

DSE 203 (Fall 2020)

The Problem of Value Matching

The Problem – String Values

- Find all authors who are in the USBooks database and in the WorldBooks database
- One Recipe
 - Find author Salutation, FName, MI, LName, Suffix from WorldBooks
 - Concatenate them, with intermediate spaces, into one string, thus forming a list of names
 - Store the list as a single column temporary table in USBooks
 - Natural Join with the author names of USBooks and report matches
- What can go wrong with this recipe?
 - 'Dr. Maya Angelou' \neq 'Maya Angelou'
 - 'Pearl Sydenstricker Buck' \neq 'Pearl S. Buck'
 - 'Mary Higgins Clark' \neq 'Higgins-Clark, Mary'

} Need inexact matching

The Problem – Numeric Values

Column A of Table R

R.A
11
13
15
120
-3

Column B of Table S

S.B
23
10
11
14
473

```
SELECT R.*  
FROM R INNER JOIN S  
ON R.A  
BETWEEN S.B- 1 AND S.B+ 2  
WHERE R.A < 100
```

A kind of Non-equijoin
Called **Band Join**

Mismatch Between Query and Data

- Find movies starring "Schwarzenegger"
 - Missing a 'g'

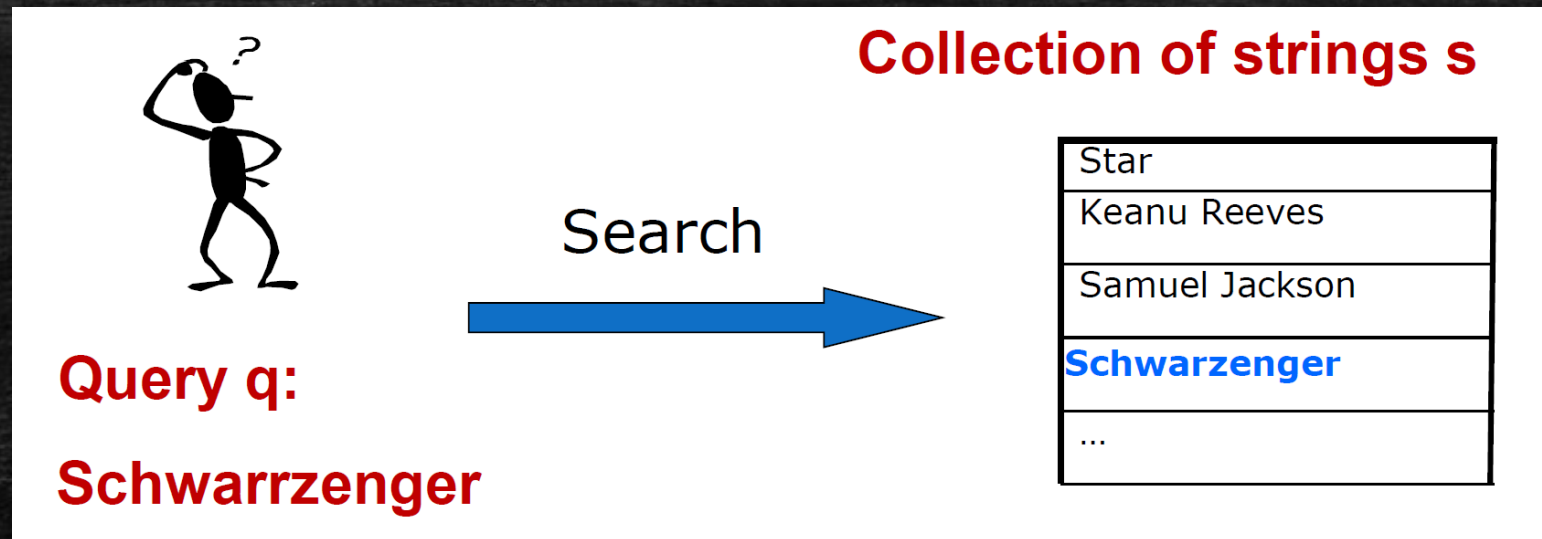
Star	Title	Year	Genre
Keanu Reeves	The Matrix	1999	Sci-Fi
Samuel Jackson	Iron man	2008	Sci-Fi
Schwarzenegger	Terminator: Dark Fate	2019	Sci-Fi
Samuel Jackson	The man	2006	Crime

What causes a Mismatch?

1. Phonetic variations: *Kohl's* versus *Coles*
2. Typographical mistakes: *Microsoft* vs. *Microsft*
3. Contextual differences: *Company* vs. *Commercial For-Profit Organization*
4. Reordered terms: *Sam Hopkins* vs. *Hopkins, Sam*
5. Prefixes and suffixes: *Amarnath Gupta* vs. *Dr. Amarnath Gupta*
6. Abbreviations, nicknames, and initials: *IBM* vs. *International Business Machines*
7. Cross-language semantics: *Private limited* vs. *私人有限*
8. Transliteration differences: *Corriander* vs. *Cilantro*, *Potato Chips* vs. *Potato Crisps*
9. Truncated letters and missing or extra spaces: *Chu Kong Transport Company* vs. *ChuKong Transport Co. Ltd.*
10. Taxonomic Difference: *Coffee* vs. *Latte*

Basic Solution

- Since “exact matches” do not work use “approximate matching”



- Output: string s such that $\text{dist}(q, s) \leq \delta$
- The $\text{dist}()$ function can be edit distance, Jaccard coefficient, cosine similarity, ...

Example – 1 Edit Distance

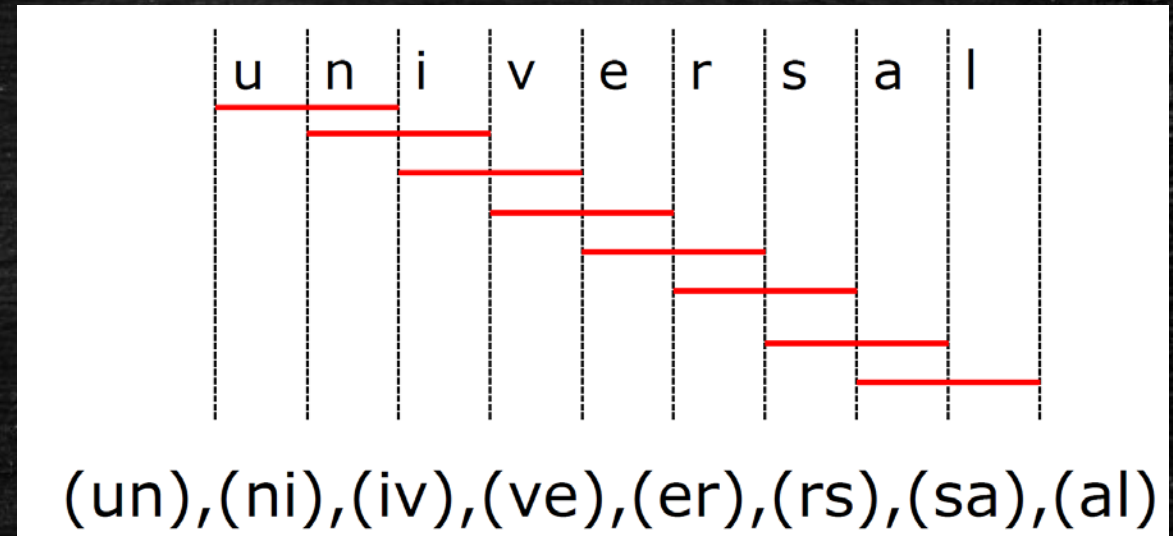
- Possibly the most commonly used metric to define string similarity
 - Also known as Levenshtein distance
- $Ed(s1,s2)$ = minimum # of operations (insertion, deletion, substitution) to change $s1$ to $s2$.
- $d(x, y)$ computes minimal cost of transforming x into y , using a sequence of operators, each with cost 1
 - Delete a character
 - Insert a character
 - Substitute a character with another
- Example: x = David Smiths, y = Davidd Simth, $d(x, y) = 4$, using following sequence
 - Inserting a character d (after David)
 - Substituting m by i
 - Substituting i by m
 - Deleting the last character of x , which is s

Edit Distance

- Models common editing mistakes
- Inserting an extra character, swapping two characters, etc.
- So smaller edit distance means higher similarity
- Can be converted into a similarity measure
 - $s(x, y) = 1 - d(x, y) / [\max(\text{length}(x), \text{length}(y))]$
 - $s(\text{David Smiths}, \text{Davidd Simth}) = 1 - 4 / \max(12, 12) = 0.67$
- Pro
 - Computation is fast using dynamic programming
- Con
 - Context insensitivity \rightarrow Amarnath vs. Armanath vs. Smarpath vs. Amaranth
 - Equal weight to all errors \rightarrow Catherine vs. Katherine vs. Xatherine

Example 2 – Jaccard Coefficient

- We are given strings s_1 and s_2
- Construct n -grams for each – $g(s_1)$ and $g(s_2)$
- $\text{Jaccard}(s_1, s_2) = g(s_1) \cap g(s_2) / g(s_1) \cup g(s_2)$
- $s_1 = \text{dave}, s_2 = \text{dav}$
- Assuming $n = 2$
 - $g(s_1) = \{\#d, da, av, ve, e\# \}$
 - $g(s_2) = \{\#d, da, av, v\# \}$
 - $\text{Jaccard}(s_1, s_2) = 3/6 = 0.5$
- Very common in practice



Example 3 – Cosine Distance

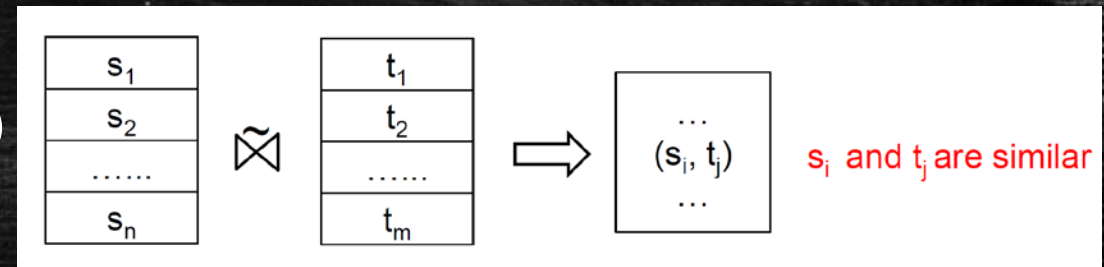
$$\text{cosine}(A, B) = \frac{\sum_{i=1}^n (A_i B_i)}{\sqrt{\sum_{i=1}^n (A_i^2) \sum_{i=1}^n (B_i^2)}}$$

- Strings
 - S₁ = Tom Hanks
 - S₂ = Ton Hank
 - 3-Gram(S₁):{Tom,om_,m_H,_Ha,Han,ank,nks}
 - 3-Gram(S₂):{Ton,on_,n_H,_Ha,Han,ank}
 - Cosine(S₁,S₂)= 3/sqrt(6x7) =0.46
- } N-gram starting from char 1

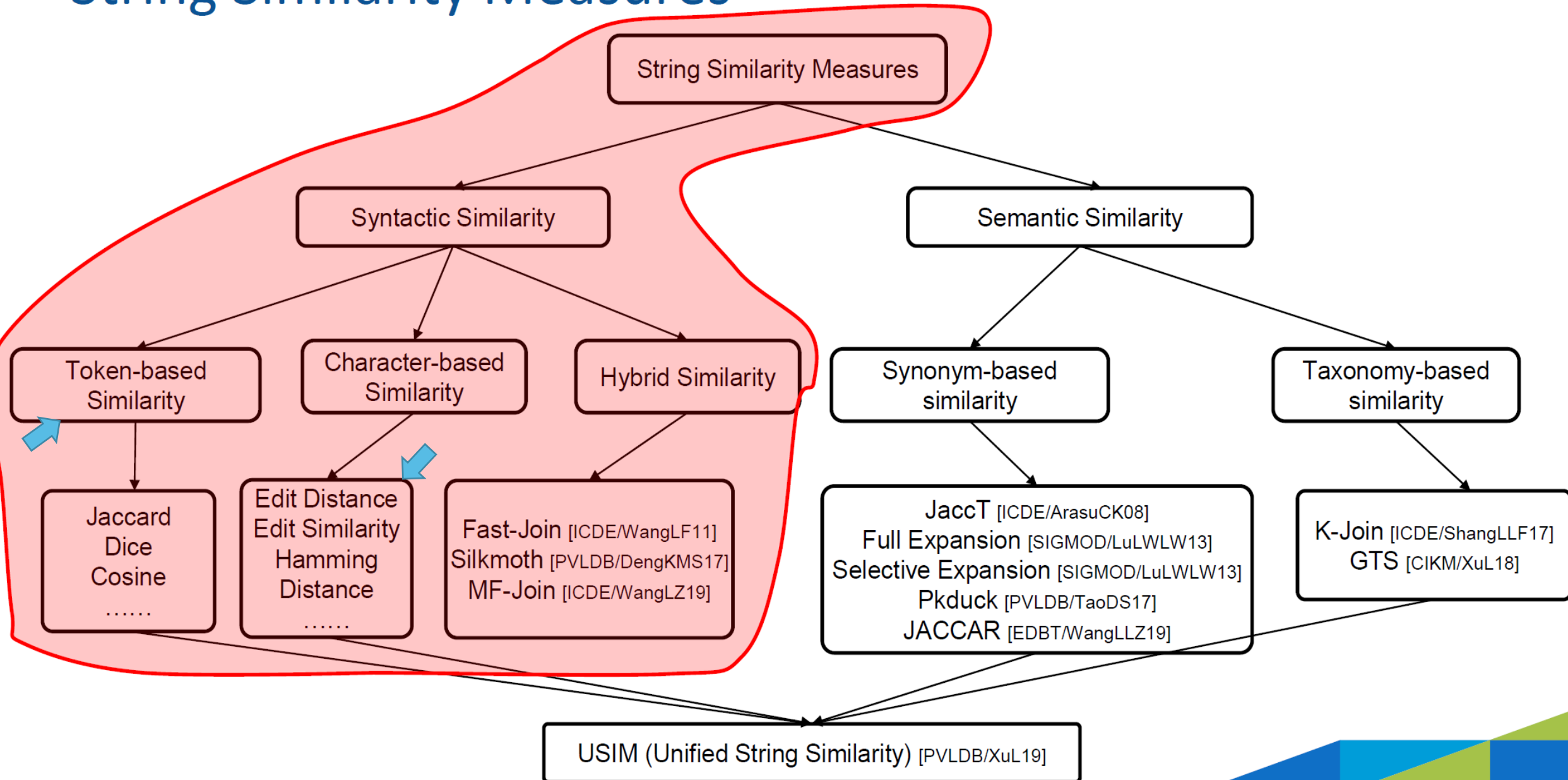
For some applications, a **word n-gram** is good enough
 Similarity Matching need to determine the **Tokenization Scheme**

How to Choose a Similarity *Strategy*?

- Accuracy
 - Does this similarity measure actually give good results *for this application*?
 - Measuring accuracy
 - Precision and Recall
 - Precision: proportion of true positive *within* the results returned
 - Recall: proportion of total number of correct results that have been returned
 - Test with several samples
- Performance/Scalability (esp. for joins)
 - If we are matching N strings with M strings
 - Can we avoid doing N X M comparisons?
 - Are there some techniques that will reduce the number of comparisons without sacrificing accuracy?



String Similarity Measures



Needleman-Wunch Measure

- Generalizes Levenshtein edit distance
- Basic idea
 - defines notion of alignment between x and y
 - assigns score to alignment
 - return the alignment with highest score
- Alignment: set of correspondences between characters of x and y , allowing for gaps

d	-	-	v	a
d	e	e	v	e

Gap Penalty

- Use a score matrix and a gap penalty
 - alignment score = sum of scores of all correspondences -
sum of penalties of all mismatches and gaps

d	-	-	v	a
d	e	e	v	e

- For the above alignment, it is 2 (for d-d) + 2 (for v-v) -1 (for a-e) -2 (for gap) = 1
- This is the alignment with the highest score, it is returned as the Needleman-Wunch score for dva and deeve.

Computing the NW Measure

- Computes similarity scores instead of distance values
- Generalizes edit costs into a **score matrix**
 - allowing for more fine-grained score modeling
 - e.g., $\text{score}(o,o) > \text{score}(a,o)$
- Generalizes insertion and deletion into gaps, and generalizes their costs from 1 to C_g

$$s(i,j) = \max \begin{cases} s(i-1,j-1) + c(x_i,y_j) \\ s(i-1,j) - c_g \\ s(i,j-1) - c_g \end{cases}$$

$$s(0,j) = -jc_g$$
$$s(i,0) = -ic_g$$

		d	e	e	v	e
	0	-1	-2	-3	-4	-5
d	-1	2	1	0	-1	-2
v	-2	1	1	0	2	1
a	-3	0	0	0	1	1

Tracing Arrows

- Mark a path from the cell on the bottom right back to the cell on the top left by following the direction of the arrows.
- From this path, the sequence is constructed by these rules:
 - A diagonal arrow represents a match or mismatch, so the letter of the column and the letter of the row of the origin cell will align.
 - A horizontal or vertical arrow represents an indel (insert/delete).
 - Horizontal arrows will align a gap ("-") to the letter of the row (the "side" sequence).
 - Vertical arrows will align a gap to the letter of the column (the "top" sequence).
- If there are multiple arrows to choose from, they represent a branching of the alignments.
- If two or more branches all belong to paths from the bottom right to the top left cell, they are equally viable alignments. In this case, note the paths as separate alignment candidates.

Needleman-Wunsch

match = 1 mismatch = -1 gap = -1

		G	C	A	T	G	C	U	
		0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5	
A	-2	0	0	1	0	-1	-2	-3	
T	-3	-1	-1	0	2	1	0	-1	
T	-4	-2	-2	-1	1	1	0	-1	
A	-5	-3	-3	-1	0	0	0	-1	
C	-6	-4	-2	-2	-1	-1	1	0	
A	-7	-5	-3	-1	-2	-2	0	0	

The alignment path is highlighted with blue arrows, showing the sequence GCGTTC aligned with AATCA. The alignment is as follows:

	G	C	G	T	T	C	A
G	1	0	-1	-2	-3	-4	-5
A	0	0	1	0	-1	-2	-3
T	-1	-1	0	2	1	0	-1
T	-2	-2	-1	1	1	0	-1
A	-3	-3	-1	0	0	0	-1
C	-4	-2	-2	-1	-1	1	0
A	-5	-3	-1	-2	-2	0	0

Sequences Best alignments

-----	-----		
GCAATGCU	GCAATGCU	GCA-TGCU	GCA-TGCU
GATTACA	GATTACA	G-ATTACA	G-ATTACA

The Affine Gap Measure

- In practice, gaps tend to be longer than 1 character
- Assigning same penalty to each character unfairly punishes long gaps
- Solution: define cost of opening a gap vs. cost of continuing the gap
 - $\text{cost}(\text{gap of length } k) = c_0 + (k - 1)c_r$
 - c_0 = cost of opening gap
 - c_r = cost of continuing gap, $c_0 > c_r$
- E.g., "David Smith" vs. "David Richardson Smith"
 - $c_0 = 1, c_r = 0.5$, alignment cost = $6 * 2 - 1 - 9 * 0.5 = 6.5$

The Smith-Waterman Measure

- What will the previous alignment algorithms do for this case?
 - “Prof. John R. Smith, Univ of Wisconsin” vs. “Dr. John R. Smith, Professor”
- Global Alignment does not work
 - find two substrings of x and y that are most similar
 - We should find “John R. Smith” in the above case → local alignment

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(a_i, b_j) \\ S_{i-1,j} + s(a_i, -) \\ S_{i,j-1} + s(-, b_j) \\ 0 \end{cases} = \max \begin{cases} S_{i-1,j-1} + 1 & a_i = b_j \\ S_{i-1,j-1} + -1 & a_i \neq b_j \\ S_{i-1,j} + -2 & b_j = - \\ S_{i,j-1} + -2 & a_i = - \\ 0 \end{cases}$$

SW extends NW

- SW makes two key changes to Needleman-Wunch
 - allows the match to restart at any position in the strings (no longer limited to just the first position)
 - if global match dips below 0, then ignore prefix and restart the match
 - after computing matrix using recurrence equation, retracing the arrows from the largest value in matrix, rather than from lower-right corner
 - this effectively ignores suffixes if the match they produce is not optimal
 - retracing ends when we meet a cell with value 0 → start of alignment

<http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Smith-Waterman>

Jaro Measure

- Mainly for comparing short strings, e.g., first/last names

- The Jaro distance d_j of two given strings s_1 and s_2 is

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3} \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{otherwise} \end{cases}$$

– where:

- m is the number of *matching characters*
- t is half the number of *transpositions* (if the i -th common character of s_1 does not match the i -th common character of s_2 , then we have a transposition)
- Two characters from s_1 and s_2 respectively, are considered *matching* only if they are the same and not farther than

$$\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1.$$

- Each character of s_1 is compared with all its matching characters in s_2
 - The number of matching (but different sequence order) characters divided by 2 defines the number of *transpositions*

Example of Jaro Measure

- $s_1 = \text{DWAYNE}, s_2 = \text{DUANE}$
- So $m = 4, |s_1| = 6, |s_2| = 5$
- $t = 0$
- $\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1 = 6/2 - 1 = 2$
- $d_j = 1/3 (4/6 + 4/5 + (4 - 0)/4) = 0.822$

-
- $s_1 = \text{jon}, s_2 = \text{ojhn}$
 - Verify that $d_j = 0.81$

Jaro-Winkler Measure

- Uses a prefix scale p
 - Gives more favorable ratings to strings that match from the beginning for a set prefix length l
- Given two strings s_1 and s_2
 - the Jaro–Winkler similarity $d_w = d_j + lp(1 - d_j)$

Given the strings s_1 MARTHA and s_2 MARHTA we find:

- $m = 6$
- $|s_1| = 6$
- $|s_2| = 6$
- There are mismatched characters T/H and H/T leading to $t = \frac{2}{2} = 1$

We find a Jaro score of:

$$d_j = \frac{1}{3} \left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6} \right) = 0.944$$

To find the Jaro-Winkler score using the standard weight $p = 0.1$, we continue to find:

$$\ell = 3$$

Thus:

$$d_w = 0.944 + (3 * 0.1(1 - 0.944)) = 0.961$$

Generalized Jaccard Measure

- Jaccard measure
 - considers overlapping tokens in both s_1 and s_2
 - 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 when the strings have errors
- 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

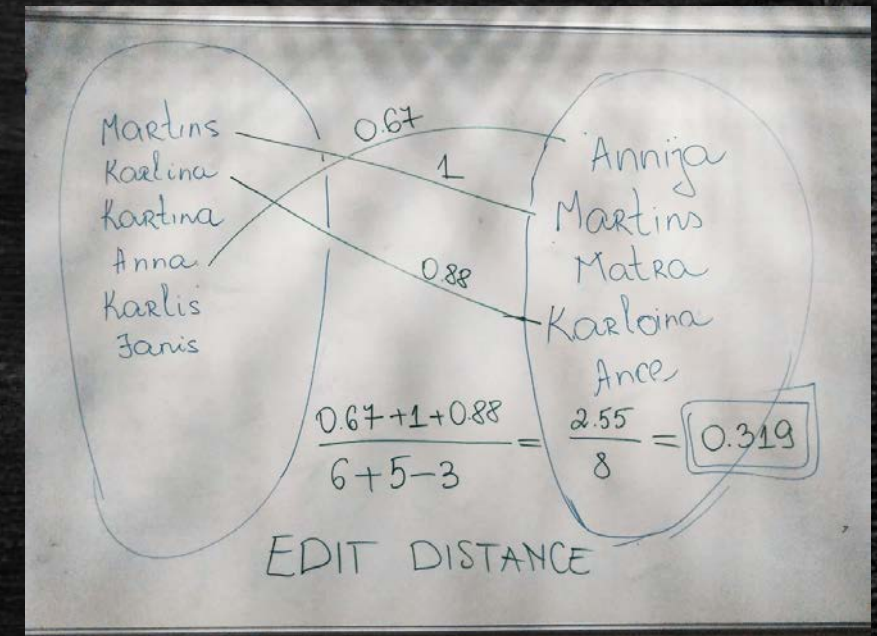
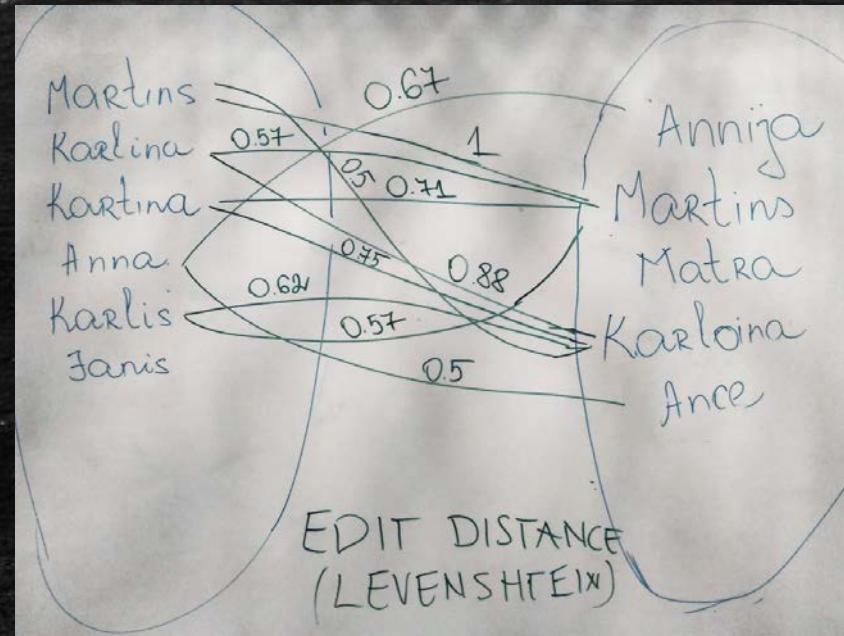
Generalized Jaccard Measure

- Let $B_x = \{x_1, \dots, x_n\}$, $B_y = \{y_1, \dots, y_m\}$
- Step 1: find token pairs that will be in the “softened” overlap set
 - apply some standard similarity measure s to compute sim score for each pair (x_i, y_j)
 - keep only those score \geq a given threshold α , this forms a bipartite graph G
 - find the maximum-weight matching M in G
 - In graph theory, a matching or independent edge set in a graph is a set of edges without common vertices
- Step 2: return normalized weight of M as generalized Jaccard score
 - $GJ(x, y) = \sum_{(x_i, y_j) \in M} s(x_i, y_j) / (|B_x| + |B_y| - |M|)$

Generalized Jaccard Measure

- $s_1 = \{\text{Kartina, Janis, Karlis, Martins, Anna, Karlina}\}$
- $s_2 = \{\text{Annija, Martins, Matra, Karloina, Ance}\}$

	EDIT distance	Jaro Winkler
kartina, annija	0.43	0.66
kartina, martins	0.71	0.81
kartina, matra	0.43	0.71
kartina, karloina	0.75	0.91
kartina, ance	0.14	0.46
janis, annija	0.33	0.7
janis, martins	0.43	0
janis, matra	0.2	0.47
janis, karloina	0.25	0.55
janis, ance	0.4	0.63
karlis, annija	0.17	0.56
karlis, martins	0.57	0.75
karlis, matra	0.17	0.58
karlis, karloina	0.62	0.89
karlis, ance	0.17	0.47
martins, annija	0.29	0.53
martins, martins	1	1



Choose a threshold of 0.5

Word Tokenization for Multiword Terms

- Words and Tokens
 - **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words
 - **Token** – An instance of a word or term occurring in a “document”
 - **Tokenization** – Splitting a sentence into a sequence of tokens
- Example
 - The Company employed approximately 201,000 people as of September 29, 2018

The	Compa ny	emplo yed	approximately	201	,	000	people	as	of	September	29	,	2018
-----	-------------	--------------	---------------	-----	---	-----	--------	----	----	-----------	----	---	------

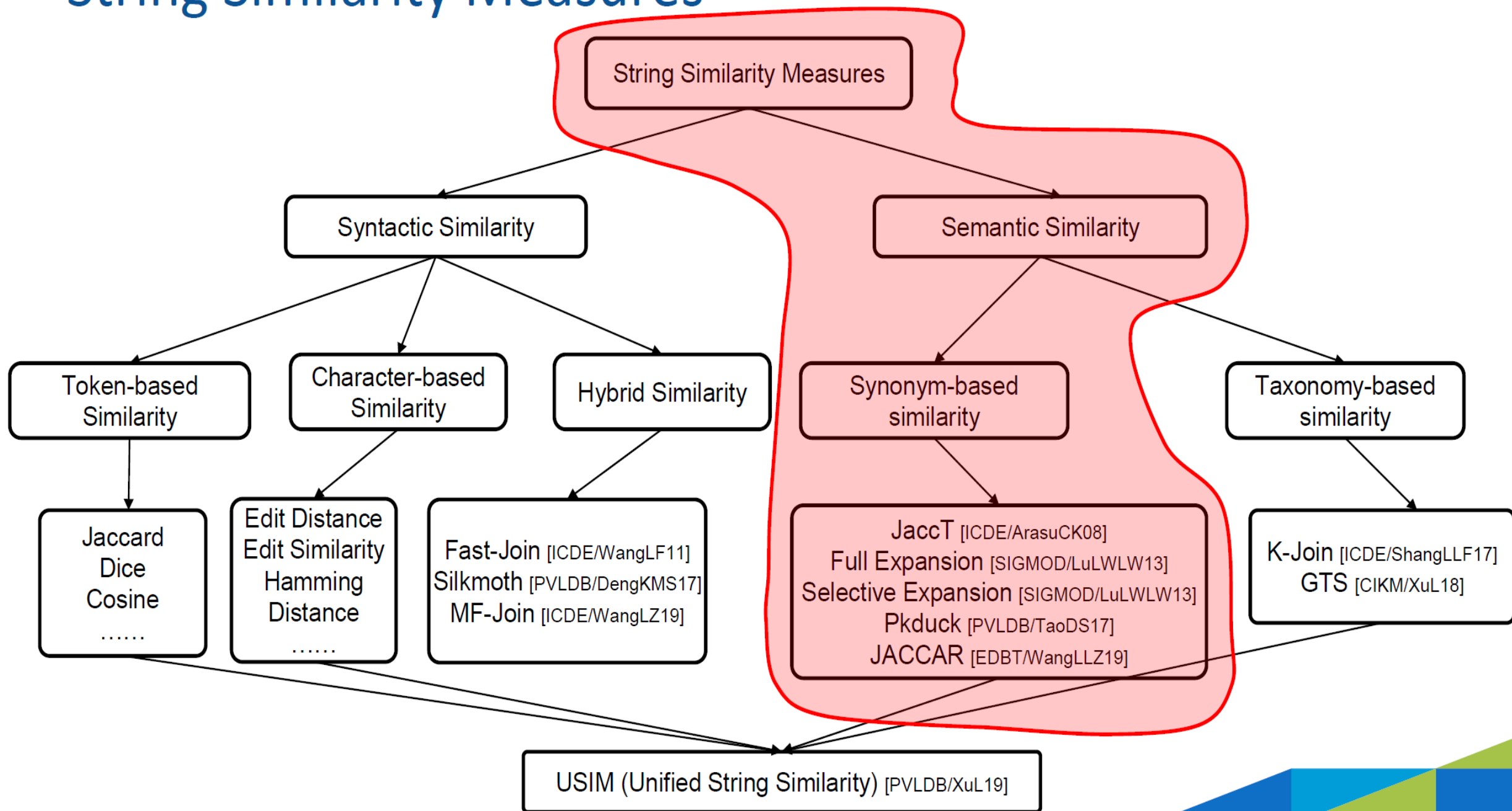
Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- the hold-him-back-and-drag-him-away maneuver
- data base
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University
- 3/20/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333

Tokenization is task dependent

- Issues to consider
 - A simple space separator is not enough: "San" and "Francisco"? "Mar 2015"?
 - Do we care about phrases? "San Francisco"? "New York"?
 - Special characters like "\$10"? Or hashtag on Twitter "#LU"
 - Apostrophes ('): "O'Connor" or "O" and "Connor"?
 - "rock 'n' roll", "Tom's place"
 - Commas: "100,000 dollars" or ("100" "000" "dollars")
 - Hyphens: "soon-to-be" or ("soon" "to" "be"), "Hewlett-Packard"
 - Email addresses, dates, URLs (Usually treated separately)
- In practice, no tokenization is perfect
- Tokenization software usually uses regular expressions in smart ways
 - Refer to chapter 3 of the NLTK book for a regular expression tokenizer

String Similarity Measures



Using Synonyms

- Synonyms, Abbreviations, Acronyms are used commonly
- We would like to match
 - UCSD <<->> University of California San Diego
 - AWS <<->> Amazon Web Service
 - CIKM <<->> ACM International Conference on Knowledge Management
<<->> ACM Int. Conf. on Knowl. Mgmt.
- How do we get these synonyms?
 - Obtained by machine learning models
 - Provided by domain experts
 - Extracted from knowledge bases (e.g., DBPedia)

Full Expansion

- Example Strings
 - S_1 = "ACM International Conference on Information and Knowledge Management China"
 - S_2 = "CIKM 2019 CN"
- Synonym Dictionary
 - CIKM = ACM International Conference on Information and Knowledge Management
 - China = CN
- Expanding all synonyms
 - S_1' = "ACM International Conference on Information and Knowledge Management China CIKM CN"
 - S_2' = "CIKM 2019 CN ACM International Conference on Information and Knowledge Management China "
 - $Jaccard(S_1', S_2') = 11/12 = 0.92$

String Transformation

- Transformation-based operation
- Enumerate all the transformed strings
 - Pick the pair with largest score

- Example

- $s = \text{"VLDB Conf"}$
 - $t = \text{"Large Database Conference"}$

$s_1 = \text{"VLDB Conf"}$

$s_2 = \text{"Very Large Database Conf"}$

$s_3 = \text{"VLDB Conference"}$

$s_4 = \text{"Very Large Database Conference"}$

$t_1 = \text{"Large Database Conference"}$

$t_2 = \text{"Large Database Conf"}$

- From synonym dictionary
 - VLDB \leftrightarrow Very Large Database
 - Conf \leftrightarrow Conference

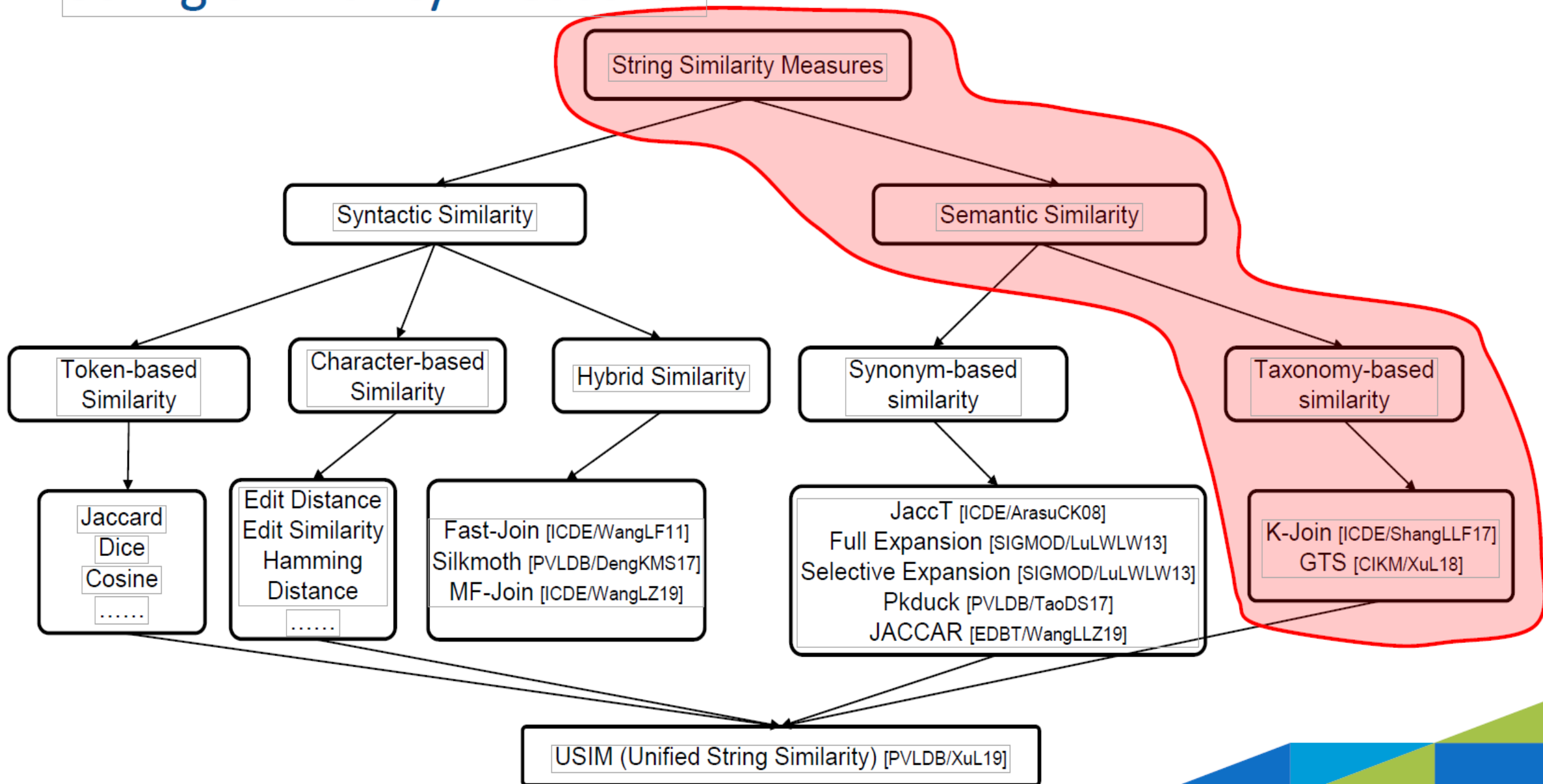
$$\text{SIM}(s, t) = \text{Jaccard}(s_4, t_1) = \frac{3}{4} = 0.75$$

We didn't really need to expand t!!

A Simple Trick

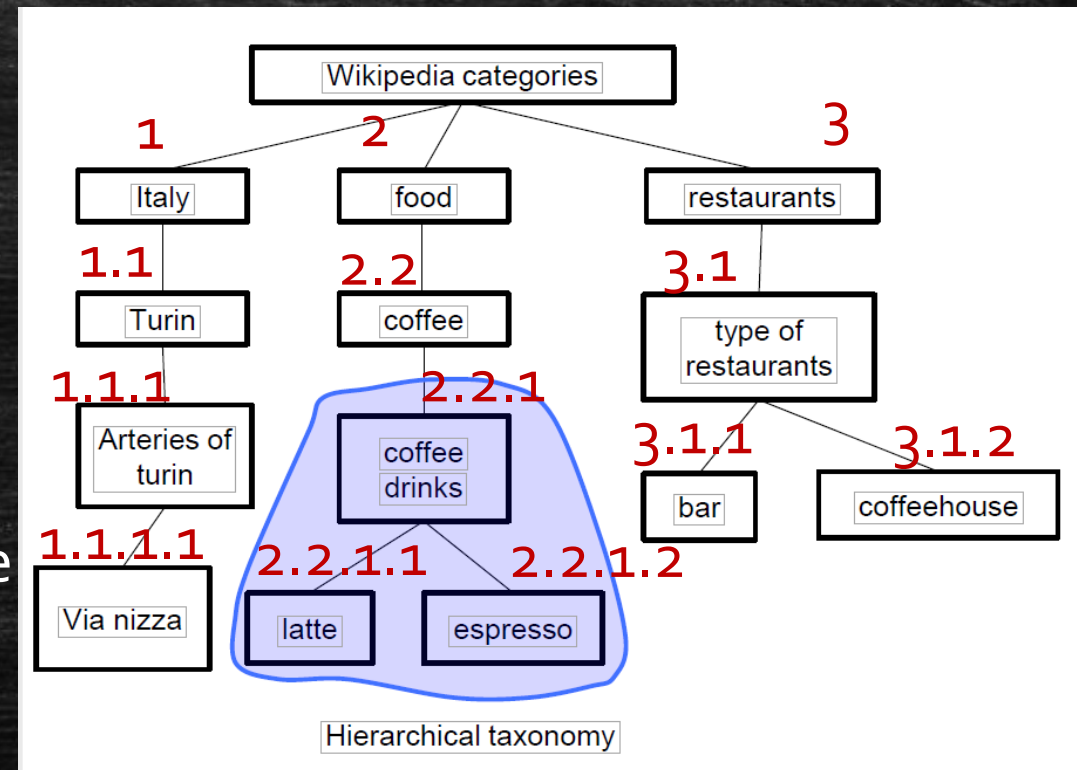
- Avoid the all-by-all comparison
- Apply the transformation on one side at a time
- Expand s and match with t – get a max matching score M
- Expand t and match with s – get a max matching score M'
- $\text{SIM}(s,t) = \max(M, M')$

String Similarity Measures



Taxonomic Substitution

- Hierarchies
 - Trees or DAGs
- Compute distance on the tree
 - s = "American latte"
 - t = "American espresso"
 - Tree distance = 2
- A Dewey Decimal System can be used to encode tree nodes



Using the Taxonomy

- Map your strings to the nearest nodes of the tree
 - American espresso <-> espresso
 - American latte <-> latte
- } From this point we will only consider the nodes
- The set similarity problem
 - $S : \{s_1, \dots, s_i\}$ and $T : \{t_1, \dots, t_j\}$ are two sets of nodes
 - Let $|s|$ refer to the depth of the node in the tree
 - Node closeness (taxonomic similarity) for any pair of nodes s, t
 - For sets S and T

$$TS(s, t) = \frac{|LCA(s, t)|}{\max(|s|, |t|)}$$

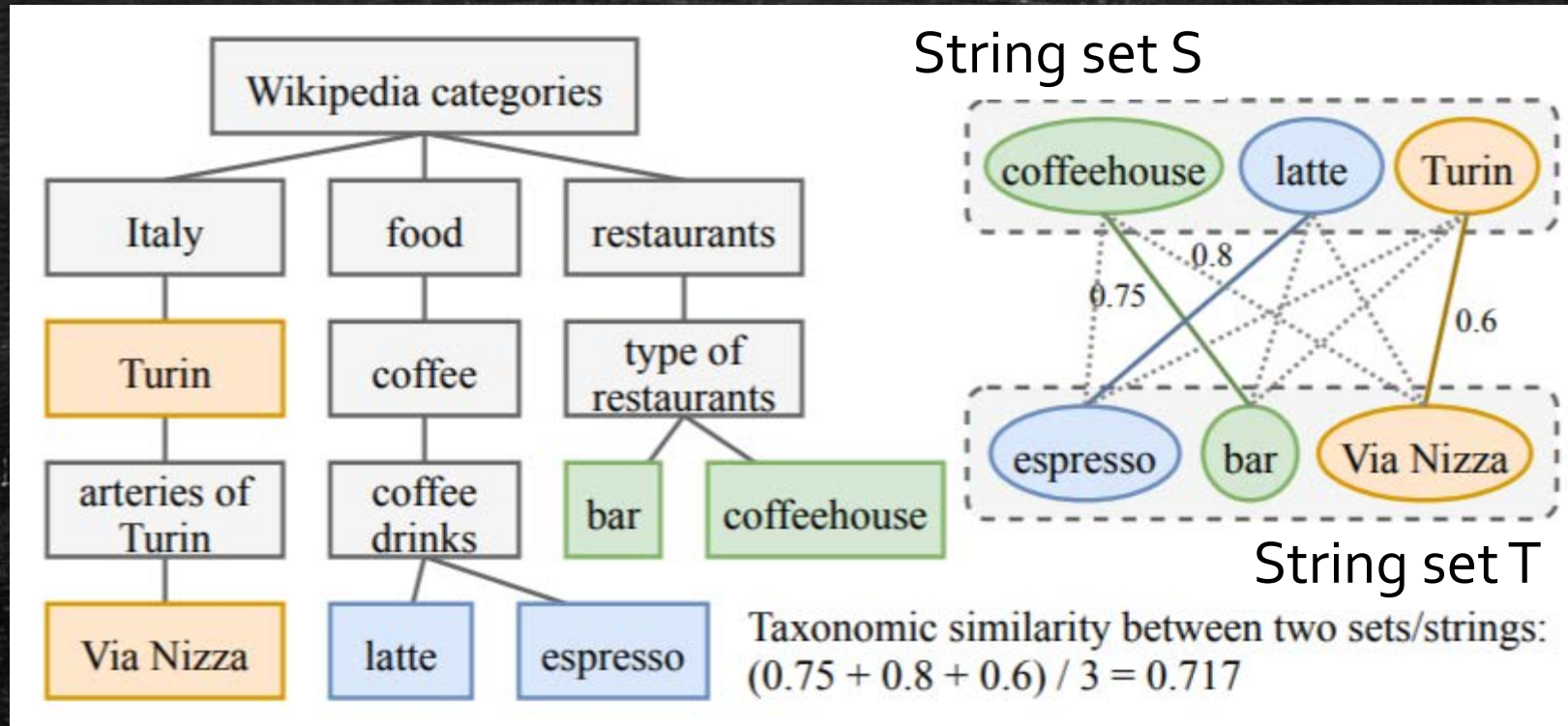
$$GTS(S, T) = \frac{W(S, T)}{\max(|S|, |T|)} = \frac{\max \sum_p \sum_q I_{pq} TS(s_p, t_q)}{\max(|S|, |T|)} \quad (2)$$

where $p \in [1, i], q \in [1, j], I_{pq} = 0$ or $1, \sum_p I_{pq} \leq 1$, and $\sum_q I_{pq} \leq 1$

$|S|$ = no. of nodes in S

Nodes will be selected only once

Example – Taxonomic Similarity



The distinctness criteria forbid any node from being selected more than once, e.g., selecting both ("latte", "espresso") and ("latte", "Turin") are not allowed

Scalability of String Matching

- Philosophy
 - Avoid an all-to-all matching
 - Create a method to identify candidates that should be matched
 - Apply similarity/distance measure only on the candidate set
- Strategy: Filter-and-Verification framework
 - If two strings are similar, they must have at least T common tokens or q -grams

Example: Size/Count Filter

- The intuition
 - Match only strings of comparable size
- Retrieves only strings in Y whose sizes make them match candidates
 - given a string $x \in X$, infer a constraint on the size of strings in Y that can possibly match x
 - uses a B-tree index to retrieve only strings that satisfy size constraints
- For Jaccard measure $J(x,y) = |x \cap y| / |x \cup y|$
 - Assume two strings x and y match if $J(x,y) \geq t$
 - One can show that given a string $x \in X$, only strings y such that $|x|/t \geq |y| \geq |x|*t$ can possibly match x
 - $|x|$ is the number of tokens in X

Threshold for Size/Count Filters

Token-based similarity

- Jaccard similarity: $T = \frac{\tau}{1 + \tau}(|r| + |s|)$;
- Cosine similarity: $T = \tau \sqrt{|r| \cdot |s|}$;
- Dice similarity: $T = \frac{\tau}{2}(|r| + |s|)$.

Character-based similarity

- Edit distance: $T = \max(|r|, |s|) - q + 1 - \tau * q$;
- Hamming distance: $T = \max(|r|, |s|) - q + 1 - \tau * q$;
- Edit similarity: $T = \max(|r|, |s|) - q + 1 - (\max(|r|, |s|) * (1 - \tau) * q)$.

s = "Information and Knowledge Management

t = "Management of Knowledge Base

Threshold $\tau = 0.6$

$T = 0.6/1.6*(4+4) = 3$ – need 3 common tokens

- Have 2 – so this pair is not to be considered

Prefix Filtering

- Key idea: if two sets share many terms \rightarrow large subsets of them also share terms
- Consider overlap measure $O(x,y) = |x \cap y|$
 - if $|x \cap y| \geq k \rightarrow$ any subset $x' \subset x$ of size at least $|x| - (k - 1)$ must overlap y
- To exploit this idea to find pairs (x,y) such that $O(x,y) \geq k$
 - given x , construct subsets x' of size $|x| - (k - 1)$
 - use an inverted index to find all y that overlap x'

Selecting the Subset Intelligently

- Recall that we select a subset x' of x and check its overlap with the entire set y
- We can do better by selecting a particular subset x' and checking its overlap with only a particular subset y' of y
- How?
 - impose an ordering O over the universe of all possible terms
 - e.g., in increasing frequency
 - reorder the terms in each $x \in X$ and $y \in Y$ according to O
 - refer to subset x' that contains the first n terms of x as the prefix of size n of x

Prefix Filtering Example

R

r_1	$\{e_1, e_5, e_6, e_8, e_9\}$
r_2	$\{e_1, e_2, e_6, e_7, e_8\}$
r_3	$\{e_1, e_2, e_4, e_7, e_8\}$
r_4	$\{e_1, e_2, e_4, e_6, e_8\}$
r_5	$\{e_4, e_5, e_6, e_7, e_9\}$


S

s_1	$\{e_2, e_3, e_5, e_6, e_9\}$
s_2	$\{e_3, e_6, e_7, e_8, e_9\}$
s_3	$\{e_2, e_4, e_7, e_8, e_9\}$
s_4	$\{e_1, e_4, e_5, e_8, e_9\}$
s_5	$\{e_1, e_5, e_6, e_7, e_8\}$

Inverted Index

Find (r, s) such that $Prefix(r) \cap Prefix(s) \neq \phi$?

Prefix(R)



r_1	$\{e_1, e_5\}$
r_2	$\{e_1, e_2\}$
r_3	$\{e_1, e_2\}$
r_4	$\{e_1, e_2\}$
r_5	$\{e_4, e_5\}$


Prefix(S)

s_1	$\{e_2, e_3\}$
s_2	$\{e_3, e_6\}$
s_3	$\{e_2, e_4\}$
s_4	$\{e_1, e_4\}$
s_5	$\{e_1, e_5\}$

Candidates

$(r_1, s_4), (r_1, s_5)$
 $(r_2, s_1), (r_2, s_3), (r_2, s_4), (r_2, s_5)$
 $(r_3, s_1), (r_3, s_3), (r_3, s_4), (r_3, s_5)$
 $(r_4, s_1), (r_4, s_3), (r_4, s_4), (r_4, s_5)$
 $(r_5, s_3), (r_5, s_4)$

Build inverted index on *Prefix(S)*



e_1	e_2	e_3	e_4	e_5	e_6
s_4 s_5	s_1 s_3	s_1 s_2	s_3 s_4	s_5	s_2

An Algorithm for Band Join

- Nature of the join
 - $R.A - c_1 \leq S.B < R.A + c_2$
 - $c_1 + c_2$ is the band
- Assumption
 - Bands are narrow
 - Each tuple joins with no more than a few tuples

Truncated Hash Band Join Algorithm

- The idea

- a non-equijoin can be performed using **hashing**
- 3-phase algorithm

- Phase 1 (building)

- for each tuple i of the relation R , truncate the value of its join attribute $R_i.A$ before hashing
- $ri = R_i.A - ((R_i.A) \bmod (c_1 + c_2))$ /* R_i is the i -th tuple of relation R */
- $hi = \text{hash}(ri)$
- install tuple R_i in entry hi of a hash table
 - All tuples falling into entry hi are chained together in a collision chain

$c_1 = 1, c_2 = 2$

$R_i.A$	ri
11	9
15	15
13	12

Truncated Hash Band Join Algorithm

- Phase 2 (Probing)

- Create a similar hash with Sj.B
 - $sj1 = Sj.B - ((Sj.B) \bmod (c1 + c2))$
- if $Sj.B < (sj1 + c2)$ then $sj2 = sj1 - (c1 + c2)$ else $sj2 = sj1 + (c1 + c2)$
 - If a band of size $c1 + c2$ of Sj.B overlaps some values of R that were truncated to the left neighbor of $sj1$ OR if it overlaps some values that were truncated to the right neighbor of $sj1$ (it cannot overlap both), we denote these neighbors as $sj2$
- Create hash as before
 - $hj1 = \text{hash}(sj1)$
 - $hj2 = \text{hash}(sj2)$

<i>Ri.A</i>	<i>ri</i>
11	9
15	15
13	12

<i>Sj.B</i>	<i>sj1</i>	<i>sj2</i>
23	21	24
10	9	6
11	9	12
14	12	15

- Phase 3 (Join)

- All tuples of R that join with Sj can be found among the tuple chains rooted at the hash table entries $hj1$ and $hj2$

<i>Ri.A-candidates for join</i>	<i>Ri.A satisfying the join predicate</i>
-	-
11	11
11 and 13	11
13 and 15	13 and 15