Blocking and Relational Entity Resolution

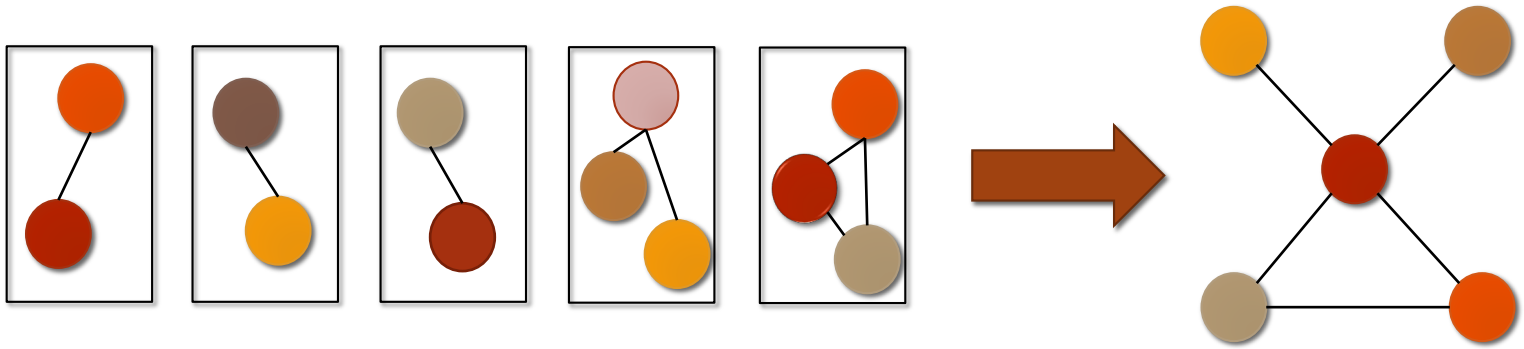
JAY PUJARA

DSCI-558 SPRING 2021

SOME SLIDES FROM HTTP://USERS.UMIACS.UMD.EDU/~GETOOR/TUTORIALS/ER_VLDB2012.PDF

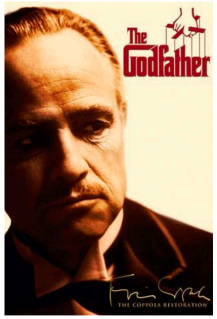
A quick reminder

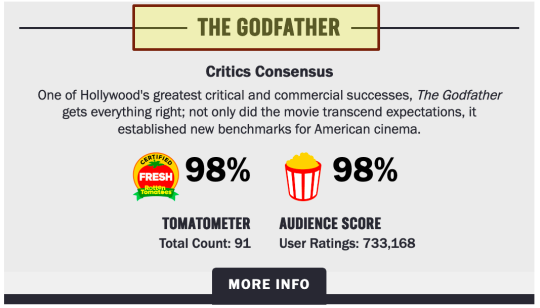
Deduplication

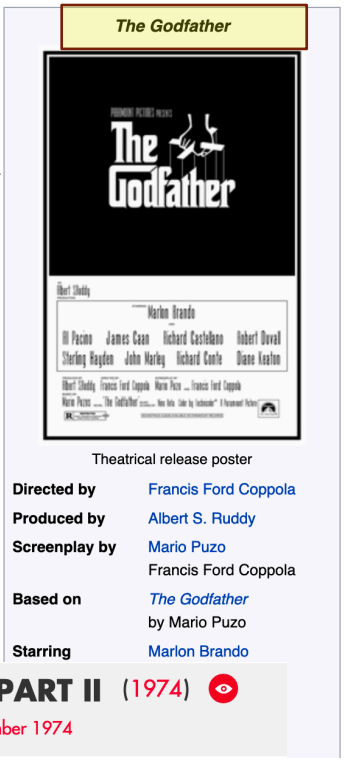


Merging Ambiguous Entities

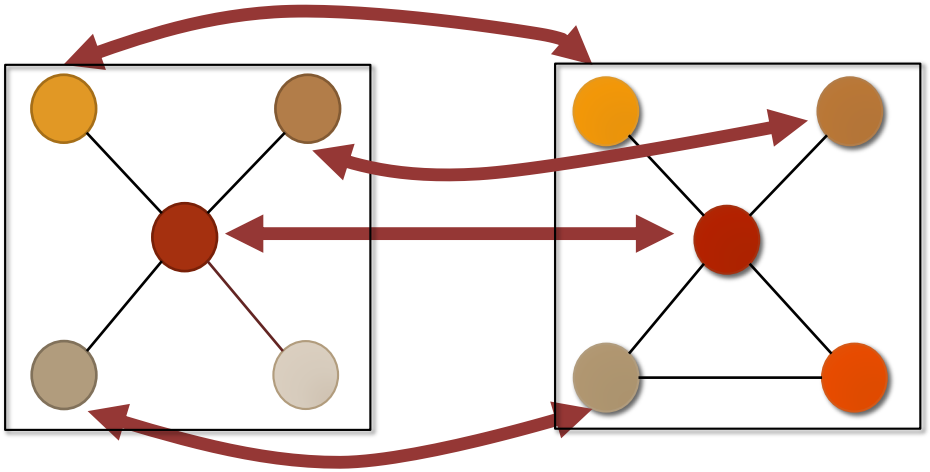
Deduplication Example





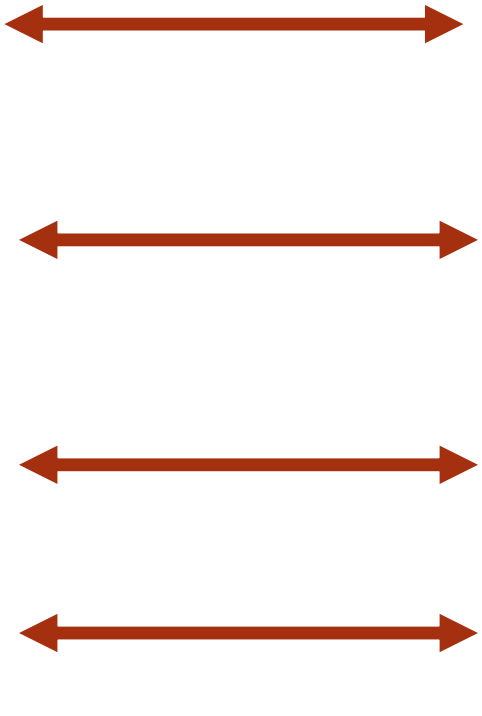


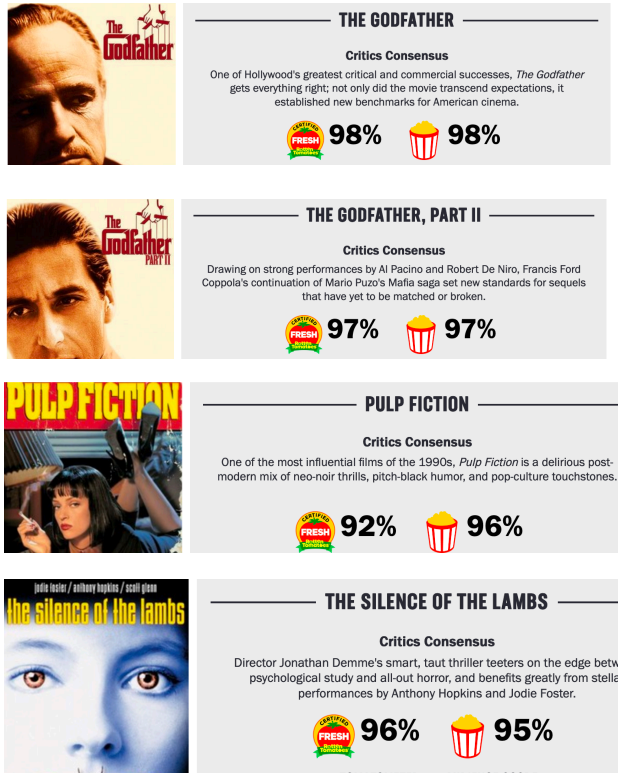
Record Linkage



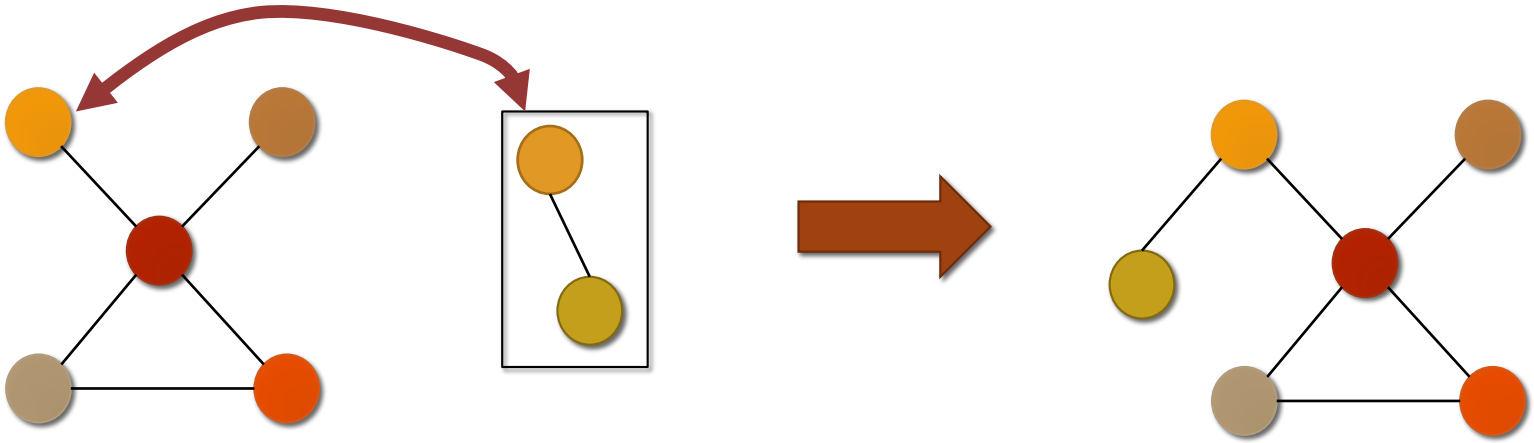
Combining KGs

Record Linkage Example





Entity Mapping/Linking



Integrating New Candidates

Entity Linking Example

TIL that in the movie The Godfather, the reason the word 'mafia' is not mentioned a single time is because mafia boss Joe Colombo, along with Frank Sinatra, threatened the films production and would only back the filming if they could change the script to their liking.

Blocking

Blocking: Motivation

- Naïve pairwise: $|R|^2$ pairwise comparisons
  - 1000 business listings each from 1,000 different cities across the world
  - 1 trillion comparisons
  - 11.6 days (if each comparison is 1 µs)

- Mentions from different cities are unlikely to be matches
  - **Blocking Criterion: City**
  - 1 billion comparisons
  - 16 minutes (if each comparison is 1 µs)

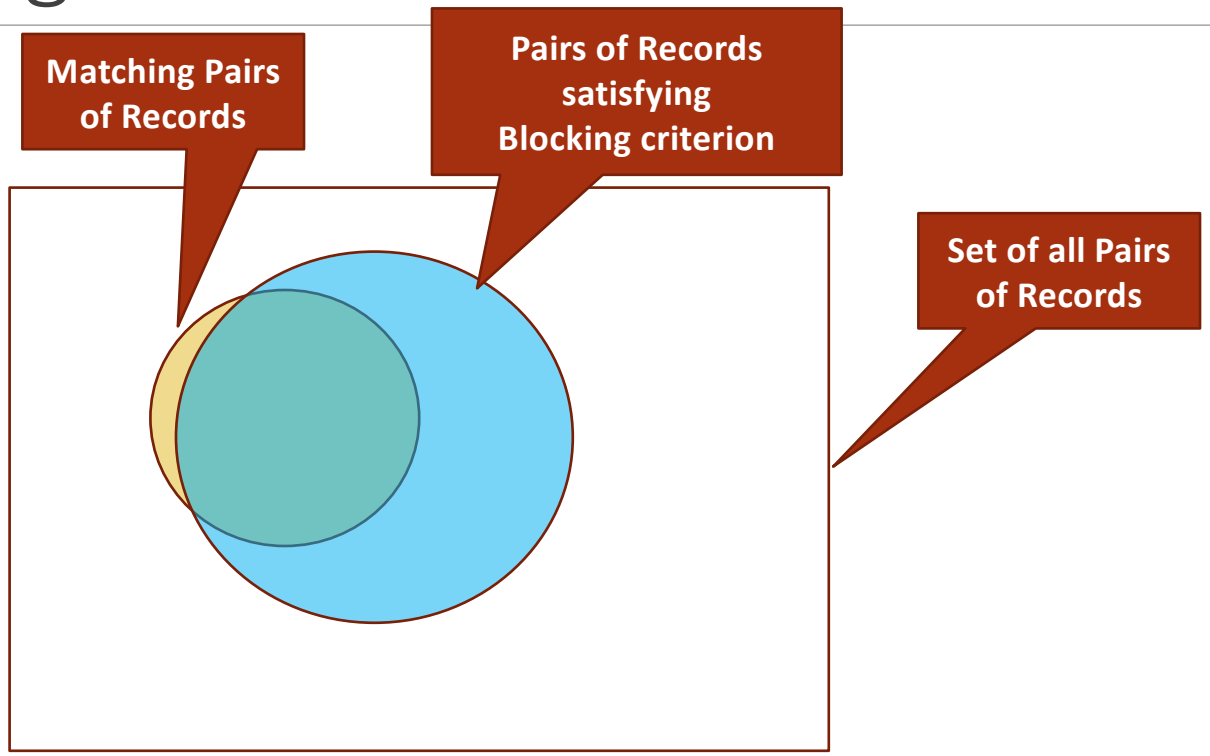
Blocking: key ideas

- Comparing each entity with all other entities is too computationally demanding – $O(N^2)$

- Need to find a way to reduce the number of comparisons

- Partition entities into ”blocks”– O(N)
- Make only within block comparisons [so if largest block is log N in size – O(N log N)]

Blocking: Motivation



Blocking: Problem Statement

Input: Set of records *R*

Output: Set of *blocks/canopies*

$$\{C_1, C_2, \ldots, C_k\}, where\ \forall_i C_i \subset R\ and \bigcup_i C_i = R$$

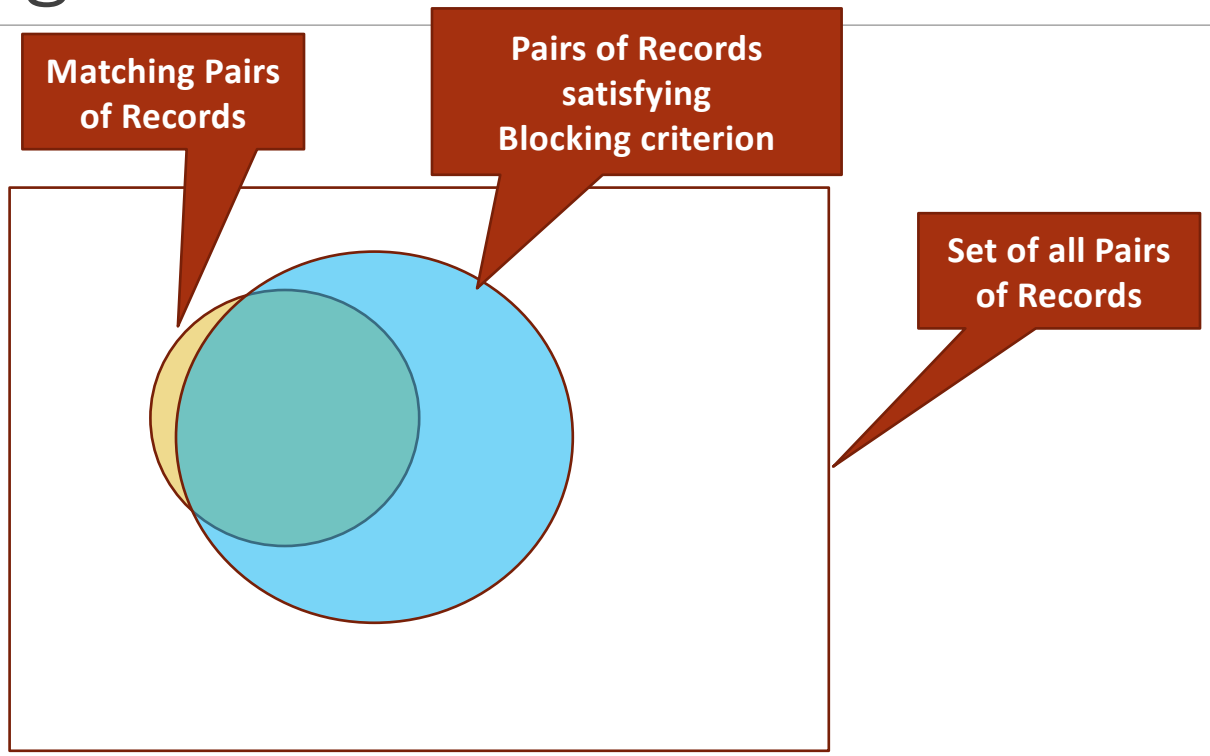
Variants:

- *Disjoint Blocking*: Each mention appears in one block.

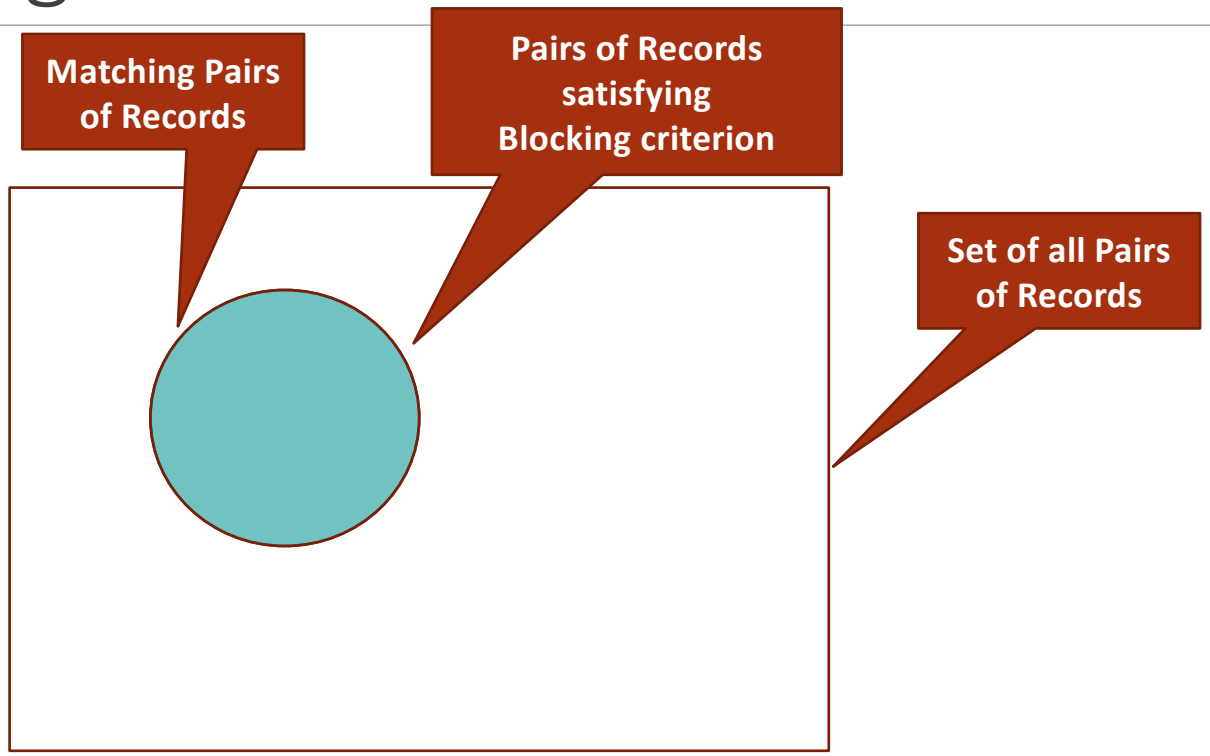
$$\forall_{i,j} C_i \cap C_j = \emptyset$$

- *Non-disjoint Blocking*: Mentions can appear in more than one block.

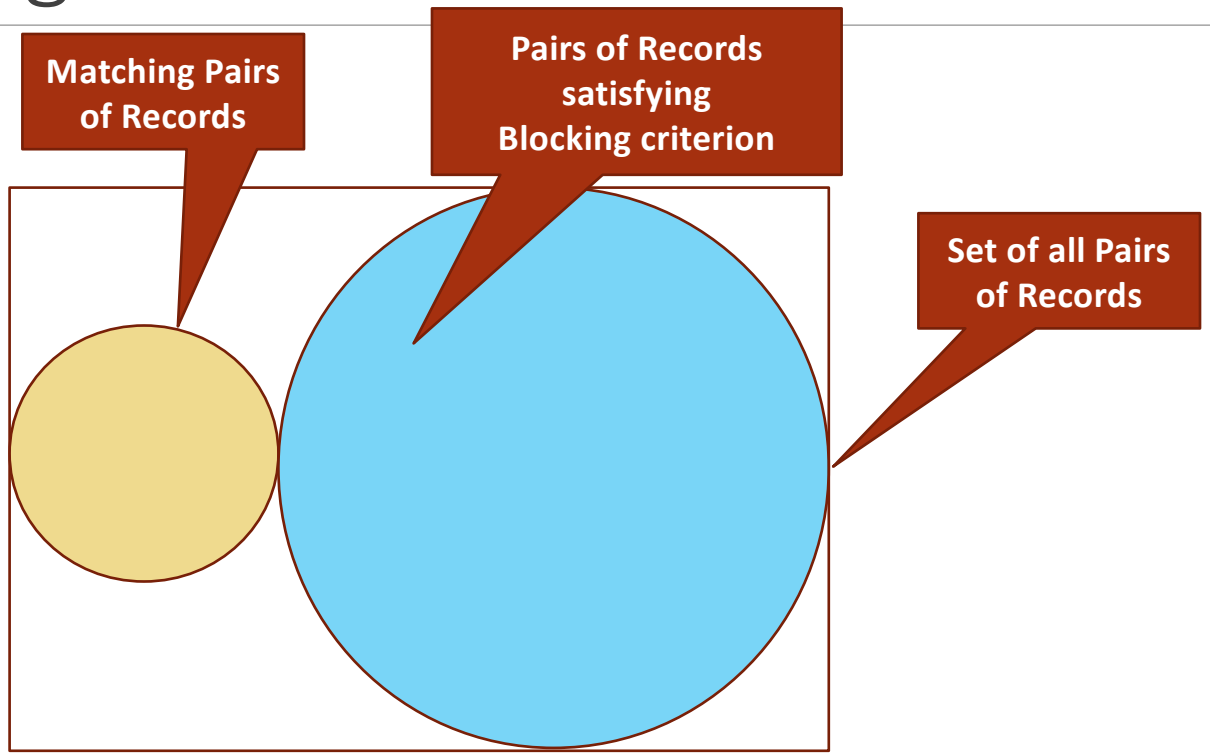
Blocking: Motivation



Blocking: Ideal scenario



Blocking: worst-case scenario



Blocking: Problem Statement

Metrics:

- Efficiency (or reduction ratio) :

$$\frac{number\ of\ pairs\ compared}{total\ number\ of\ pairs\ in\ R \times R}$$

- Recall* (or pairs completeness) :

$$\frac{number\ of\ true\ matches\ compared}{number\ of\ true\ matches\ in\ R \times R}$$

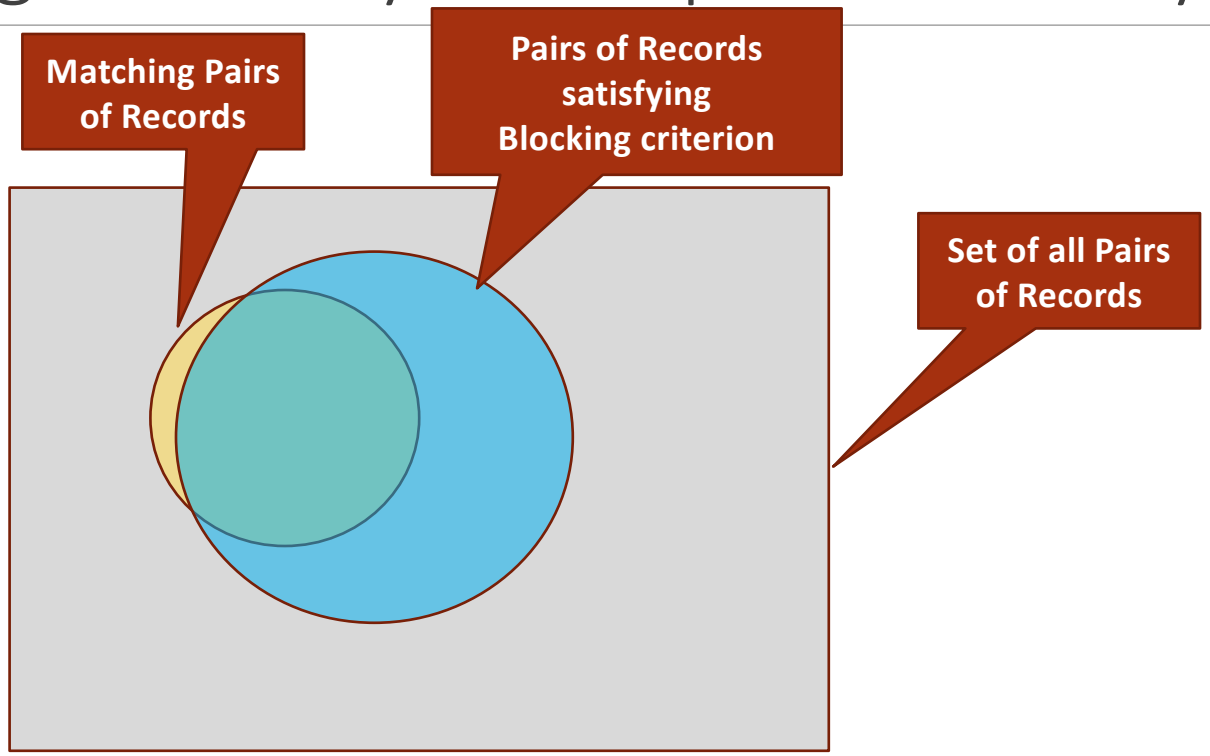
- Precision* (or pairs quality) :

$$\frac{number\ of\ true\ matches\ compared}{number\ of\ matches\ compared}$$

- Max Canopy Size: $max_i\ |C_i|$

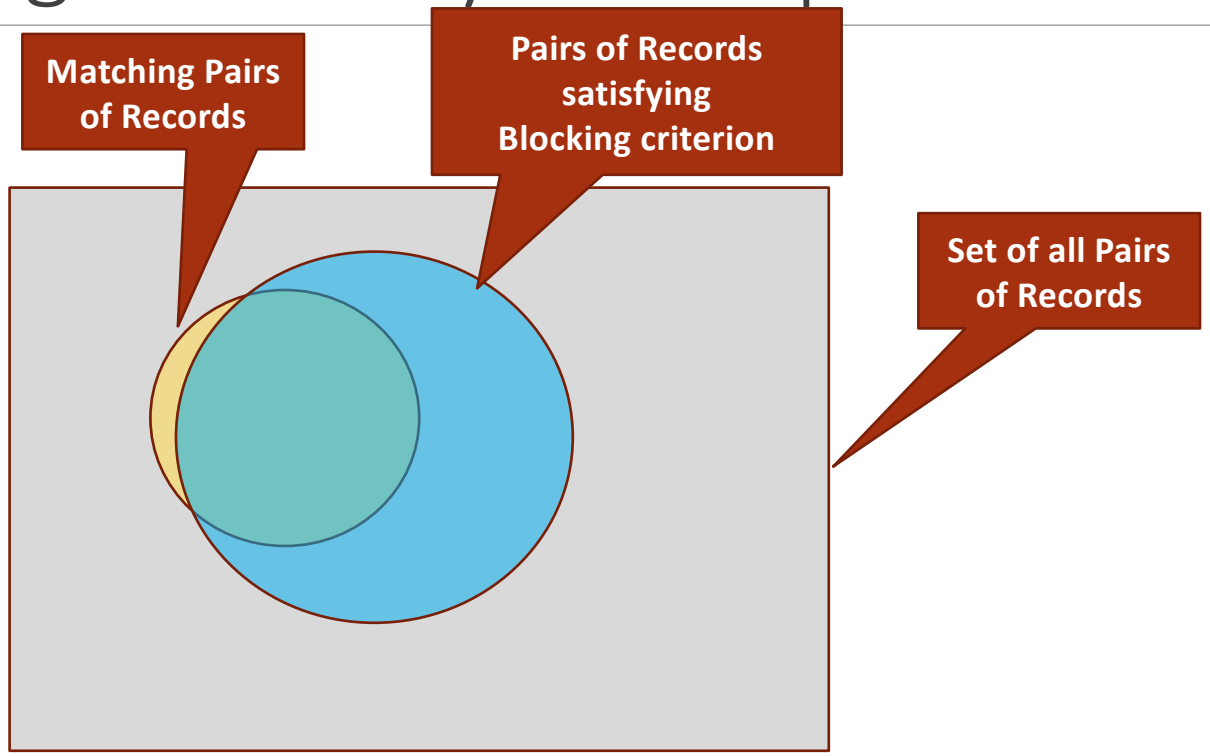
**Need to know ground truth in order to compute this metric*

Blocking: How do you compute efficiency?



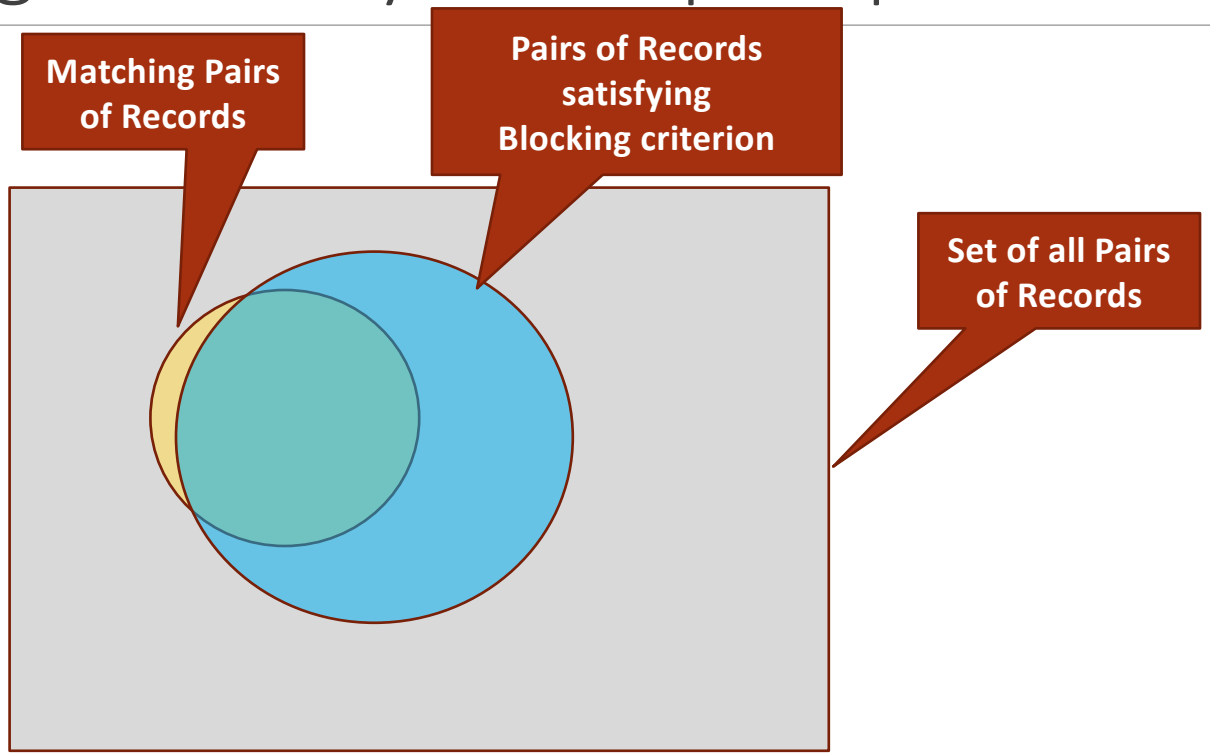
$$\frac{number\ of\ pairs\ compared}{total\ number\ of\ pairs\ in\ R \times R}$$

Blocking: How do you compute recall?



$$\frac{number\ of\ true\ matches\ compared}{number\ of\ true\ matches\ in\ R \times R}$$

Blocking: How do you compute precision?



$$\frac{number\ of\ true\ matches\ compared}{number\ of\ matches\ compared}$$

How do we block?

- Feature-based blocking keys
- Clustering or sorting
- Hashing

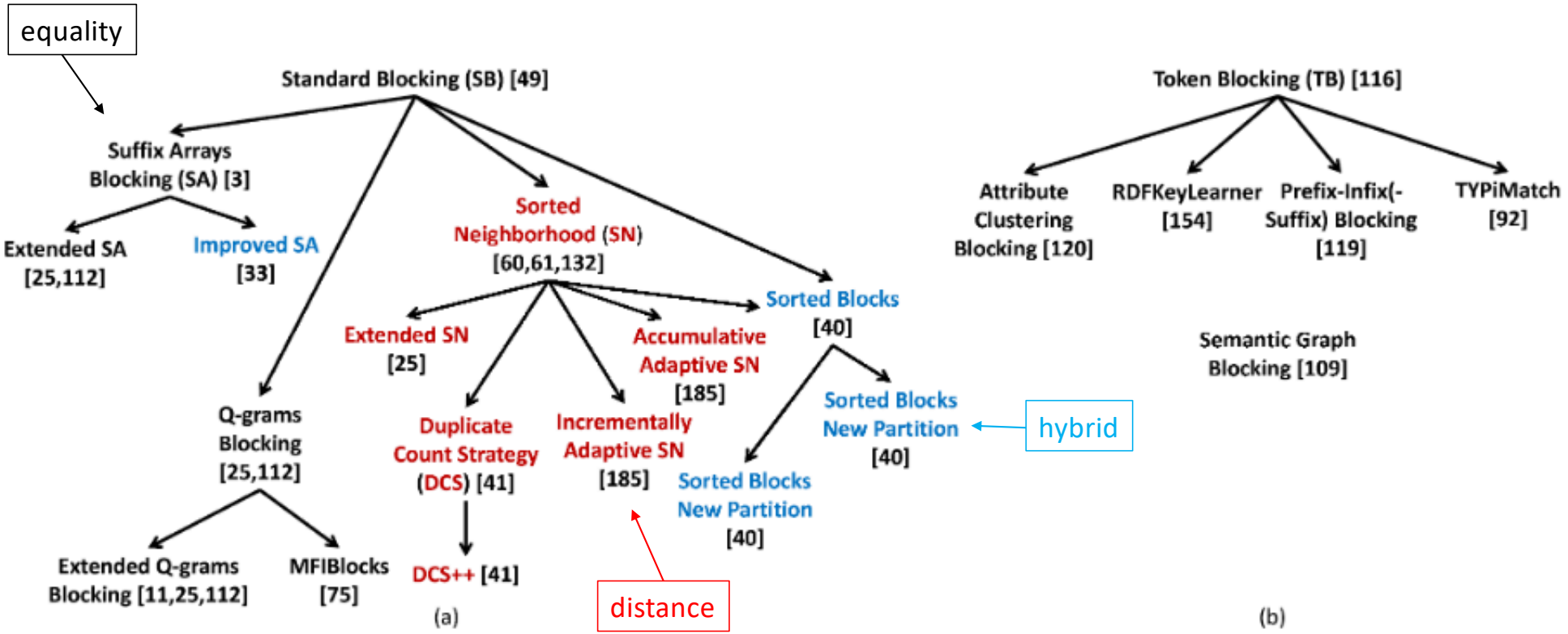
Table 1. Taxonomy of the Block Building methods discussed in Sections 3.2 and 3.3.

Method	Key type	Redundancy awareness	Constraint awareness	Matching awareness
Standard Blocking (SB) [49]	hash-based	redundancy-free	lazy	static
Suffix Arrays Blocking (SA) [3]	hash-based	redundancy-positive	proactive	static
Extended Suffix Arrays Blocking [25, 112]	hash-based	redundancy-positive	proactive	static
Improved Suffix Arrays Blocking [33]	hash-based	redundancy-positive	proactive	static
Q-Grams Blocking [25, 112]	hash-based	redundancy-positive	lazy	static
Extended Q-Grams Blocking [11, 25, 112]	hash-based	redundancy-positive	lazy	static
MFIBlocks [75]	hash-based	redundancy-positive	proactive	static
Sorted Neighborhood (SN) [60, 61, 132]	sort-based	redundancy-neutral	proactive	static
Extended Sorted Neighborhood [25]	sort-based	redundancy-neutral	lazy	static
Incrementally Adaptive SN [185]	sort-based	redundancy-neutral	proactive	static
Accumulative Adaptive SN [185]	sort-based	redundancy-neutral	proactive	static
Duplicate Count Strategy (DCS) [41]	sort-based	redundancy-neutral	proactive	dynamic
DCS++ [41]	sort-based	redundancy-neutral	proactive	dynamic
Sorted Blocks [40]	hybrid	redundancy-neutral	lazy	static
Sorted Blocks New Partition [40]	hybrid	redundancy-neutral	proactive	static
Sorted Blocks Sliding Window [40]	hybrid	redundancy-neutral	proactive	static
(a) Non-learning, schema-aware methods.				
ApproxRBSetCover [16]	hash-based	redundancy-positive	lazy	static
ApproxDNF [16]	hash-based	redundancy-positive	lazy	static
Blocking Scheme Learner (BSL) [100]	hash-based	redundancy-positive	lazy	static
Conjunction Learner [21] (semi-supervised)	hash-based	redundancy-positive	lazy	static
BGP [48]	hash-based	redundancy-positive	lazy	static
CBlock [146]	hash-based	redundancy-positive	proactive	static
DNF Learner [54]	hash-based	redundancy-positive	lazy	dynamic
FisherDisjunctive [72] (unsupervised)	hash-based	redundancy-positive	lazy	static
(b) Learning-based (supervised), schema-aware methods.				
Token Blocking (TB) [116]	hash-based	redundancy-positive	lazy	static
Attribute Clustering Blocking [120]	hash-based	redundancy-positive	lazy	static
RDFKeyLearner [154]	hash-based	redundancy-positive	lazy	static
Prefix-Infix(-Suffix) Blocking [119]	hash-based	redundancy-positive	lazy	static
TYPiMatch [92]	hash-based	redundancy-positive	lazy	static
Semantic Graph Blocking [109]	-	redundancy-neutral	proactive	static
(c) Non-learning, schema-agnostic methods.				
Hetero [73]	hash-based	redundancy-positive	lazy	static
Extended DNF BSL [74]	hash-based	redundancy-positive	lazy	static
(d) Learning-based (unsupervised), schema-agnostic methods.				

Broad list of blocking techniques

See https://arxiv.org/pdf/1905.06167.pdf

Genealogy of blocking techniques

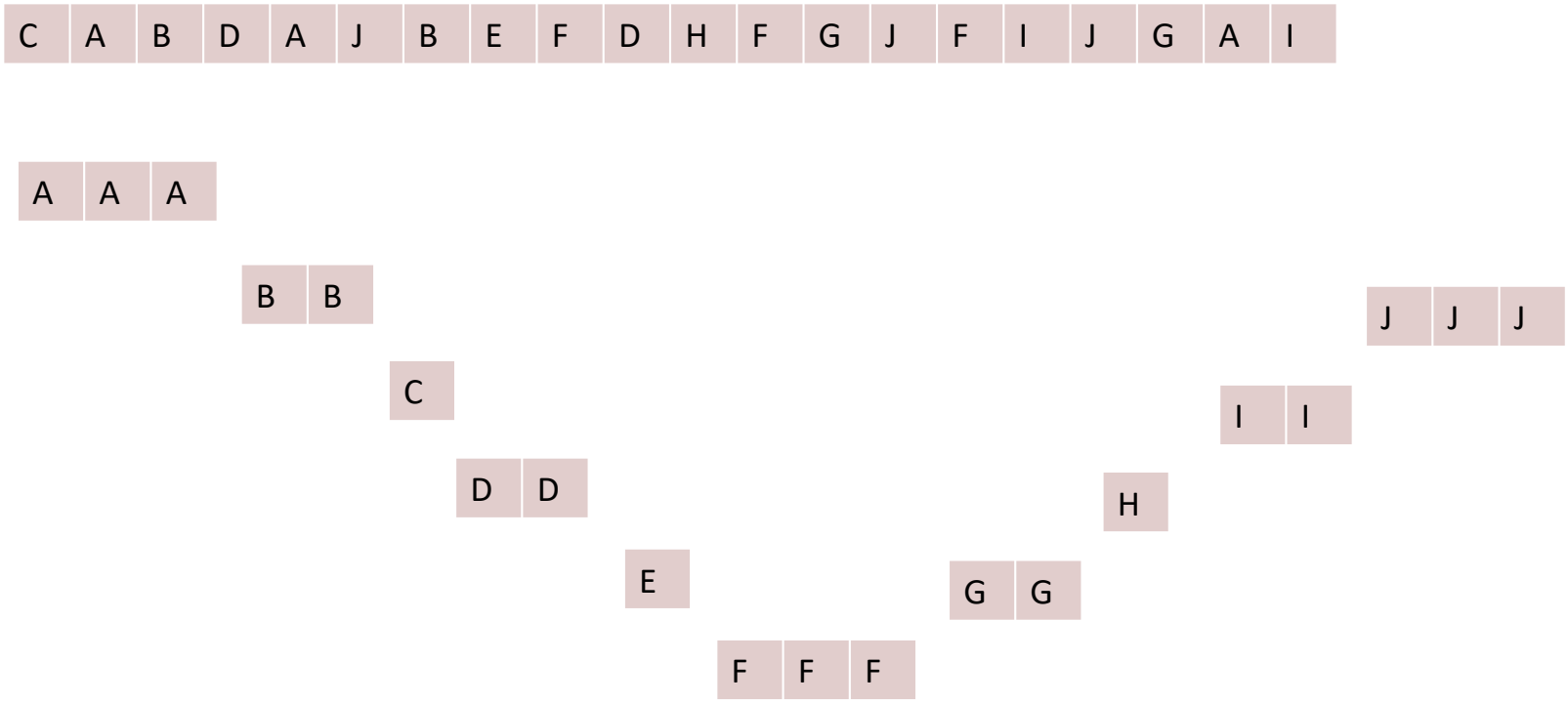


Feature-based blocking

Simple/Standard Blocking

- Pick a key: an attribute or feature of each entity
- Put all entities with that key in the same block
- Perform entity resolution within each block

Simple Blocking: Index on a feature



Simple Blocking: Inverted Index on a Predicate

Examples of blocking predicates (keys):

- First three characters of last name
- City + State + Zip
- Character or Token n-grams
- Minimum infrequent n-grams

Key choices for blocking methods

- Key selection: learn the keys, or use expert knowledge/heuristics?
- Schema awareness: what do we know about the attributes?
- Key type: exact equality, similarity-based, or hybrid?
- Redundancy: entity in one or multiple blocks? Does matching in multiple blocks increase the match probability?

Extensions to simple blocking

- Frequency limits
- Adaptive keys based on frequency
- Learning keys based on data (more later)

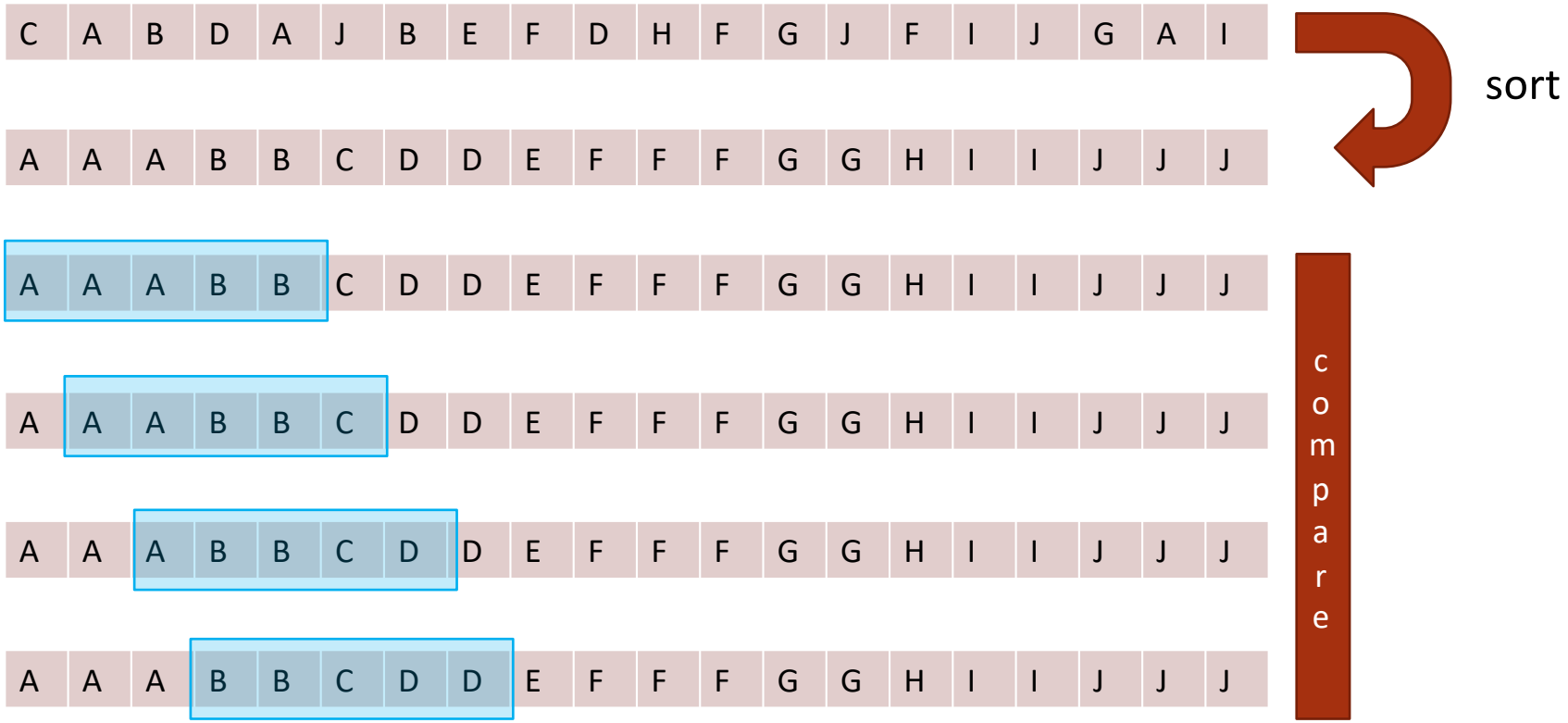
Sort or cluster-based blocking

Sorted Neighborhood Blocking

- Pick an attribute
- Sort all entities based on that attribute
- Use a sliding window of |K| entities and compare all pairs

Sorted Neighborhood Approach [Hernandez et al SIGMOD'95]

Sorted Neighborhood Blocking

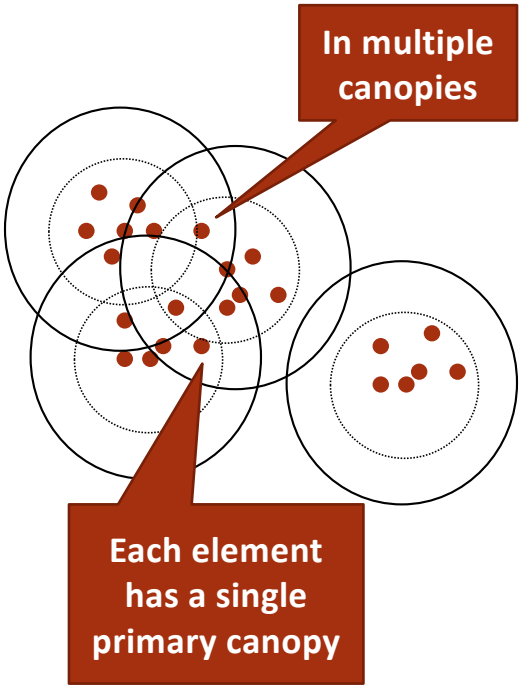


Canopy Clustering [McCallum et al KDD'00]

Input: Mentions *M*,
d(x,y), a distance metric,
thresholds $T_1 > T_2$

Algorithm:

1. Pick a random element *x* from *M*
2. Create new canopy C_x using mentions y s.t. $d(x,y) < T_1$
3. Delete all mentions *y* from *M* s.t. $d(x,y) < T_2$ *(from consideration in this algorithm)*
4. Return to Step 1 if *M* is not empty.



Hash-based blocking

Hash-based blocking

- Hash based blocking
  - Each block C_i is associated with a hash key h_i.
  - Mention x is hashed to C_i if $hash(x) = h_i$.
  - Within a block, all pairs are compared.
  - Each hash function results in disjoint blocks.

- What *hash* function?
  - Deterministic function of attribute values
  - Boolean Functions over attribute values [Bilenko et al ICDM'06, Michelson et al AAAI'06, Das Sarma et al CIKM '12]
  - **minHash** (min-wise independent permutations) [Broder et al STOC'98] + locality sensitive hashing

minHash (Minwise Independent Permutations)

- Let F_x be a set of features for mention *x*
  - (predicates of) attribute values
  - character ngrams
  - optimal blocking functions …
- Let π be a random permutation of features in F_x
  - E.g., order imposed by a random hash function
- *minHash(x)* = minimum element in F_x according to π

Why minHash?

Surprising property: For a random permutation π,

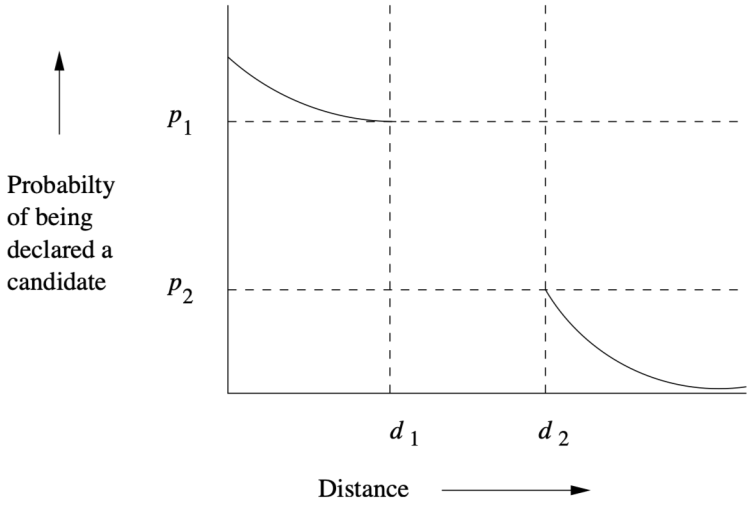
$$P[minHash(x) = minHash(y)] = \frac{F_x \cap F_y}{F_x \cup F_y}$$

Locality Sensitive Hashing Functions

Suppose *d* is a distance metric on a domain.

A family of functions **F** is said to be *(d1, d2, p1, p2)-sensitive* if for all f in **F**,

- If d(x,y) < d1,
  then P[f(x) = f(y)] > p1
- If d(x,y) > d2,
  then P[f(x) = f(y)] < p2



Locality sensitive family for Jaccard

- Jaccard distance = 1- Jaccard similarity = $1 - \frac{F_x \cap F_y}{F_x \cup F_y}$

- minHash is one example of locality sensitive family that can strongly distinguish pairs that are close from pairs that are far.

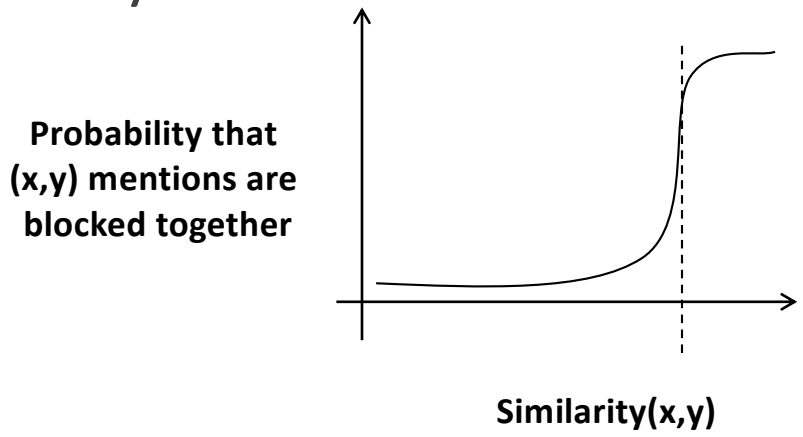
- The family of minHash functions is a (*d1*, *d2*, *1-d1*, *1-d2*)-sensitive family for the Jaccard distance.

Blocking based on minHash

Surprising property: For a random permutation π,

$$P[minHash(x) = minHash(y)] = \frac{F_x \cap F_y}{F_x \cup F_y}$$

How to build a blocking scheme such that only pairs with Jaccard similarity > s fall in the same block (with high prob)?



Probability Refresher

- What is the probability that I flip a fair coin three times and get a HEADS and another HEADS and another HEADS?

  ½ * ½ * ½ = 1/8

- What is the probability that I flip a fair coin three times and I get a HEADS or a HEADS or a HEADS?

  1 - (½ * ½ * ½) = 7/8

Amplifying a Locality-sensitive family

- AND construction:
  - Construct a new family F' consisting of r members of F
  - f in F' = $\{f_1, f_2, \ldots, f_r\}$
  - f(x) = f(y) iff for all i, $f_i(x) = f_i(y)$
  - If F is (d1, d2, p1, p2)-sensitive, then F' is (d1, d2, $p1^r$, $p2^r$)-sensitive

- OR construction:
  - Construct a new family F' consisting of b members of F
  - f in F' = $\{f_1, f_2, \ldots, f_b\}$
  - f(x) = f(y) iff there exists i, $f_i(x) = f_i(y)$
  - If F is (d1, d2, p1, p2)-sensitive,
    then F' is (d1, d2, $1-(1-p1)^b$, $1-(1-p2)^b$)-sensitive

minHash Analysis

- Let F be a (0.2, 0.6, 0.8, 0.4)-sensitive family of minHash functions
  - Pairs with Jaccard similarity > 0.8 are close, and similarity < 0.4 are far

- Let F1 be the family constructed using a "band of r=5 minHashes" **(AND construction on F)**
  - F1 is (0.2, 0.6, 0.8^5, 0.4^5) = (0.2, 0.6, 0.328, 0.01)-sensitive

'Far' objects are blocked together with very low probability

minHash Analysis

- F is (0.2, 0.6, 0.8, 0.4)-sensitive minHash
- F1 is a "band of r=5 minHashes" **(AND construction on F)**
  - F1 is (0.2, 0.6, 0.8^5, 0.4^5) = (0.2, 0.6, 0.328, 0.01)-sensitive

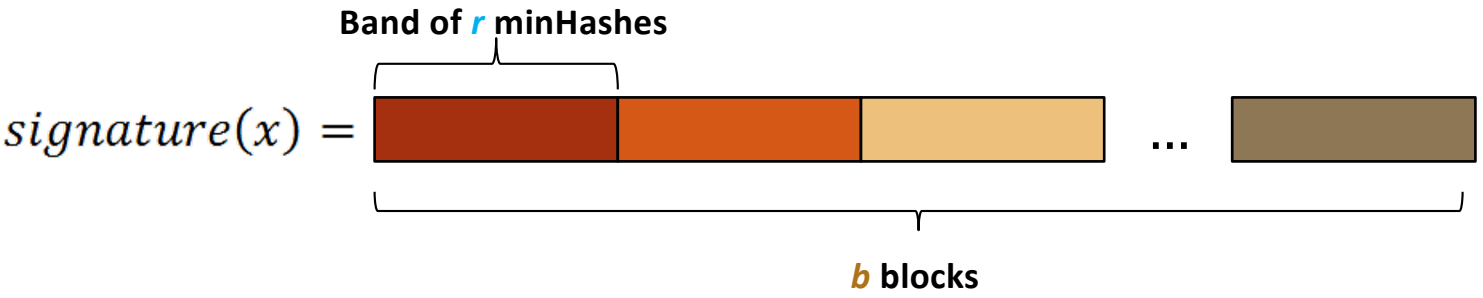
- Let F2 be the family constructed using "b = 10 bands of r=5 minHashes each" **(OR construction on F1)**
  - F2 is (0.2, 0.6, $1 - (1-0.8^5)^{10}$, $1 - (1-0.4^5)^{10}$)
    = (0.2, 0.6, 0.98 0.09)-sensitive

'Close' objects are blocked together with very high probability

'Far' objects are blocked together with very low probability

Blocking using minHashes

- Compute minHashes using *r* * *b* permutations (hash functions)



- Signatures that match on ***1 out of b*** bands, go to the same block.

minHash Analysis

- F is minHash family
  - (0.2, 0.6, 0.8, 0.4)-sensitive
- F1 is a "band of r=5 minHashes"
  - (0.2, 0.6, 0.328, 0.01)-sensitive
- F2 is "b=20 bands of r=5 minHashes each"
  - (0.2, 0.6, 0.98, 0.09)-sensitive

$r = 5, k = 20$

Sim(s)	P(same block)
0.9	0.9986
0.8	0.98
0.7	0.84
0.6	0.55
0.5	0.27
0.4	0.09
0.3	0.02
0.2	0.003
0.1	0.00009

minHash Analysis

$r = 5, k = 20$

False Negatives: (missing matches)

P(pair x,y not in the same block with Jaccard sim = s) $= (1 - s^r)^k$

should be very low for high similarity pairs

False Positives: (blocking non-matches)

P(pair x,y in the same block with Jaccard sim = s) $= k \times s^r$

Sim(s)	P(not same block)
0.9	10^{-8}
0.8	0.00035
0.7	0.025
0.6	0.2
0.5	0.52
0.4	0.81
0.3	0.95
0.2	0.994
0.1	0.9998

Summary of Hash-based Blocking

- Complex boolean functions can be built to optimize recall using a training set of matches and non-matches
- Locality sensitive hashing functions can strongly distinguish pairs that are close from pairs that are far.
- AND and OR construction help amplify the distinguishing capability of locality sensitive functions.
- Can design good LSH family for many distance metrics.

Learning to block

Learning Optimal Blocking Functions

- Using one or more blocking predicates may be insufficient
  - 2,376,206 American's shared the surname Smith in the 2000 US
  - NULL values may create large blocks.

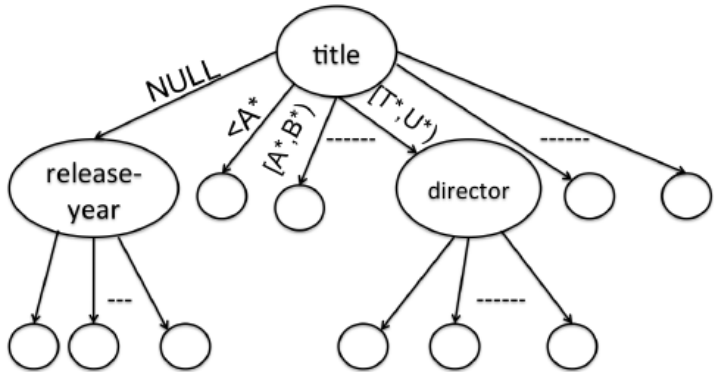
- Solution: Construct blocking predicates by combining simple predicates

Complex Blocking Predicates

- Conjunction of predicates [Michelson et al AAAI'06, Bilenko et al ICDM'06]
  - {City} AND {last four digits of phone}

- Chain-trees [Das Sarma et al CIKM'12]
  - **If** ({City} = NULL or LA) **then** {last four digits of phone} AND {area code}
    else {last four digits of phone} AND {City}

- BlkTrees [Das Sarma et al CIKM'12]

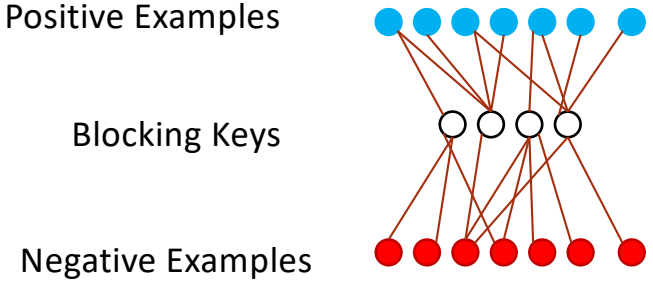


Learning an Optimal predicate

[Bilenko et al ICDM '06]

- Find k blocking predicates that eliminate the most non-matches, while retaining almost all matches.
  - Need a training set of positive and negative pairs

- Algorithm Idea: Red-Blue Set Cover

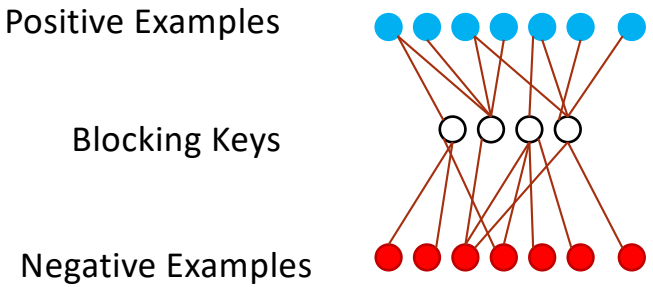


Pick k Blocking keys such that
(a) At most ε blue nodes are not covered
(b) Number of red nodes covered is minimized

Learning an Optimal function

[Bilenko et al ICDM '06]

- Algorithm Idea: Red-Blue Set Cover



Pick k Blocking keys such that
(a) At most ε blue nodes are not covered
(b) Number of red nodes covered is minimized

- Greedy Algorithm:
  - Construct "good" conjunctions of blocking predicates $\{p_1, p_2, \ldots\}$.
  - Pick k conjunctions $\{p_{i1}, p_{i2}, \ldots, p_{ik}\}$, such that the following is minimized

$$\frac{number\ of\ new\ blue\ nodes\ covered\ by\ p_{i_j}}{number\ of\ red\ nodes\ covered\ by\ p_{i_j}}$$

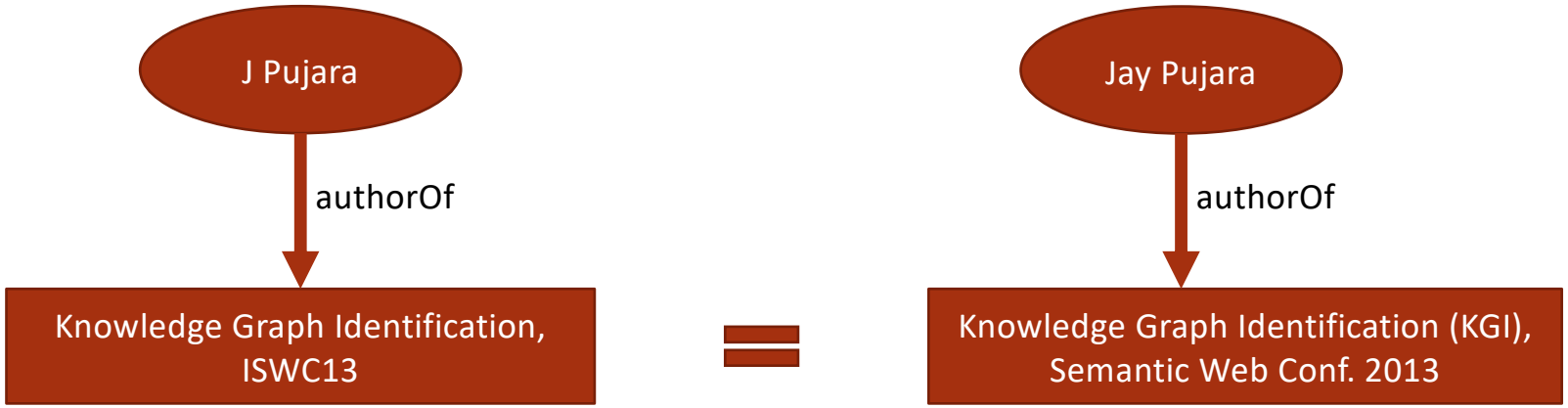
Collective, Relational Entity Resolution

MATERIALS FROM: GENERIC STATISTICAL RELATIONAL ENTITY RESOLUTION IN KNOWLEDGE GRAPHS (PUJARA & GETOOR, 2016) AVAILABLE AT HTTPS://ARXIV.ORG/ABS/1607.00992

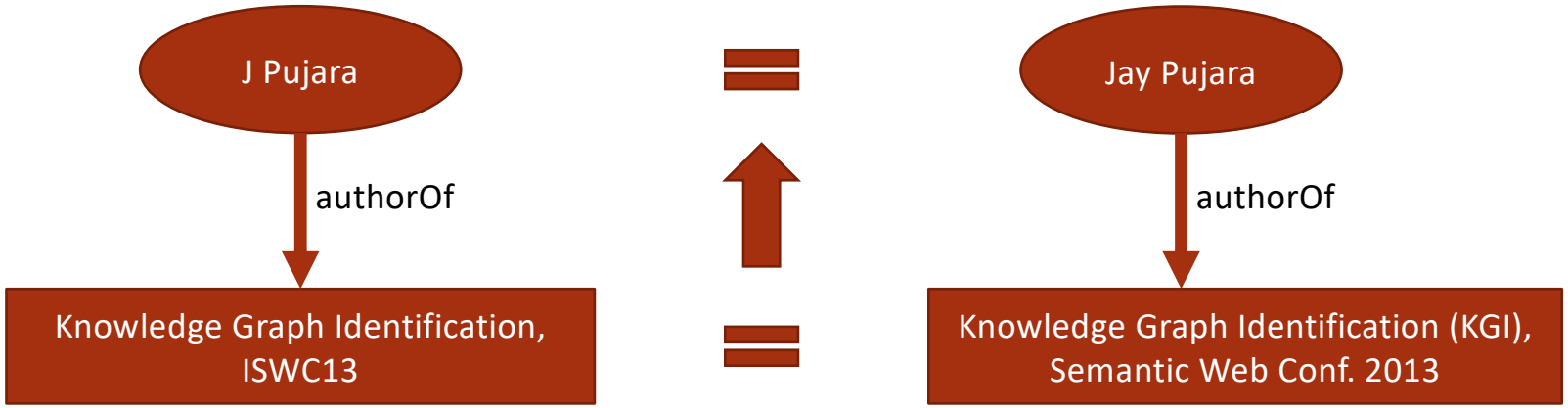
Common ER problem scenario

Common ER problem scenario

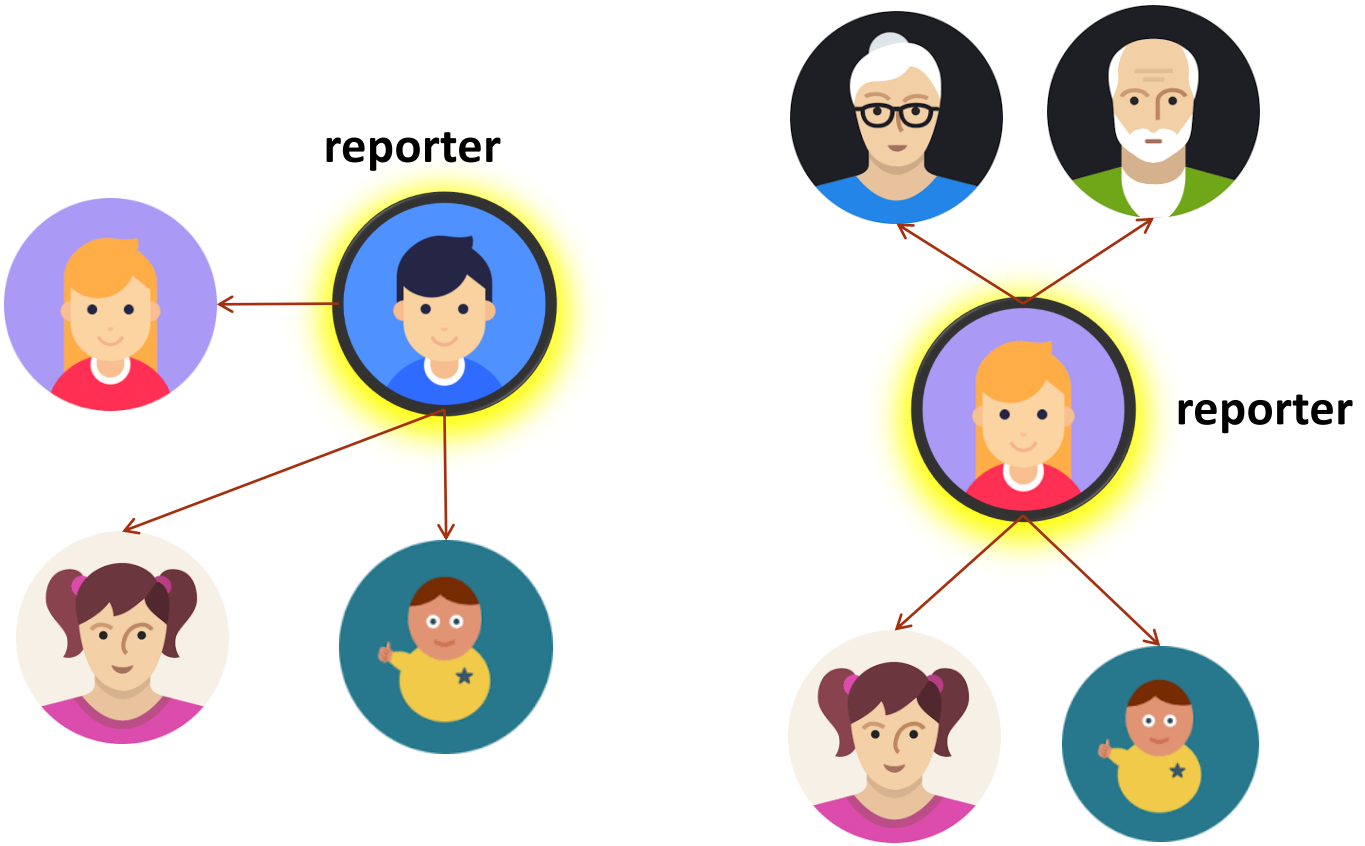
Common ER problem scenario



Common ER problem scenario



Example: Healthcare Records



Core problem: correlated ER decisions

- Most ML focused on independent instances
- KG entity resolution decisions are related
- Creates a dependency between matches
- Combine local ER with collective ER

Using PSL for collective KG ER

- **Goal:** encode ER dependencies in a set of rules
- Use soft-logic values to capture similarities
- Use logic to capture the constraints

Encoding relational entity resolution

	Local	Collective
Basic	String similarity	Transitivity, Sparsity, Functionality
New Entity	Prior	Sparsity penalty
Knowledge Graph	Type-based	Relation-based
Domain-specific	Restricted type-based	Restricted relation-based

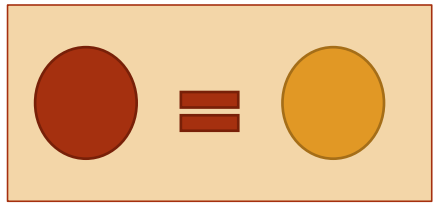
Basic rules

Nothing is the same without evidence

$$\neg \textsc{Same}(E_1, E_2) \qquad (1)$$

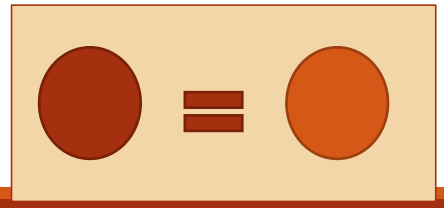
Candidate pairs (things blocked together) may be the same

$$\textsc{CandSame}(E_1, E_2) \Rightarrow \textsc{Same}(E_1, E_2) \qquad (2)$$



Similar candidate pairs (things blocked together) may be the same

$$\textsc{CandSame}(E_1, E_2) \wedge \textsc{Sim}(E_1, E_2) \Rightarrow \textsc{Same}(E_1, E_2) \qquad (3)$$



Core collective ER ideas

- **Symmetry:** If M1 matches with M2, then M2 must match with M1
- **Transitivity**: If M1 and M2 match, M2 and M3 match, then M1 and M3 match
- **Sparsity**: If M1 matches with M2, then M1 cannot match with M3

Core collective rules

Symmetry

$$\textsc{Same}(E_1, E_2) \Rightarrow \textsc{Same}(E_2, E_1) \quad (4)$$

Transitivity

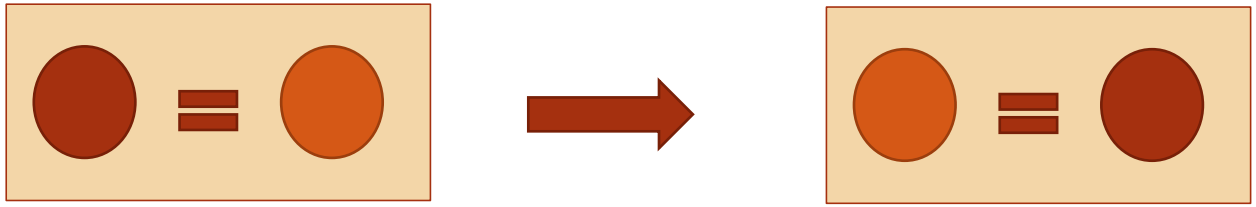
$$\textsc{CandSame}(A, B) \wedge \textsc{CandSame}(B, C) \wedge \textsc{CandSame}(A, C) \wedge \textsc{Same}(A, B) \wedge \textsc{Same}(B, C) \Rightarrow \textsc{Same}(A, C) \quad (5)$$

Sparsity

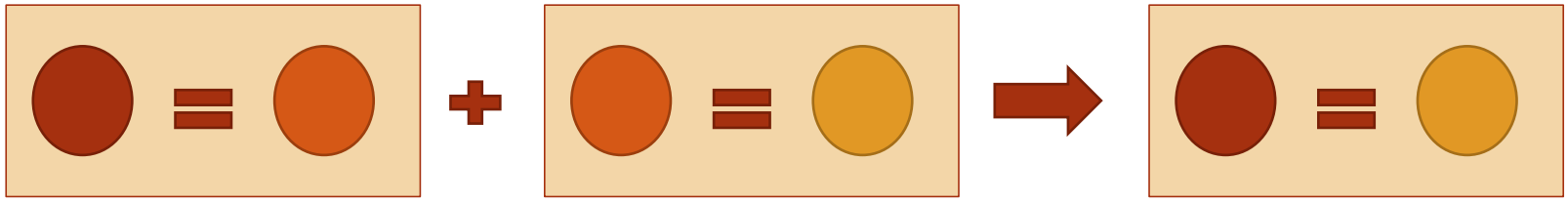
$$\textsc{CandSame}(A, B) \wedge \textsc{CandSame}(A, C) \wedge \textsc{Same}(A, B) \Rightarrow \neg\textsc{Same}(A, C) \quad (6)$$

Core collective rules

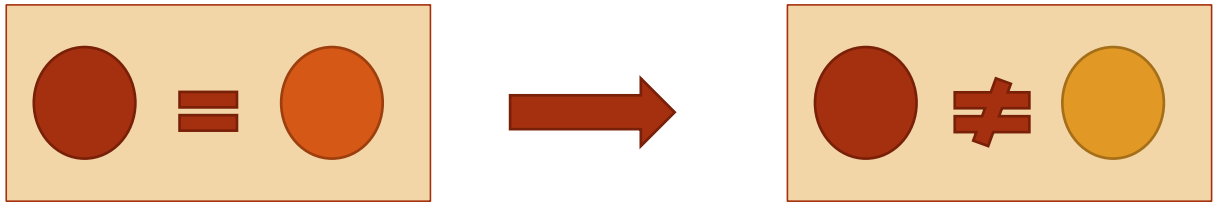
Symmetry



Transitivity



Sparsity



Dealing with new entities?

- What happens when an entity is missing from the target KG?

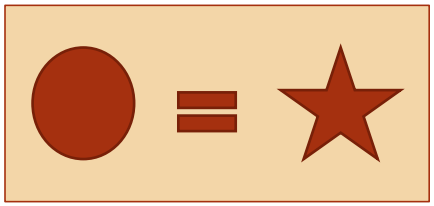
Dealing with new entities?

- What happens when an entity is missing from the target KG?
  - Skolemization: create a new constant for the entity
  - Avoid penalizing predictors that require attributes or relationships when linking to a new entity

New entity rules

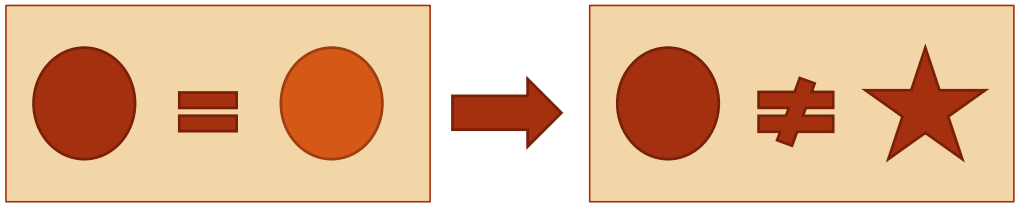
Linking to new entities

$$\textsc{CandSame}(E_1, E_2) \wedge \textsc{NewEntity}(E_1) \Rightarrow \textsc{Same}(E_1, E_2) \quad (7)$$



Avoiding new entities

$$\textsc{Same}(E_1, E_2) \wedge \textsc{CandSame}(E_1, E_3) \wedge \textsc{NewEntity}(E_3) \Rightarrow \neg\textsc{Same}(E_1, E_3) \quad (8)$$



Local KG-based ER rules

Entity labels/attributes should agree

$$\textsc{CandSame}(E_1, E_2) \wedge \textsc{Sim}(E_1, E_2) \wedge \textsc{Lbl}(E_1, L) \wedge \textsc{Lbl}(E_2, L) \Rightarrow \textsc{Same}(E_1, E_2) \quad (9)$$

Entity labels/attributes should not disagree

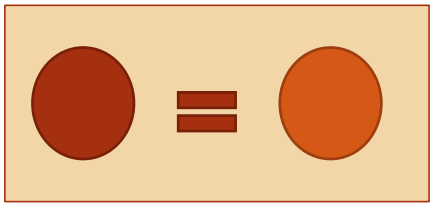
$$\textsc{CandSame}(E_1, E_2) \wedge \textsc{Lbl}(E_1, L) \wedge \neg\textsc{Lbl}(E_2, L) \wedge \neg\textsc{NewEntity}(E_2) \Rightarrow \neg\textsc{Same}(E_1, E_2) \quad (10)$$

Entity labels/attributes should be compatible

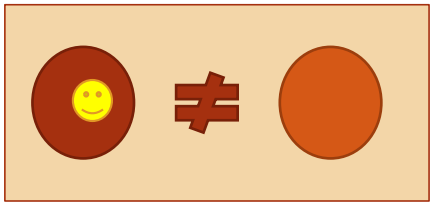
$$\textsc{CandSame}(E_1, E_2) \wedge \textsc{Lbl}(E_1, L_1) \wedge \textsc{Lbl}(E_2, L_2) \wedge \textsc{Mut}(L_1, L_2) \Rightarrow \neg\textsc{Same}(E_1, E_2) \quad (11)$$

Local KG-based ER rules

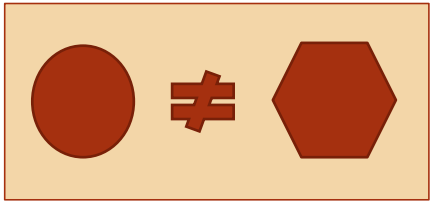
Entity labels/attributes should agree



Entity labels/attributes should not disagree



Entity labels/attributes should be compatible



Collective KG-based ER rules

Relational equivalence:
Matching entities should relate to objects in the same way

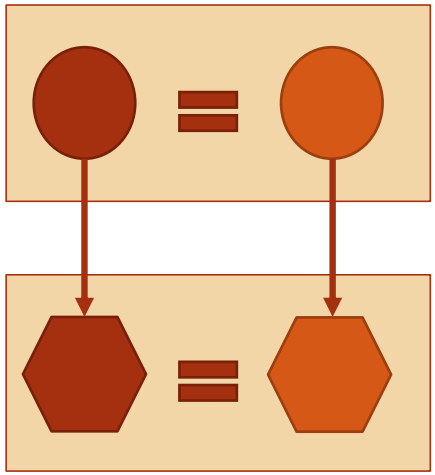
$$\begin{aligned} &\textsc{CandSame}(E_1, E_2) \wedge \textsc{CandSame}(O_1, O_2) \\ &\wedge \textsc{Sim}(E_1, E_2) \wedge \textsc{Same}(O_1, O_2) \\ &\wedge \textsc{Rel}(E_1, O_1, R) \wedge \textsc{Rel}(E_2, O_2, R) \\ &\Rightarrow \textsc{Same}(E_1, E_2) \end{aligned} \tag{12}$$

Relational parity:
If entities do not relate to objects in the same way they should not match

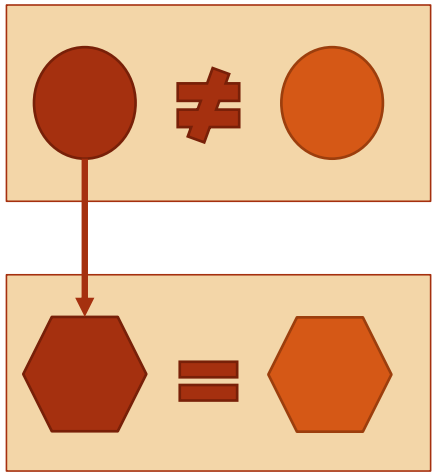
$$\begin{aligned} &\textsc{CandSame}(E_1, E_2) \wedge \textsc{CandSame}(O_1, O_2) \\ &\wedge \textsc{Same}(O_1, O_2) \wedge \neg\textsc{Rel}(E_1, O_1, R) \\ &\wedge \neg\textsc{NewEntity}(E_1) \wedge \neg\textsc{NewEntity}(O_1) \\ &\wedge \textsc{Rel}(E_2, O_2, R) \\ &\Rightarrow \neg\textsc{Same}(E_1, E_2) \end{aligned} \tag{13}$$

Collective KG-based ER rules

Relational equivalence:
Matching entities
should relate to objects
in the same way



Relational parity:
If entities do not relate
to objects in the same
way they should not
match



Local domain-specific rules

$$\begin{aligned}\textsc{CandSame}(E_1, E_2) \;&\wedge\; \textsc{Sim}(E_1, E_2) \\ &\wedge\; \textsc{Rel}(E_1, L, \texttt{release\_type}) \\ &\wedge\; \textsc{Rel}(E_2, L, \texttt{release\_type}) \\ &\Rightarrow\; \textsc{Same}(E_1, E_2) \qquad (15)\end{aligned}$$

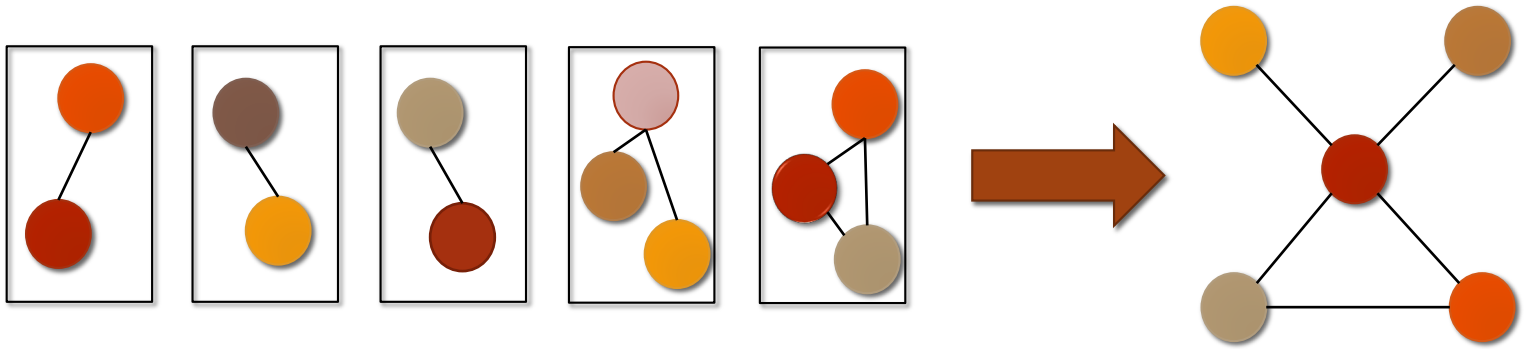
$$\begin{aligned}\textsc{CandSame}(E_1, E_2) \;&\wedge\; \textsc{Sim}(E_1, E_2) \\ &\wedge\; \textsc{Lbl}(E_1, \texttt{artist}) \\ &\wedge\; \textsc{Lbl}(E_2, \texttt{artist}) \\ &\Rightarrow\; \textsc{Same}(E_1, E_2) \qquad (16)\end{aligned}$$

Collective domain-specific rules

$$\begin{aligned} &\textsc{CandSame}(E_1, E_2) \wedge \textsc{Sim}(E_1, E_2) \\ \wedge\ &\textsc{CandSame}(O_1, O_2) \wedge \textsc{Same}(E_2, E_1) \\ &\wedge \textsc{Rel}(E_1, O_1, \texttt{release\_album}) \\ &\wedge \textsc{Rel}(E_2, O_2, \texttt{release\_album}) \\ &\Rightarrow \textsc{Same}(O_1, O_2) \end{aligned} \tag{17}$$

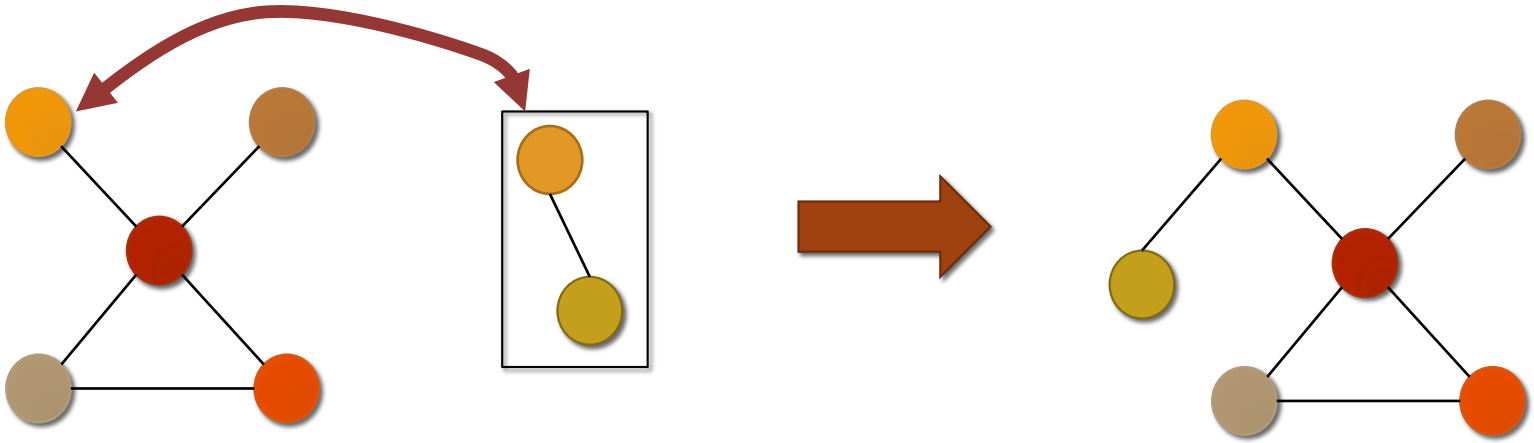
$$\begin{aligned} &\textsc{CandSame}(E_1, E_2) \wedge \textsc{CandSame}(O_1, O_2) \\ \wedge\ &\textsc{Sim}(E_1, E_2) \wedge \textsc{Sim}(O_1, O_2) \\ &\wedge \textsc{Rel}(E_1, O_1, \texttt{album\_artist}) \\ &\wedge \textsc{Rel}(E_2, O_2, \texttt{album\_artist}) \\ &\wedge \textsc{Rel}(E_1, G, \texttt{album\_genre}) \\ &\wedge \textsc{Rel}(E_2, G, \texttt{album\_genre}) \\ &\wedge \textsc{Rel}(O_1, G, \texttt{artist\_genre}) \\ &\wedge \textsc{Rel}(O_2, G, \texttt{artist\_genre}) \\ &\wedge \textsc{Same}(O_1, O_2) \\ &\Rightarrow \textsc{Same}(E_1, E_2) \end{aligned} \tag{18}$$

Deduplication



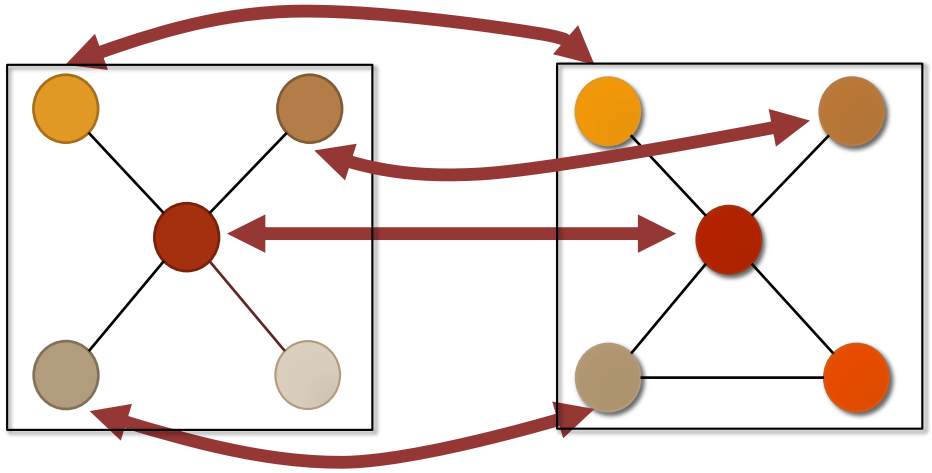
Integrating extractions

Entity Mapping/Linking



Extending a KG with new information

Record Linkage



Merging KGs

	Local/ collective	New extractions	Extend KG	Multiple KGs
Negative prior (1)	L	Y	Y	Y
Positive prior (2)	L	Y	Y	Y
Similarities (3)	L	Y	Y	Y
Symmetry (4)	C	Y	Y	Y
Transitivity (5)	C	Y	N	Y
Sparsity (6)	C	N	N	Y
New Entity prior (7)	L	N	Y	Y
New Entity penalty (8)	C	N	Y	Y
Label agreement (9)	L	N?	Y	Y
Label disagreement (10)	L	N?	Y?	Y?
Label exclusion (11)	L	Y	Y	Y
Relational agreement (12)	C	N?	Y	Y
Relational disagreement (13)	C	N?	Y?	Y?
Relational exclusion (14)	C	Y	Y	Y
Domain-specific categorical relations (15)	L	Y	Y	Y
Domain-specific prior (16)	L	Y	Y	Y
Domain-specific relations (17)	C	Y?	Y	Y
Domain-specific relational criteria (18)	C	Y?	Y	Y

Evaluation results

Method	AUPRC	F1	Prec.	Recall	Inf. Time (s)
Basic, Local	0.267	0.333	0.214	0.759	5
Basic & KG, Local	0.247	0.426	0.298	0.747	220
Basic, All	0.307	0.446	0.333	0.675	8
Basic & KG, All	**0.351**	**0.453**	0.383	0.554	4000

Table 3: Comparing the performance of knowledge graph entity resolution rules in for the NELL dataset. Performance improves by adding knowledge graph features and collective features, with the best performance with both.

Method	AUPRC	F1	F1 (Exist)	F1 (New)
Basic & NewEntity, Local	0.416	0.734	0.169	0.744
Basic & Domain, All; NewEntity, Local	0.569	0.805	**0.305**	0.831
Basic & Domain & NewEntity, All	**0.724**	**0.840**	0.070	**0.895**

Table 4: Comparing the performance of knowledge graph entity resolution rules when merging MusicBrainz entities into the Knowledge Graph. Due to a skew toward new entities, the collective new entity rules dramatically improve overall performance, but with a substantial drop in the F1 measure for existing entities

Key takeaways

- Dependencies are important in KG ER
- Different KG construction settings require respecting different dependency types
- PSL is a flexible way of using these dependencies