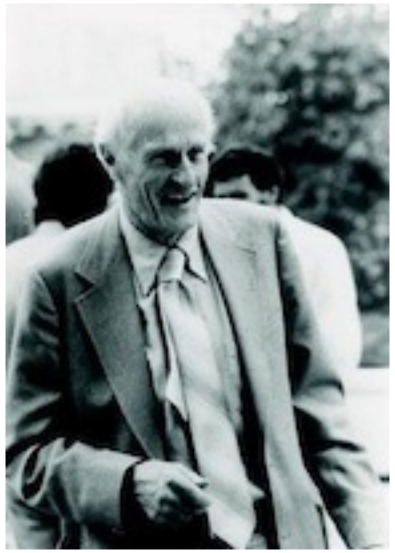
Regular Expressions: More Disjunction

- Woodchuck is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	`woodchuck`
`yours|mine`	`yours`
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	`Woodchuck`

Regular Expressions: ? *+ .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n		begin begun begun beg3n


Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ $

Pattern	Matches
`^[A-Z]`	<u>P</u>alo Alto
`^[^A-Za-z]`	<u>1</u> <u>“</u>Hello”
`\.$`	The end<u>.</u>
`.$`	The end<u>?</u> The end<u>!</u>

Example

- Find me all instances of the word “the” in a text.

  `the`

  Misses capitalized examples

  `[tT]he`

  Incorrectly returns `other` or `theology`

  `[^a-zA-Z][tT]he[^a-zA-Z]`

Errors

- The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other)

False positives (Type I errors)

2. Not matching things that we should have matched (The)

False negatives (Type II errors)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing
- For hard tasks, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

SIT330-770: Natural Language Processing

Week 3.2- More Regular Expressions: Substitutions and ELIZA

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Substitutions

- Substitution in Python and UNIX commands:

```
s/regexp1/pattern/
```

e.g.:

```
s/colour/color/
```

Capture Groups

- Say we want to put angles around all numbers:

the 35 boxes → *the <35> boxes*

- Use parens () to "capture" a pattern into a numbered register (1, 2, 3…)
- Use \1 to refer to the contents of the register

```
s/([0-9]+)/<\1>/
```

Capture groups: multiple registers

- `/the (.*)er they (.*), the \1er we \2/`
- Matches

  the faster they ran, the faster we ran

- *But not*

  the faster they ran, the faster we ate

But suppose we don't want to capture?

- Parentheses have a double function: grouping terms, and capturing
- Non-capturing groups: add a ?: after paren:
- E.g.: `/(?:some|a few) (people|cats) like some \1/`
  - matches
    - `some cats like some cats`
  - but not
    - `some cats like some some`

Lookahead assertions

- `(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**
- `(?! pattern)` true if a pattern does not match
- How to match, at the beginning of a line, any single word that doesn't start with “Volcano”:
  - `/^(?!Volcano)[A-Za-z]+/`

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist
  - Joseph Weizenbaum, 1966.
- Uses pattern matching to match, e.g.,:
  - `“I need X”`

and translates them into, e.g.

  - `“What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike.

IN WHAT WAY

They're always bugging us about something or other. CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

- s/.* I’M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
- s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
- s/.* all .*/IN WHAT WAY?/
- s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/

SIT330-770: Natural Language Processing

Week 3.3 - Words and Corpora

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

How many words in a sentence?

- "I do uh main- mainly business data processing"
  - Fragments, filled pauses
- "Seuss's cat in the hat is different from other cats!"
  - **Lemma**: same stem, part of speech, rough word sense
    - cat and cats = same lemma
  - **Wordform**: the full inflected surface form
    - cat and cats = different wordforms

How many words in a sentence?

they lay back on the San Francisco grass and looked at the stars and their

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 15 tokens (or 14)
  - 13 types (or 12) (or 11?)

How many words in a corpus?

N = number of tokens

V = vocabulary = set of types, **|*V*|** is size of vocabulary

Heaps Law = Herdan's Law = $|V| = kN^{\beta}$, where often $0.67 < \beta <$0.75

i.e., vocabulary size grows with > square root of the number of word tokens

	Tokens = N	**Types = |V|**
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Corpora

Words don't appear out of nowhere!

A text is produced by

- a specific writer(s),
- at a specific time,
- in a specific variety,
- of a specific language,
- for a specific function.

Corpora vary along dimension like

- **Language**: 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAE Twitter posts might include forms like "*iont*" *(I don't)*
- **Code switching**, e.g., Spanish/English, Hindi/English:

  S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)

  [For the first time I get to see @username actually being hateful! it was beautiful:)]

  H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe

  ["he was and will remain a friend ... don't worry ... but have faith"]
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, SES

Corpus datasheets

Gebru et al (2020), Bender and Friedman (2018)

Motivation:

- Why was the corpus collected?
- By whom?
- Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled? Was there consent? Pre-processing?

- **+Annotation process, language variety, demographics, etc.**

SIT330-770: Natural Language Processing

Week 3.4 - Word tokenization

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Text Normalization

- Every NLP task requires text normalization:
  1. Tokenizing (segmenting) words
  2. Normalizing word formats
  3. Segmenting sentences

Space-based tokenization

- A very simple way to tokenize
  - For languages that use space characters between words
    - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
  - Segment off a token between instances of spaces
- Unix tools for space-based tokenization
  - The "tr" command
  - Inspired by Ken Church's UNIX for Poets
  - Given a text file, output the word tokens and their frequencies

Simple Tokenization in UNIX

- Given a text file, output the word tokens and their frequencies

```
tr -sc ’A-Za-z’ ’\n’ < shakes.txt
                                  | sort
                                  | uniq −c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

The first step: tokenizing

```
tr -sc ’A-Za-z’ ’\n’ < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

```
A
A
A
A
A
A
A
A
A
...
```

More counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954  d
```

What happened here?

Issues in Tokenization

- Can't just blindly remove punctuation:
  - m.p.h., Ph.D., AT&T, cap'n
  - prices ($45.55)
  - dates (01/02/06)
  - URLs (http://www.stanford.edu)
  - hashtags (#nlproc)
  - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
  - "are" in we're, French "je" in j'ai, "le" in l'honneur
- When should multiword expressions (MWE) be words?
  - New York, rock 'n' roll

Tokenization in NLTK

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Tokenization in NLTK

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r''' (?x) # set flag to allow verbose regexps
...     ([A-Z]\.)+ # abbreviations, e.g. U.S.A.
```

Tokenization in languages without spaces

- Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

- How do we decide where the token boundaries should be?

Word tokenization in Chinese

- Chinese words are composed of characters called "**hanzi**" (or sometimes just "**zi**")
- Each one represents a meaning unit called a morpheme.
- Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

- 3 words?

姚明 进入 总决赛

YaoMing reaches finals

- 5 words?

姚 明 进入 总 决赛

Yao Ming reaches overall finals

- 7 characters? (don't use words at all):

姚 明 进 入 总 决 赛

Yao Ming enter enter overall decision game

Word tokenization / segmentation

- So, in Chinese it's common to just treat each character (zi) as a token.
  - So, the **segmentation** step is very simple
- In other languages (like Thai and Japanese), more complex word segmentation is required.
  - The standard algorithms are neural sequence models trained by supervised machine learning.

SIT330-770: Natural Language Processing

Week 3.5 - Byte Pair Encoding

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

Another option for text tokenization

- Instead of
  - white-space segmentation
  - single-character segmentation
- **Use the data** to tell us how to tokenize.
- **Subword tokenization** (because tokens can be parts of words as well as whole words)

Subword tokenization

- Three common algorithms:
  - **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
  - **Unigram language modeling tokenization** (Kudo, 2018)
  - **WordPiece** (Schuster and Nakajima, 2012)
- All have 2 parts:
  - A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

- Let vocabulary be the set of all individual characters

  = {A, B, C, D,…, a, b, c, d….}

- Repeat:
  - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- Until *k* merges have been done.

BPE token learner algorithm

```
function BYTE-PAIR ENCODING(strings $C$, number of merges $k$) returns vocab $V$

    $V \leftarrow$ all unique characters in $C$          # initial set of tokens is characters
    for $i = 1$ to $k$ do                                  # merge tokens til $k$ times
        $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in $C$
        $t_{NEW} \leftarrow t_L + t_R$                     # make new token by concatenating
        $V \leftarrow V + t_{NEW}$                         # update the vocabulary
        Replace each occurrence of $t_L$, $t_R$ in $C$ with $t_{NEW}$     # and update the corpus
    return $V$
```

Byte Pair Encoding (BPE) Addendum

- Most subword algorithms are run inside space-separated tokens.
- So we commonly first add a special end-of-word symbol '__' before space in training corpus
- Next, separate into letters.

BPE token learner

- Original (very fascinating🙄) corpus:
- low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new
- Add end-of-word tokens, resulting in this vocabulary:

```
Corpus                      Vocabulary
5 l o w _                   _, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _
```

BPE

```
Corpus                              Vocabulary
5 l o w _                           _, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _
```

Merge [e r] to [er]

```
Corpus                              Vocabulary
5 l o w _                           _, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _
```

```
Corpus                         Vocabulary
5 l o w _                      _, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _
```

Merge [er _] to [er_]

```
Corpus                         Vocabulary
5 l o w _                      _, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _
```

BPE

Corpus	Vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _	
6 n e w er_	
3 w i d er_	
2 n e w _	

Merge [n e] to [ne]

Corpus	Vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, **ne**
2 l o w e s t _	
6 **ne** w er_	
3 w i d er_	
2 **ne** w _	

BPE

- The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, 1, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, 1, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, 1, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, 1, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, 1, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every `[e r]` to `[er]`, then merge `[er _]` to `[er_]`, etc.

- Result:
  - Test set "`n e w e r _`" would be tokenized as a full word
  - Test set "`l o w e r _`" would be two tokens: "`low er_`"

Properties of BPE tokens

- Usually include frequent words and frequent subwords
  - Which are often morphemes like *-est* or *–er*
- A **morpheme** is the smallest meaning-bearing unit of a language
  - *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

SIT330-770: Natural Language Processing

Week 3.6 - Word Normalization and other issues

Dr. Mohamed Reda Bouadjenek

School of Information Technology, Faculty of Sci Eng & Built Env

Word Normalization

- Putting words/tokens in a standard format
    - U.S.A. or USA
    - uhhuh or uh-huh
    - Fed or fed
    - am, is, be, are

Case folding

- Applications like IR: reduce all letters to lower case
  - Since users tend to use lower case
  - Possible exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***
- For sentiment analysis, MT, Information extraction
  - Case is helpful (***US*** versus ***us*** is important)

Lemmatization

- Represent all words as their lemma, their shared root = dictionary headword form:
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - Spanish quiero ('I want'), quieres ('you want')

    $\rightarrow$ querer 'want'

  - *He is reading detective stories*

    $\rightarrow$ *He be read detective story*

Lemmatization is done by Morphological Parsing

- Morphemes:
  - The small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Parts that adhere to stems, often with grammatical functions
- Morphological Parsers:
  - Parse *cats* into two morphemes *cat* and *s*
  - Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

Stemming

- Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

→

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note.

Porter Stemmer

- Based on a series of rewrite rules run in series
  - A cascade, in which output of each pass fed to next pass
- Some sample rules:

ATIONAL	$\longrightarrow$	ATE (e.g., ATIONAL$\longrightarrow$ATE)
ING	$\longrightarrow$	ϵ if stem contains vowel (e.g., motoring $\longrightarrow$ motor)
SSES	$\longrightarrow$	SS (e.g., grasses $\longrightarrow$ grass)

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
- Uygarlastiramadiklarimizdanmissinizcasina
- \`(behaving) as if you are among those whom we could not civilize'
- Uygar \`civilized' + las \`become'

  + tir \`cause' + ama \`not able'

  + dik \`past' + lar 'plural'

  + imiz 'p1pl' + dan 'abl'

  + mis 'past' + siniz '2pl' + casina 'as if'

Sentence Segmentation

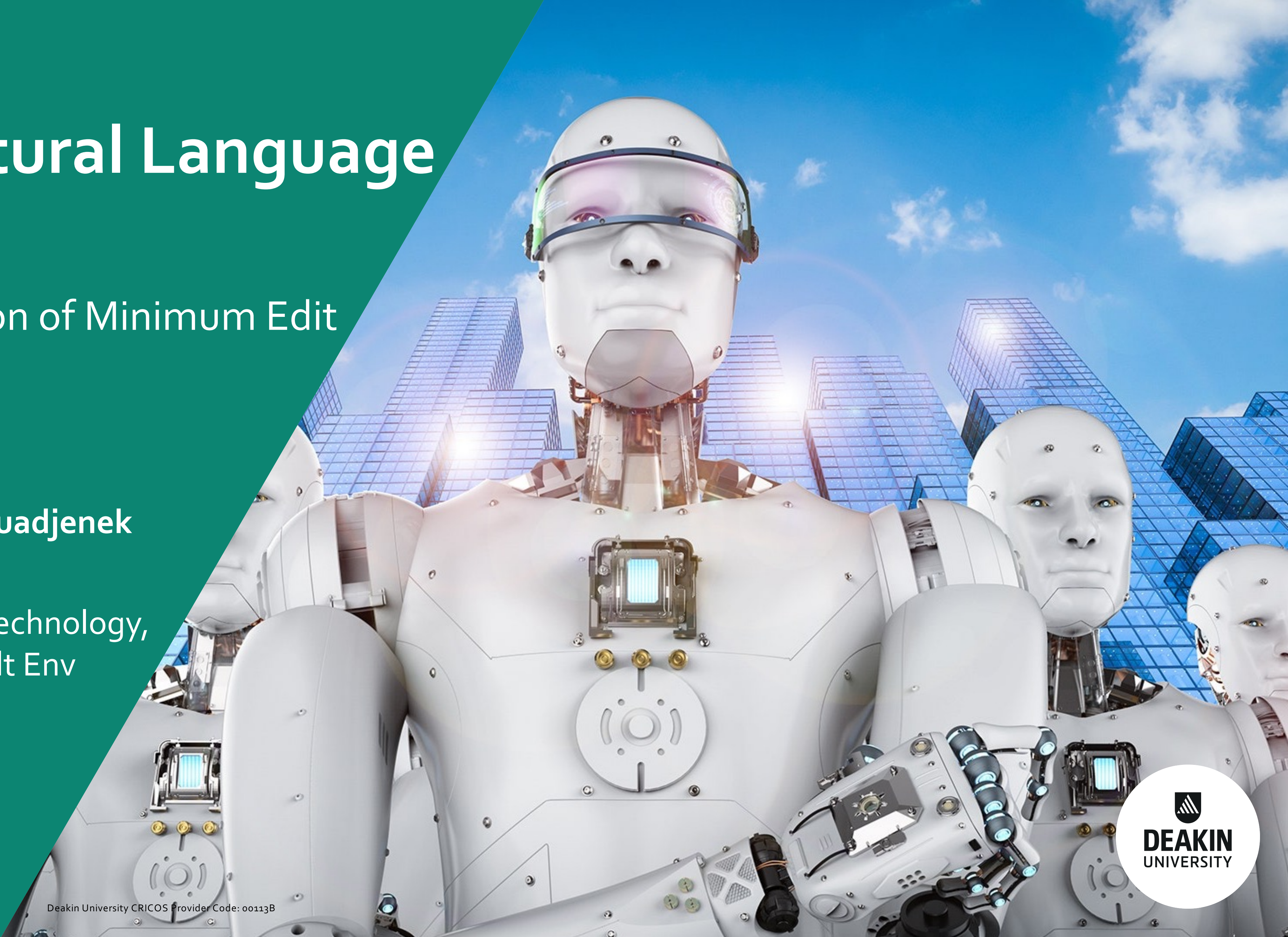
- !, ? mostly unambiguous but **period** "." is very ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Common algorithm: Tokenize first: use rules or ML to classify a period as either (a) part of the word or (b) a sentence-boundary.
  - An abbreviation dictionary can help
- Sentence segmentation can then often be done by rules based on this tokenization.

SIT770: Natural Language Processing

Week 3.7 - Definition of Minimum Edit Distance

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env



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
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

  - Resulting alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

- Also for Machine Translation, Information Extraction, Speech Recognition

Edit Distance

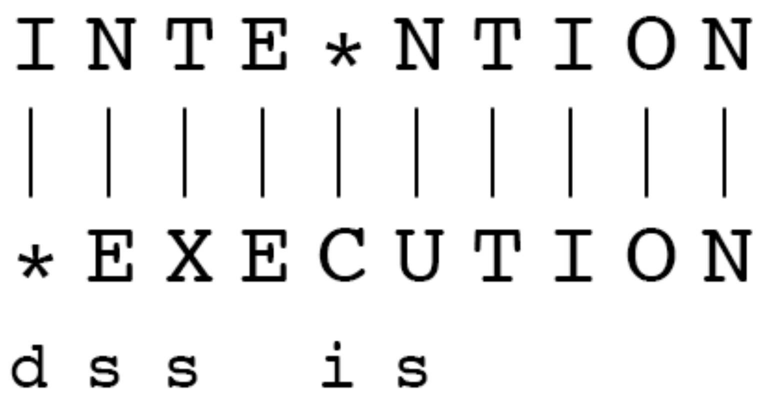
- The minimum edit distance between two strings
- Is the minimum number of editing operations
    - Insertion
    - Deletion
    - Substitution
- Needed to transform one into the other

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



- 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
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- An alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

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

Other uses of Edit Distance in NLP

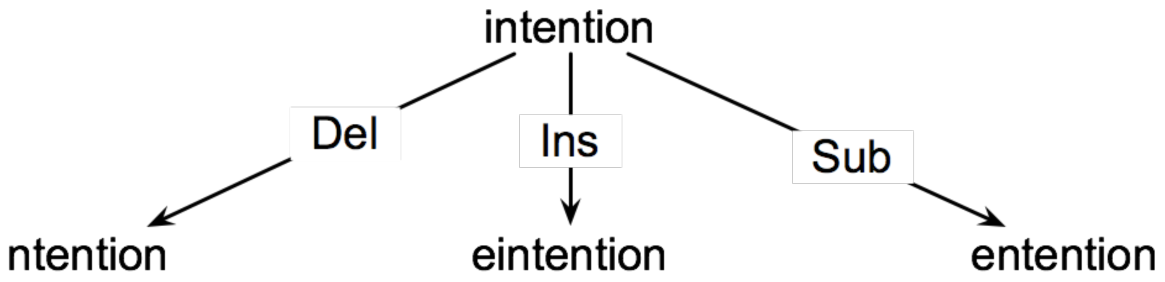
- Evaluating Machine Translation and speech recognition

```
R Spokesman confirms     senior government adviser was appointed
H Spokesman said     the senior            adviser was appointed
                S      I              D                      I
```

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford Professor Jennifer Eberhardt announced yesterday
  - for Professor Eberhardt…

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 revisted 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*)

SIT770: Natural Language Processing

Week 3.8 - Computing Minimum Edit Distance

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env

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…M
          For each  j = 1…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

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

The Edit Distance Table

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

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

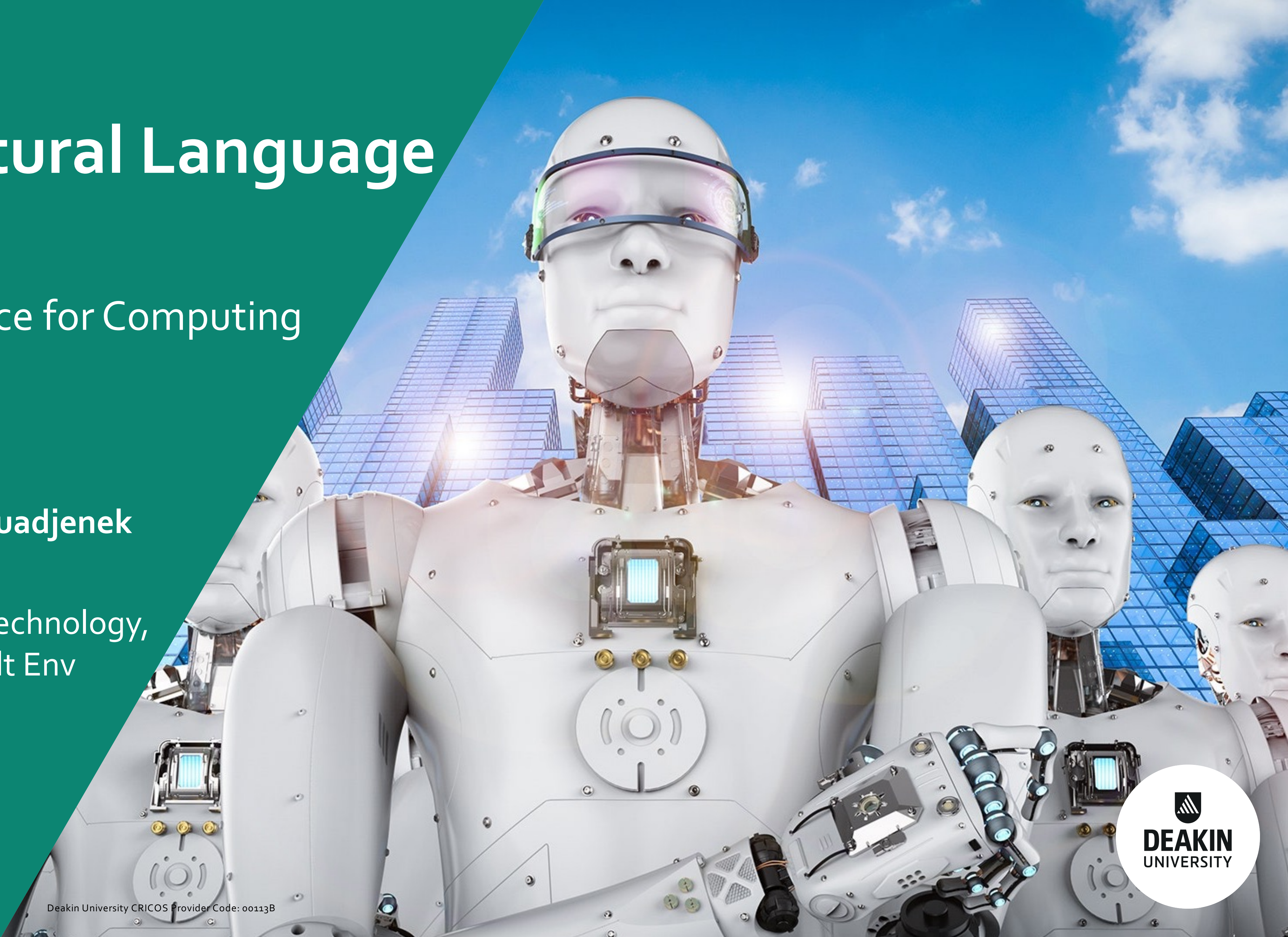
N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	**0**	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

SIT770: Natural Language Processing

Week 3.9 - Backtrace for Computing Alignments

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env



Computing alignments

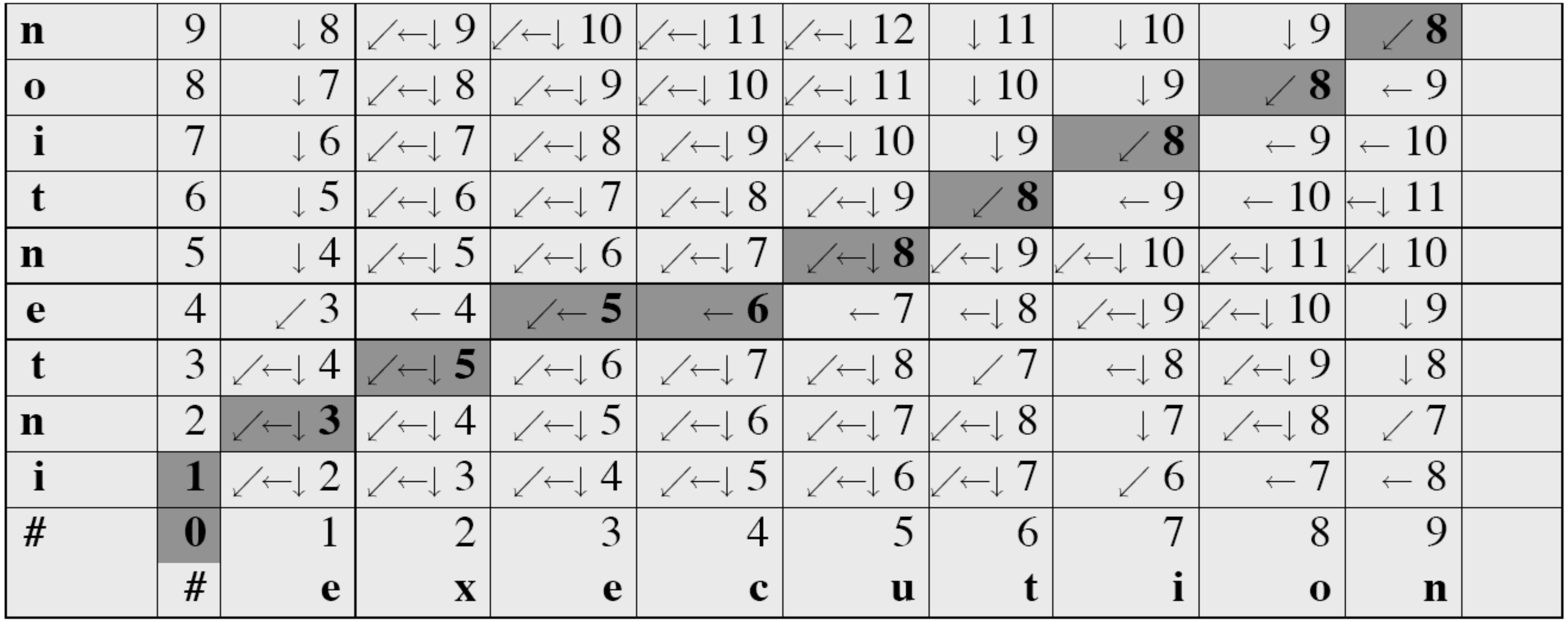
- Edit distance isn’t sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the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

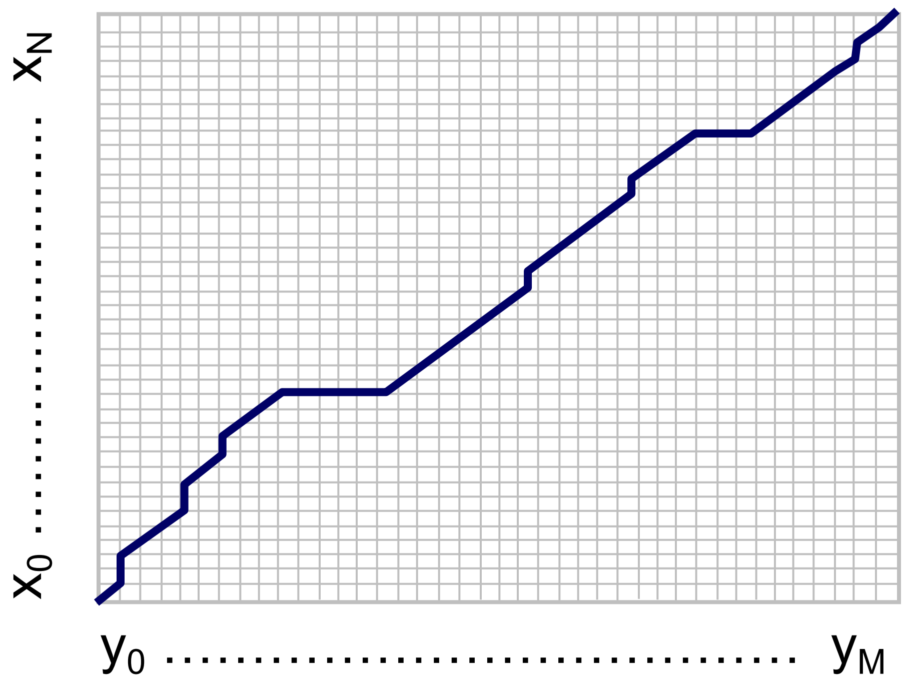
Recurrence Relation:

```
For each  i = 1…M
    For each  j = 1…N
```

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

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

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

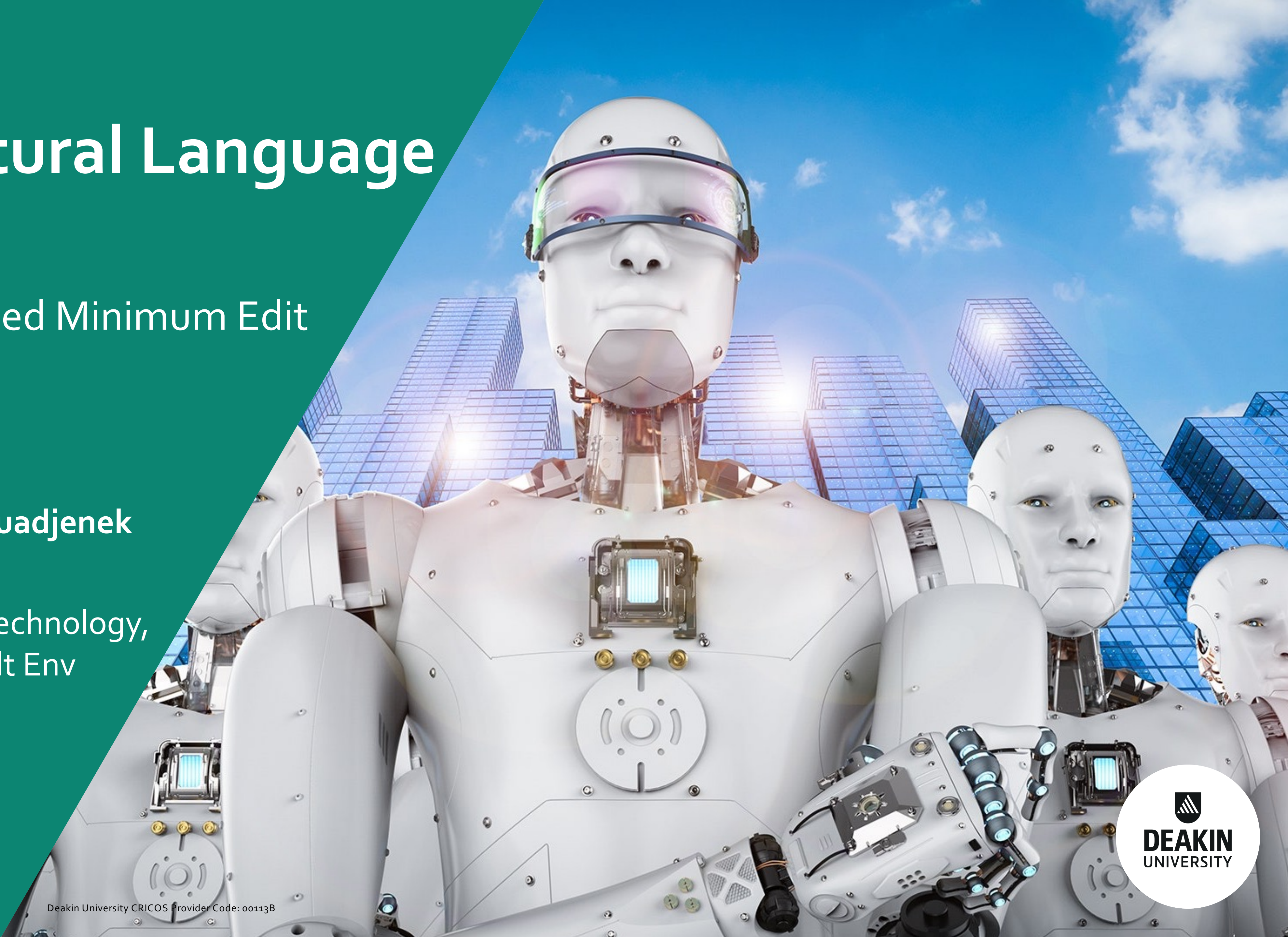
- Time:
  - O(nm)
- Space:
  - O(nm)
- Backtrace
  - O(n+m)

SIT770: Natural Language Processing

Week 3.10 - Weighted Minimum Edit Distance

Dr. Mohamed Reda Bouadjenek

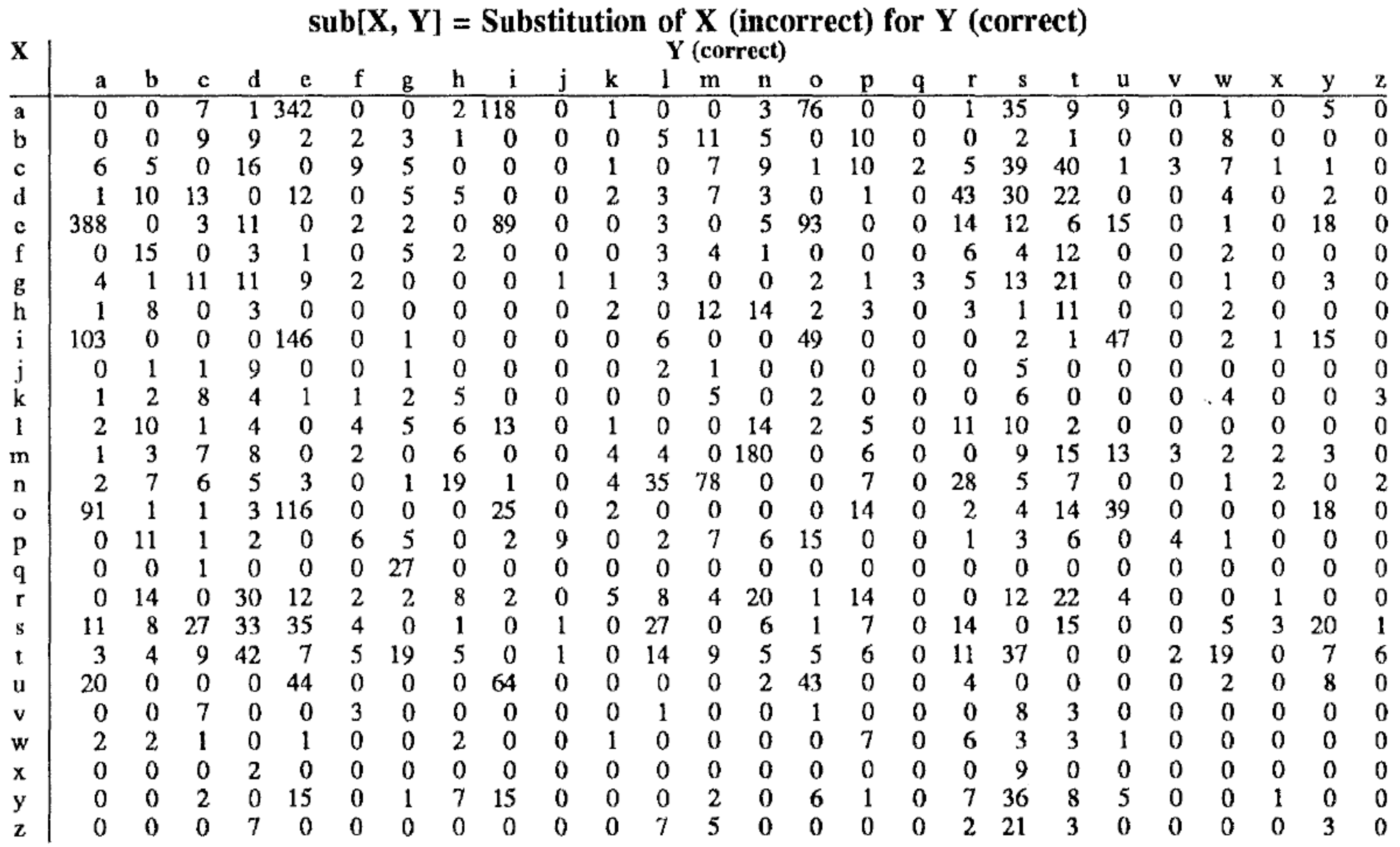
School of Information Technology,
Faculty of Sci Eng & Built Env



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

~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 — - + =
Tab Q W E R T Y U I O P { [}]
Caps Lock A S D F G H J K L : ; " '
Shift Z X C V B N M < , > . ? / Shift
Ctrl Alt Alt

Weighted Min Edit Distance

- Initialization:

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

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

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

- Recurrence Relation:

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

- Termination:

D(N,M) is distance

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.”

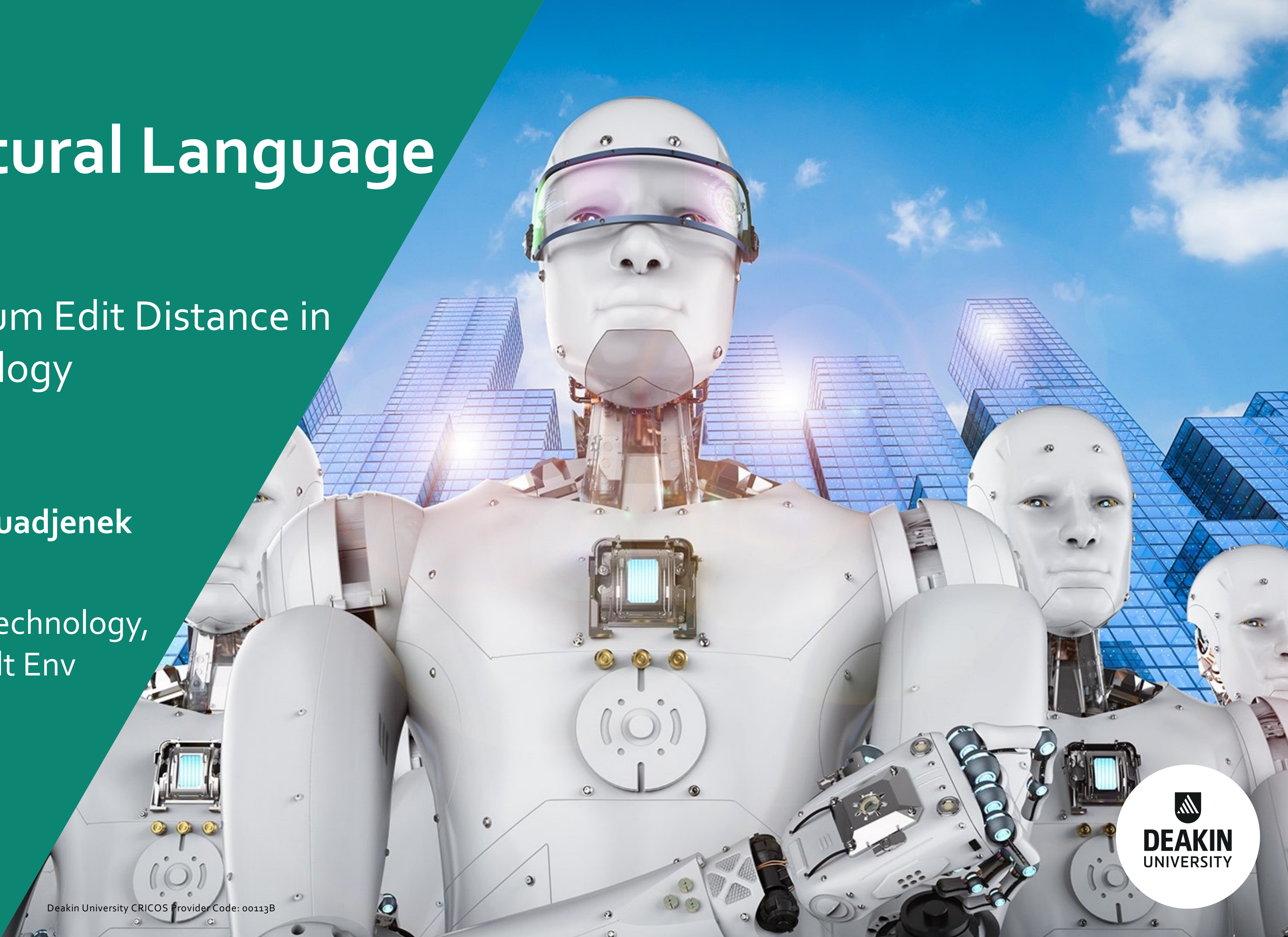
Richard Bellman, “Eye of the Hurricane: an autobiography” 1984.

SIT770: Natural Language Processing

Week 3.11 - Minimum Edit Distance in Computational Biology

Dr. Mohamed Reda Bouadjenek

School of Information Technology,
Faculty of Sci Eng & Built Env



Sequence Alignment

AGGCTATCACCTGACCTCCAGGCCGATGCCC

TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---

T**AG**-**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

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
```

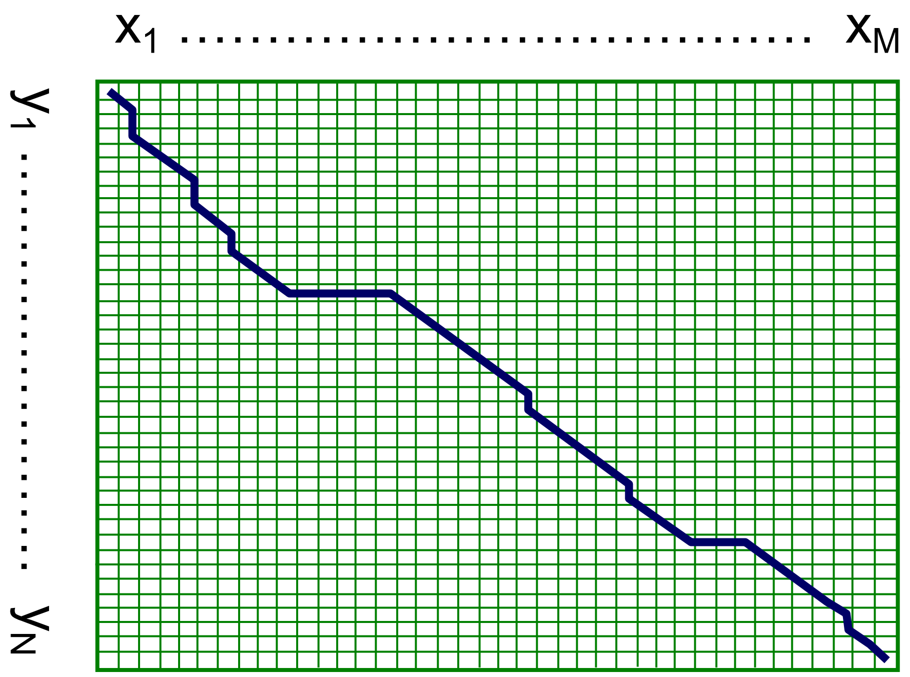
- Recurrence Relation:

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

- Termination:

```
D(N,M) is distance
```

The Needleman-Wunsch Matrix



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

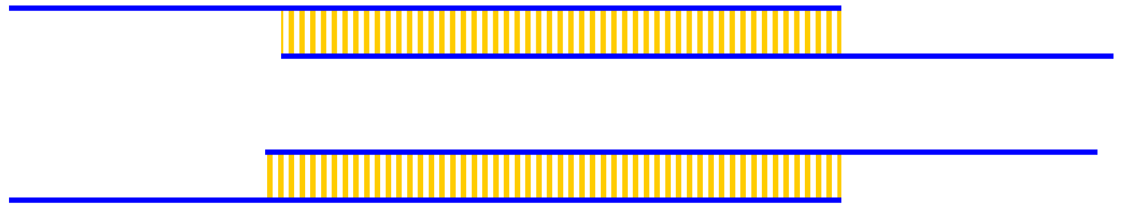
A variant of the basic algorithm:

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

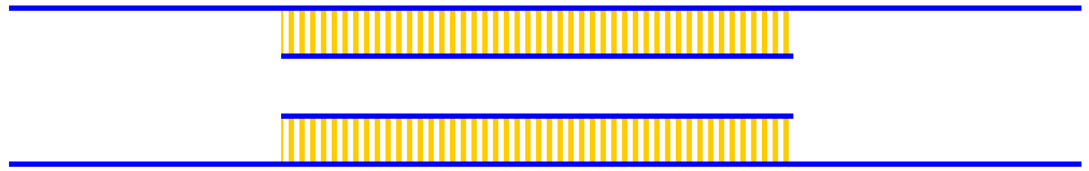
```
----------CTATCACCTGACCTCCAGGCCGATGCCCCTTCCGGC
GCGAGTTCATCTATCAC--GACCGC--GGTCG--------------
```

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

Different types of overlaps

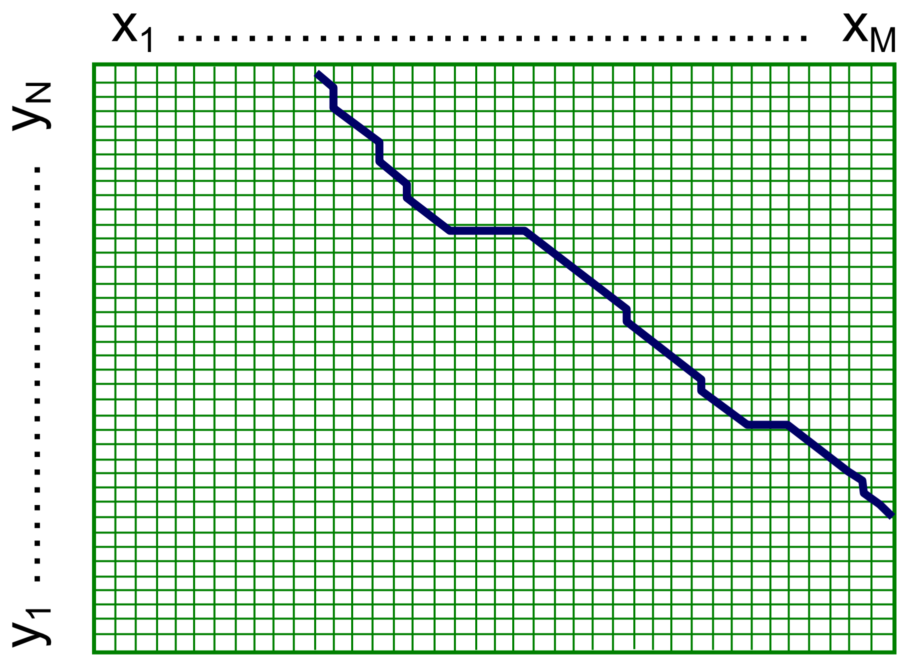


Example:
2 overlapping "*reads*" from a sequencing project



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

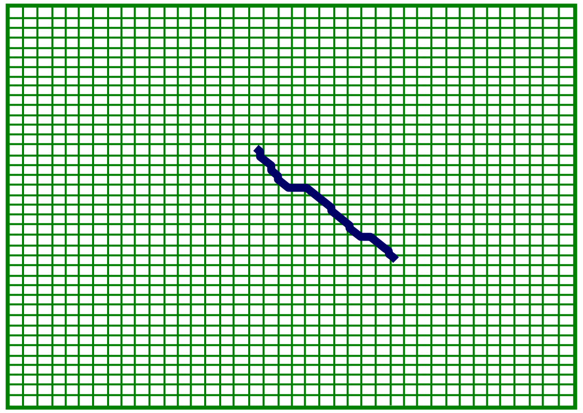
$$F_{OPT} = \max \begin{cases} \max_i F(i, N) \\ \max_j F(M, j) \end{cases}$$

The Local Alignment Problem

Given two strings $x = x_1......x_M$,
$y = y_1......y_N$

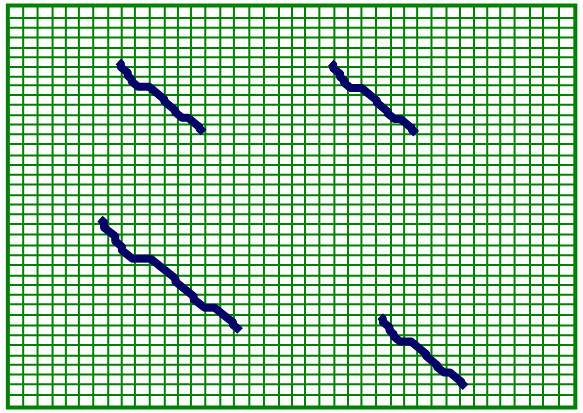
Find substrings x', y' whose similarity (optimal global alignment value) is maximum

x = aaaacccc$\boxed{\text{cgggg}}$tta
y = ttcc$\boxed{\text{cggga}}$accaacc



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…

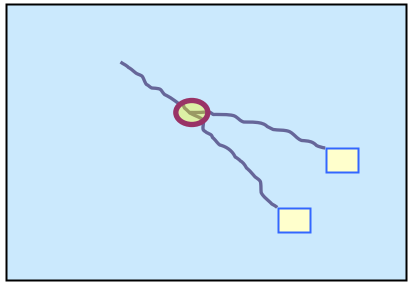
$$F_{OPT} = \max_{i,j} F(i, j)$$

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



Local alignment example

X = ATCAT

Y = ATTATC

Let:

m = 1 (1 point for match)

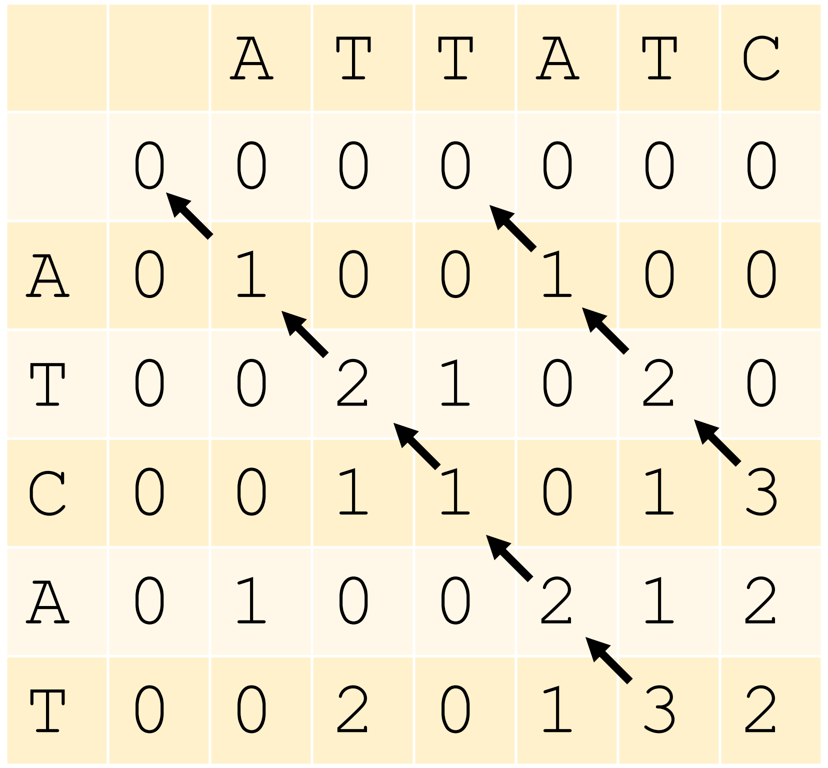
d = 1 (-1 point for del/ins/sub)

		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

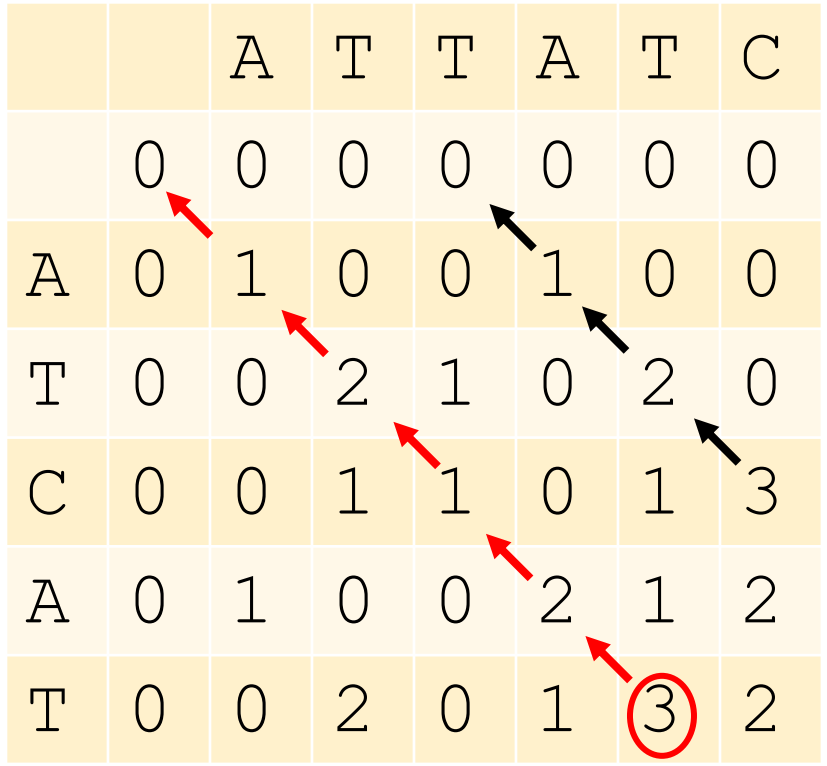


		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0	1	0	0	1	0	0
T	0	0	2	1	0	2	0
C	0	0	1	1	0	1	3
A	0	1	0	0	2	1	2
T	0	0	2	0	1	3	2

Local alignment example

X = **ATCAT**

Y = **ATTAT**C

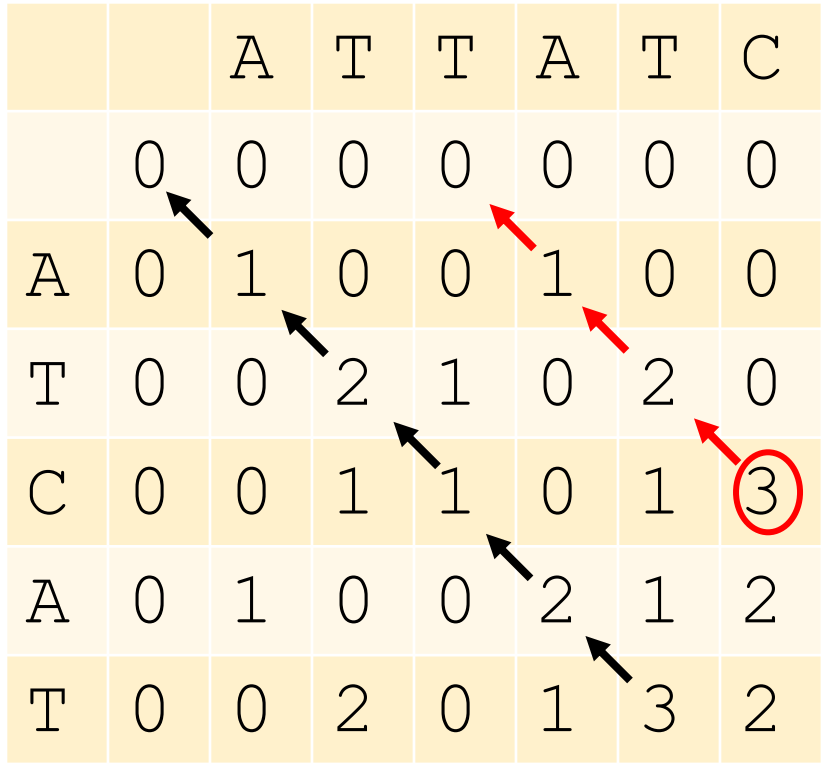


		A	T	T	A	T	C
	0	0	0	0	0	0	0
A	0	1	0	0	1	0	0
T	0	0	2	1	0	2	0
C	0	0	1	1	0	1	3
A	0	1	0	0	2	1	2
T	0	0	2	0	1	$\boxed{3}$	2

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