Entitytity Resolution part2

Outline

1. Abstract Problem Statement
2. Algorithmic Foundations of ER
3. Scaling ER to Big-Data (Intruduction)
4. ER for web Data
5. ER and BigData (Continue)

Entity Resolution

Application 1: Data Cleaning

- Identifying duplicates in SCHUFA database [WNJLS08]
  - Schufa is the main credit scoring company for individuals in Germany
  - Duplicates enter the database due to incomplete information and caution
  - Domain specific duplicate detection on relational data

Person

PID	Firstname	Surname	Birthdate
1	John	Doe	29.03.1970
2	Jonathan	Doe	29.03.1907
3	Michael	Mustermann	

Contract

CID	Partner	*PID*
123	NZ Bank	1
456	National Bank of New Zealand	2
789	Foreign Bank	3

Address

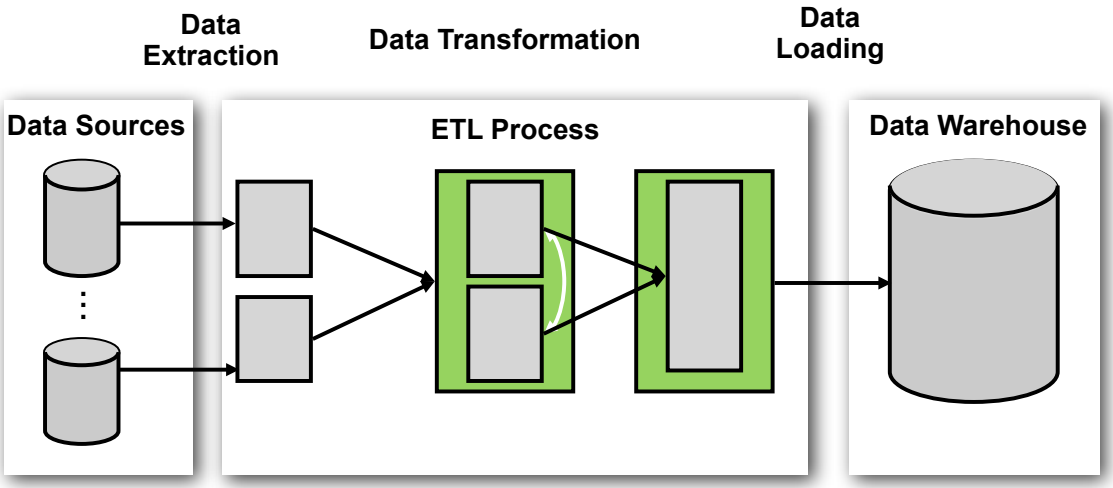
Street	City	*PID*
Main Street 1	Auckland	1
Large Place 2	Christchurch	1
Large Place 2	Christchurch	2
Dorfstrasse 567	Kleinstadt	3

History

Event	Date	*CID*
Open	29.03.1988	123
Update Student to Pro	01.01.1993	123

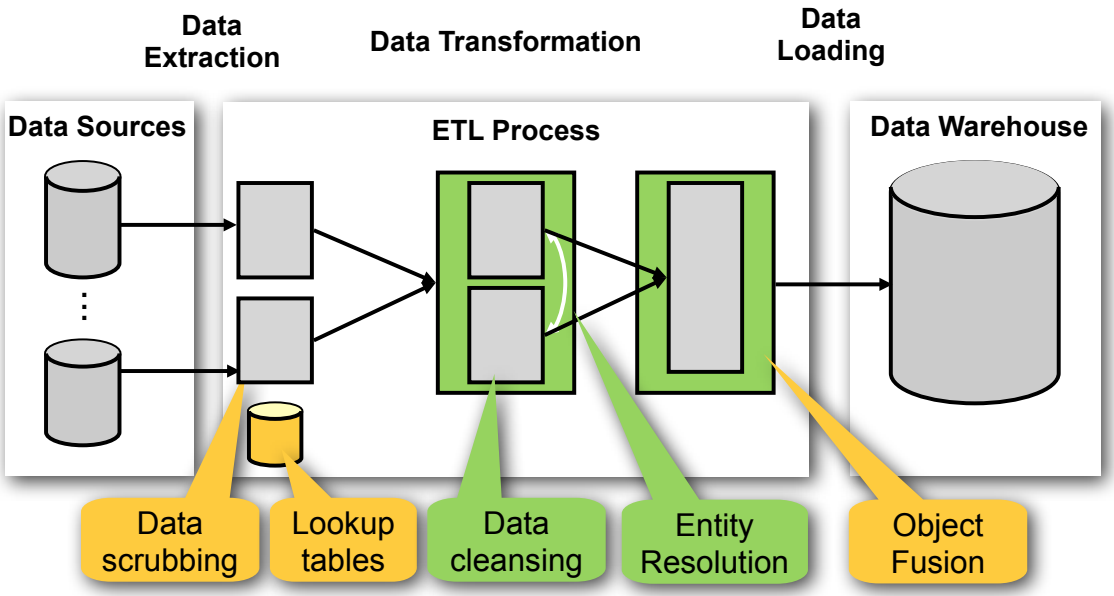
Entity Resolution

Application 2: Data Warehousing (ETL)



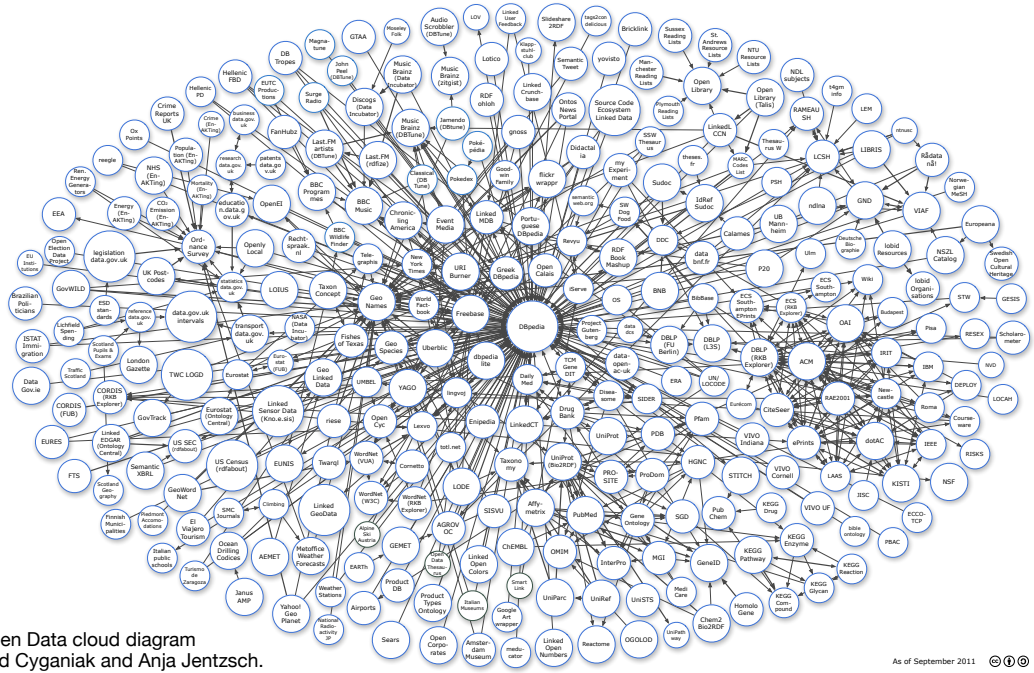
Entity Resolution

Application 2: Data Warehousing (ETL)



Entity Resolution

Application 3: Linking Open Data



Linked Open Data cloud diagram
by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

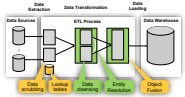
As of September 2011

Entity Resolution Solution Space

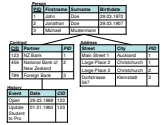
Type of data

- Graph

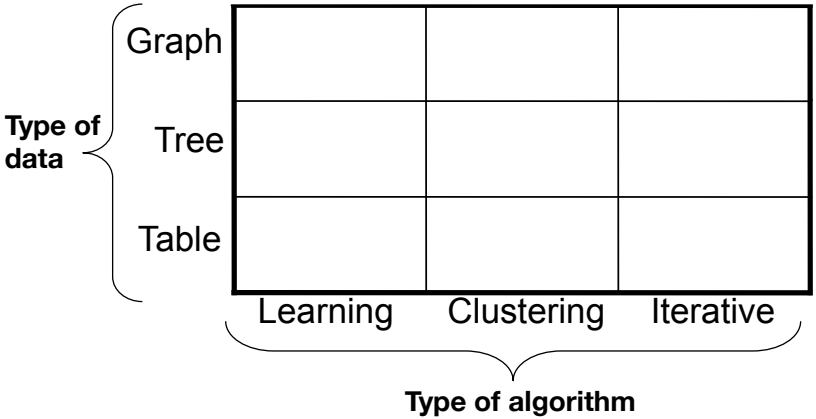
- Tree



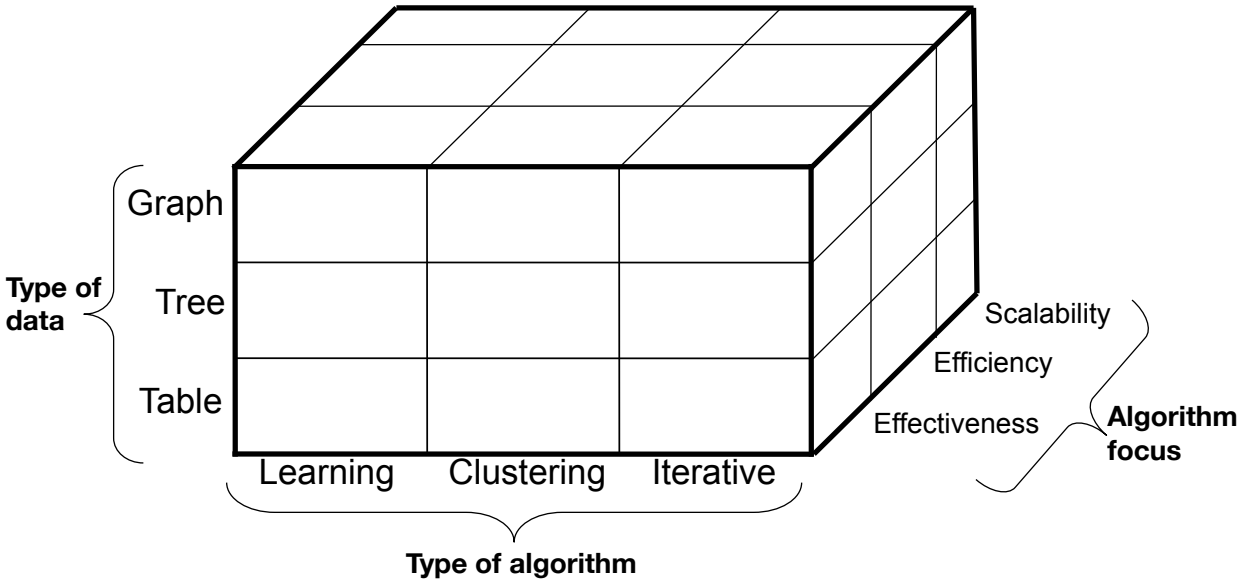
- Table



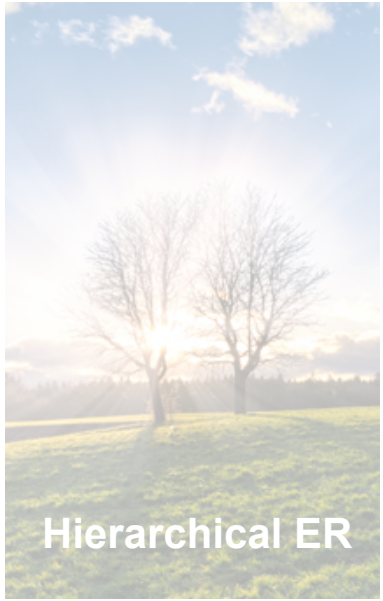
Entity Resolution Solution Space

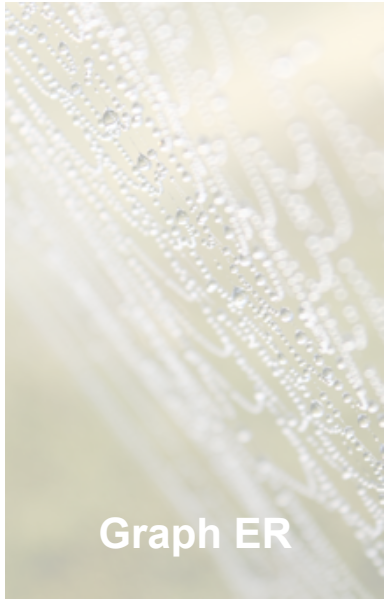


Entity Resolution Solution Space

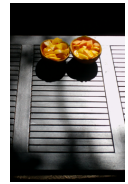


Agenda





Relational Entity Resolution



- Schemas of compared data (tuples) are identical
- Focuses on variations of the data
- In data integration context, we assume that data with different schema have been mapped to a common global schema prior to entity resolution.
- In Semantic Web, we assume a global ontology to describe equivalences between attribute names.

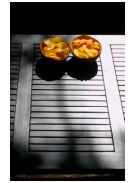
Title	Genre	Year	Director
Troy	Action	2004	Petersen
Troj	History		Petersen

typo

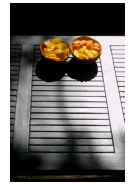
contradiction

missing value

Pairwise Comparisons

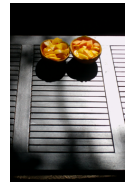


Pairwise Comparisons



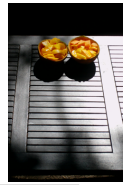
- Given two data sets *A* and *B*
- General idea
  - Form the **Cartesian product** A x B.
  - For each pair, decide if it is a **duplicate**:
    - Based on a **similarity measure** and a threshold
    - Based on a **rule-based** classifier
  - Identification of **duplicate clusters** (transitive closure)

Pairwise Comparisons



- Given two data sets *A* and *B*
- General idea
  - Form the **Cartesian product** A x B.
  - For each pair, decide if it is a **duplicate**:
    - Based on a **similarity measure** and a threshold
    - Based on a **rule-based** classifier
  - Identification of **duplicate clusters** (transitive closure)
- Problems
  - Number and complexity of comparisons (**efficiency**)
  - Quality of duplicate classification (**effectiveness**)
  - Processing of large volumes of data (**scalability**)

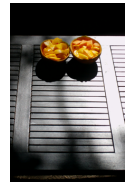
Pairwise Comparisons



- Given two data sets *A* and *B*
- General idea
  - Form the **Cartesian product** A x B.
  - For each pair, decide if it is a **duplicate**:
    - Based on a **similarity measure** and a threshold
    - Based on a **rule-based** classifier
  - Identification of **duplicate clusters** (transitive closure)
- Problems
  - Number and complexity of comparisons (**efficiency**)
  - Quality of duplicate classification (**effectiveness**)
  - Processing of large volumes of data (**scalability**)

Cosine similarity,
Jaro-Winkler similarity

Pairwise Comparisons

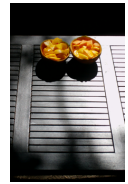


- Given two data sets *A* and *B*
- General idea
  - Form the **Cartesian product** A x B.
  - For each pair, decide if it is a **duplicate**:
    - Based on a **similarity measure** and a threshold
    - Based on a **rule-based** classifier
  - Identification of **duplicate clusters** (transitive closure)
- Problems
  - Number and complexity of comparisons (**efficiency**)
  - Quality of duplicate classification (**effectiveness**)
  - Processing of large volumes of data (**scalability**)

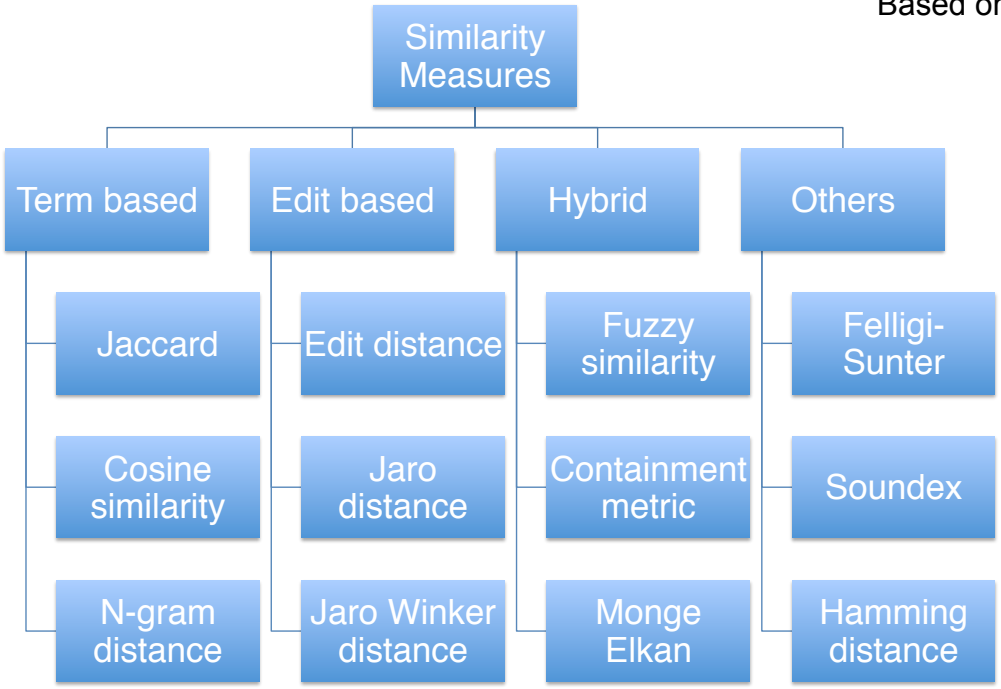
Cosine similarity,
Jaro-Winkler similarity

Sorted Neighborhood Method

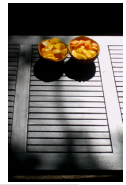
Similarity Measures



Based on [HB12]



Similarity Measures
Jaro-Winkler Similarity



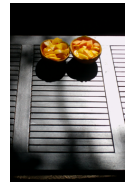
- The **Jaro-Winkler similarity** [WT91] of two strings *s1* and s2 that share a **common prefix** ρ is defined as:

$$SimJaroWinkler(s1, s2) = SimJaro(s1, s2) + |\rho| \cdot f \cdot (1 - SimJaro(s1, s2))$$

- *f* is a **user-defined coefficient** that "corrects" the similarity computed using the Jaro similarity [Jaro89], denoted *SimJaro(s1,s2)* to account for the common prefix of *s1* and *s2*.

- Hence, a **common prefix increases the Jaro-Winkler similarity**.

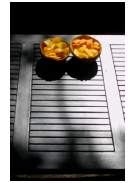
Similarity Measures
Jaro-Winkler Similarity - Example



Given s1 = Spears and s2 = Spaers, simJaro(s1, s2) = 0.944:

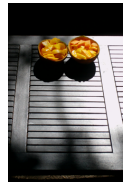
- All characters belong to the set of characters shared among *s1* and *s2*.
  - For *S, p, r, et s,* positions are identical
  - For *e* and *a,* the positions vary by *1, which is below the lower bound* for transpositions (computed as *0.5 * 6 – 1)*
  - *Hence,* ***σ = {S,p,e,a,r,s}.***
- When traversing *s1* and *s2*, we observe:
  - 1st common character in *s1 (S)* = 1st common character in *s2*
  - 2nd common character in *s1 (p)* = 2nd common character in *s2*
  - **But** 3rd common character in *s1 (e)* ≠ 3rd common character in *s2*
  - A second transposition occurs for the fourth character in *s1 (a)*
  - Remaining positions are identical.
  - Hence**,** ***t = 2***
- *SimJaro(s1,s2) = 1/3 * (6/6 + 6/6 + (6 - 0.5 * 2)/6 = 0.944*

Sorted Neighborhood Method



- Idea
  - Create **partitions**
  - Perform comparisons **only within a partition**
- The initial algorithm [HS95]
  1. Create Key:
     - Creates a key value based on relevant attribute values.
  2. Sort:
     - Sort tuples in lexicographical order of their generated keys.
  3. Merge:
     - Slide a window (of fixed size *w*) over the sorted data.
     - Limit to comparisons of tuple pairs falling in the same window.

Sorted Neighborhood Method



ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

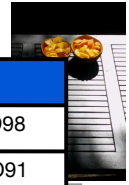
Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

Create key

1.

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95



Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

Create key

1.

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95

2. Sort

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

Create key

1.

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95

2. Sort

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

Merge

3.

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

Create key

1.

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95

2. Sort

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

Merge

3.

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

classify(18,113) → duplicates

Sorted Neighborhood Method

ID	Title	Year	Genre
17	Mask of Zorro	1998	Adventure
18	Addams Family	1991	Comedy
25	Rush Hour	1998	Comedy
31	Matrix	1999	Sci-Fi
52	Return of Dschafar	1994	Children
113	Adams Family	1991	Comedie
207	Return of Djaffar	1995	Children

Create key

1.

ID	Key
17	MSKAD98
18	DDMCO91
25	RSHCO98
31	MTRSC99
52	RTRCH94
113	DMSCO91
207	RTRCH95

2. Sort

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

Merge

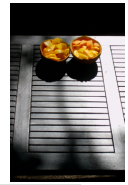
3.

ID	Key
18	DDMCO91
113	DMSCO91
17	MSKAD98
31	MTRSC99
25	RSHCO98
52	RTRCH94
207	RTRCH95

classify(18,113) → duplicates

classify(52,207) → duplicates

Sorted Neighborhood Method
Key Generation Phase



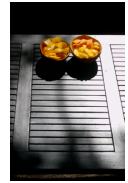
- **Key**: for a given tuple t, its key consists of a sequence of attribute value substrings taken from t.

- **Quality** of entity resolution strongly depends on the choice of the key.

- The key is virtual and is **not necessarily unique.**

  - It only serves to sort the tuples.

Sorted Neighborhood Method
Sort Phase



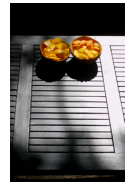
- **Sort** tuples according to the lexicographic order of their generated keys.

- Goal:
  Equivalent tuples (duplicates) are **sorted close to each other.**

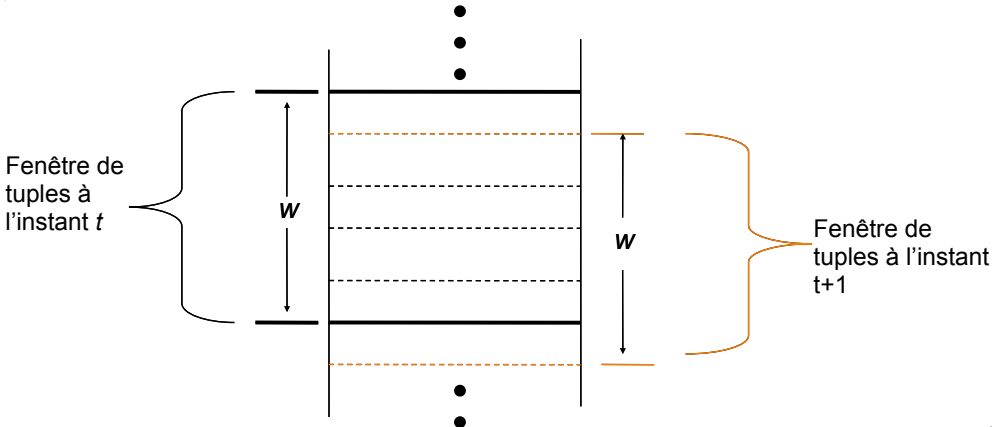
- Different sorting strategies (Quicksort, AlphaSort, etc.)

  - For scalability, use of a DBMS for efficient secondary memory access (two passes over the data).

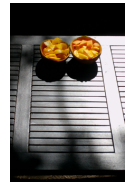
Sorted Neighborhood Method
Comparison Phase



- A window of predefined fixed size *w* goes over the sorted data.
- $2 \leq w \leq N$
- Only compare tuples that fall in the same window.
- HS95 proposes a rule-based classifier to detect duplicates, but any similarity measure can be used as well.

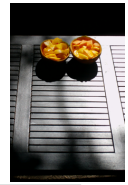


Sorted Neighborhood Method
Discussion



- Complexity
  - *N* : number of tuples
  - w: window size
  - In theory:
    - *O(N) + O(N logN) + O(w N) = O(N logN) when w < logN;*
    - *O(wN) otherwise*
  - In practice:
    - **Three scans** of the relational data stored on disk.

Relational Entity Resolution



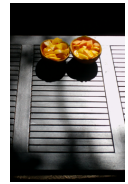
Extensions to the Sorted Neighborhood Method

- Multi-Pass Sorted Neighborhood Method [HS95,HS98]
- Sorted-Neighborhood for XML data [PHN06]
- Automatic adjustment of the window size [YLKG07, DNSW12]

Other means to reduce the number of pairwise comparisons

- Blocking (partitioning w.r.t. one or more attribute values)
- Recall-maintaining filter functions (upper / lower bound for similarity measures) [ACG02,WN04]

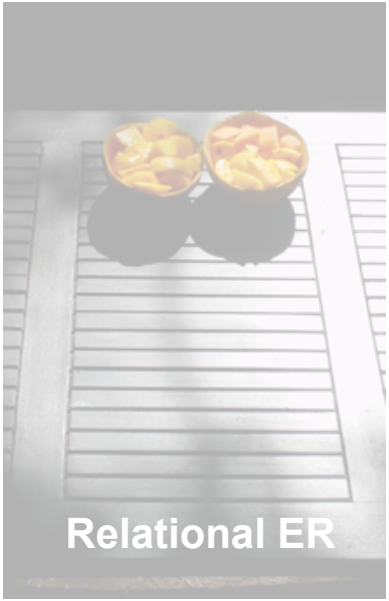
Relational Entity Resolution

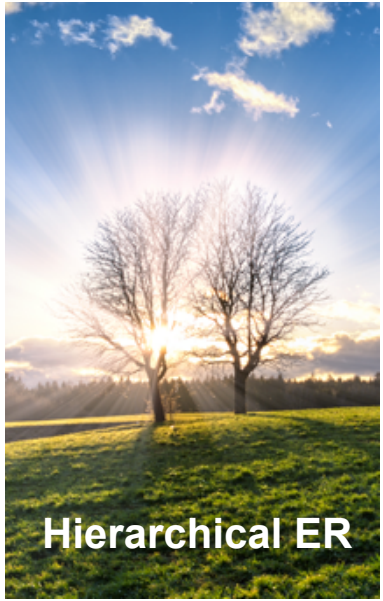


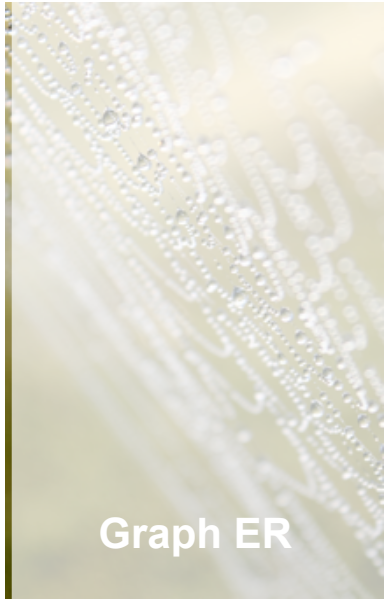
Many alternative approaches to Relational Entity Resolution

- Survey by Elmagarmid et. al. [EIV07]
- Book by Naumann et. al. [NH10]
- Survey by Peter Christen [Christen12]

Agenda







Hierarchical Entity Resolution

Examples of hierarchically organized data

- Relational star / snowflake schema [ACG02]
- Hierarchical XML data [CHL10]

Specialized similarity measures and algorithms

- **Containment metric** as similarity measure [ACG02]
- **Top-down** & **bottom-up** algorithms

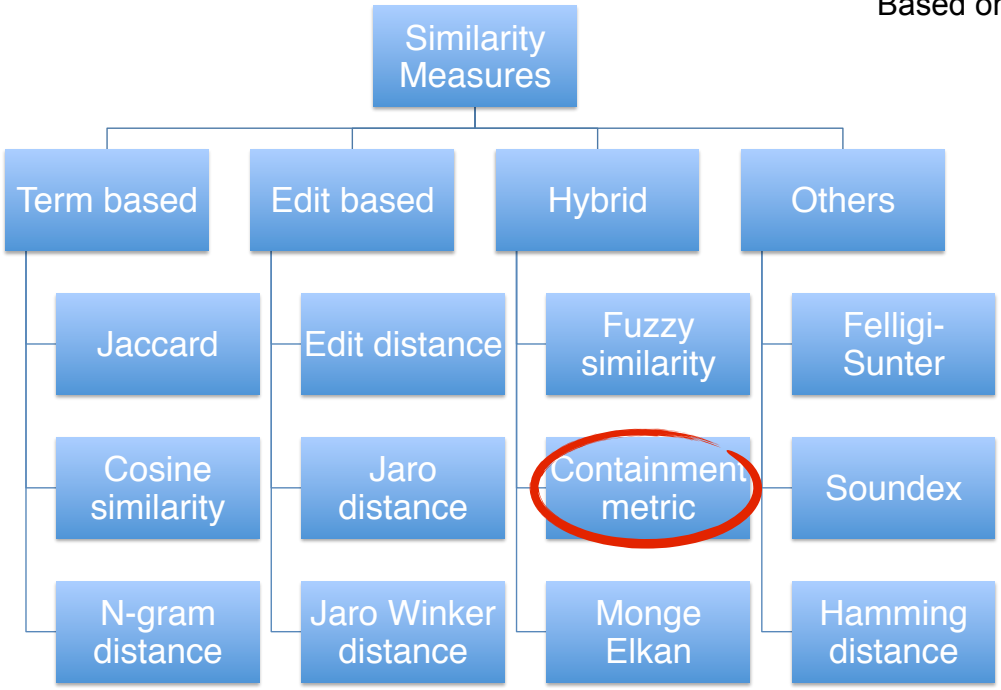
Hierarchical Relational Schema

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2/10
F2	Gottvatter, The	72	

Similarity Measures

Based on [HB12]



Comtainment Metric

- **Hybrid similarity measure** [ACG02] considering
  - Similarity of attribute values (*tcm*)
  - Similarity of children sets reached by following foreign keys (*fkcm*)
- Similarity of **attribute values**
  - Divide tuples into tokens → token sets *TS*
  - Compute the edit distance between token sets
  - Determine weight of each token using IDF [BR99]
  - The **token similarity metric *tcm*** measures which fraction of one tuple *T* is covered by the other tuple *T'*.

$$tcm(T,T') = \frac{\sum idf(TS(T) \cap TS(T'))}{\sum idf(TS(T))}$$

Containment Metric

- Similarity of **children sets**
  - The children set of a tuple *T* includes all tuples referencing *T* from other relations by means of a foreign key.
    → **Children sets** *CS*
  - **Foreign-key containment metric** (*fkcm*) measures at what extent the children set of a tuple *T* is covered by the children set of a tuple *T'*.

$$fkcm(T,T') = \frac{|CS(T) \cap CS(T')|}{|CS(T)|}$$

Containment Metric

- **Combining tcm and fkcm:**
  - Both *tcm* and *fkcm* are assigned an IDF **weight**.
  - Use of a **classification function**:
    pos(x) = 1 if x > 0,
    -1 otherwise
  - Threshold for *tcm*: *s1*
  - *Threshold for fkcm: s2*
  - Classification of pairwise comparison between *T* and *T'* using

$$pos(IDF(TS) * pos(tcm(T,T') - s1) + IDF(CS) * pos(fkcm(T,T') - s2))$$

  - If final result equals 1, then duplicate, otherwise non-duplicate.

Containment Metric
Example

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2/10
F2	Gottvatter, The	72	

Containment Metric
Example

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2/10
F2	Gottvatter, The	72	

1. Token sets:
TS(F1) = {The, Godfather, 1972, 9.2/10}
TS(F2) = {Gottvatter, The, 72}

2. Attribute similarities
The = The, Godfather = Gottvatter, 1972 = 72.

3. Weights
For simplification, we assume all tokens have equal weight.

4. Token containment metric
tcm(F1,F2) = ¾, tcm(F2,F1) = 1

Containment Metric
Example

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

ID	Name	Year	Rating
F1	The Godfather	1972	9.2/10
F2	Gottvatter, The	72	

1. Token sets:
TS(F1) = {The, Godfather, 1972, 9.2/10}
TS(F2) = {Gottvatter, The, 72}

2. Attribute similarities
The = The, Godfather = Gottvatter, 1972 = 72.

3. Weights
For simplification, we assume all tokens have equal weight.

4. Token containment metric
tcm(F1,F2) = ¾, tcm(F2,F1) = 1

5. Children co-occurrence
fkcm(F1,F2) = 1, *fkcm(F2,F1)* = ½

Containment Metric
Example

ID	Name	Year	Rating
F1	The Godfather	1972	9.2/10
F2	Gottvatter, The	72	

ID	Actor	Film
S1	Al Pacino	F1
S2	Al Pacino	F2
S3	Marlon Brando	F2

1. Token sets:
TS(F1) = {The, Godfather, 1972, 9.2/10}
TS(F2) = {Gottvatter, The, 72}

2. Attribute similarities
The = The, Godfather = Gottvatter,
1972 = 72.

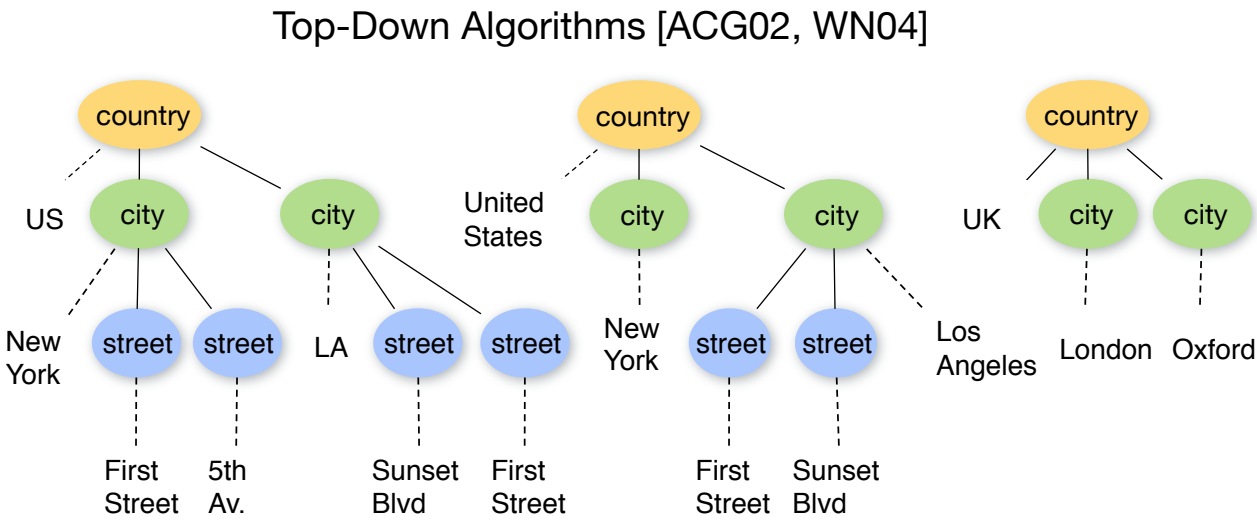
3. Weights
For simplification, we assume all tokens have equal weight.

4. Token containment metric
tcm(F1,F2) = ¾, tcm(F2,F1) = 1

5. Children co-occurrence
fkcm(F1,F2) = 1, *fkcm(F2,F1)* = ½

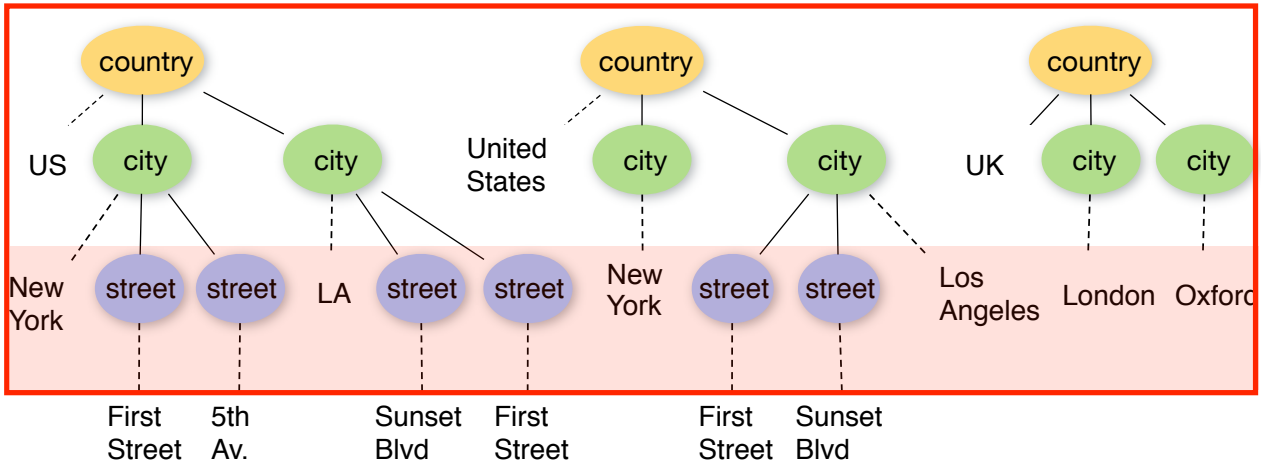
6. **Combination of both metrics**
(*s1* = *s2* = 0.5, weights = 1)
pos(pos(3/4 - 0.5) + pos(1 - 0.5) = 1
→ F1 and F2 duplicates

Top-Down Algorithms



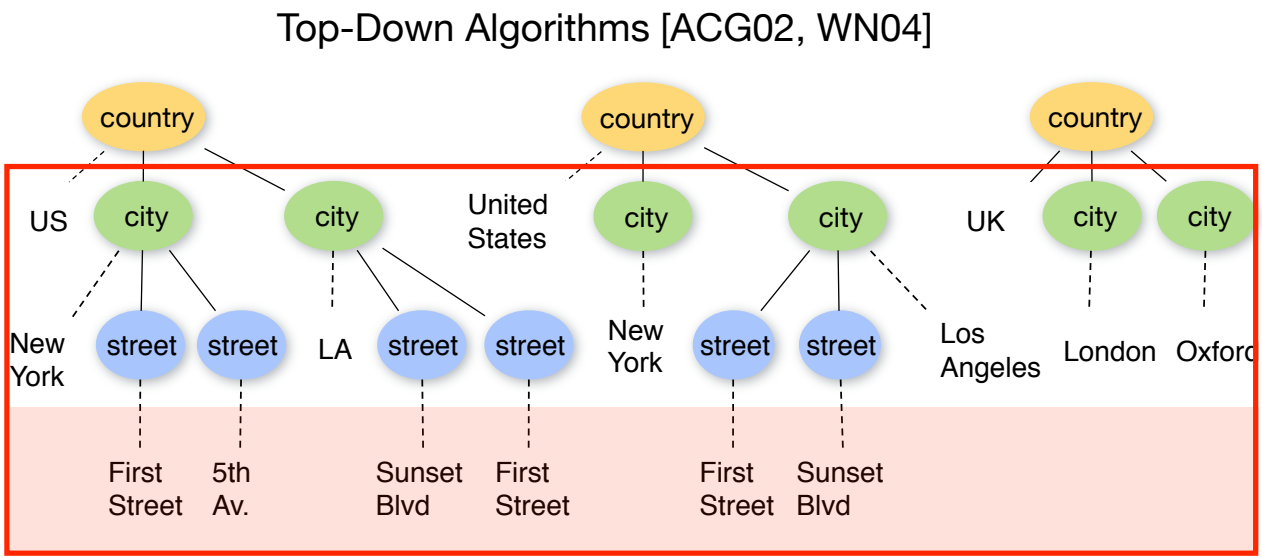
Top-Down Algorithms

Top-Down Algorithms [ACG02, WN04]



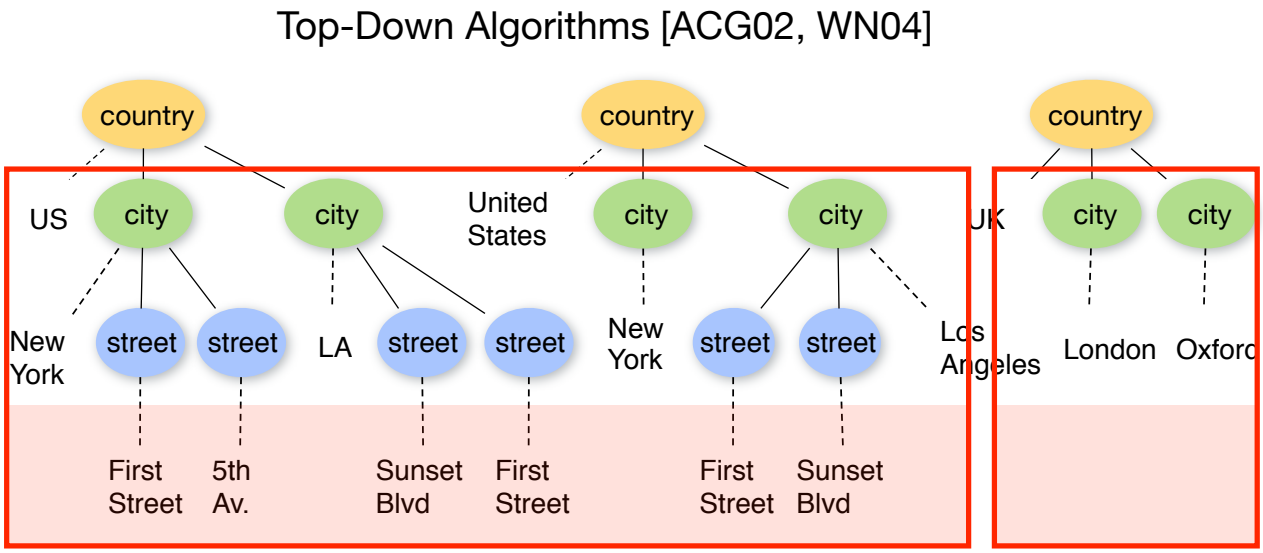
1. Compare top-level elements using description similarity and exact children co-occurrence

Top-Down Algorithms



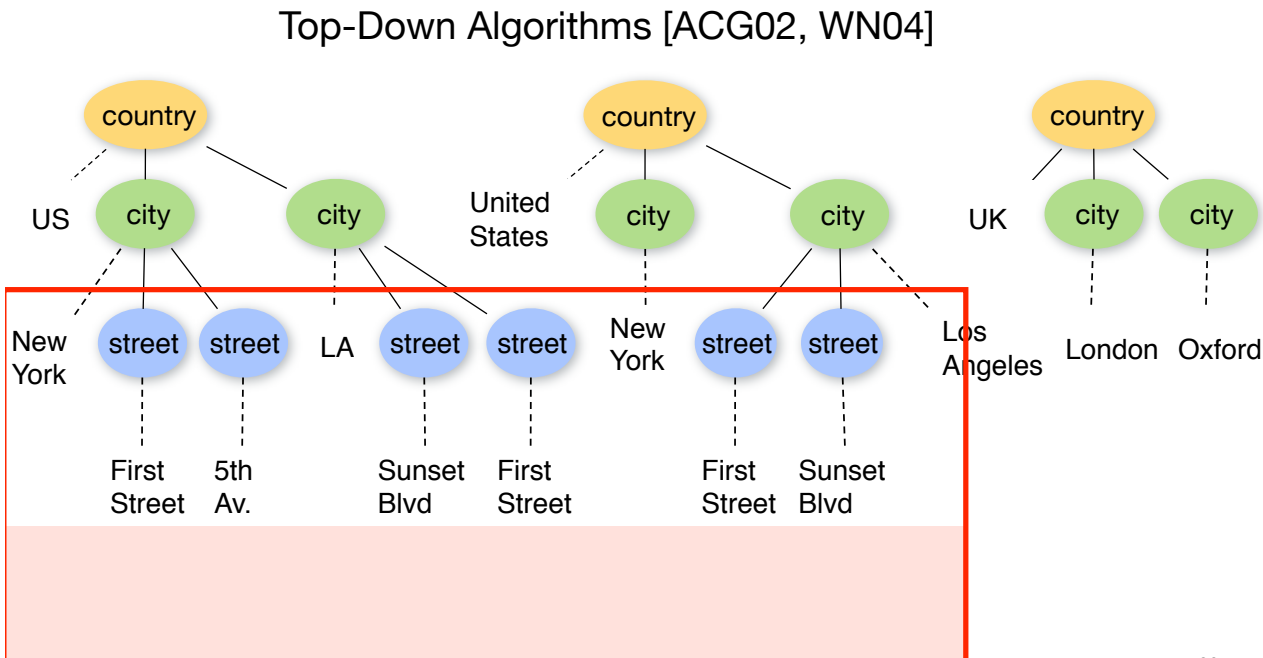
1. Compare top-level elements using description similarity and exact children co-occurrence

Top-Down Algorithms

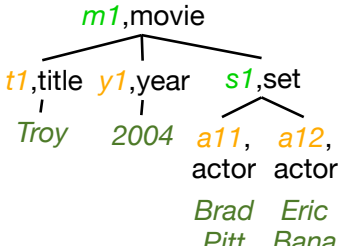


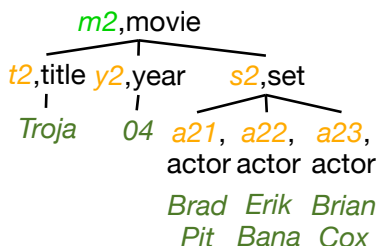
2. Only search for duplicates among children with common ancestor.

Top-Down Algorithms

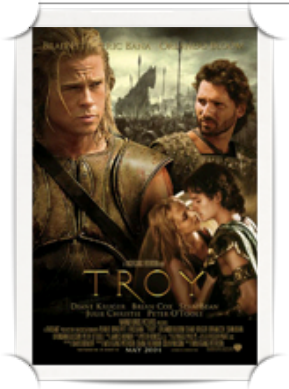


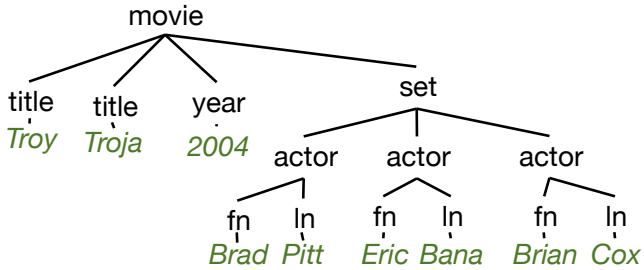
XML Duplicates



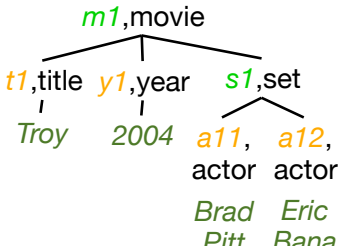


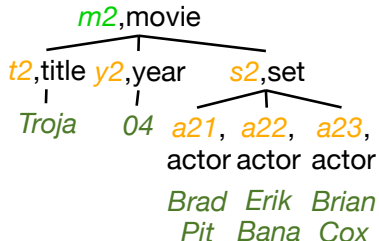
XML Duplicates



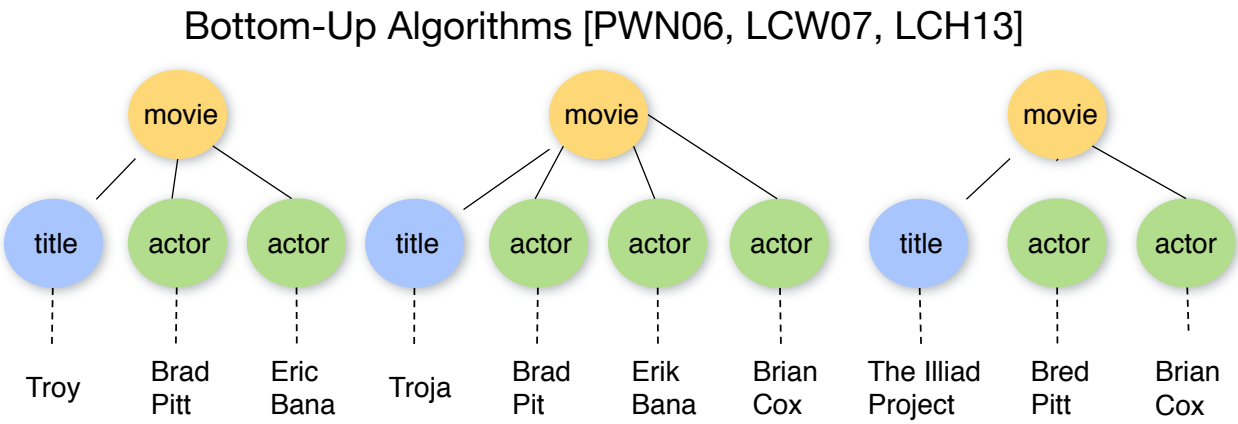


Data Cleansing

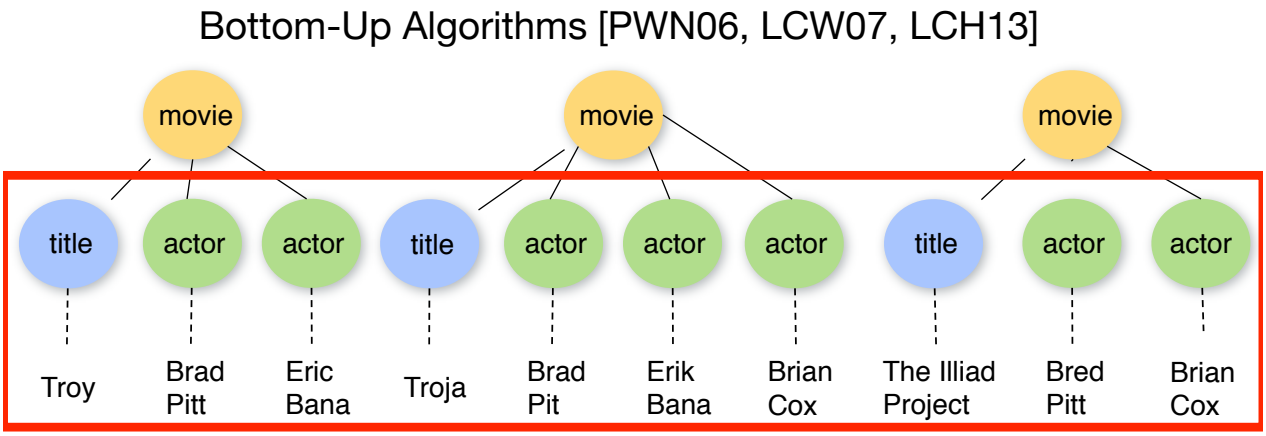




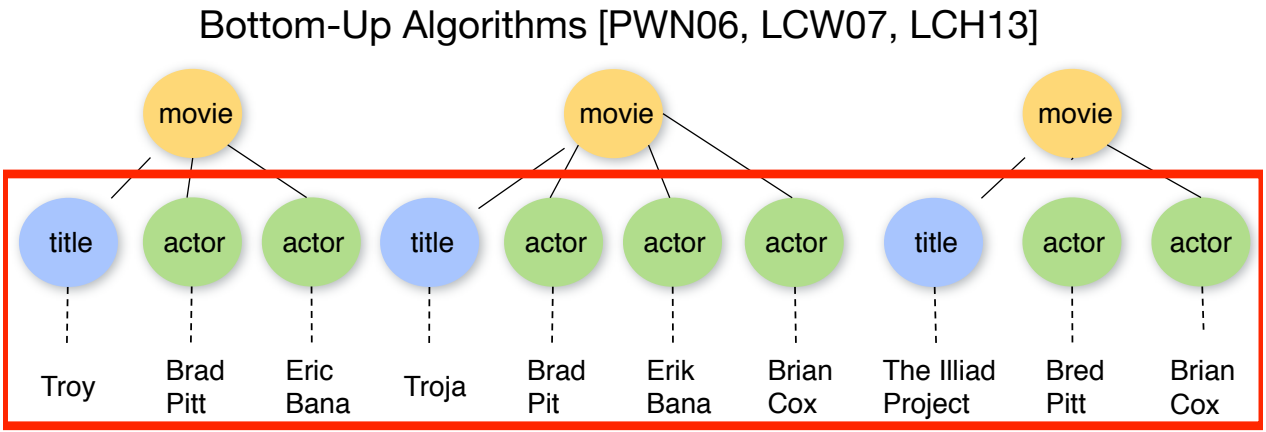
Bottom-Up Algorithms



Bottom-Up Algorithms



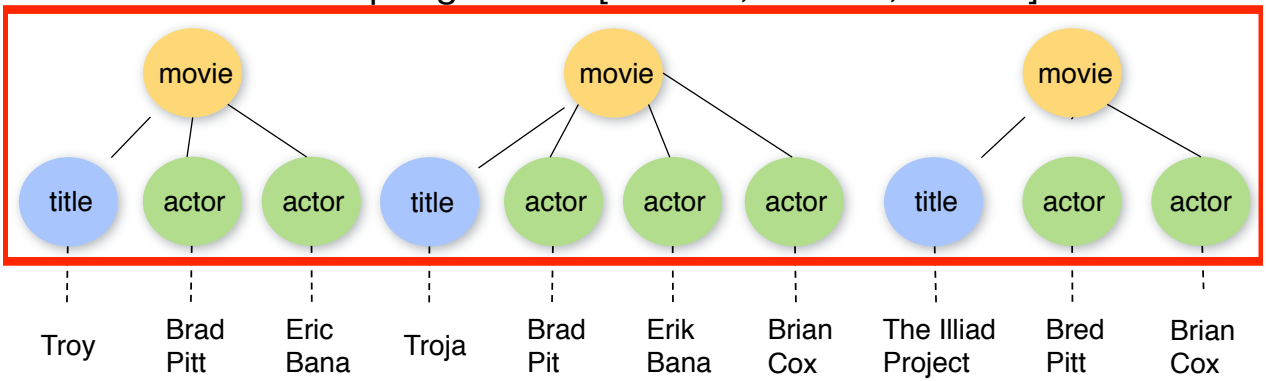
Bottom-Up Algorithms



1. On **leaf** level, perform **pairwise comparisons** to detect duplicates. (e.g., apply sorted neighborhood method on leaf candidates [PWN06])

Bottom-Up Algorithms

Bottom-Up Algorithms [PWN06, LCW07, LCH13]

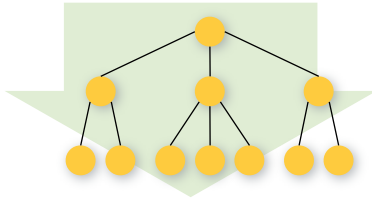
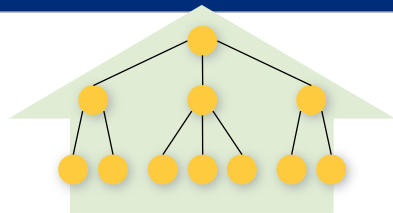


2. **Propagate** duplicate decisions to **parent level** and perform comparisons one level up, **taking into account identified child duplicates** (e.g., propagate similarities that reflect duplicate probability through a Bayesian Network [LCW07]).

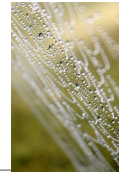
Hierarchical Entity Resolution

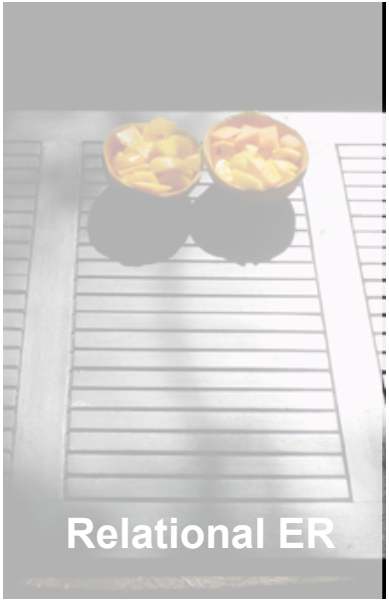
Similarity measures

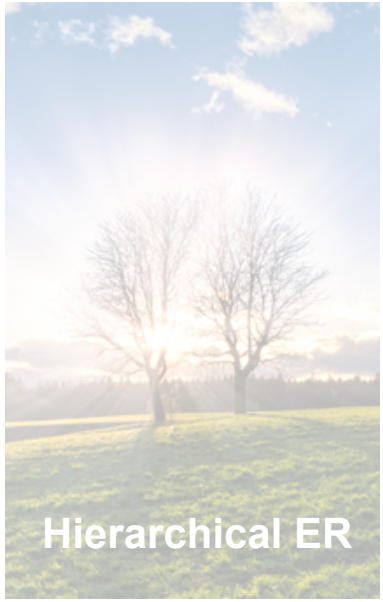
- Consider attribute **values** (text value data)
- Consider similarity of **children sets** (referencing tuples or child XML elements)

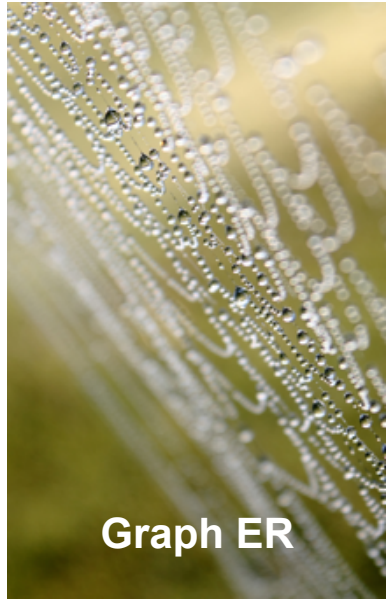
Top-down [WN04]	Bottom-up [PWN06]
	
Hierarchy represents 1:N relationships between candidate entities.	Hierarchy represents candidate entities that can be in M:N relationship.
Prune comparisons of descendants that do not have same or similar ancestors.	Sorted neighborhood method applied to every candidate type.
$O(\|T_1\|^2+\|T_2\|^2+\ldots+ \|T_n\|^2)$	$O(\|T_1\| \log\|T_1\|+\ldots + \|T_n\| \log\|T_n\|)$

Agenda

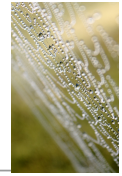






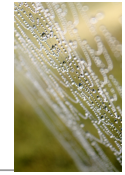


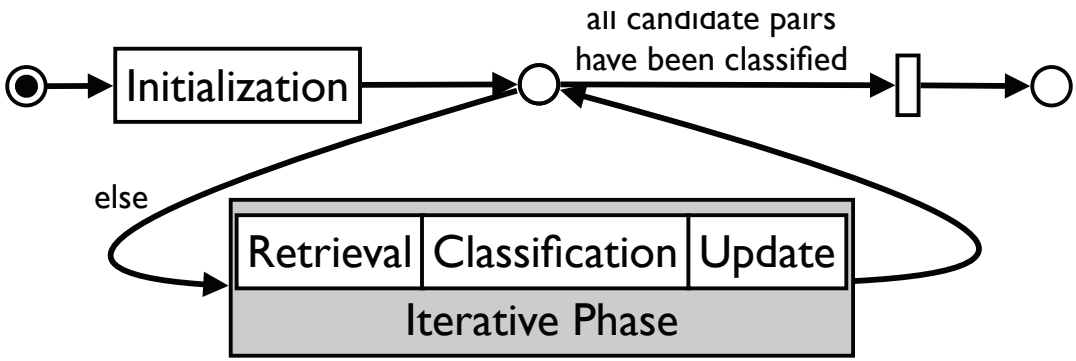
Graph Data



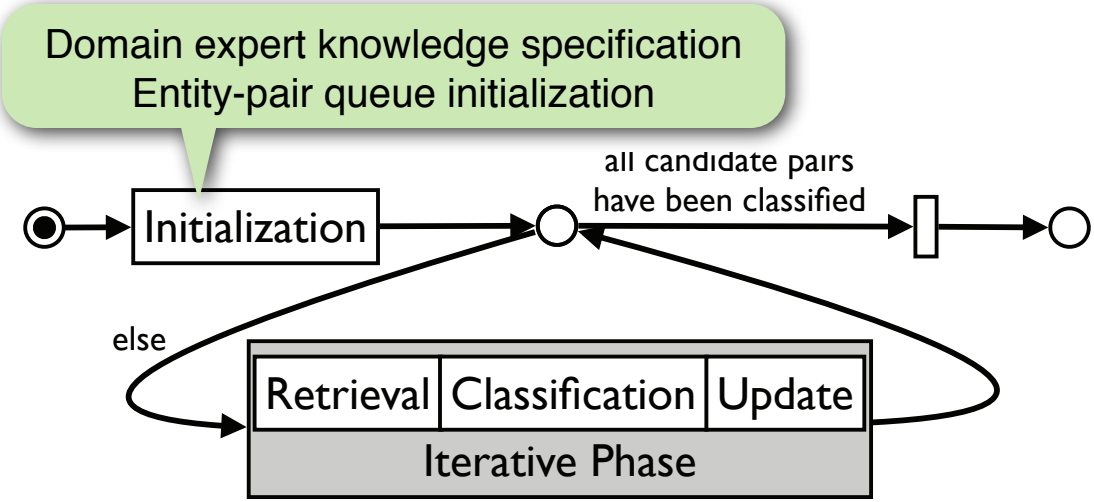
- In the most general case, data not only form a tree, but a **graph**
  - LOD graph
  - General relational schema
  - Domain-knowledge about entity relationships
  - ...
- In graph data, there is **no clear order of comparisons** (top down, bottom-up?)
- Several algorithms for Entity Resolution in Graph Data have been proposed [DHM05, WN06, BG07, ...].
- Many of them conform to a **general framework** [HNST12]

Graph Entity Resolution Framework

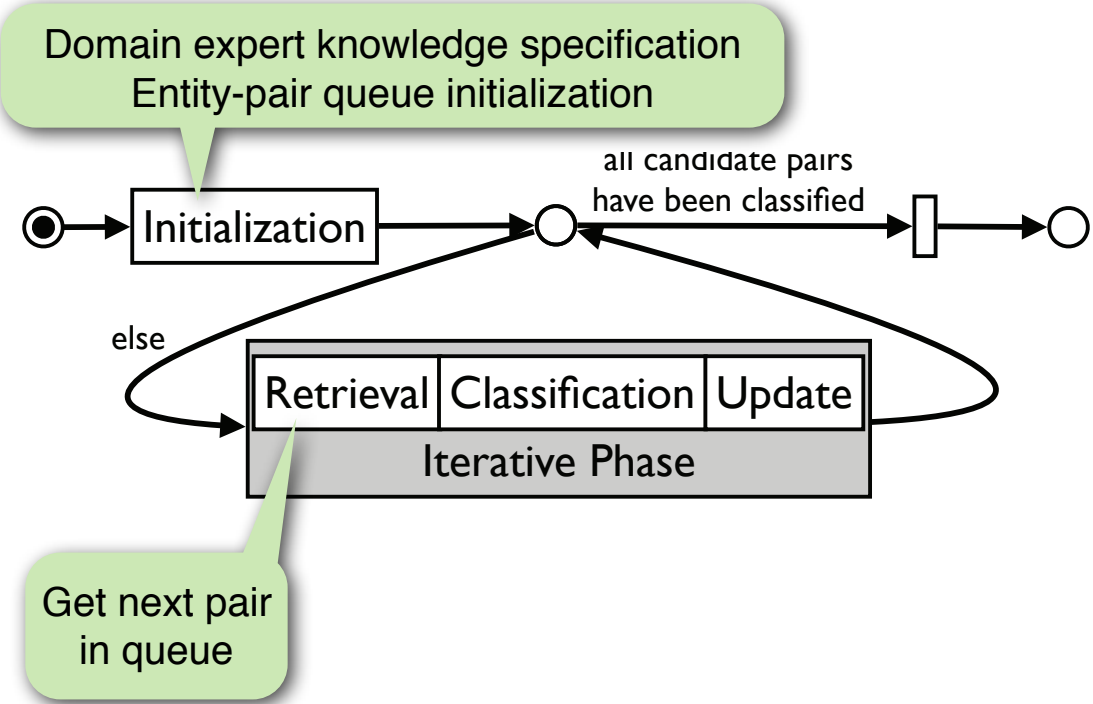




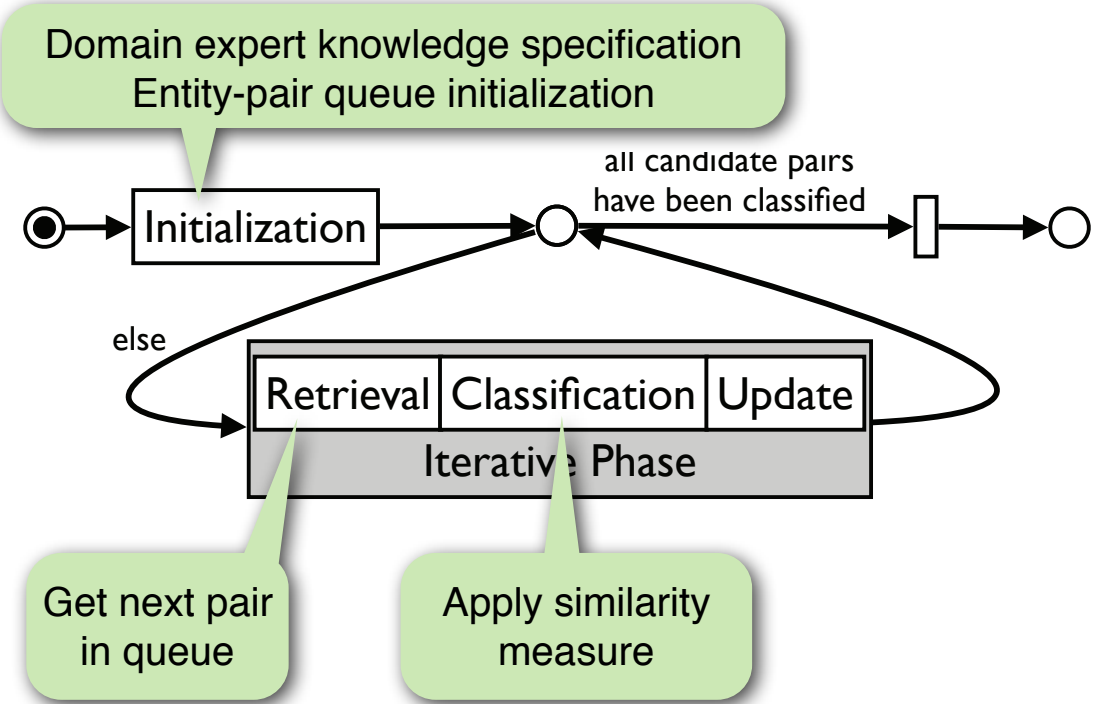
Graph Entity Resolution Framework



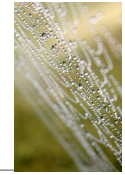
Graph Entity Resolution Framework

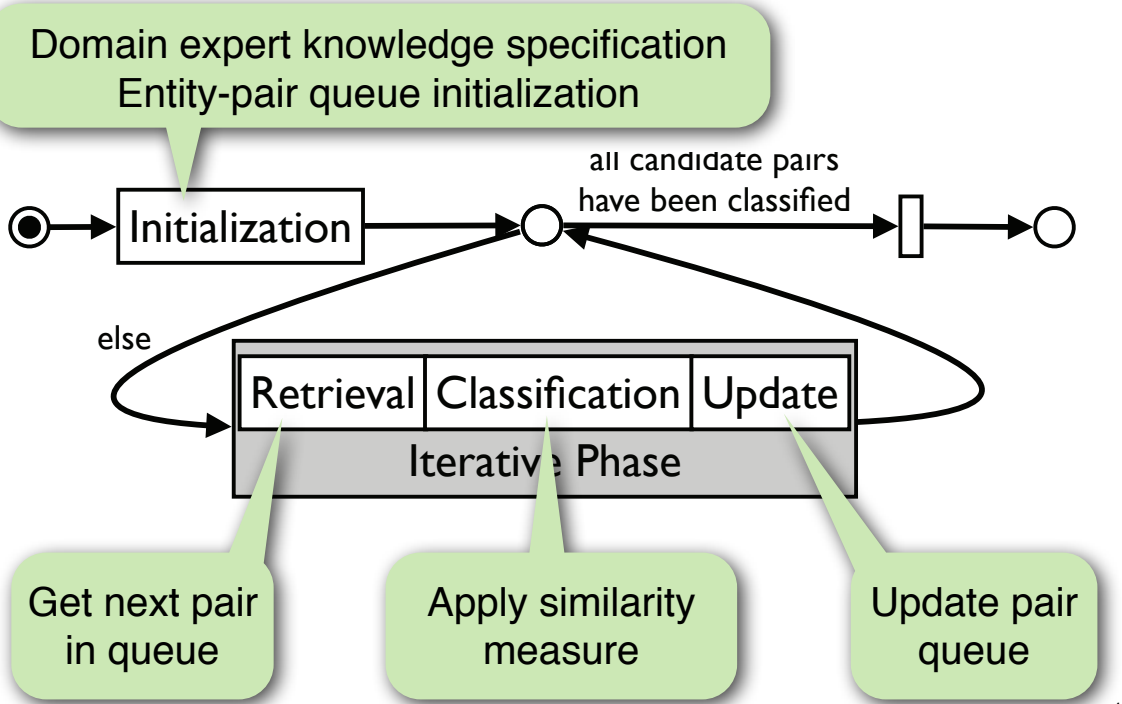


Graph Entity Resolution Framework

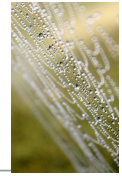


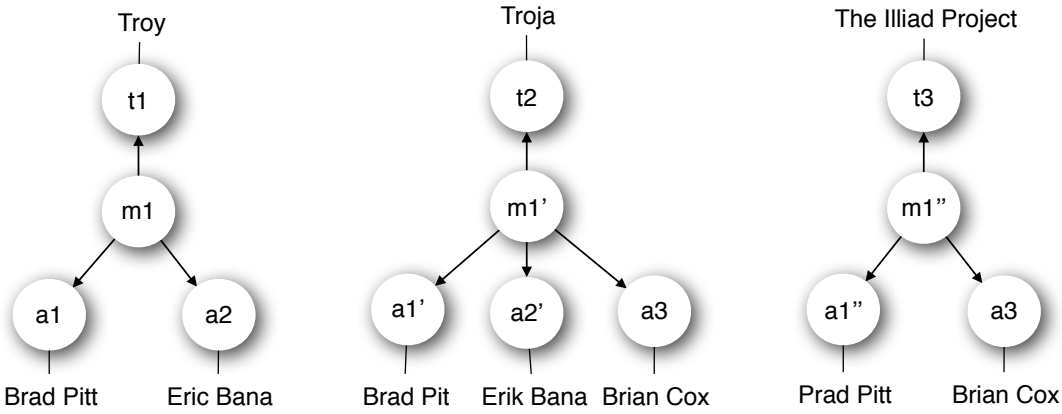
Graph Entity Resolution Framework



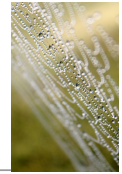


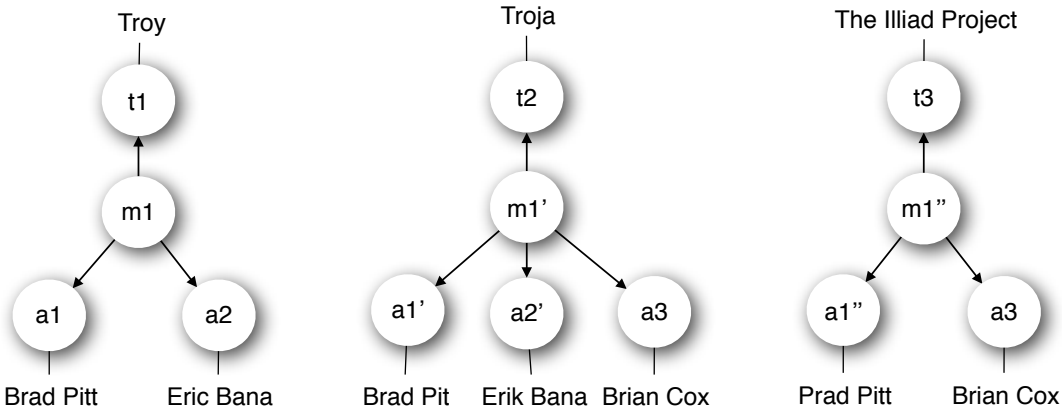
Domain Expert Knowledge Specification





Domain Expert Knowledge Specification

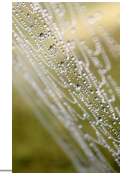


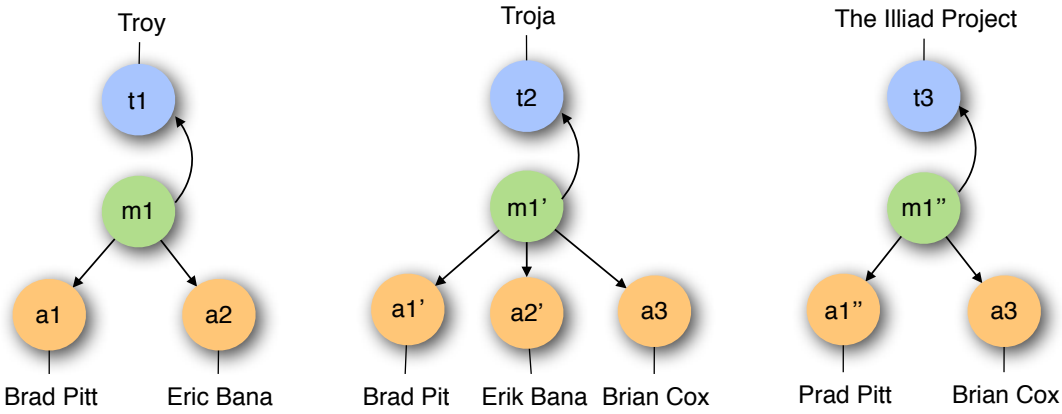


Domain expert specifies

- Duplicate **candidates** (e.g., movie, actor, title)
- (Additional) **relationships** between candidates (e.g., title → movie)

Domain Expert Knowledge Specification

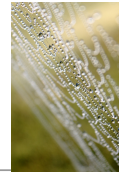


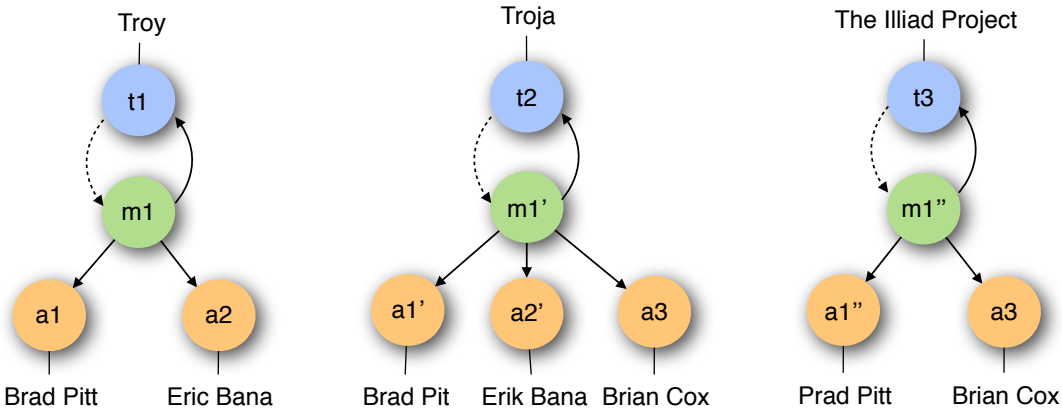


Domain expert specifies

- Duplicate **candidates** (e.g., movie, actor, title)
- (Additional) **relationships** between candidates (e.g., title → movie)

Domain Expert Knowledge Specification

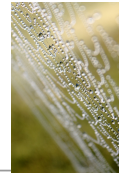


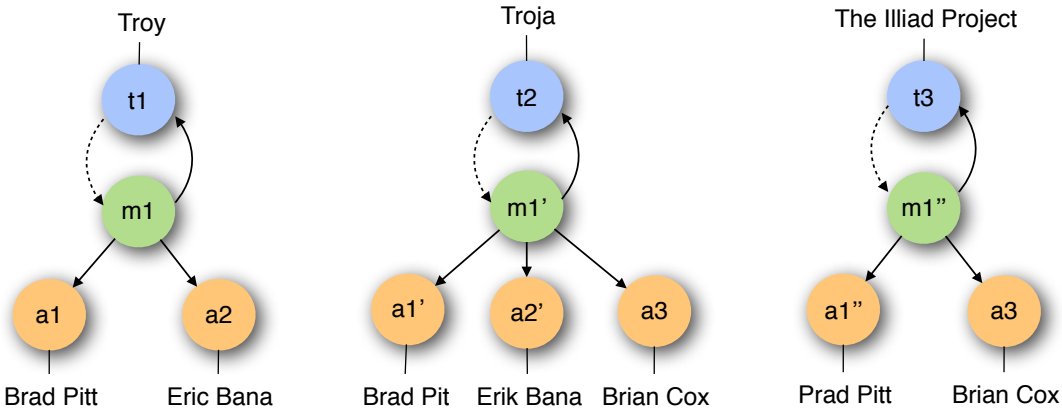


Domain expert specifies

- Duplicate **candidates** (e.g., movie, actor, title)
- (Additional) **relationships** between candidates (e.g., title → movie)

Domain Expert Knowledge Specification

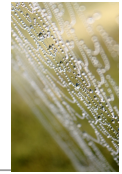


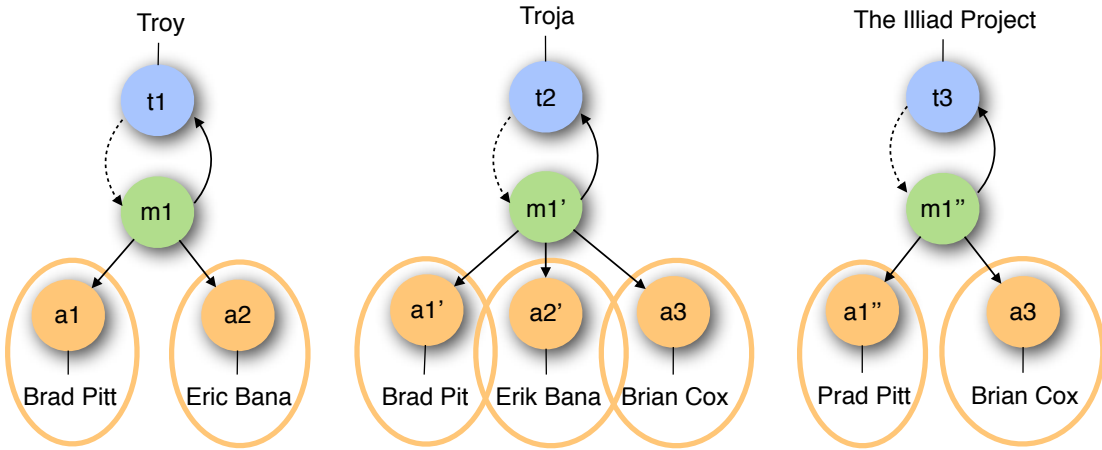


For **pairwise similarity computation**, domain expert also selects what information is **relevant** for comparisons

- **Object description** (attribute values)
- **Influencing neighbo**r candidates

Domain Expert Knowledge Specification

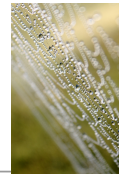


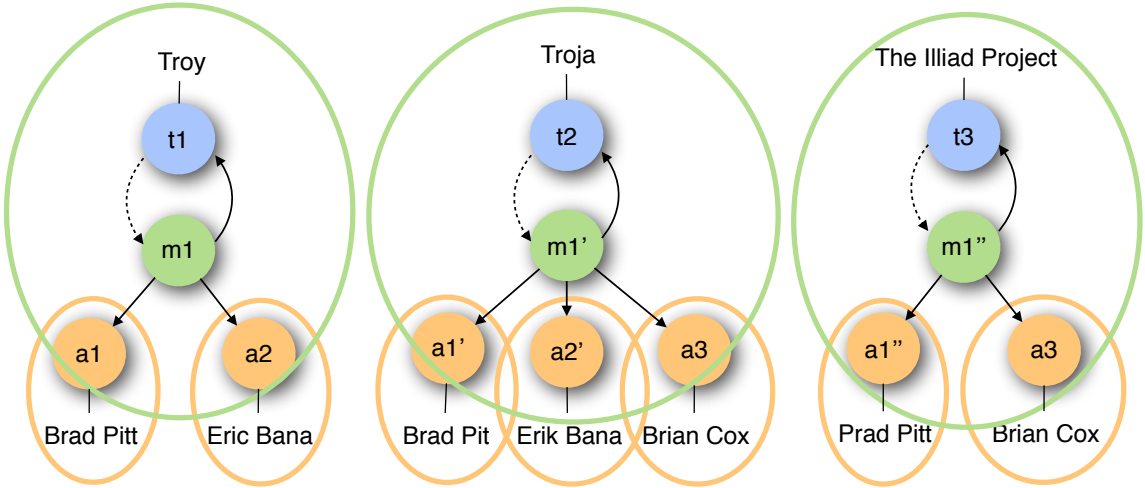


For **pairwise similarity computation**, domain expert also selects what information is **relevant** for comparisons

- **Object description** (attribute values)
- **Influencing neighbo**r candidates

Domain Expert Knowledge Specification

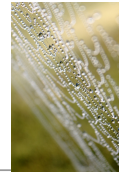


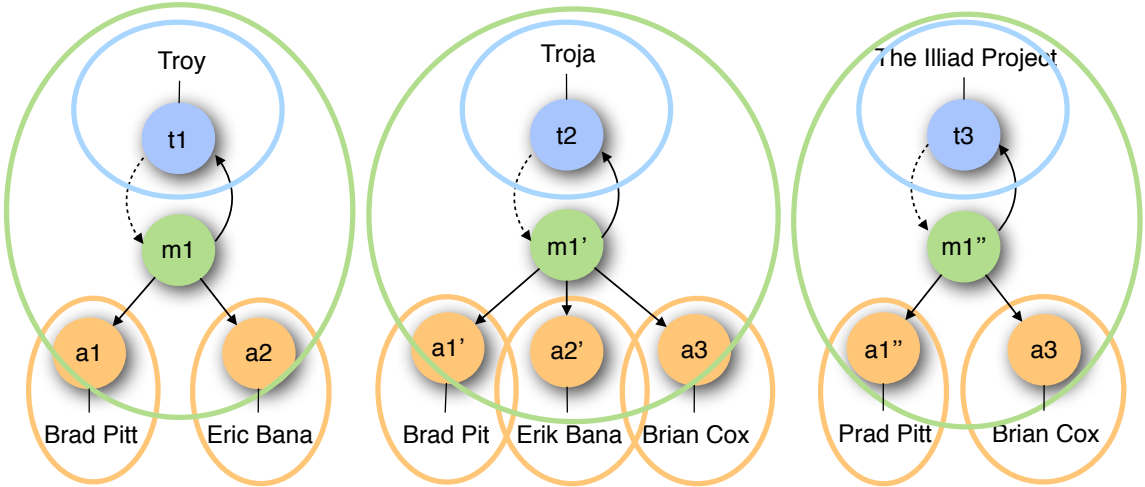


For **pairwise similarity computation**, domain expert also selects what information is **relevant** for comparisons

- **Object description** (attribute values)
- **Influencing neighbor** candidates

Domain Expert Knowledge Specification

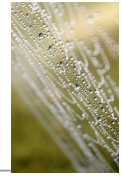




For **pairwise similarity computation**, domain expert also selects what information is **relevant** for comparisons

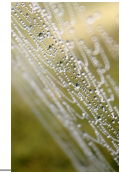
- **Object description** (attribute values)
- **Influencing neighbo**r candidates

Similarity Measure Template



$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

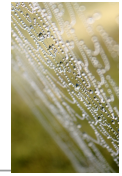
Similarity Measure Template



weight functions
(e.g., IDF)

$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

Similarity Measure Template

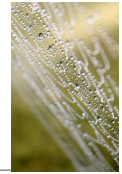


weight functions (e.g., IDF)

Set of similar **object descriptions**

$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

Similarity Measure Template



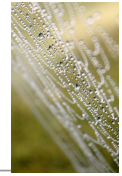
weight functions (e.g., IDF)

Set of similar **object descriptions**

Set of similar **influencing neighbors**

$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

Similarity Measure Template



weight functions (e.g., IDF)

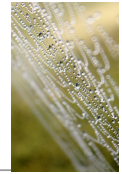
Set of similar **object descriptions**

Set of similar **influencing neighbors**

$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

Set of different **object descriptions**

Similarity Measure Template



weight functions (e.g., IDF)

Set of similar **object descriptions**

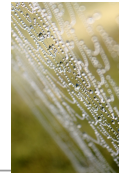
Set of similar **influencing neighbors**

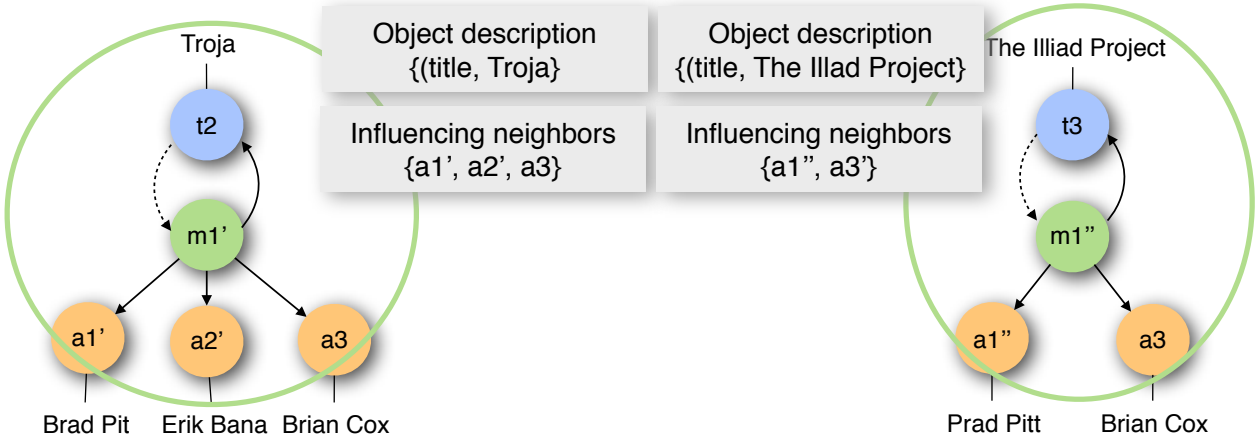
$$sim(c, c') = \frac{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx})}{w_{od}(N_{od}^{\approx}) + w_{ip}(N_{ip}^{\approx}) + w_{od}(N_{od}^{\neq}) + w_{ip}(N_{ip}^{\neq})}$$

Set of different **object descriptions**

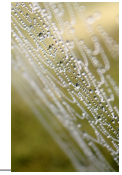
Set of different **influencing neighbors**

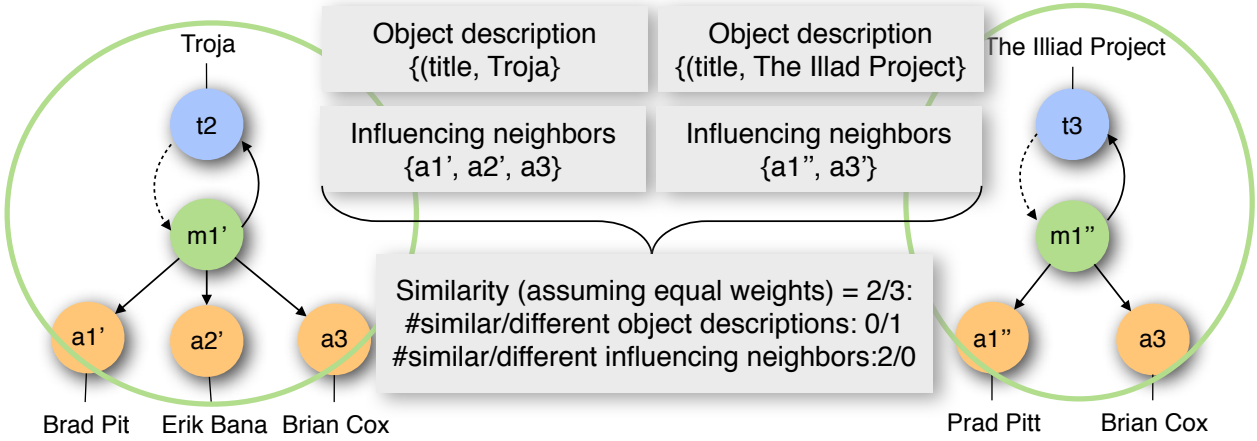
Similarity Measure Template
Example [WN05]



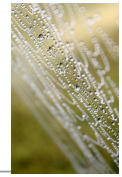


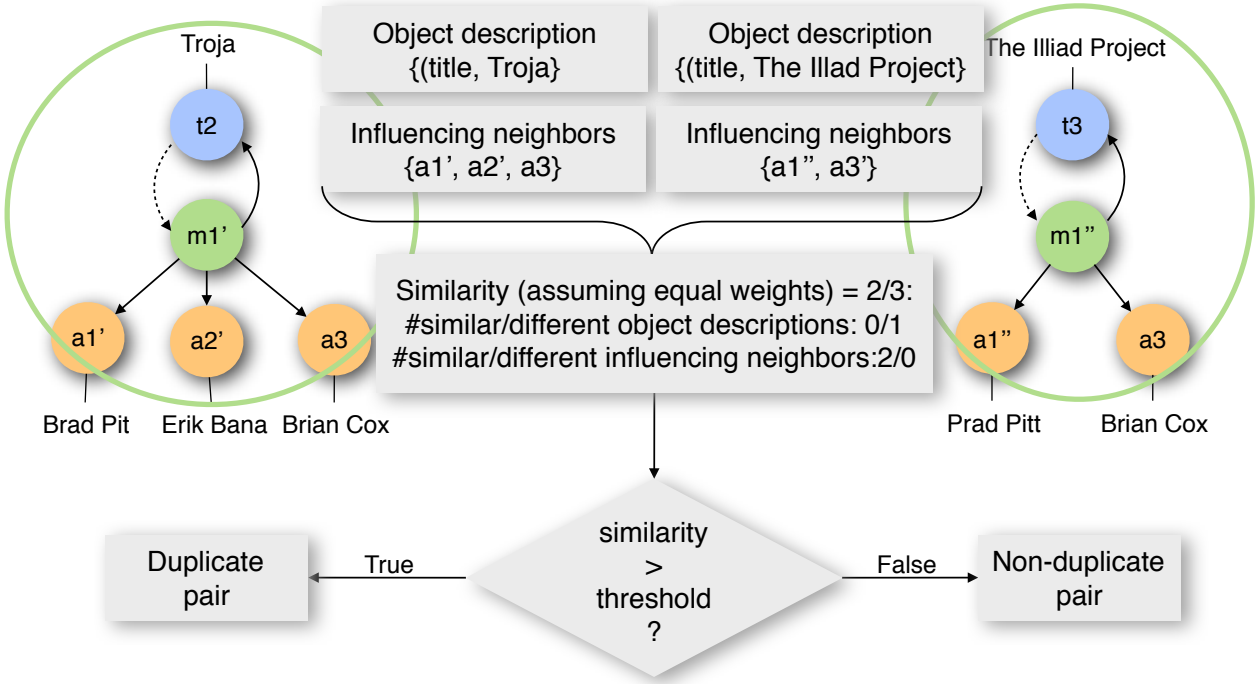
Similarity Measure Template
Example [WN05]



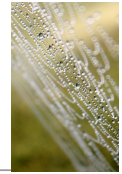


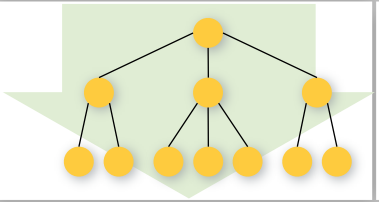
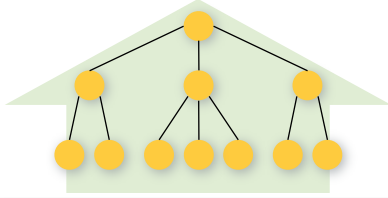
Similarity Measure Template
Example [WN05]



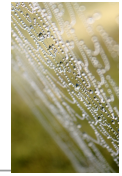


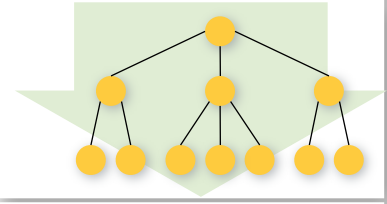
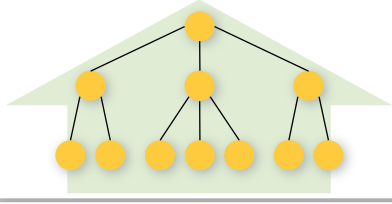
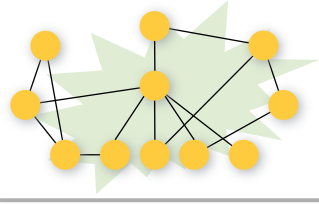
Entity Pair Queue



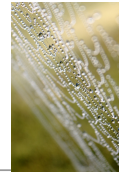
Top-down [WN04]	Bottom-up [PWN06]
	
Hierarchy represents **1:N** relationships between candidate entities.	Hierarchy represents candidate entities that can be in **M:N** relationship.
Prune comparisons of descendants that do not have same or similar ancestors.	Sorted neighborhood method applied to every candidate type.
$O(\|T1\|2+\|T2\|2+\ldots+ \|Tn\|2)$	$O(\|T1\| log\|T1\|+\ldots + \|Tn\| log\|Tn\|)$

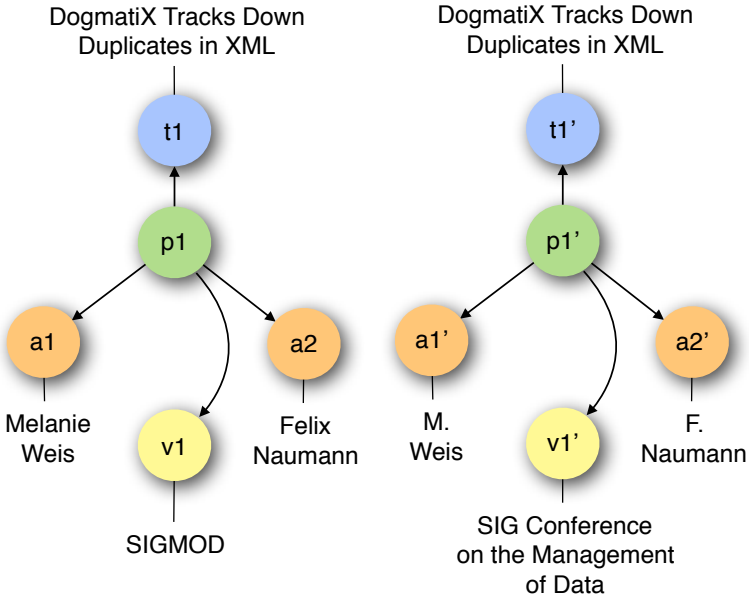
Entity Pair Queue



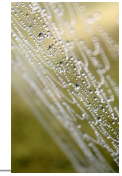
Top-down [WN04]	Bottom-up [PWN06]	Graph duplicate detection [WN06]
		
Hierarchy represents **1:N** relationships between candidate entities.	Hierarchy represents candidate entities that can be in **M:N** relationship.	Edges represent all kinds of relationships
Prune comparisons of descendants that do not have same or similar ancestors.	Sorted neighborhood method applied to every candidate type.	Pairs can be compared more than once →reduce re-comparisons by maintaining an priority queue
O(|T1|2+|T2|2+…+ |Tn|2)	*O(|T1| log|T1|+… + |Tn| log|Tn|)*	*O(|T1|2+|T2|2+…+|Tn|2)*

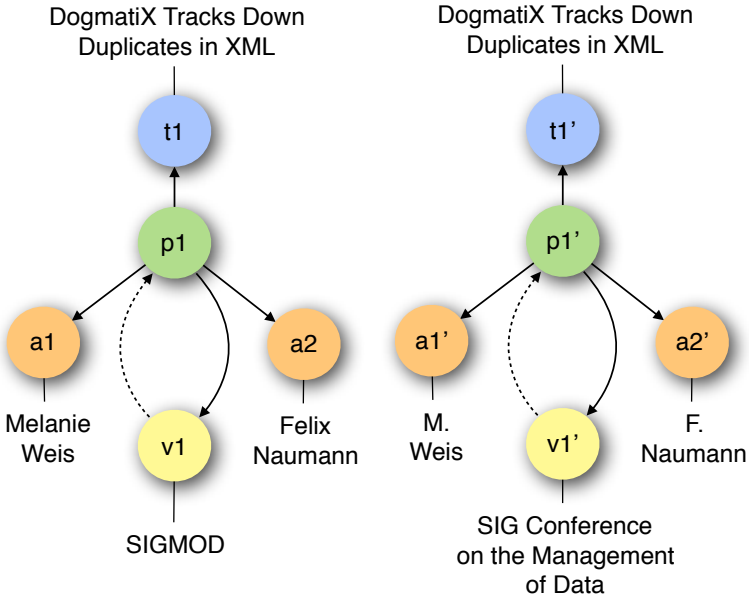
Entity Pair Queue



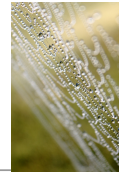


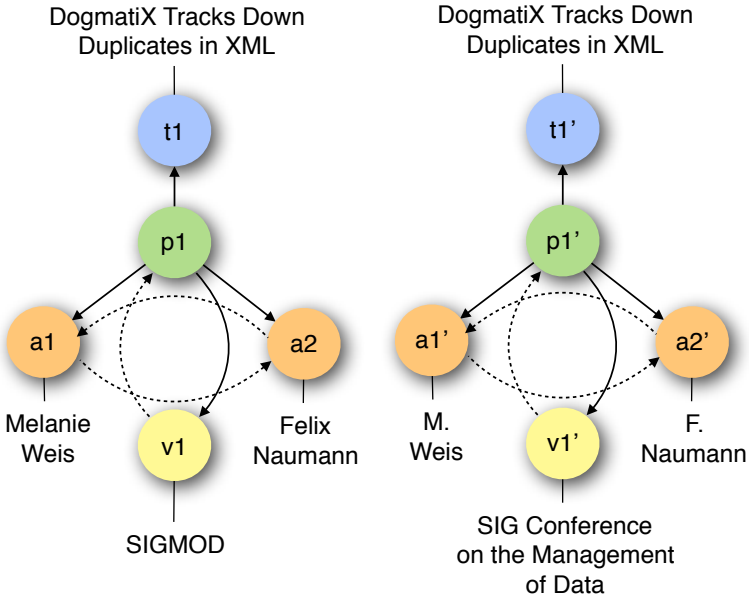
Entity Pair Queue



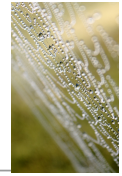


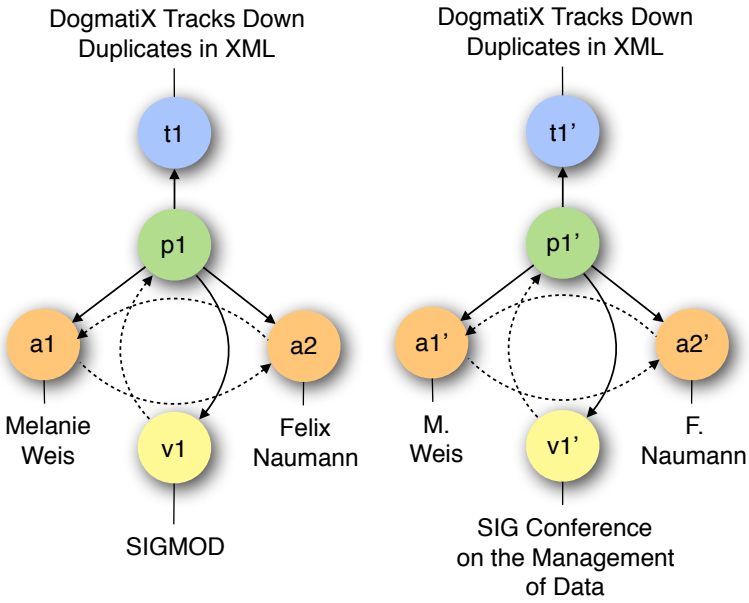
Entity Pair Queue

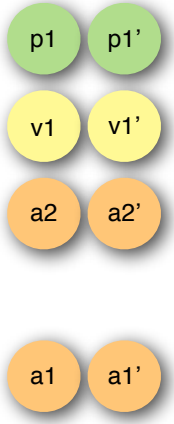




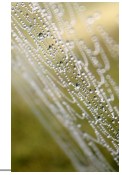
Entity Pair Queue

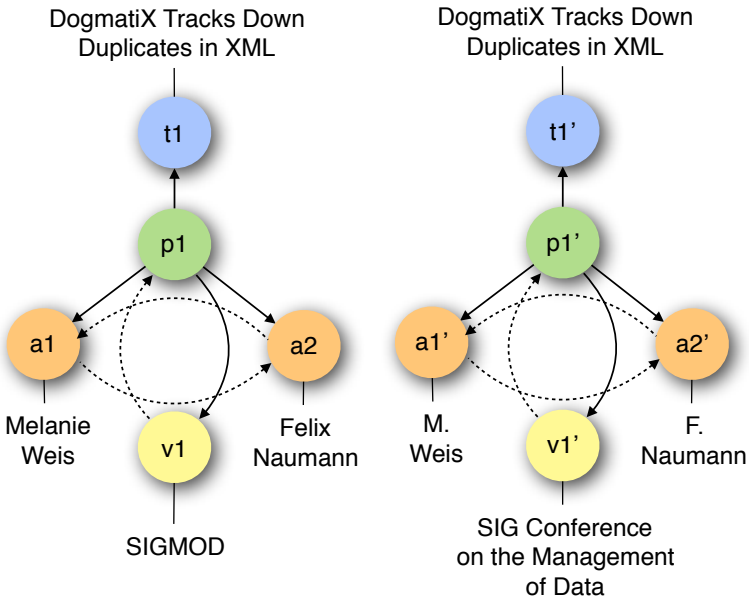


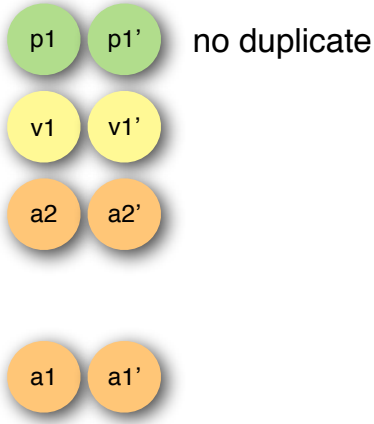




Entity Pair Queue

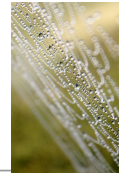


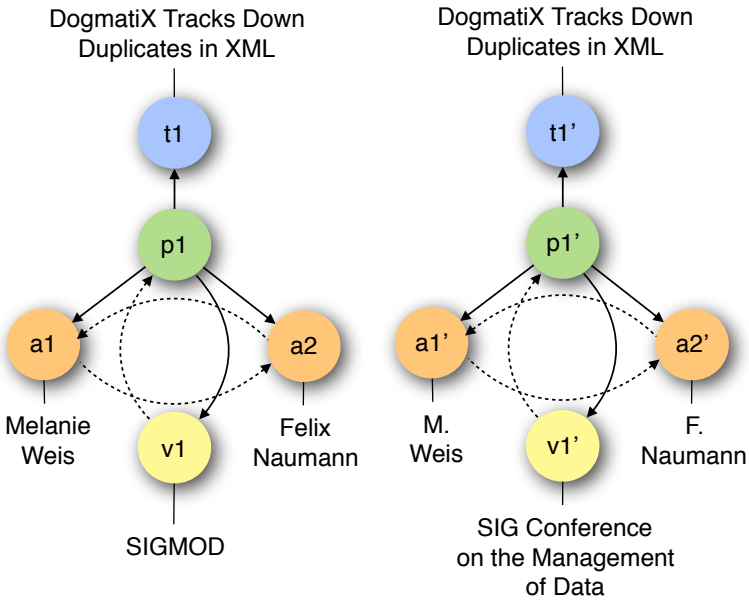


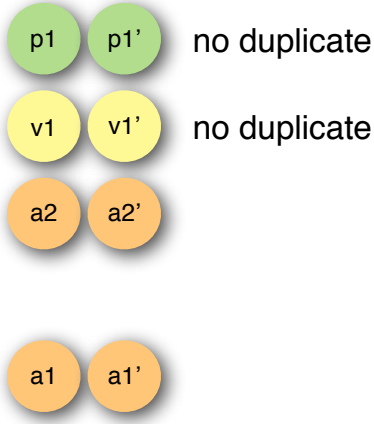


no duplicate

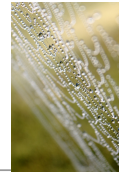
Entity Pair Queue

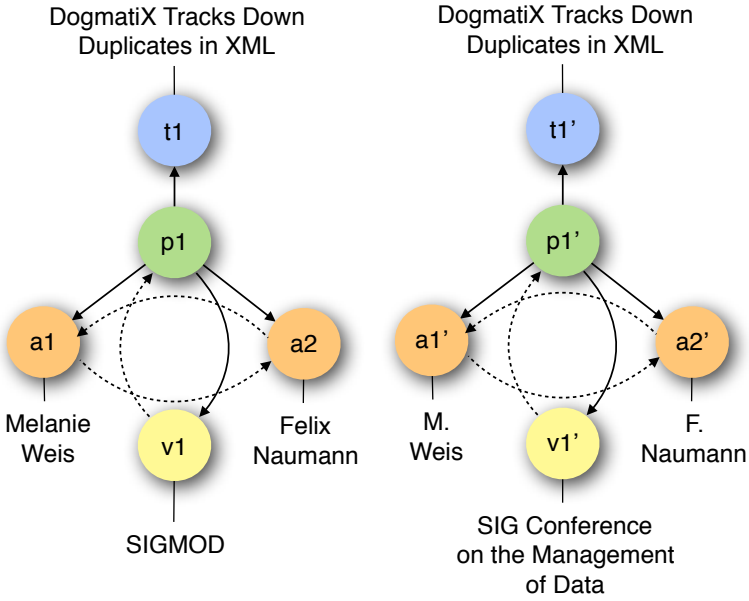


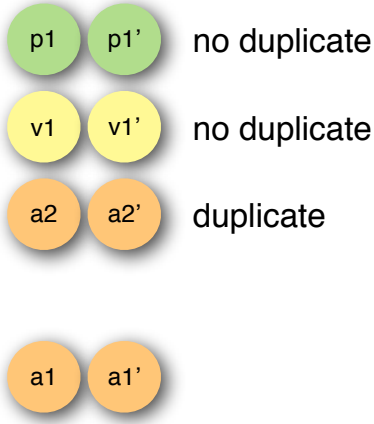




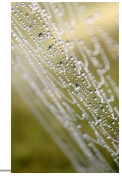
Entity Pair Queue

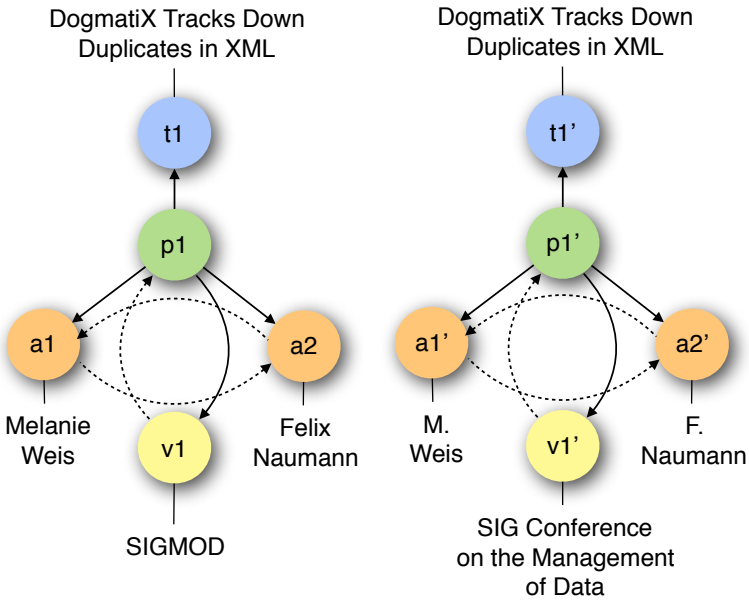


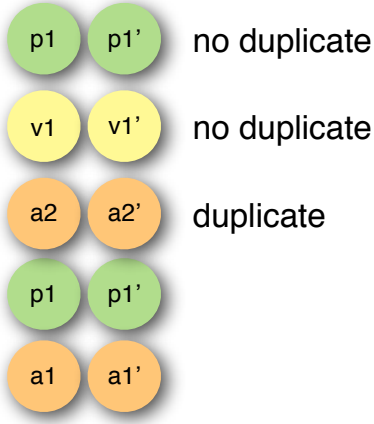




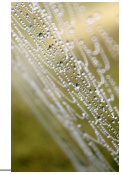
Entity Pair Queue

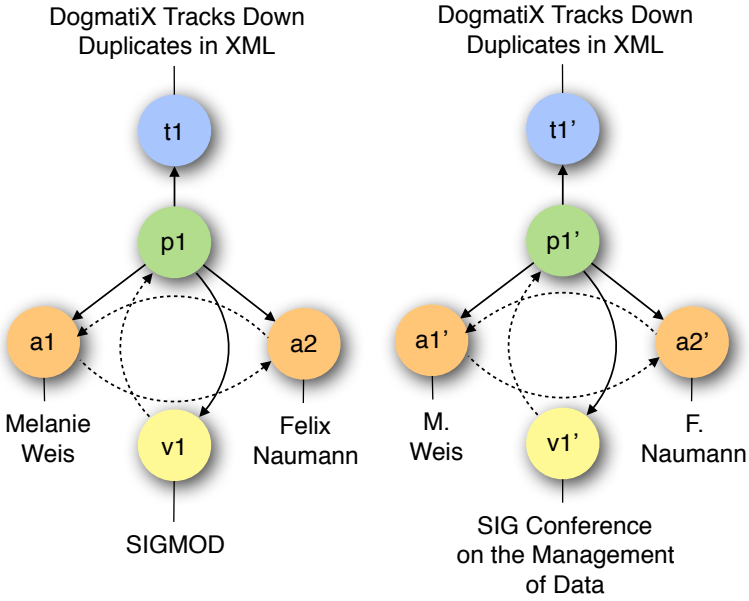


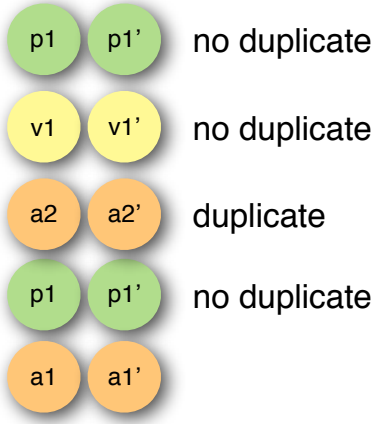




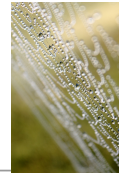
Entity Pair Queue

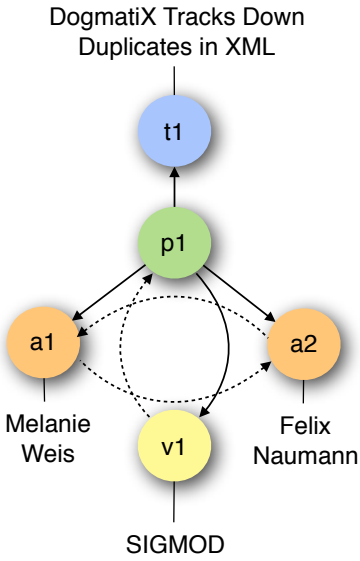


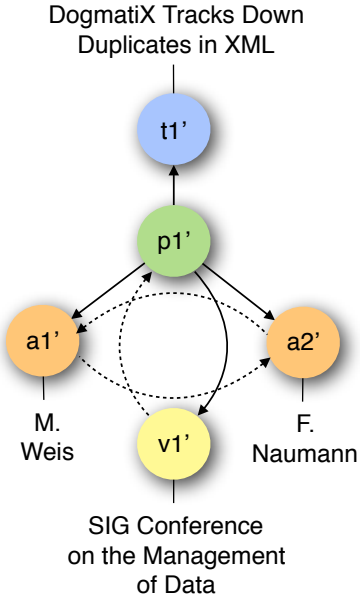


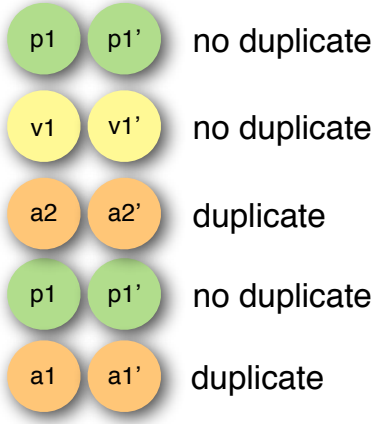


Entity Pair Queue

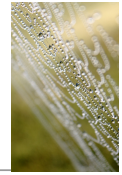


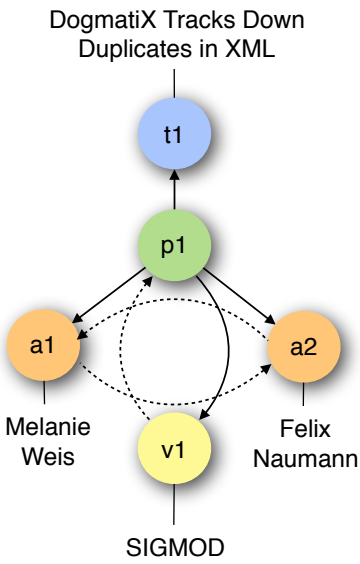


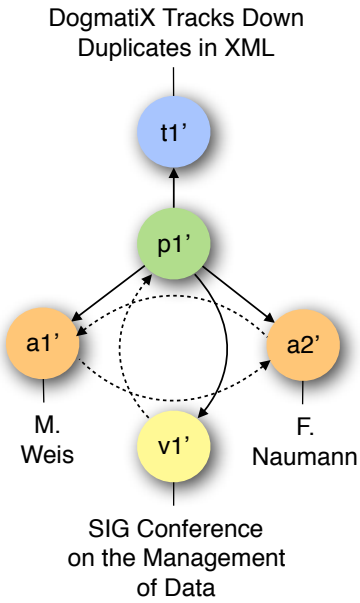


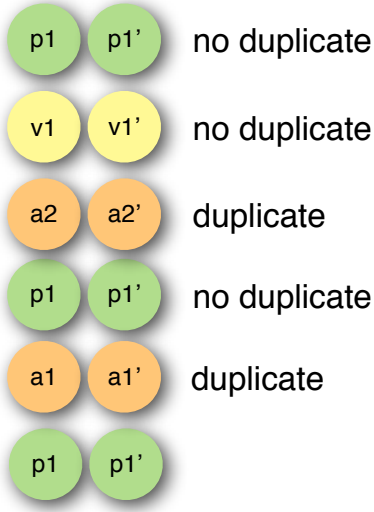


Entity Pair Queue

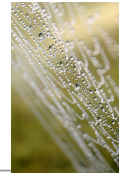


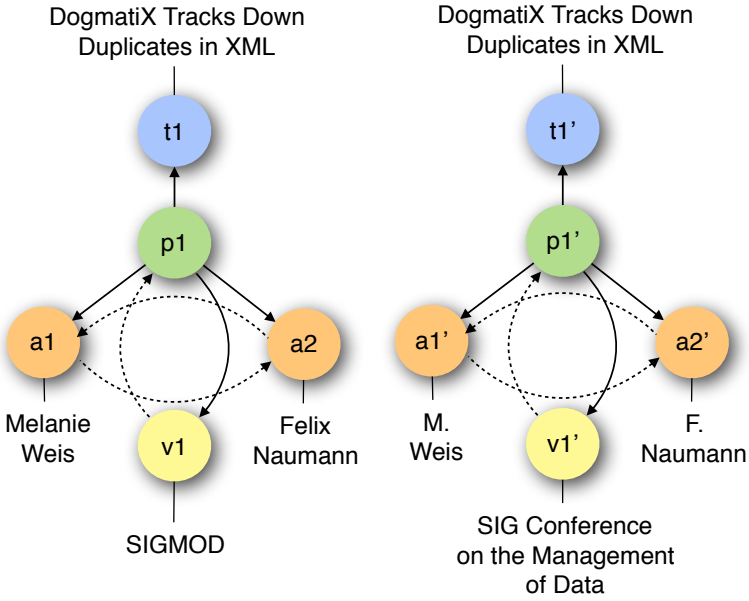


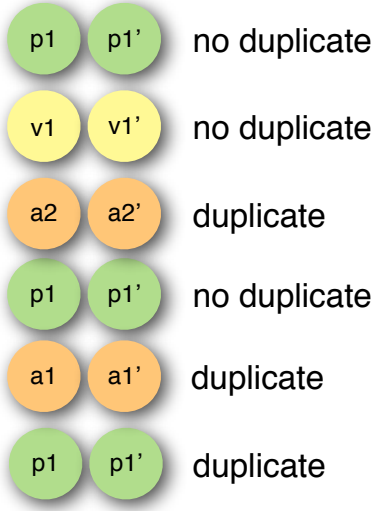




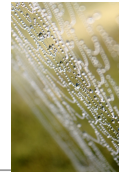
Entity Pair Queue

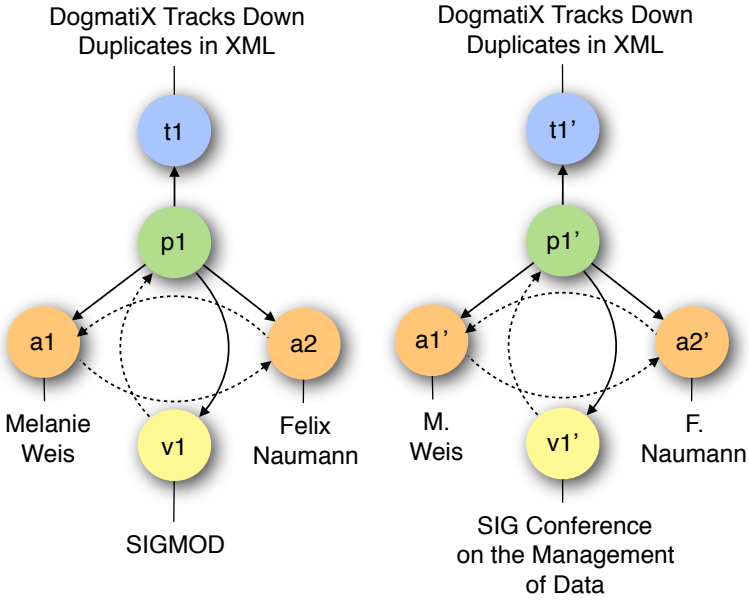


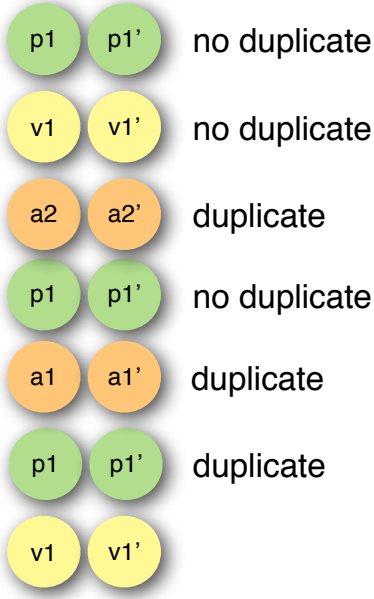




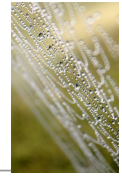
Entity Pair Queue

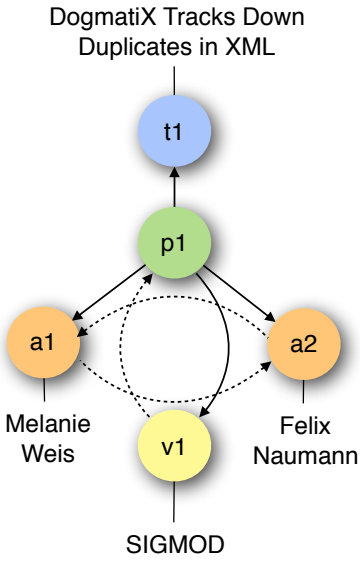


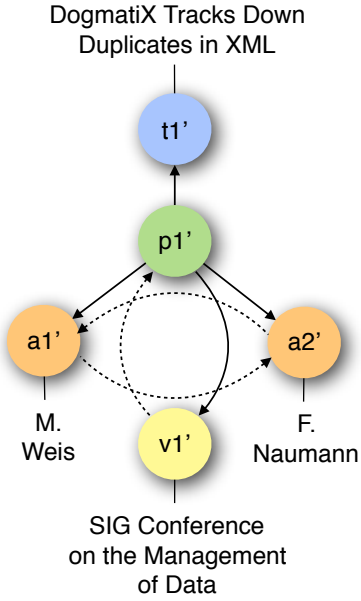


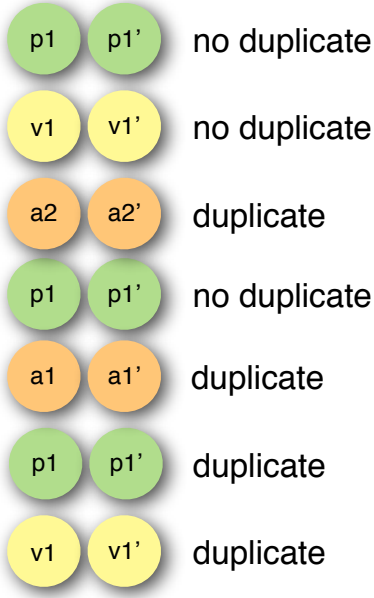


Entity Pair Queue

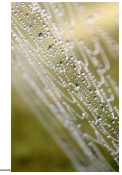


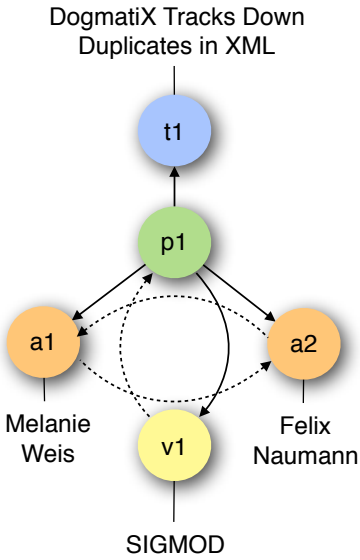


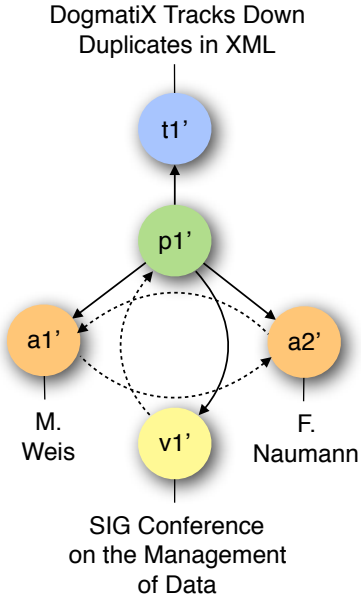


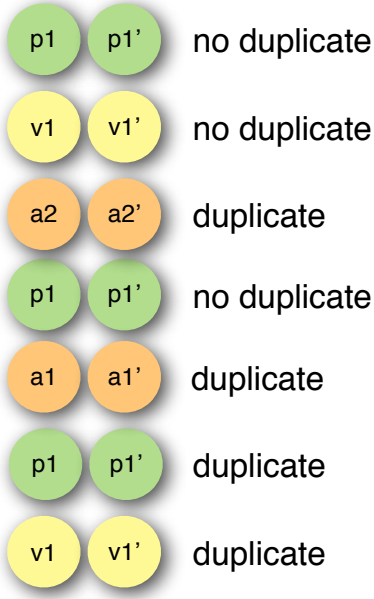


Entity Pair Queue



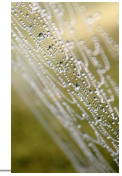


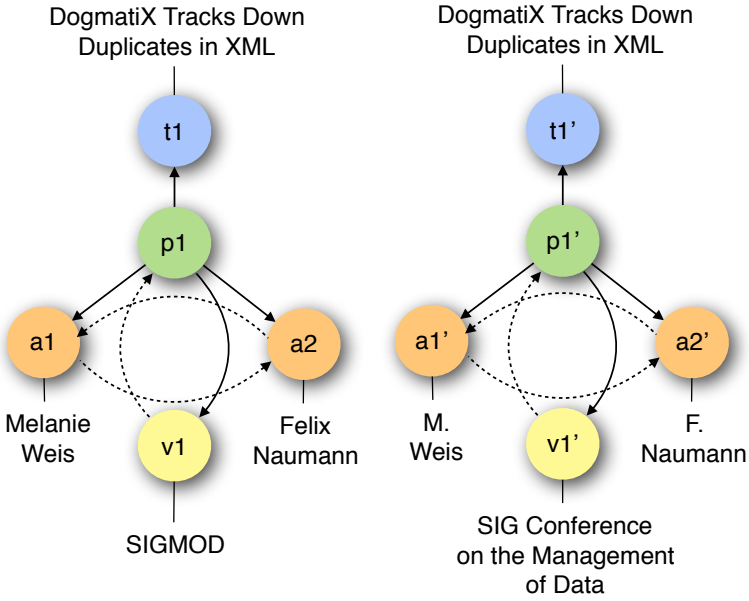


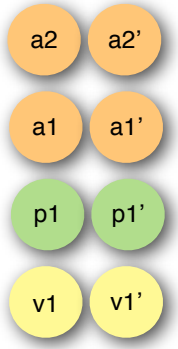


7 comparisons
(3 re-comparisons)

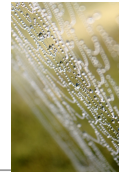
Entity Pair Queue

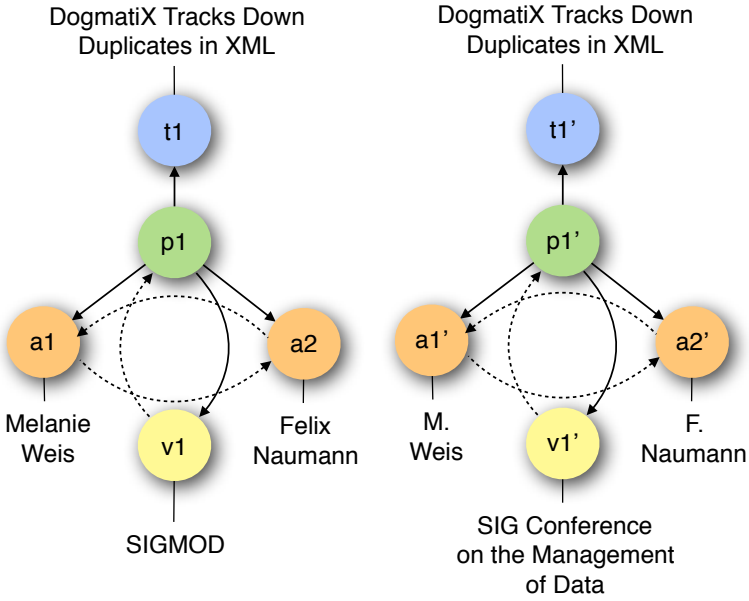


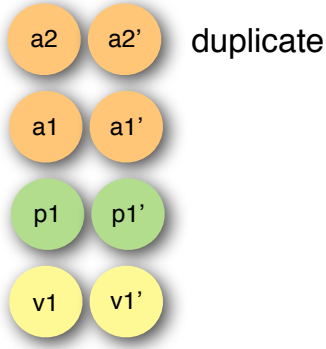




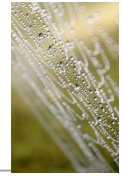
Entity Pair Queue

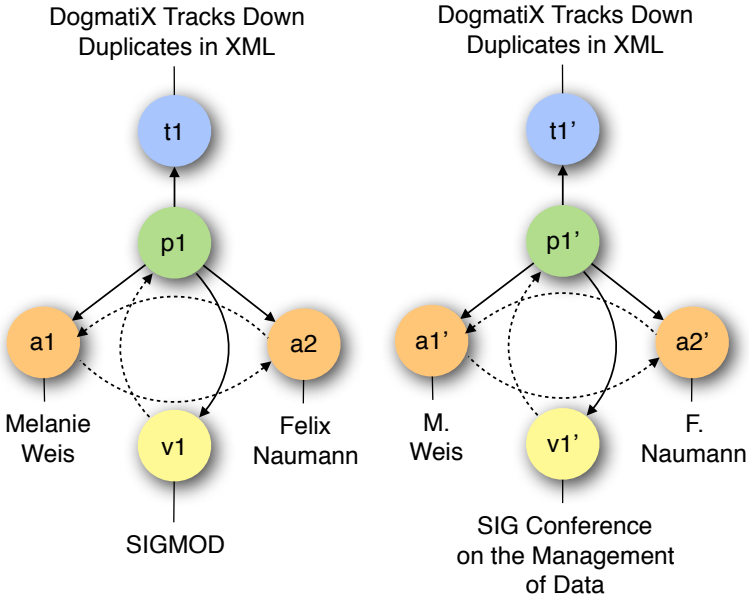


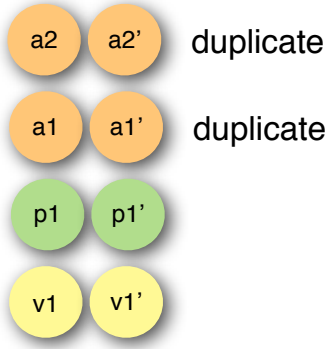




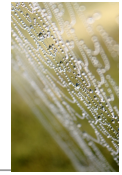
Entity Pair Queue

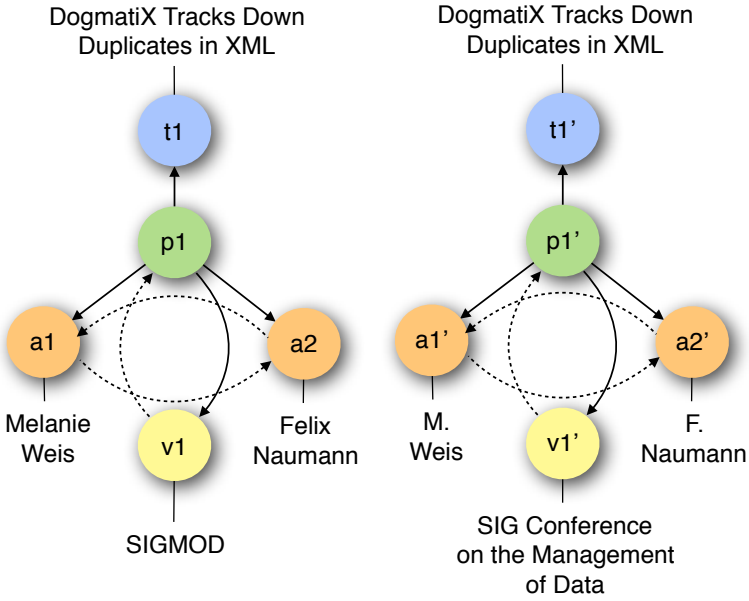


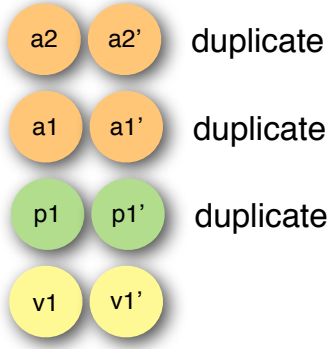




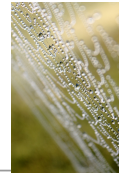
Entity Pair Queue

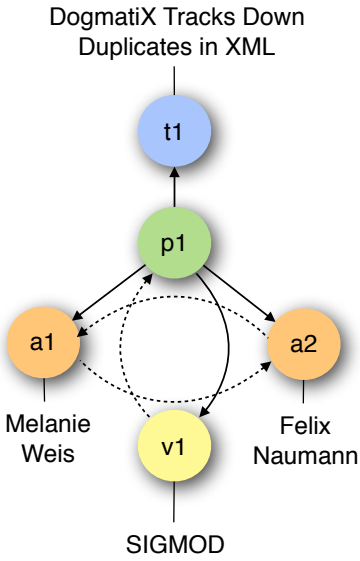


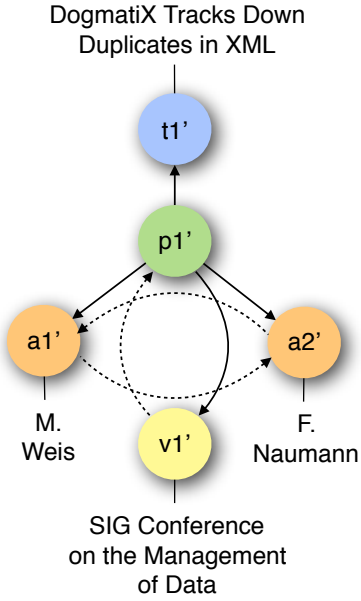


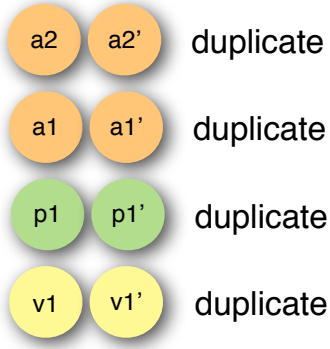


Entity Pair Queue

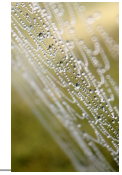


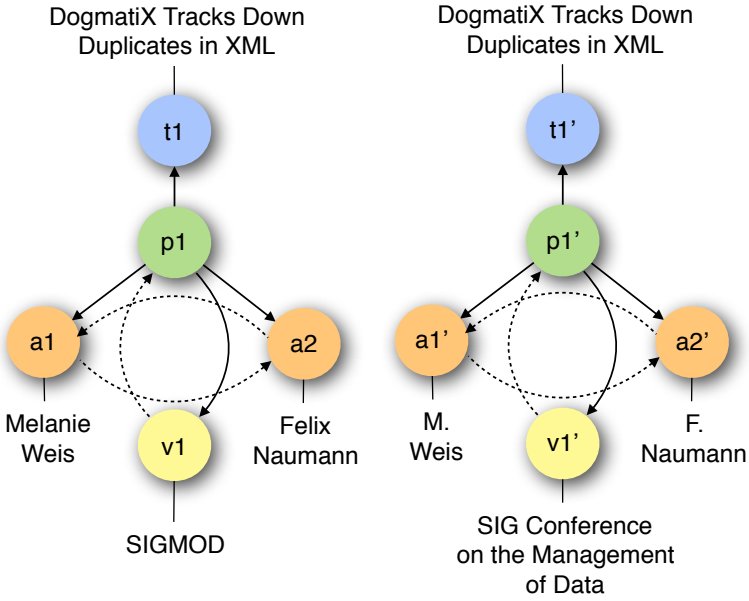


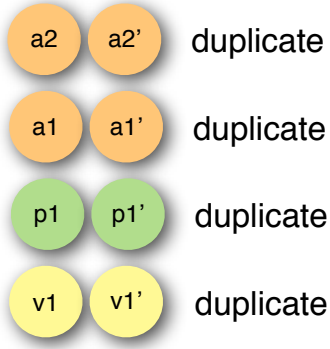




Entity Pair Queue

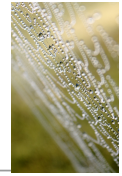






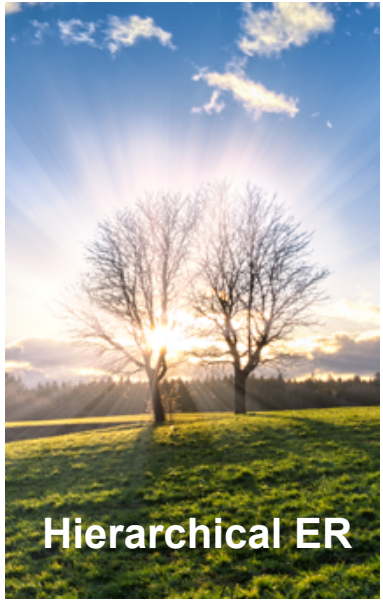
4 comparisons
(0 re-comparisons)

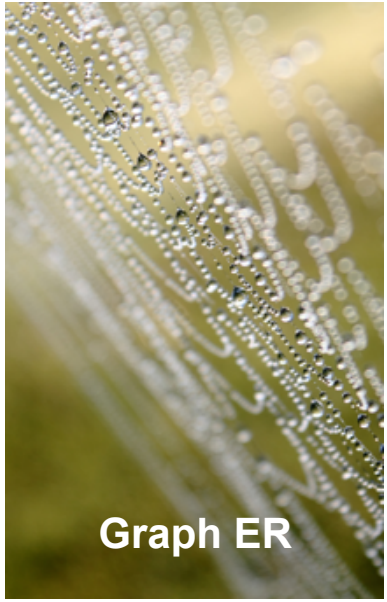
Entity Pair Queue to Manage Comparison Order



- Queue **maintenance** necessary whenever a **duplicate is found**.
- In general, goal of maintaining the priority queue is to r**educe the number of re-comparisons**.
- Different strategies
  - Based on **heuristics** (degree of nodes in graph) [WN06]
  - Based on calculation of (approximate) **similarities** [BG07].
  - Based on different **edge types** (FIFO, LIFO) [DHM05]
  - **Lazy** maintenance [HNST12]

Closing Remarks



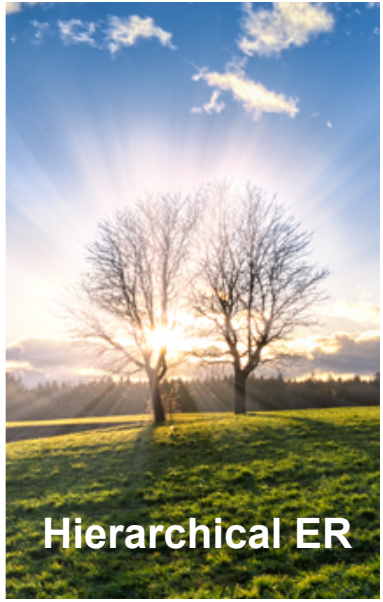


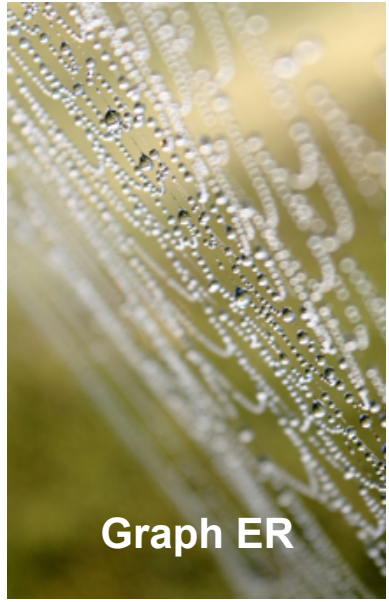
Closing Remarks

Similarity Measures:
- Cosine similarity
- Jaro & Jaro Winkler

Sorted Neighborhood:
- Effective
- Efficient
- Scalable

Relational ER





Closing Remarks

Similarity Measures:
- Cosine similarity
- Jaro & Jaro Winkler

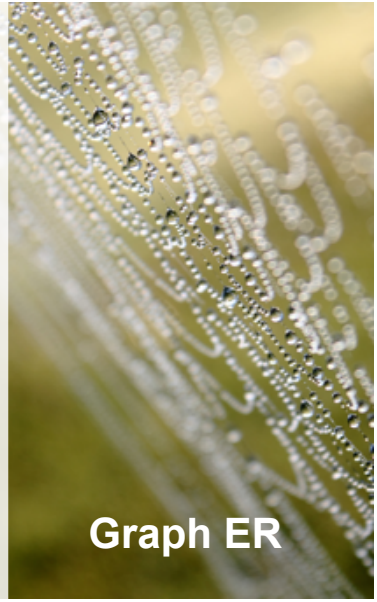
Sorted Neighborhood:
- Effective
- Efficient
- Scalable

Relational ER

Similarity Measures:
- Containment Metric

Comparison Orders:
- Top-Down (1:N)
- Bottom-Up (M:N)

Hierarchical ER



Closing Remarks

Relational ER

Similarity Measures:
- Cosine similarity
- Jaro & Jaro Winkler

Sorted Neighborhood:
- Effective
- Efficient
- Scalable

Hierarchical ER

Similarity Measures:
- Containment Metric

Comparison Orders:
- Top-Down (1:N)
- Bottom-Up (M:N)

Graph ER

Similarity Measure:
- Template
- Expert knowledge

Comparison Order:
- PQ maintenance
- Reduce number of re-comparisons