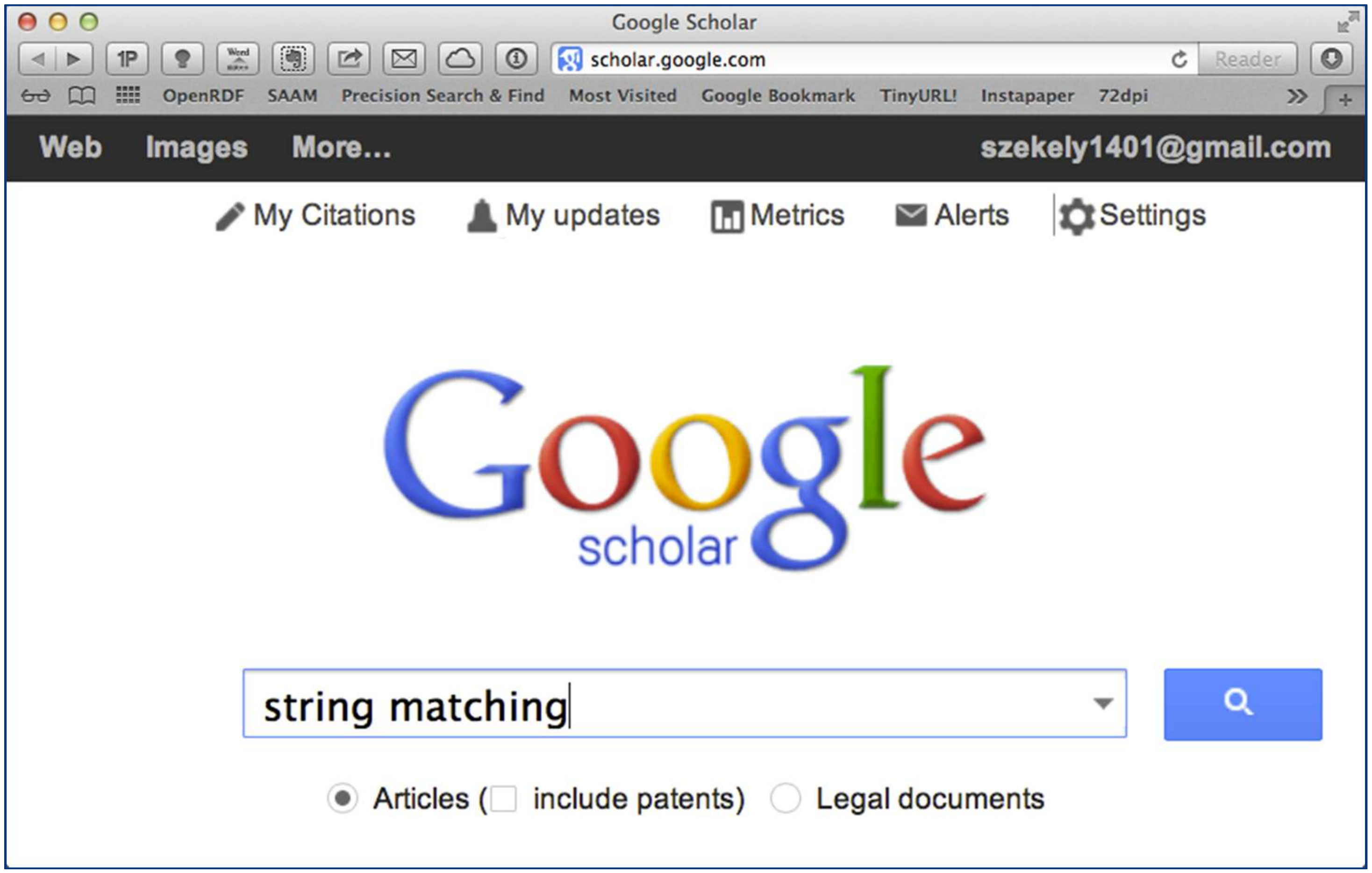
String Matching

Pedro Szekely & Craig Knoblock
University of Southern California

Isn’t the Problem Solved?

String.equalsIgnoreCase(String x)

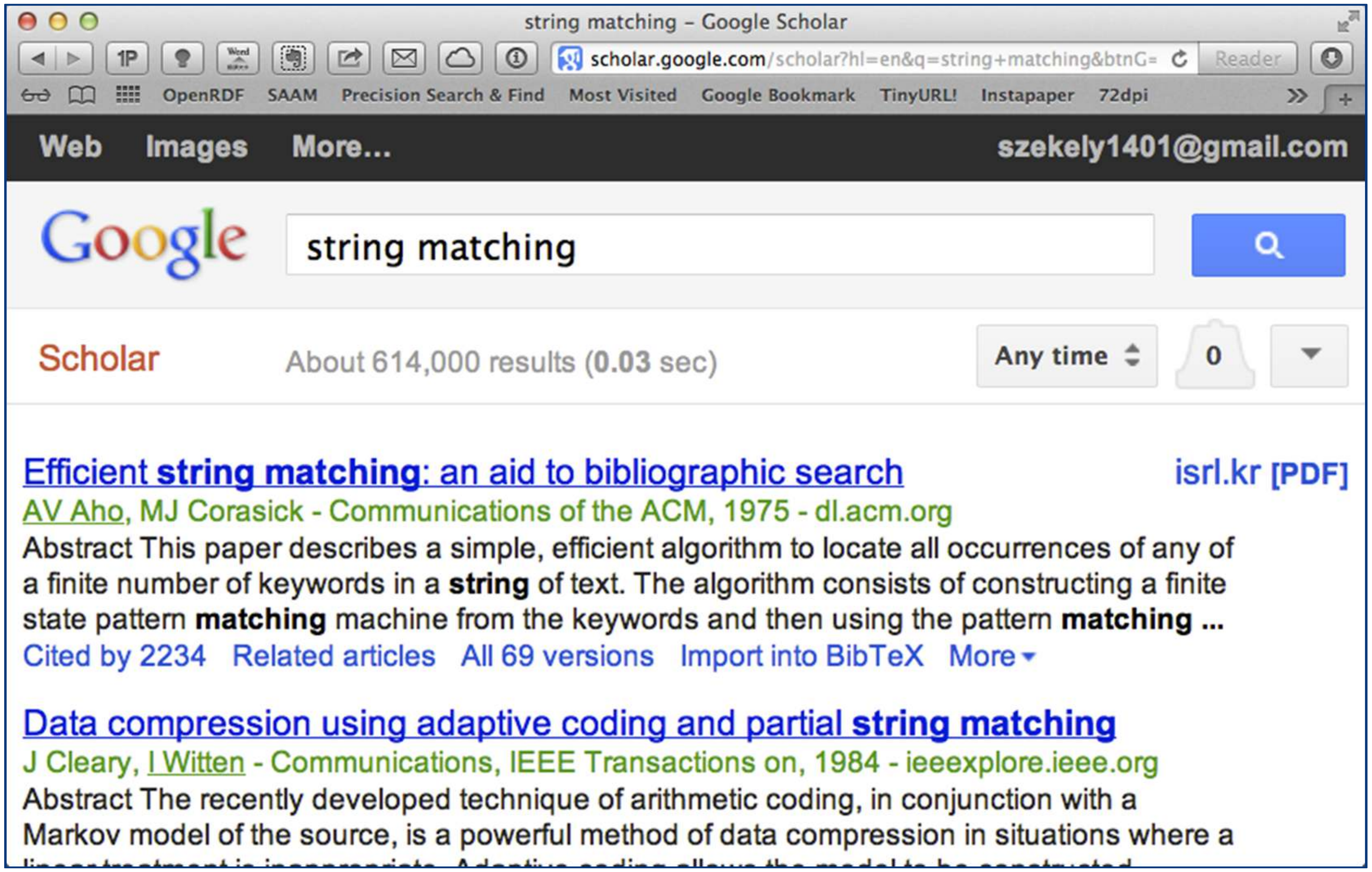
How Many Publications?



string matching

(● Articles (☐ include patents) ○ Legal documents)

Why So Many?



Scholar About 614,000 results (**0.03** sec)

Efficient **string matching**: an aid to bibliographic search isrl.kr **[PDF]**
AV Aho, MJ Corasick - Communications of the ACM, 1975 - dl.acm.org
Abstract This paper describes a simple, efficient algorithm to locate all occurrences of any of a finite number of keywords in a **string** of text. The algorithm consists of constructing a finite state pattern **matching** machine from the keywords and then using the pattern **matching** ...
Cited by 2234 Related articles All 69 versions Import into BibTeX More

Data compression using adaptive coding and partial **string matching**
J Cleary, I Witten - Communications, IEEE Transactions on, 1984 - ieeexplore.ieee.org
Abstract The recently developed technique of arithmetic coding, in conjunction with a Markov model of the source, is a powerful method of data compression in situations where a

slide by Pedro Szekely

Multiple John Singer Sargents?

```
dallas:John_Singer_Sargent
  a foaf:Person;
  :dateOfBirth "1856" ;
  :dateOfDeath "1925" ;
  :name "John Singer Sargent" .
```

```
ima:John_Singer_Sargent
  a foaf:Person;
  :dateOfBirth "1856" ;
  :dateOfDeath "1925" ;
  :name "John S. Sargent" .
```

Multiple John Singer Sargents?

```
dallas:John_Singer_Sargent
   a foaf:Person;
   :dateOfBirth "1856" ;
   :dateOfDeath "1925" ;
   :name "John Singer Sargent" .
```

string_match(

) = ???

```
ima:John_Singer_Sargent
   a foaf:Person;
   :dateOfBirth "1856" ;
   :dateOfDeath "1925" ;
   :name "John S. Sargent" .
```

String Matching Problem

myMatchFunction(x, y)

yourMatchFunction(x, y)

What does it mean that one is better than the other?

Problem Definition

Given X and Y sets of strings

Find pairs (x, y)
such that both x and y
refer to the same real world entity

`"John S. Sargent"`

`"John Singer Sargent"`

Problem Definition

Given X and Y sets of strings

Find pairs (x, y)
such that both x and y
refer to the same real world entity

We can use precision and recall to evaluate algorithms

Problem Definition

Given X and Y sets of strings

Find pairs (x, y)
such that both x and y
refer to the same real world entity

fraction of pairs found that are correct

We can use precision and recall to evaluate algorithms

fraction of pairs found

Why Strings Don’t Match Perfectly?

typos	"Joh” vs “John”
OCR errors	"J0hn” vs “John”
formatting conventions	”03/17” vs “March 17”
abbreviations	”J. S. Sargent” vs “John Singer Sargent”
nick names	”John” vs “Jock”
word order	”Sargent, John S.” vs “John S. Sargent”

slide by Pedro Szekely

Similiarity Measure

Similiarity	**s**(x, y) in [0, 1]
	better
Distance	**d**(x, y) in [0, ∞)

Types of Similarity Metrics

- Sequence based
- Set based
- Hybrid
- Phonetic

Sequence Based Metrics

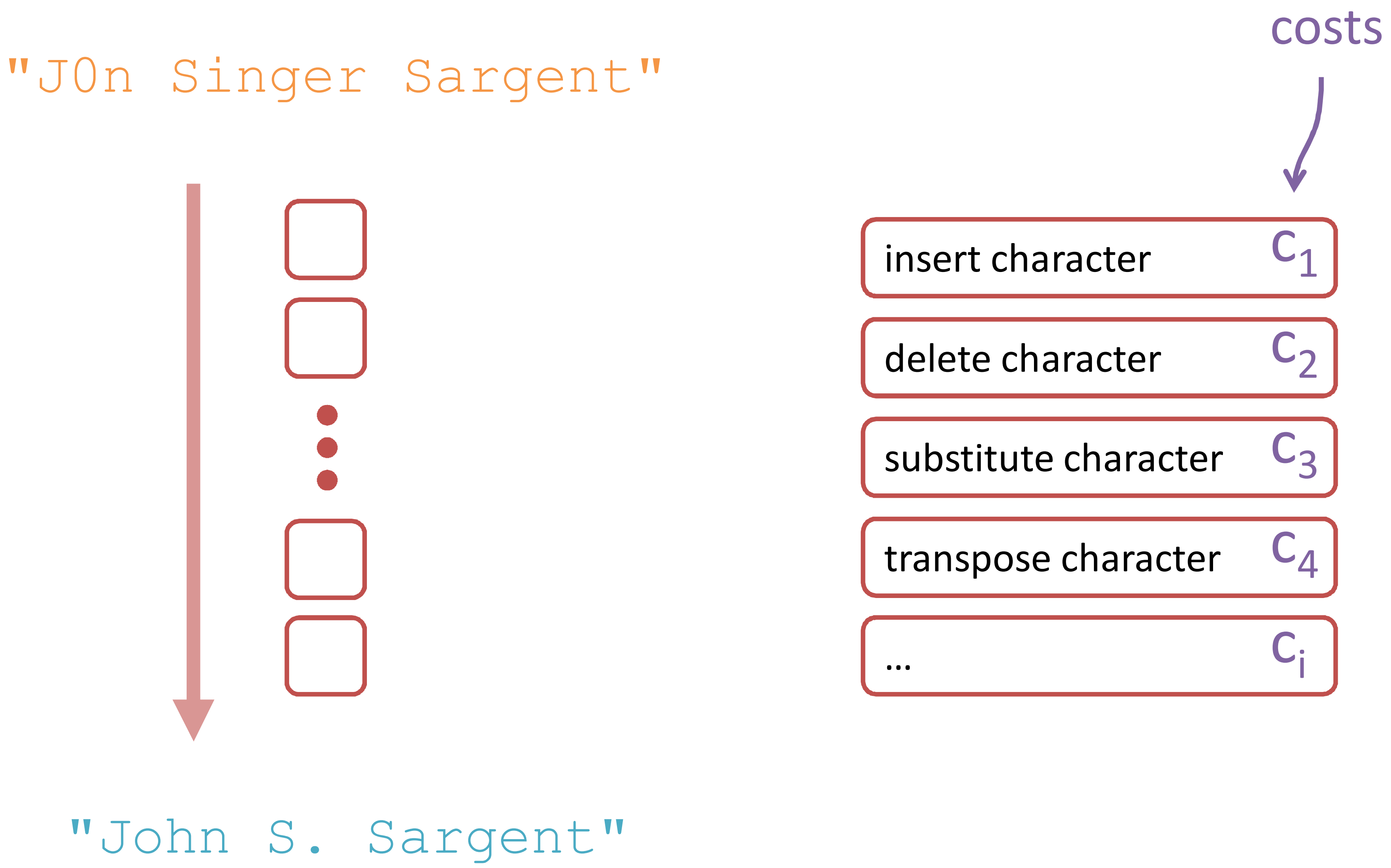
Edit Distance

"J0n Singer Sargent"

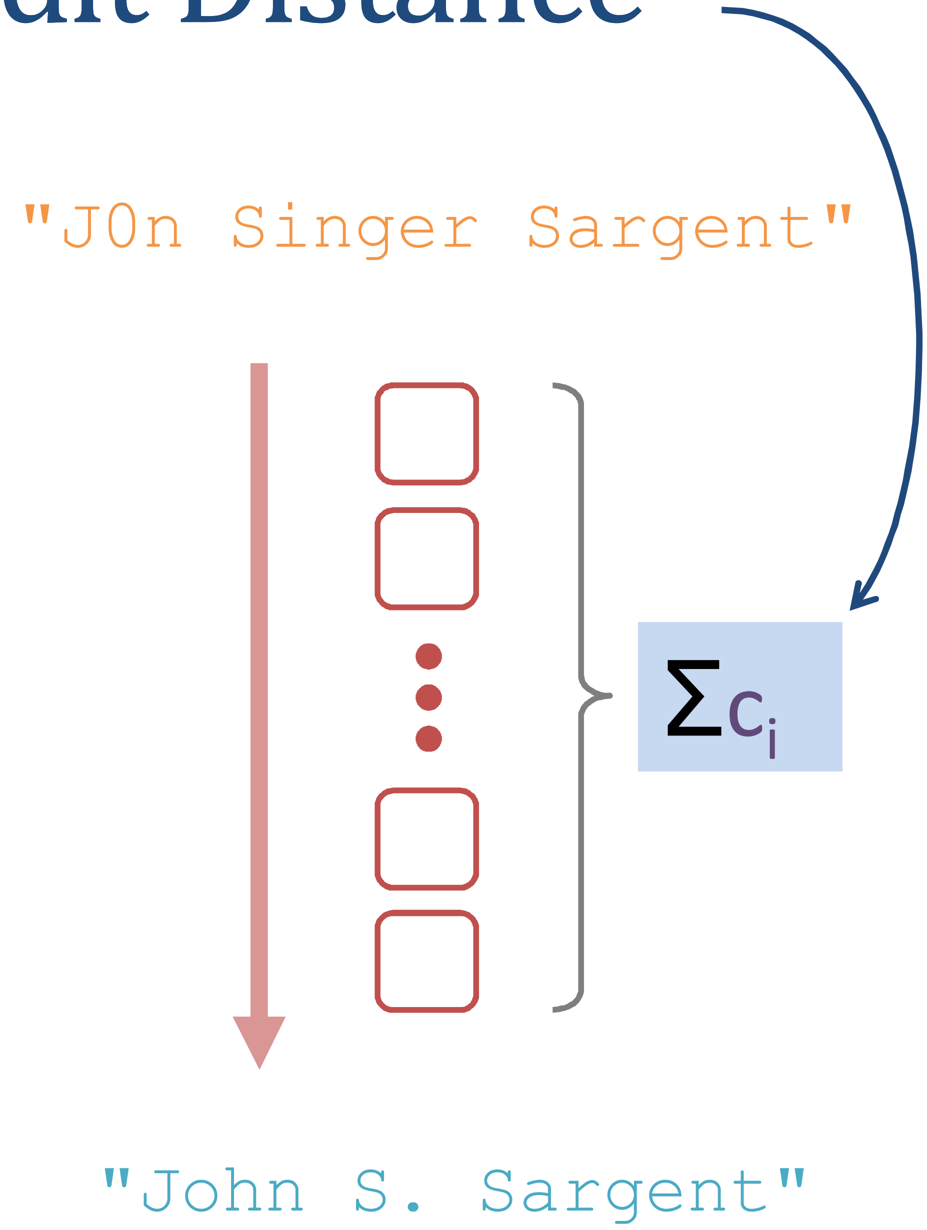
"John S. Sargent"

- insert character
- delete character
- substitute character
- transpose character
- …

Edit Distance



Edit Distance



costs

insert character	c_1
delete character	c_2
substitute character	c_3
transpose character	c_4
…	c_i

Levenshtein Distance

costs

Edit distance:

insert character	1
delete character	1
substitute character	1

lev(x, y) is the minimum cost to transform x to y

Online calculator: http://planetcalc.com/1721/

Computing Levenshtein Distance

lev(x, y) is the minimum cost to transform x to y

Definitions

$x = x_1 x_2 \ldots x_n$

$y = y_1 y_2 \ldots y_m$

$d(i,j) = \mathrm{lev}(x_1 x_2 \ldots \underline{x_i},\ y_1 y_2 \ldots \underline{y_j})$

$d(0,0) = \mathrm{lev}(\text{“”}, \text{“”}) = 0$

We want $d(n,\ m)$

Computing Levenshtein Distance

```
d(0,0) = 0
```

Computing Levenshtein Distance

$d(0,0) = 0$

$x_1x_2 \ldots x_{i-1}x_i$ is a suffix of x $y_1y_2 \ldots y_{j-1}y_j$ is a suffix of y

Computing Levenshtein Distance

$d(0,0) = 0$

$x_1 x_2 \ldots x_{i-1} x_i$ is a prefix of x $y_1 y_2 \ldots y_{j-1} y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	$d(i-1,j-1)$	keep x_i

Computing Levenshtein Distance

$d(0,0) = 0$

$x_1 x_2 \ldots x_{i-1} x_i$ is a prefix of x $y_1 y_2 \ldots y_{j-1} y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	$d(i-1, j-1)$	keep x_i
$x_i \mathrel{!}= y_j$		delete x_i insert y_j after x_i replace x_i with y_j

Computing Levenshtein Distance

$d(0,0) = 0$

$x_1x_2 \ldots x_{i-1}x_i$ is a prefix of x $y_1y_2 \ldots y_{j-1}y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	$d(i-1,j-1)$	keep x_i
$x_i \,!\!= y_j$	$d(i-1,j) + 1$	delete x_i insert y_j after x_i replace x_i with y_j

Computing Levenshtein Distance

`d(0,0) = 0`

$x_1x_2 \dots x_{i-1}x_i$ is a prefix of x $y_1y_2 \dots y_{j-1}y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	`d(i-1,j-1)`	keep x_i
x_i != y_j	`d(i-1,j) + 1`	delete x_i
	`d(i,j-1) + 1`	insert y_j after x_i
		replace x_i with y_j

Computing Levenshtein Distance

`d(0,0) = 0`

$x_1 x_2 \ldots x_{i-1} x_i$ is a prefix of x $\qquad$ $y_1 y_2 \ldots y_{j-1} y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	`d(i-1,j-1)`	keep x_i
x_i != y_j	`d(i-1,j) + 1`	delete x_i
	`d(i,j-1) + 1`	insert y_j after x_i
	`d(i-1,j-1) + 1`	replace x_i with y_j

Computing Levenshtein Distance

$d(0,0) = 0$

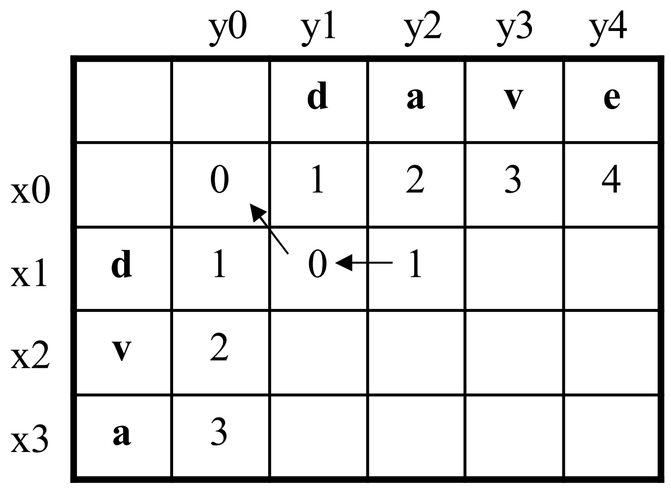
$x_1x_2 \ldots x_{i-1}x_i$ is a prefix of x $y_1y_2 \ldots y_{j-1}y_j$ is a prefix of y

Case	Distance	Operation
$x_i = y_j$	$d(i-1,j-1)$	keep x_i
$x_i \mathrel{!=} y_j$	$d(i-1,j) + 1$	delete x_i
	$d(i,j-1) + 1$	insert y_j after x_i
	$d(i-1,j-1) + 1$	replace x_i with y_j

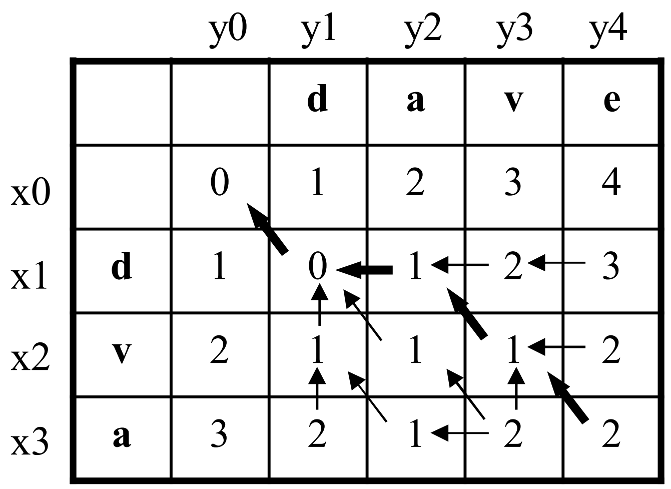
$d(i,j) = $ minimum

Computing Levenshtein Distance Using Dynamic Programming

- x = dva, y = dave



		y0	y1	y2	y3	y4
			d	**a**	**v**	**e**
x0		0	1	2	3	4
x1	**d**	1	0	1		
x2	**v**	2				
x3	**a**	3				



		y0	y1	y2	y3	y4
			d	**a**	**v**	**e**
x0		0	1	2	3	4
x1	**d**	1	0	1	2	3
x2	**v**	2	1	1	1	2
x3	**a**	3	2	1	2	2

```
x = d – v a
    | | | |
y = d a v e
```

substitute a with e
insert a (after d)

- Cost of dynamic programming is O(|x||y|)

Levenshtein Distance Complexity

Dynamic programming algorithm

Time Complexity = O (N * M), Space Complexity = O (N * M)

Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity
Alexandr Andoni, Robert Krauthgamer, Krzysztof Onak

We present a near-linear time algorithm that approximates the edit distance between two strings within a polylogarithmic factor; specifically, for strings of length n and every fixed epsilon>0, it can compute a $(\log n)^{O(1/\text{epsilon})}$ approximation in $n^{(1+\text{epsilon})}$ time.

http://arxiv.org/abs/1005.4033

Levenshtein Distance Examples

```
lev(John Singer Sargent,
    John S. Sargent)          =
```

Levenshtein Distance Examples

```
lev(John Singer Sargent,
    John S. Sargent)          = 5
```

Levenshtein Distance Examples

```
lev(John Singer Sargent,
    John S. Sargent)          =  5

lev(John Singer Sargent,
    Jane Klinger Sargent)     =
```

Levenshtein Distance Examples

```
lev(John Singer Sargent,
    John S. Sargent)          =  5

lev(John Singer Sargent,
    Jane Klinger Sargent)     =  5
```

Levenshtein Distance Examples

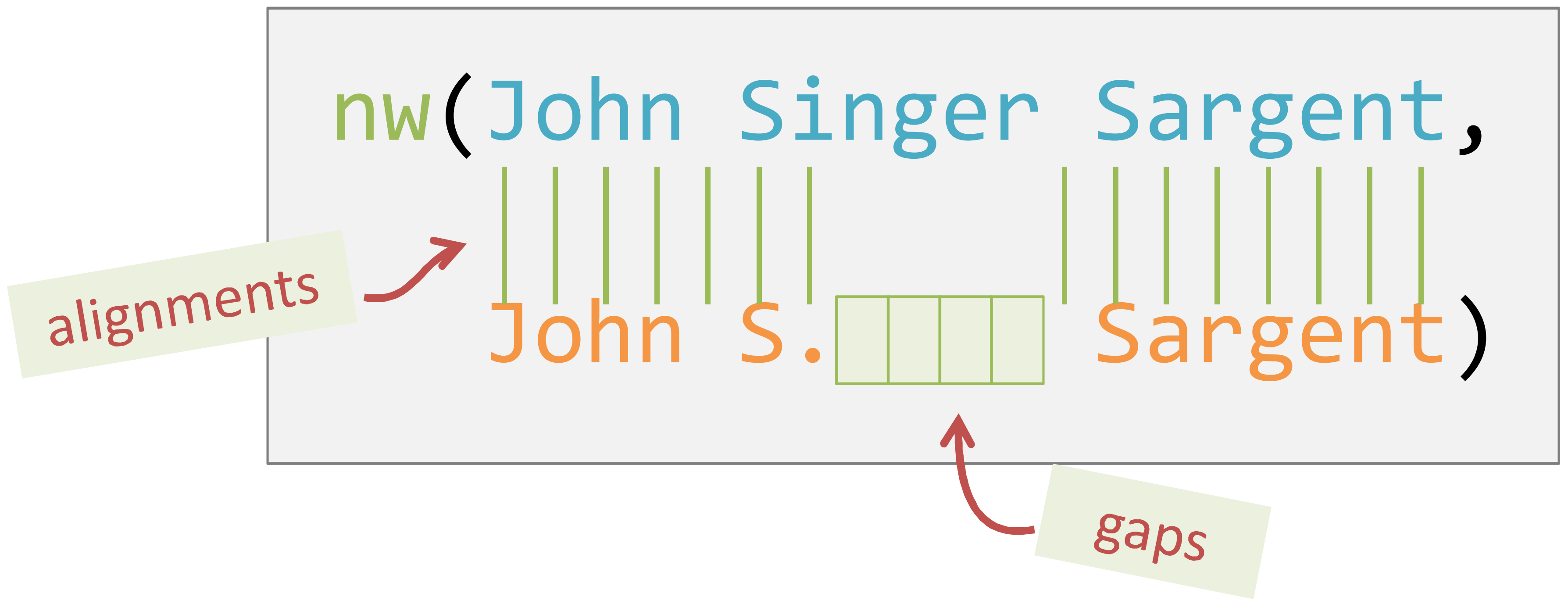
Too high a cost for deleting a sequence of characters

```
lev(John Singer Sargent,
    John S.     Sargent)        =   5


lev(John Singer Sargent,
    Jane Klinger Sargent)       =   5
```

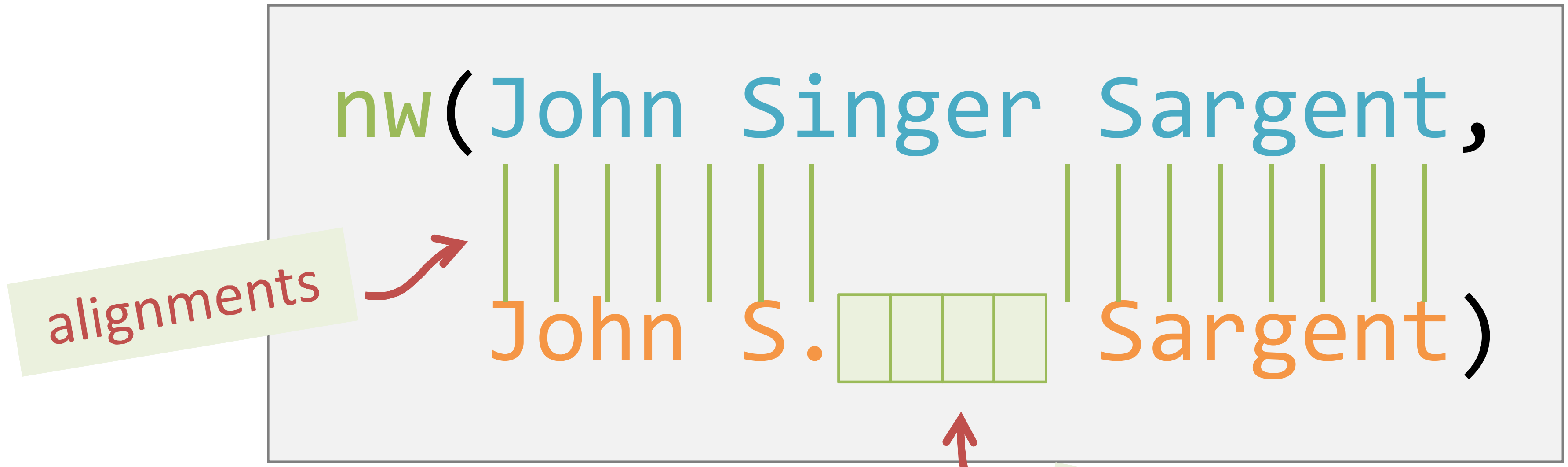
Needleman-Wunch Measure

Generalization of levenstein(x, y)



Needleman-Wunch Measure

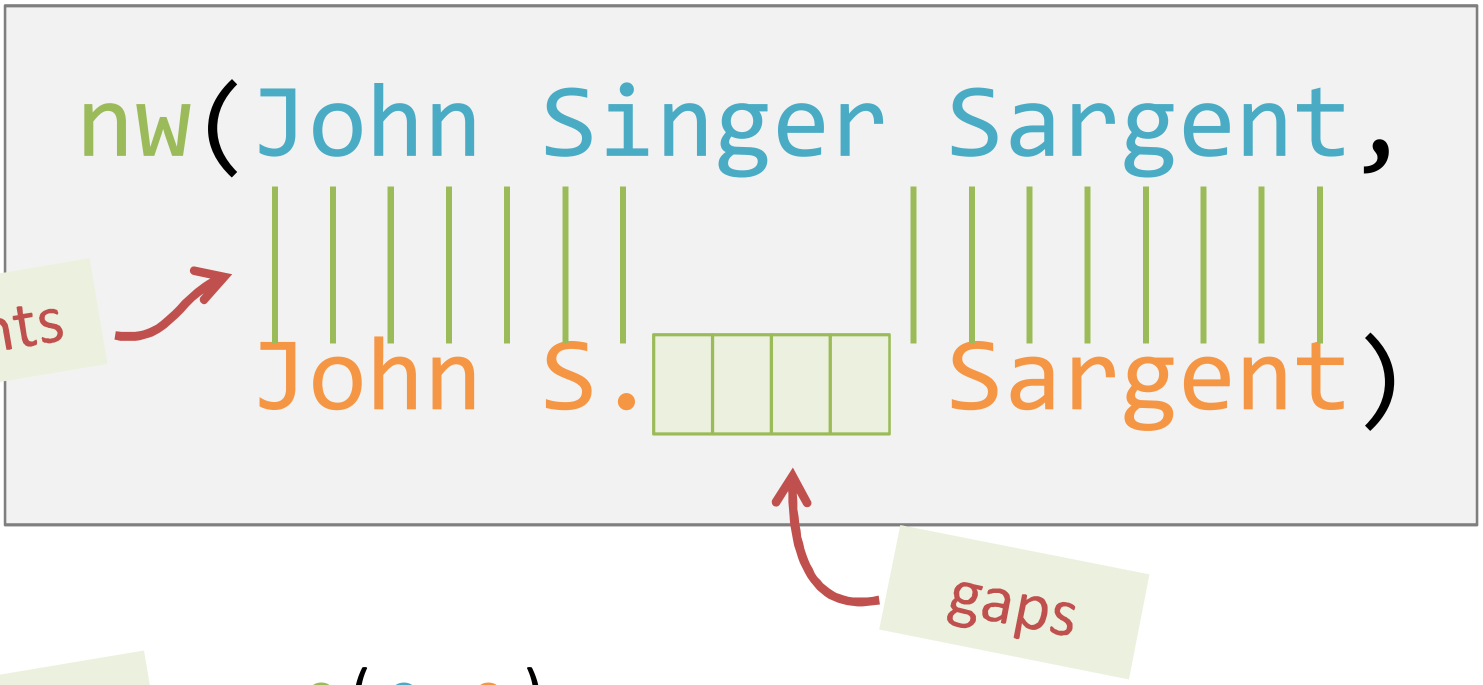
Generalization of levenstein(x, y)



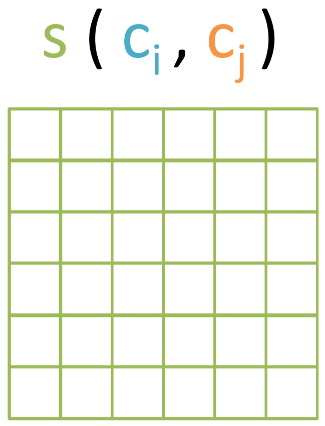
alignment score matrix
$s(c_i, c_j)$

Needleman-Wunch Measure

Generalization of levenstein(x, y)



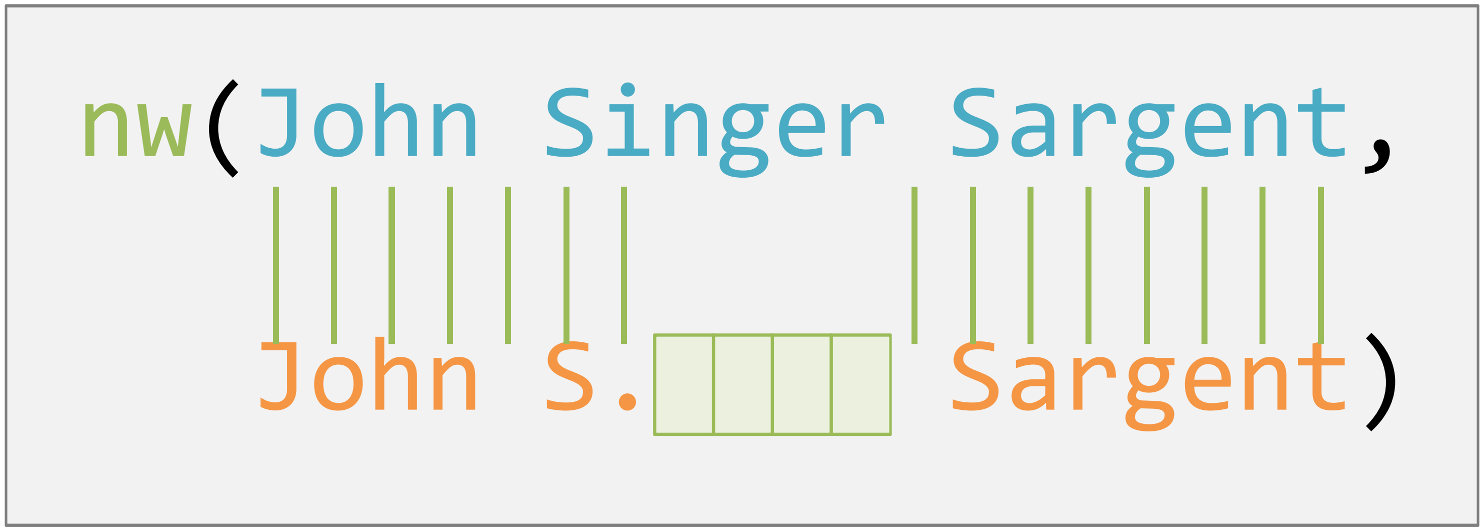
Alignment score matrix



gap score

Needleman-Wunch Measure

Generalization of levenstein(x, y)

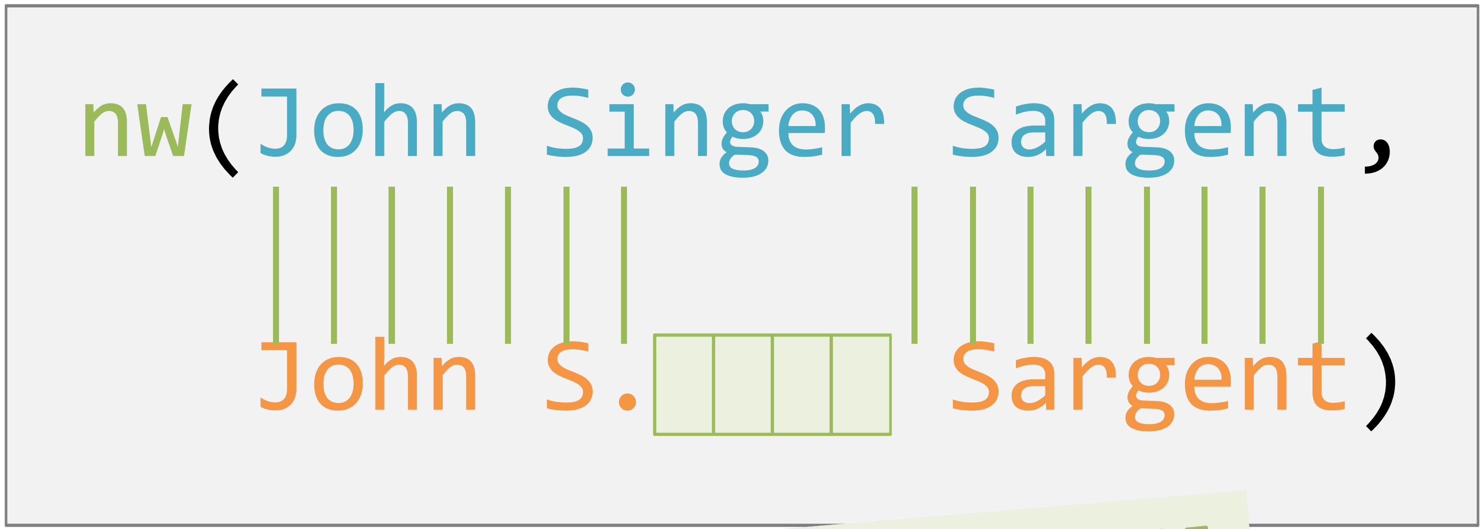


$$s(c_i, c_j) = \begin{cases} 2 & \text{if } c_i = c_j \\ -1 & \text{if } c_i \mathrel{!=} c_j \end{cases}$$

gap-score = -0.5

Needleman-Wunch Measure

Generalization of levenstein(x, y)



nw(John Singer Sargent,

John S. Sargent)

$2 * 14 + (-1) * 1 + (-0.5) * 4 = 25$

$$s(c_i, c_j) = \begin{cases} 2 & \text{if } c_i = c_j \\ -1 & \text{if } c_i \mathrel{!=} c_j \end{cases}$$

gap-score = -0.5

Comparison

	Levenshtein	**Needleman-Wunch**
Costs	1	matrix
Operations	insert/delete	gaps
Result	distance	similarity

OCR errors "J0hn" vs "John"

```
score(o,0) = -0.2

score(m,0) = -1.0
```

lower penalty

slide by Pedro Szekely

Needleman-Wunch Example

```
nw(John Singer Sargent,
   John S.     Sargent)
```

2 * 14 + (-1) * 1 + (-0.5) * 4 = 25

2 * 14 + (-1) * 3 + (-0.5) * 1 = 24.5

```
nw(John S inger Sargent,
   Jane Klinger Sargent)
```

Needleman-Wunch Example

```
nw(John Singer Sargent,
   John S.     Sargent)
```

2 * 14 + (-1) * 1 + (-0.5) * 4 = 25

2 * 14 + (-1) * 1 + (-0.5) * 8 = 23

```
nw(John Stanislaus Sargent,
   John S.         Sargent)
```

Needleman-Wunch Example

```
nw(John Singer Sargent,
   John S.     Sargent)
```

+ (-1) * 1 + (-0.5) * 4 = 25

Longer gaps are penalized more

Bad for names

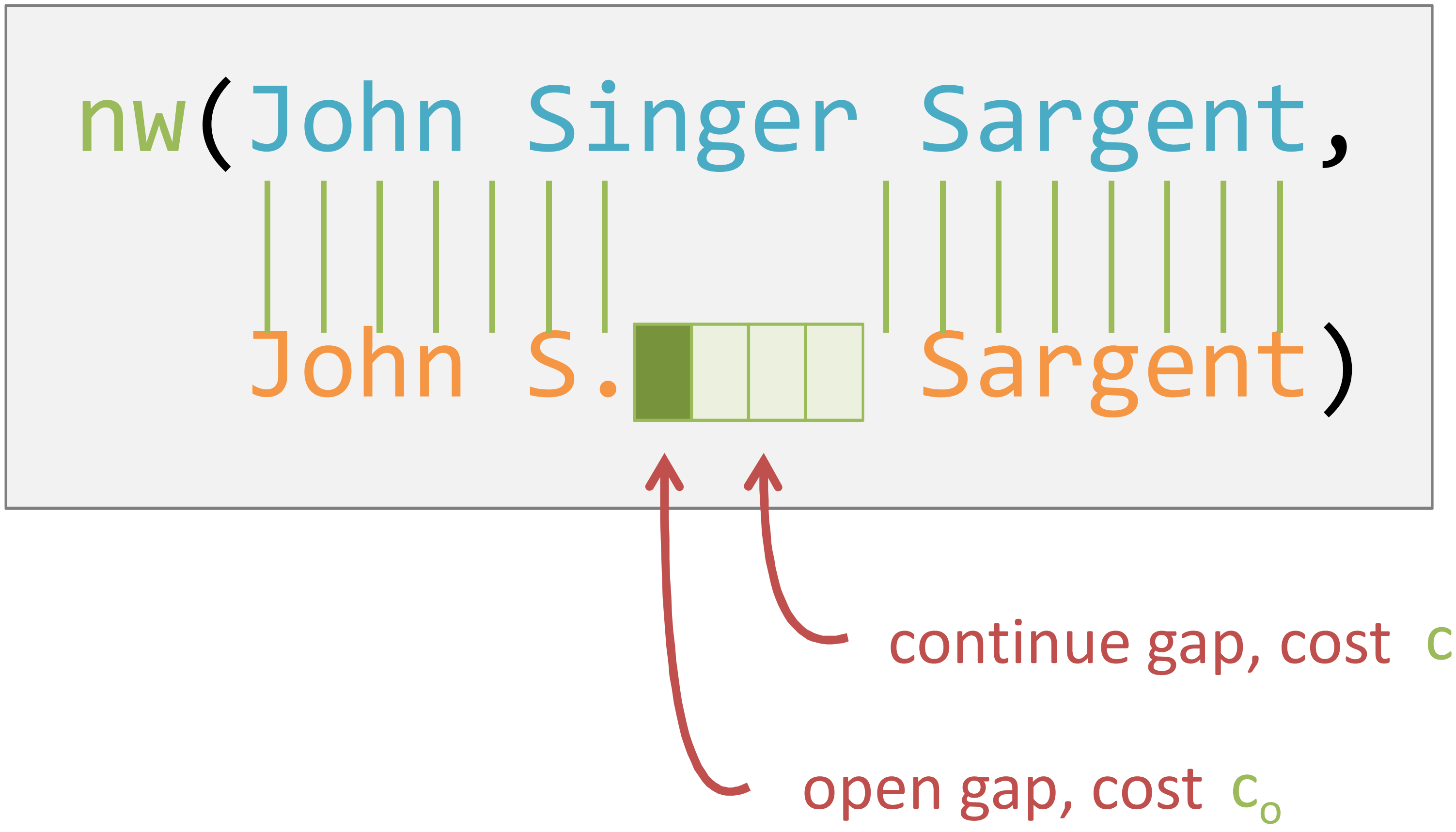
2 * 14 + (-1) * 1 + (-0.5) * 8 = 23

```
nw(John Stanislaus Sargent,
   John S.         Sargent)
```

slide by Pedro Szekely

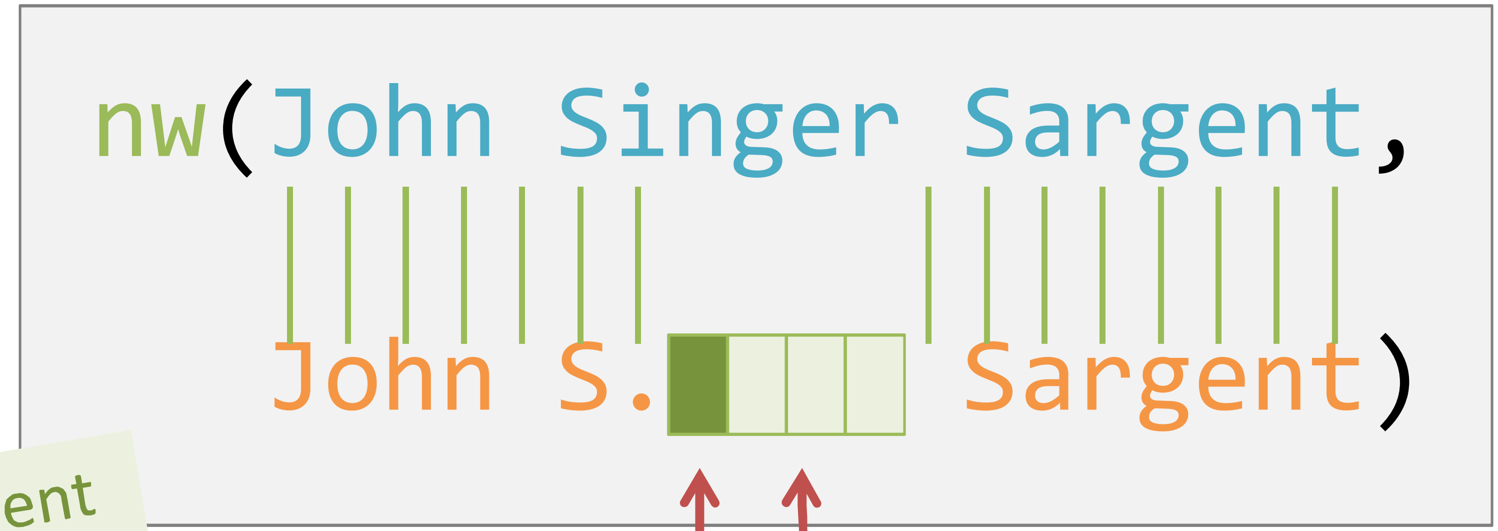
Affine Gap Measure

Generalization of needleman-wunch(x, y)

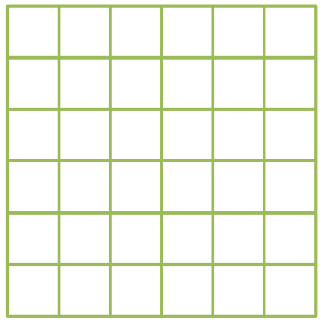


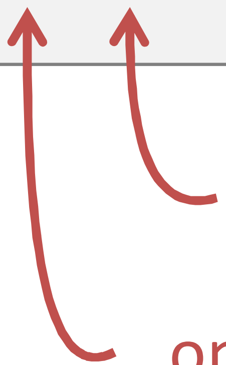
Affine Gap Measure

Generalization of needleman-wunch(x, y)



$s(c_i, c_j)$





continue gap, cost c_r

open gap, cost c_o

total gap cost: $c_o + (|gap| - 1) * c_r$

Smith-Waterman

Global alignment
fully align sequences, opening gaps as needed

```
F T F T A L I L L A V A V
F - - T A L – L L A – A V
```

Needleman-Wunch

Local alignment
find best subsequences to align

```
F T F T A L I L L A V A V
- - F T A L – L L A V - -
```

Smith-Waterman: local alignment version of Needleman-Wunch

slide by Pedro Szekely

Smith-Waterman Example

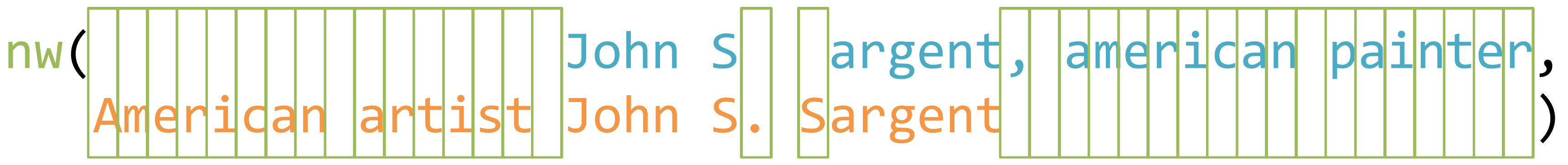
```
match(John Sargent, american painter,
      American artist John S. Sargent)
```

```
nw(               John S   argent, american painter,
   American artist John S. Sargent                  )
```

Needleman-Wunch: significant gap penalty

Smith-Waterman Example

```
match(John Sargent, american painter,
      American artist John S. Sargent)
```



```
nw(                John S   argent, american painter,
   American artist John S. Sargent                   )
```

Needleman-Wunch: significant gap penalty

Smith-Waterman Example

```
match(John Sargent, american painter,
      American artist John S. Sargent)
```

```
nw(                John S   argent, american painter,
   American artist John S. Sargent                  )
```

Needleman-Wunch: significant gap penalty

```
sw(                John S   argent, american painter,
   American artist John S. Sargent                  )
```

Smith-Waterman: identifies similar subsequences

Jaro Similarity Measure

- Get points for having characters in common
  - but only if they are "close by"

- Get points for common characters in the same order
  - lose points for transpositions

Jaro Similarity Measure jaro(x, y)

$$\text{max-distance} = \frac{\max(\,|x|\,,\,|y|\,)}{2} - 1$$

$$x_i \text{ matches } y_j \text{ if } \begin{cases} x_i = y_j \\ |i - j| \le \text{max-distance} \end{cases}$$

m = number of matching characters

t = number of transpositions (of matching characters)

Jaro Similarity Measure

$$\text{max-distance} = \frac{\max(\,|x|\,,\,|y|\,)}{2} - 1$$

m = number of matching characters

t = number of transpositions

$$\text{jaro}(x, y) = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m}\right) & \end{cases}$$

Jaro Example

```
lev(DIXON, DICKSONX)  = 4     (4/8 = 0.5)

jaro(DIXON, DICKSONX) = ???
```

Jaro Example

	D	I	X	O	N
D	1	0	0	0	0
I	0	1	0	0	0
C	0	0	0	0	0
K	0	0	0	0	0
S	0	0	0	0	0
O	0	0	0	1	0
N	0	0	0	0	1
X	0	0	1	0	0

Jaro Example

	D	I	X	O	N
D	1	0	0	0	0
I	0	1	0	0	0
C	0	0	0	0	0
K	0	0	0	0	0
S	0	0	0	0	0
O	0	0	0	1	0
N	0	0	0	0	1
X	0	0	0	0	0

$|x| = 5$

$|y| = 8$

max-distance

$= (8/2) - 1$

$= 3$

$m = 4$

$t = 0$

Jaro Example

	D	I	X	O	N
D	1	0	0	0	0
I	0	1	0	0	0
C	0	0	0	0	0
K	0	0	0	0	0
S	0	0	0	0	0
O	0	0	0	1	0
N	0	0	0	0	1
X	0	0	0	0	0

$|x| = 5$

$|y| = 8$

$m = 4$

$t = 0$

$$\frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m}\right)$$

$$\frac{1}{3}\left(\frac{4}{5} + \frac{4}{8} + \frac{4 - 0}{4}\right)$$

$$= 0.767$$

Jaro Example

```
lev(DIXON, DICKSONX)  = 4    (4/8 = 0.5)

jaro(DIXON, DICKSONX) = 0.767
```

Jaro-Winkler Measure

Give a bonus if there is a common prefix

```
jwProximity(x,y,boostThreshold,prefixSize)
     = jaro(x,y) <= boostThreshold
     ? jaro (x,y)
     : jaro (x,y)
       + 0.1 * prefixMatch(x,y,prefixSize)
             * (1.0 - jaro(x,y))
```

```
prefixMatch(x,y,prefixSize) =
  min(prefixSize, common-prefix(x,y))
```

```
boostThreshold = 0.7
    prefixSize = 4
```

Jaro-Winkler Measure

```
jwProximity(x,y,boostThreshold,prefixSize)
     = jaro(x,y) <= boostThreshold
     ? jaro (x,y)
     : jaro (x,y)
       + 0.1 * prefixMatch(x,y,prefixSize)
             * (1.0 - jaro(x,y))
```

```
boostThreshold = 0.7
    prefixSize = 4
```

```
jaro(DIXON, DICKSONX) = 0.767

jwProximity(DIXON, DICKSONX) = 0.767 + 0.1*2*(1 – 0.767)
                             = 0.767 + 0.2*0.233
                             = 0.767 + 0.0466
                             = 0.8136
```

Set-Based Metrics

Set-Based Metrics

Generate set of tokens from the strings

Measure similarity between the sets of tokens

Tokenizing a String

Words

Tokenizing a String

Words

q-grams: substrings of length q

“david smith” 3-grams
{##d, #da, dav, avi, …, h##}

Jaccard Measure

B_x = tokens(x)

B_y = tokens(y)

$$\text{jaccard}(x,y) = \frac{|B_x \cap B_y|}{|B_x \cup B_y|}$$

jaccard(dave, dav)

B_x = {#d, da, av, ve, e#}

B_y = {#d, da, av, v#}

jaccard(x,y) = 3/6

TF/IDF Measure

TF = term frequency

IDF = inverse document frequency

x = Apple Corporation, CA

y = IBM Corporation, CA

z = Apple Corp

…

blah blah Corporation

$$\text{lev}(x, y) \; \boxed{\begin{matrix} > \\ = \\ < \end{matrix}} \; \text{lev}(x, z)$$

???

TF/IDF Measure

TF = term frequency

IDF = inverse document frequency

x = Apple Corporation, CA

y = IBM Corporation, CA

z = Apple Corp

…

blah blah Corporation

lev(x, y) > lev(x, z)

… but intuitively (x, z) is a better match

TF/IDF Measure

TF = term frequency

IDF = inverse document frequency

x = Apple Corporation, CA

y = IBM Corporation, CA

z = Apple Corp

…

blah blah Corporation

frequent term should not carry as much weight

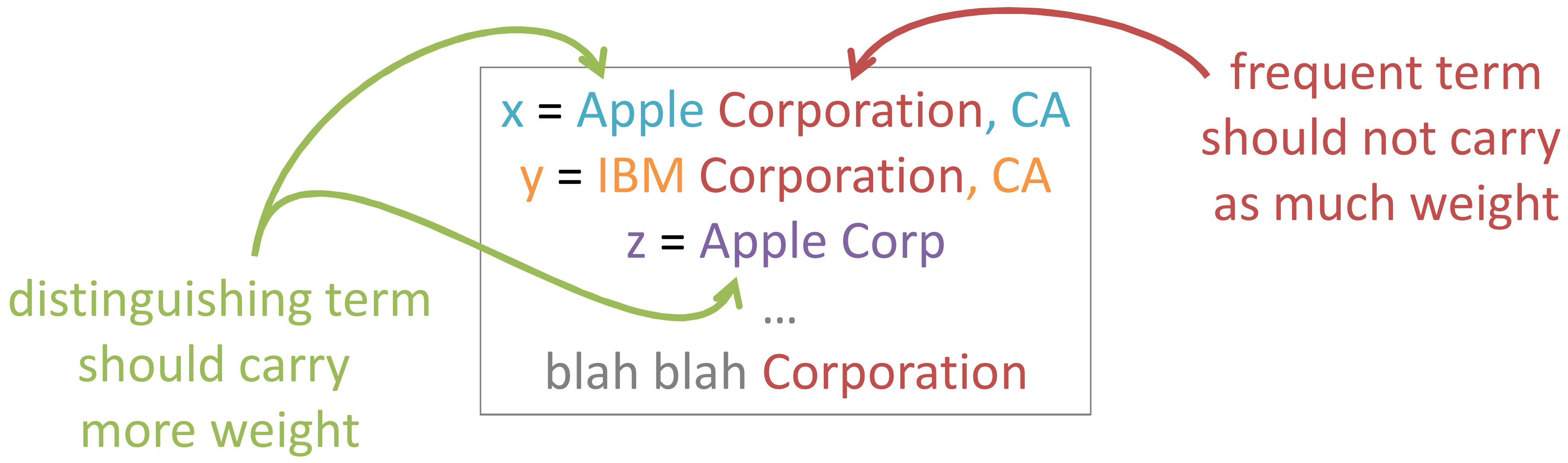
lev(x, y) > lev(x, z)

… but intuitively (x, z) is a better match

TF/IDF Measure

TF = term frequency

IDF = inverse document frequency



x = Apple Corporation, CA

y = IBM Corporation, CA

z = Apple Corp

…

blah blah Corporation

distinguishing term should carry more weight
frequent term should not carry as much weight

lev(x, y) **>** lev(x, z)

… but intuitively (x, z) is a better match

Term Frequencies and Inverse Document Frequencies

- Assume x and y are taken from a collection of strings
- Each string is converted into a bag of terms called a document
- term frequency tf(t,d) =
  - number of times term t appears in document d
- inverse document frequency idf(t) =
  - N / N_d, number of documents in collection divided by number of documents that contain t
  - note: in practice, idf(t) is often defined as $\log(N / N_d)$

slide from Anhai Doan, Alon Halevy, Zachary Ives

Example

$x = aab \Rightarrow B_x = \{a, a, b\}$

$y = ac \Rightarrow B_y = \{a, c\}$

$z = a \Rightarrow B_z = \{a\}$

$tf(a, x) = 2$	$idf(a) = 3/3 = 1$
$tf(b, x) = 1$	$idf(b) = 3/1 = 3$
...	$idf(c) = 3/1 = 3$
$tf(c, z) = 0$	

Feature Vectors

- Each document d is converted into a feature vector v_d
- v_d has a feature $v_d(t)$ for each term t
  - value of $v_d(t)$ is a function of TF and IDF scores
  - here we assume $v_d(t) = tf(t,d) * idf(t)$

$x = aab \Rightarrow B_x = \{a, a, b\}$

$y = ac \Rightarrow B_y = \{a, c\}$

$z = a \Rightarrow B_z = \{a\}$

$tf(a, x) = 2$ $\quad$ $idf(a) = 3/3 = 1$

$tf(b, x) = 1$ $\quad$ $idf(b) = 3/1 = 3$

... $\quad$ $idf(c) = 3/1 = 3$

$tf(c, z) = 0$

	a	**b**	**c**
$\mathbf{v_x}$	2	3	0
$\mathbf{v_y}$	3	0	3
$\mathbf{v_z}$	3	0	0

TF/IDF Similarity Score

- Let p and q be two strings, and T be the set of all terms in the collection
- Feature vectors $\mathbf{v}_p$ and $\mathbf{v}_q$ are vectors in the |T|-dimensional space wher each dimension corresponds to a term
- TF/IDF score of p and q is the cosine of the angle between $\mathbf{v}_p$ and $\mathbf{v}_q$
  - s(p,q) = $\sum_{t \in T} v_p(t) * v_q(t) \; /[\sqrt{\sum_{t \in T} v_p(t)^2} * \sqrt{\sum_{t \in T} vq(t)^2}]$

TF/IDF Similarity Score

- Score is high if strings share many frequent terms
  - terms with high TF scores
- Unless these terms are common in other strings
  - i.e., they have low IDF scores
- Dampening TF and IDF as commonly done in practice
  - use v_d(t) = log(tf(t,d) + 1) * log(idf(t)) instead of
    v_d(t) = tf(t,d) * idf(t)
- Normalizing feature vectors
  - v_d(t) = v_d(t) / $\sqrt{\sum_{\{t \in T\}} v_d(t)^2}$

Hybrid Similarity Measures

Hybrid Measures

Do the set-based thing
but
use a similiarity metric for each element of the set

x = Apple Corporation, CA
y = IBM Corporation, CA
z = Aple Corp
…
blah blah Corporation

Aple mispelt

Generalized Jaccard Measure

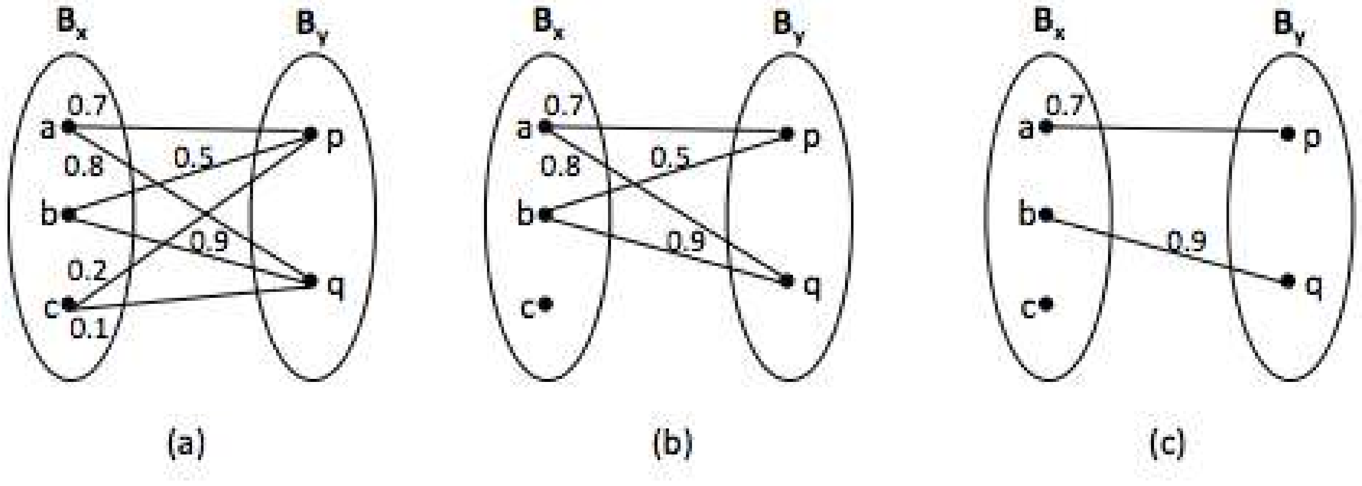
- Jaccard measure
  - considers overlapping tokens in both x and y
  - a token from x and a token from y must be identical to be included in the set of overlapping tokens
  - this can be too restrictive in certain cases
- Example:
  - matching taxonomic nodes that describe companies
  - “Energy & Transportation” vs. “Transportation, Energy, & Gas”
  - in theory Jaccard is well suited here, in practice Jaccard may not work well if tokens are commonly misspelled
    - e.g., energy vs. eneryg
  - generalized Jaccard measure can help such cases

Generalized Jaccard Measure

- Let $B_x = \{x_1, \ldots, x_n\}$, $B_y = \{y_1, \ldots, y_m\}$
- Step 1: find token pairs that will be in the "softened" overlap set
  - apply a similarity measure s to compute sim score for each pair (x_i, y_j)
  - keep only those score > a given threshold α, this forms a bipartite graph G
  - find the maximum-weight matching M in G
- Step 2: return normalized weight of M as generalized Jaccard score
  - $GJ(x,y) = \sum_{(xi,yj) \text{ in } M} s(x_i,y_j) / (|B_x| + |B_y| - |M|)$

slide from Anhai Doan, Alon Halevy, Zachary Ives

Generalized Jaccard Example



α = 0.5

- Generalized Jaccard score: (0.7 + 0.9)/(3 + 2 – 2) = 0.53

slide from Anhai Doan, Alon Halevy, Zachary Ives

The Soft TF/IDF Measure

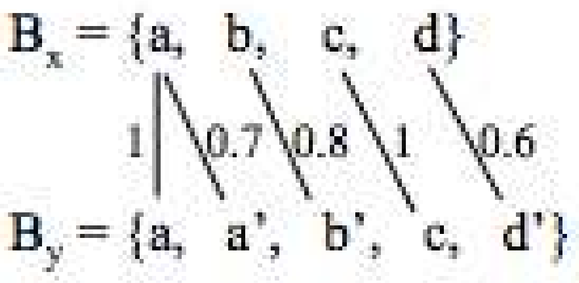
- Similar to generalized Jaccard measure, except that it uses TF/IDF measure as the "higher-level" sim measure
  - e.g., "Apple Corporation, CA", "IBM Corporation, CA", and "Aple Corp", with Apple being misspelled
- Step 1: compute close(x,y,k):
  set of all terms t ϵ B_x that have at least one close term u ϵ B_y, i.e., s'(t,u) >= k
  - s' is a basic sim measure (e.g., Jaro-Winkler), k prespecified
- Step 2: compute s(x,y) as in traditional TF/IDF score, but weighing each TF/IDF component using s'
  - $s(x,y) = \sum_{t \epsilon \text{close}(x,y,k)} v_x(t) * v_y(u^*) * s'(t,u^*)$
  - $u^* \epsilon B_y$ maximizes $s'(t,u)$ $\forall$ $u \epsilon B_y$

Soft TF/IDF Example

$x = abcd$

$y = aa'b'cd'$

(a)



$close(x, y, 0.75) = \{a, b, c\}$

(b)

$$
\begin{aligned}
s(x,y) = {} & v_x(a) \,.\, v_y(a) \,.\, 1 + \\
& v_x(b) \,.\, v_y(b') \,.\, 0.8 + \\
& v_x(c) \,.\, v_y(c) \,.\, 1
\end{aligned}
$$

(c)

Monge-Elkan Measure

- Break strings x and y into multiple substrings
  - x = A_1 ... A_n , y = B_1 ... B_m
- Compute
  - $s(x,y) = 1/n * \sum_{i=1}^{n} max_{j=1}^{m} s'(\mathbf{A}_i,\mathbf{B}_j)$
  - s' is a secondary sim measure, such as Jaro-Winkler
  - Intuitively, we ignore the order of the matching of substrings and only consider the best match for substrings of x in y

Monge-Elkan Measure

$$s(x,y) = 1/n * \sum_{i=1}^{n} max_{j=1}^{m} s'(\mathbf{A_i},\mathbf{B_j})$$

$x = A_1A_2$ $y = B_1B_2B_3$

$$\frac{\max(s'(A_1,B_1), s'(A_1,B_2), s'(A_1,B_3)) + \max(s'(A_2,B_1), s'(A_2,B_2), s'(A_2,B_3))}{2}$$

s' could be any metric, e.g., levenshtein

Monge-Elkan Measure

$$s(x,y) = 1/n * \sum_{i=1}^{n} max_{j=1}^{m} s'(A_i,B_j)$$

x = Comput. Sci. and Eng. Dept., University of California, San Diego

y = Department of Computer Science, Univ. of Calif., San Diego

what s' should we use?

levenshtein
needleman-wunch
affine-gap
smith-waterman
jaro
jaro-winkler

Phonetic Similarity Measures

Phonetic Similarity Measures

- Match strings based on their sound, instead of appearances
- Very effective in matching names, which often appear in different ways that sound the same
  - e.g., Meyer, Meier, and Mire; Smith, Smithe, and Smythe
- Soundex is most commonly used

The Soundex Measure

- Used primarily to match surnames
  - maps a surname x into a 4-letter code
  - two surnames are judged similar if share the same code
- Algorithm to map x into a code:
  - Step 1: keep the first letter of x, subsequent steps are performed on the rest of x
  - Step 2: remove all occurences of W and H. Replace the remaining letters with digits as follows:
    - replace B, F, P, V with 1, C, G, J, K, Q, S, X, Z with 2, D, T with 3, L with 4, M, N with 5, R with 6
  - Step 3: replace sequence of identical digits by the digit itself
  - Step 4: Drop all non-digit letters, return the first four letters as the soundex code

The Soundex Measure

- Example: x = Ashcraft
  - after Step 2: A226a13, after Step 3: A26a13, Step 4 converts this into A2613, then returns A261
  - Soundex code is padded with 0 if there is not enough digits
- Example: Robert and Rupert map into R163
- Soundex fails to map Gough and Goff, and Jawornicki and Yavornitzky
  - designed primarily for Caucasian names, but found to work well for names of many different origins
  - does not work well for names of East Asian origins
    - which uses vowels to discriminate, Soundex ignores vowels

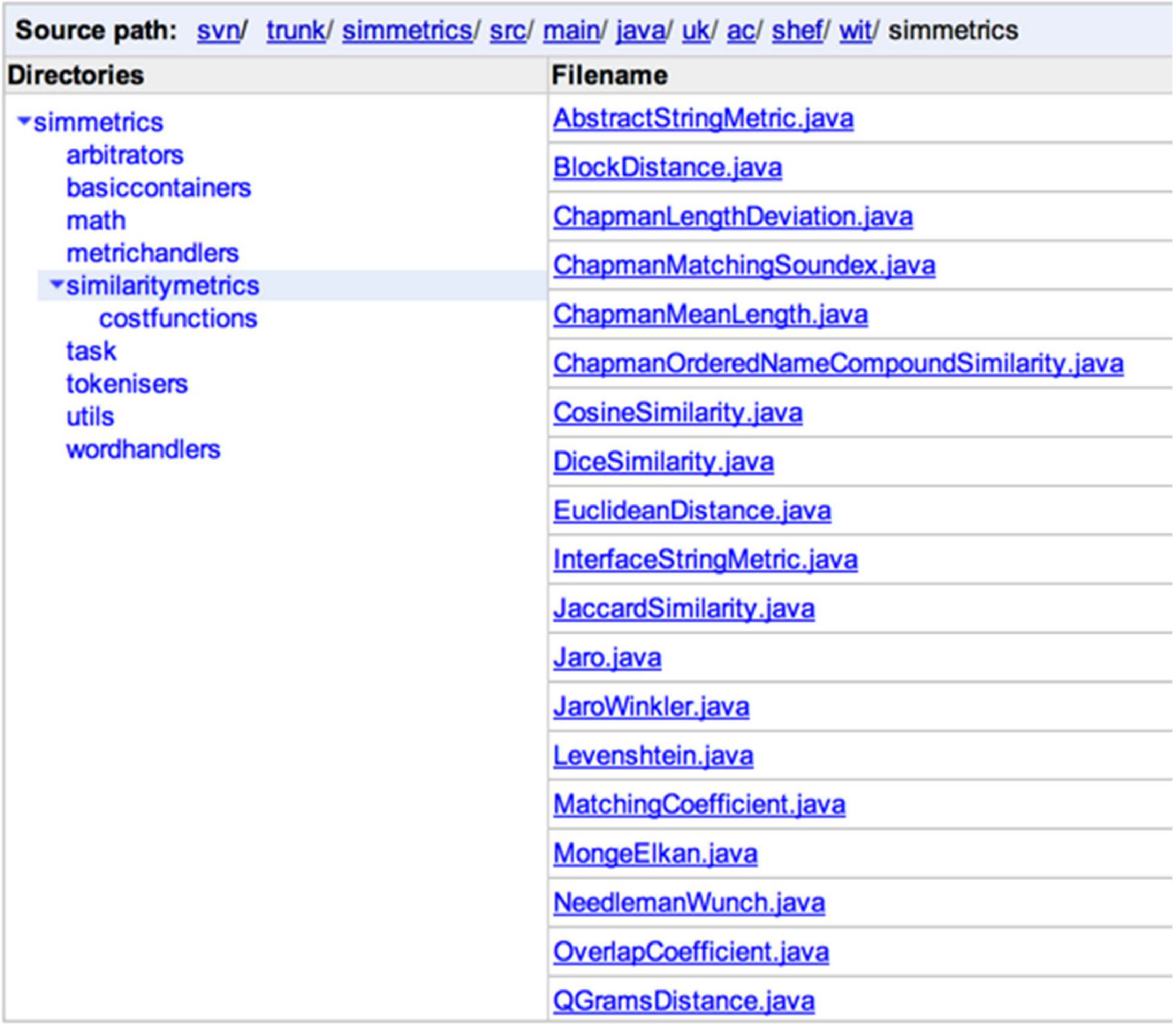
Other Readings

- http://en.wikipedia.org/wiki/String_metric
- http://en.wikipedia.org/wiki/Approximate_string_matching
- http://en.wikipedia.org/wiki/Edit_distance
- http://en.wikipedia.org/wiki/Levenshtein_distance
- http://en.wikipedia.org/wiki/Jaro–Winkler_distance
- http://en.wikipedia.org/wiki/Smith–Waterman_algorithm
- http://en.wikipedia.org/wiki/Jaccard_index

- http://alias-i.com/lingpipe/demos/tutorial/stringCompare/read-me.html
- http://www.gettingcirrius.com/2011/01/calculating-similarity-part-2-jaccard.html
- http://en.wikipedia.org/wiki/Sequence_alignment#Pairwise_alignment

Software

- http://code.google.com/p/java-similarities/source/browse/trunk/simmetrics/src/main/java/uk/ac/shef/wit/simmetrics/
- http://sourceforge.net/projects/secondstring/
- http://planetcalc.com/1721/ (Levenshtein calculator)



Source path: svn/ trunk/ simmetrics/ src/ main/ java/ uk/ ac/ shef/ wit/ simmetrics

Directories	Filename
▾simmetrics	AbstractStringMetric.java
arbitrators	BlockDistance.java
basiccontainers	ChapmanLengthDeviation.java
math	ChapmanMatchingSoundex.java
metrichandlers	ChapmanMeanLength.java
▾similaritymetrics	ChapmanOrderedNameCompoundSimilarity.java
costfunctions	CosineSimilarity.java
task	DiceSimilarity.java
tokenisers	EuclideanDistance.java
utils	InterfaceStringMetric.java
wordhandlers	JaccardSimilarity.java
	Jaro.java
	JaroWinkler.java
	Levenshtein.java
	MatchingCoefficient.java
	MongeElkan.java
	NeedlemanWunch.java
	OverlapCoefficient.java
	QGramsDistance.java