The five generations of Entity Resolution

George Papadakis
gpapadis@di.uoa.gr

OM 2023, Athens
November 7, 2023
Structure Outline

• Introduction
• The First Four Generations
• The Fifth Generation: Leveraging External Knowledge
• Challenges and Final Remarks
Part A – Introduction

- Motivation
- Preliminaries

• The First Four Generations
• The Fifth Generation: Leveraging External Knowledge
• Challenges and Final Remarks
Motivation

- Entities: an invaluable asset for numerous current applications and systems
- Encode a large part of our knowledge
Many names, descriptions, or IDs (URIs) are used for the same real-world “entity”

Example:
Matching, Linkage, Reconciliation, etc.

- Many names, descriptions, or IDs (URIs) are used for the same real-world “entity”
- Example:
Matching, Linkage, Reconciliation, etc.

- Many names, descriptions, or IDs (URIs) are used for the same real-world “entity”

Example:

Matching, Linkage, Reconciliation, etc.

- Many names, descriptions, or IDs (URIs) are used for the same real-world “entity”

Example:

London 런던 লন্ডন लंडन ロンドン

capital of UK, host city of the IV Olympic Games, host city of the XIV Olympic Games, future host of the XXX Olympic Games, city of the Westminster Abbey, city of the London Eye, the city described by Charles Dickens in his novels, ...

http://sws.geonames.org/2643743/

...
Disambiguation, Deduplication, etc.

- Plethora of different “entities” have the same name
- Example:

  - London, KY
  - London, Laurel, KY
  - London, OH
  - London, Madison, OH
  - London, AR
  - London, Pope, AR
  - London, TX
  - London, Kimble, TX
  - London, MO
  - London, London, MI
  - London, London, Monroe, MI
  - London, Uninc Conecuh County, AL
  - London, Uninc Conecuh County, Conecuh, AL
  - London, Uninc Shelby County, IN
  - London, Uninc Shelby County, Shelby, IN
  - London, Deerfield, WI
  - London, Deerfield, Dane, WI
  - London, Uninc Freeborn County, MN
  - ...

Disambiguation, Deduplication, etc.

- Plethora of different “entities” have the same name
- Example:
  - London, KY
  - London, Laurel, KY
  - London, OH
  - London, Madison, OH
  - London, AR
  - London, Pope, AR
  - London, TX
  - London, Kimble, TX
  - London, MO
  - London, Lena
  - London, London, MI
  - London, London, Monroe, MI
  - London, Uninc Conecuh County, AL
  - London, Uninc Conecuh County, Conecuh, AL
  - London, Uninc Shelby County, IN
  - London, Uninc Shelby County, Shelby, IN
  - London, Deerfield, WI
  - London, Deerfield, Dane, WI
  - London, Uninc Freeborn County, MN
  - ...
Entities in today’s settings

• Content providers provide valuable information describing (part of) real-world “entities”

• ER is required for data integration, link discovery, query answering, Web / object-oriented searching, etc.

News about London
REUTERS

Wiki pages about the London

Social networks in London

Reviews on hotels in London

Pictures and tags about London

Videos and tags for London
Entity Resolution

• Identifies and aggregates the different entity profiles that describe the same objects [1,2,3,4]
• Primary usefulness:
  – Improves data quality and integrity
  – Fosters re-use of existing data sources
• Example application domains:
  – Linked Data
  – Building Knowledge Graphs
  – Census data
  – Price comparison portals
Types of Entity Resolution

• The given entity collections can be of two types: \textit{clean} + \textit{dirty} [3,5]

• Clean:
  – Duplicate-free data
  – E.g., DBLP, ACM Digital Library, Wikipedia, Freebase

• Dirty:
  – Contain duplicate entity profiles
  – E.g., Google Scholar, CiteseerX
Types of Entity Resolution

• Based on the quality of input, we distinguish entity resolution into 3 sub-tasks:

1. Clean-Clean ER (a.k.a. Record Linkage in databases)
2. Dirty-Clean ER  
   Equivalent to Dirty ER
3. Dirty-Dirty ER  
   (a.k.a. Deduplication in databases)
References


• Introduction

Part B – Four Generations

– Generation 1: tackling Veracity
– Generation 2: tackling Volume and Veracity
– Generation 3: tackling Variety, Volume and Veracity
– Generation 4: tackling Velocity, Variety, Volume and Veracity

• The Fifth Generation:
  Leveraging External Knowledge

• Challenges and Final Remarks
Generation 1: Tackling Veracity

- Earliest approach
- Scope:
  - Structured data
- Goal:
  - Achieve high accuracy despite inconsistencies, noise, or errors in entity profiles
- Assumptions:
  - Known schema → custom, schema-based solutions
Step 1: Schema Alignment / Matching

• Scope:
  – Record Linkage

• Goal:
  – Create mappings between equivalent attributes of the two schemata, e.g., profession \equiv job

• Types of Solutions:
  – Structure-based
  – Instance-based
  – Hybrid
Step 1: Schema Alignment / Matching

- Taxonomy of Main Schema Matching Methods (in chronological order)

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>Type of Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupid [1]</td>
<td>Structure-based</td>
<td>Name similarity, Constraints, Contextual similarity</td>
</tr>
<tr>
<td>Similarity Flooding [2]</td>
<td>Structure-based</td>
<td>Name similarity, Contextual similarity</td>
</tr>
<tr>
<td>COMA [3]</td>
<td>Hybrid</td>
<td>Name similarity, Constraints, Contextual similarity</td>
</tr>
<tr>
<td>Distribution-based [4]</td>
<td>Instance-based</td>
<td>Value distribution</td>
</tr>
</tbody>
</table>
Step 2: Blocking

• Scope:
  – Both Deduplication and Record Linkage

• Goal:
  – ER is an inherently quadratic problem, $O(n^2)$: every entity has to be compared with all others
  – Blocking groups similar entities into blocks
    • Comparisons are executed only inside each block
    • Complexity is now quadratic to the size of the block (much smaller than dataset size!)
Input: Entity Collection \( E \)

E.g.: For a dataset with 100,000 entities: 
\(~10^{10}\) comparisons, 
If 0.05 msec each \( \rightarrow \rangle 100 \) hours in total
General Principles of Blocking

1. Represent each entity by *one or more* signatures called **blocking keys**
   – Focus on **string values**

2. Place into blocks all entities having the **same or similar** blocking key

3. Two matching profiles can be **detected** as long as they co-occur in at least one block
   – **Trade-off** between recall and precision!
## Taxonomy of Blocking Methods [1]

<table>
<thead>
<tr>
<th>Method</th>
<th>Key Type</th>
<th>Redundancy awareness</th>
<th>Matching awareness</th>
<th>Key selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suffix Arrays [3] + [4,5]</td>
<td>Hash-based</td>
<td>Red.-positive</td>
<td>Static</td>
<td>Non-learning</td>
</tr>
<tr>
<td>Sorted Neighborhood [9] + [4,10]</td>
<td>Sort-based</td>
<td>Red.-neutral</td>
<td>Static</td>
<td>Non-learning</td>
</tr>
<tr>
<td>Sorted Blocks [12]</td>
<td>Hybrid</td>
<td>Red.-neutral</td>
<td>Static</td>
<td>Non-learning</td>
</tr>
<tr>
<td>Blocking Scheme Learner [14]</td>
<td>Hash-based</td>
<td>Red.-positive</td>
<td>Static</td>
<td>Learning-based</td>
</tr>
<tr>
<td>FisherDisjunctive [16]</td>
<td>Hash-based</td>
<td>Red.-positive</td>
<td>Static</td>
<td>Learning-based</td>
</tr>
</tbody>
</table>

**Generation 1: Tackling Veracity**
Genealogy Tree of Non-learning Blocking Methods [1]

Generation 1: Tackling Veracity

- **Standard Blocking (SB)** [2]
  - **Suffix Arrays Blocking (SA)** [3]
    - **Extended SA** [4]
    - **Improved SA** [5]
  - **Q-grams Blocking** [6]
    - **Extended Q-grams Blocking** [4]
    - **MFIBlocks** [7]
  - **Sorted Neighborhood (SN)** [9]
    - **Extended SN** [4]
    - **Duplicate Count Strategy (DCS)** [11]
    - **DCS++** [11]
    - **Accumulative Adaptive SN** [10]
    - **Incrementally Adaptive SN** [10]
  - **Sorted Blocks** [12]
    - **Sorted Blocks New Partition** [12]
Step 3: Matching

- Estimates the similarity of candidate matches.

- Input
  - A set of blocks
    - Every distinct comparison in any block is a candidate match

- Output
  - Similarity Graph
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)
Evolution of Matching

Active Learning Methods [1,2]

Supervised Methods [3,4]

Probabilistic Methods [5,6]  Unsupervised Methods [7,8]

Learning-based Methods

Rule-based Methods [6]

Collective Methods [9,10,11]

All are heavily based on string similarity measures [6].
Step 4: Clustering

- Partitions the matched pairs into **equivalence clusters**
  i.e., groups of entity profiles describing the same real-world object

- **Input**
  - Similarity Graph:
    - Nodes → entities
    - Edges → candidate matches
    - Edge weights → matching likelihood (based on similarity score)

- **Output**
  - Equivalence Clusters
Clustering Algorithms

• For Clean-Clean ER [1][2][4]:
  • They rely on the 1-1 constraint:
    • every entity from the source dataset matches with at most one entity from the target dataset

• For Dirty ER [3]:
  • “Unconstrained algorithms”: able to predict the correct number of clusters
  • Goal: Sets of clusters that
    • maximize the intra-cluster weights
    • minimize the inter-cluster edge weights

• For both tasks:
  • Need to scale well
    — Time complexity $< O(n^2)$
  • Need to be robust with respect to characteristics of the data
    — E.g., distribution of the duplicates
Schema Matching References


Matching References

1. K. Qian, L. Popa, P. Sen. Active Learning for Large-Scale Entity Resolution. CIKM 2017: 1379-1388
Clustering References


Generation 2: Tackling **Volume** and Veracity

- Same workflow as Generation 1
- Scope:
  - (tens of) millions of structured entity profiles
- Goals:
  - High accuracy despite noise
  - High time efficiency despite the size of data
- Assumptions:
  - Known schema → custom, schema-based solutions
Solution: Parallelization

Two types:

• **Multi-core parallelization**
  – Single system → shared memory
  – Distribute processing among available CPUs

• **Massive parallelization**
  – Cluster of independent systems
  – **Map-Reduce** paradigm [1]
    • Data partitioned across the nodes of a cluster
    • Fault-tolerant, optimized execution
    • **Map Phase**: transforms a data partition into (key, value) pairs
    • **Reduce Phase**: processes pairs with the same key
Parallelization Methods per Step

• Blocking
  • Dedoop [2]
  • MapReduce-based Sorted Neighborhood [3]

• Matching
  – Multi-core approaches [7][8]
  – MapReduce-based: Emphasis on load balancing
    • BlockSplit & PairRange [4][5]
    • Dis-Dedup [6]
    • Message-passing framework [9]

• Clustering
  • Fast Multi-source Entity Resolution (FAMER) framework [10][11]
Generation 2 References

G3: Tackling Variety, Volume and Veracity

• Scope:
  – User-generated Web Data
    Voluminous, (semi-)structured datasets.
    • BTC09: 1.15 billion triples, 182 million entities.

Users are free to add attribute values and/or attribute names
→ unprecedented levels of schema heterogeneity.
• Google Base: 100,000 schemata for 10,000 entity types
• BTC09: 136,000 predicates

Several datasets produced by automatic information extraction techniques → noise, tag-style values.
### Example of Web Data

#### DATASET 1

**Entity 1**
- **name**: United Nations Children’s Fund
- **acronym**: unicef
- **headquarters**: California
- **address**: Los Angeles, 91335

#### DATASET 2

**Entity 3**
- **organization**: unicef
- **California**
- **status**: active
- **Los Angeles, 91335**

**Entity 4**
- **firstName**: Ann
- **lastName**: Veneman
- **residence**: California
- **zip_code**: 90210

---

**Loose Schema Binding**
- Split values

**Attribute Heterogeneity**
- Noise

---

Generation 3: Tackling Variety, Volume & Veracity
Ontology Matching

- Schema Matching → not scalable
  → not effective (more complex tasks)

- For details, see the series of the “International Workshop on Ontology Matching”
Unlike Blocking in G1/G2, it considers all attribute values and completely ignores all attribute names → **schema-agnostic functionality**

**Core approach:** **Token Blocking** [1]

1. Given an entity profile, extract all tokens that are contained in its attribute values.
2. Create one block for every distinct token with frequency > 2 → each block contains all entities with the corresponding token.

**Pros:**
- Parameter-free
- Efficient
- Unsupervised
Example of Token Blocking

**DATASET 1**

- **Entity 1**
  - name=United Nations Children’s Fund
  - acronym=unicef
  - headquarters=California

- **Entity 2**
  - name=Ann Veneman
  - position=unicef
  - address=California

**DATASET 2**

- **Entity 3**
  - organization=unicef
  - hdq=California
  - status=active

- **Entity 4**
  - firstName=Ann
  - lastName=Veneman
  - residence=California

Generation 3: Tackling Variety, Volume & Veracity
Genealogy of Block Building Techniques [8]

- Token Blocking (TB) [1]
  - Attribute Clustering Blocking [2]
  - RDFKeyLearner [6]
  - Prefix-Infix(-Suffix) Blocking [3]
- TYPiMatch [4]

Semantic Graph Blocking [5]

MapReduce-based parallelizations in [7]
Block Processing

- High **Recall** due to redundancy
- Low **Precision** due to:
  1. the blocks are overlapping $\rightarrow$ **redundant comparisons**
  2. high number of comparisons between irrelevant entities $\rightarrow$ **superfluous comparisons**

**Solution:**

restructure the original blocks so as to increase precision at no significant cost in recall
Block Processing Techniques

Generic approach

– Assign a matching likelihood score to each item
– Discard items with low costs

Block-centric methods

• Block Purging [1,2,3]
• Block Filtering [4]
• Block Clustering [5]
Comparison Cleaning Methods [17]

Spectral Neighborhood (SPAN) [15]

Comparison Propagation [6]

Transitive LSH [16]

Comparison Pruning [7]

Weighted Edge Pruning (WEP) [8]

Comparison Scheduling [14]

Cardinality Edge Pruning (CEP) [8]

Weighted Node Pruning (WNP) [4,8]

Low Entity Co-occurrence Pruning (LECP) [13]

Extended Canopy Clustering [9,10]

Cardinality Node Pruning (CNP) [4,8]

Canopy Clustering [12]

Low Block Co-occurrence Pruning (LBCP) [13]

Reciprocal Cardinality Node Pruning (ReCNP) [4]

Reciprocal Weighted Node Pruning (ReWNP) [4]

BLAST [11]

CooSlicer [13]

Large Block Size Pruning (LBSP) [13]

Low Block Co-occurrence Excluder (LBCE) [13]

Generation 3: Tackling Variety, Volume & Veracity
Collective approaches to tackle Variety
Most methods are crafted for Clean-Clean ER
General outline of SiGMa [1], PARIS [2], LINDA [3], RiMOM-IM [4,5]
  - Bootstrap with a few reliable seed matches.
  - Using value and neighbor similarity, propagate initial matches to neighbors.
  - Order candidate matches in descending overall similarity
  - Iteratively mark the top pair as a match if it satisfies a constraint
  - Recompute the similarity of the neighbors
  - Update candidate matches order
MinoanER [6] performs a specific number of steps, rather than iterating until convergence
• Methods of G1 & G2 are still applicable
  – Only difference: similarity scores extracted in a schema-agnostic fashion, not from specific predicates
4. Y. Ma, T. Tran. TYPiMatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. WSDM 2013: 325-334


Entity Matching References


G4: Tackling **Velocity**, Variety, Volume and Veracity

**Scope:**
- Applications with increasing data volume and time constraints
  - Loose ones (e.g., minutes, hours) \(\rightarrow\) Progressive ER
  - Strict ones (i.e., seconds) \(\rightarrow\) Real-time (On-line) ER

**End-to-end workflows for Progressive ER**
Progressive Entity Resolution

Unprecedented, increasing volume of data → applications requiring partial solutions to produce useful results

may require some pre-processing

generate most of the benefit much earlier
Outline Progressive ER

• Requires:
  – Improved Early Quality
  – Same Eventual Quality

• Prioritization
  – Defines **optimal processing order** for a set of entities
  – Static Methods [1,2]:
    • Guide which records to compare first, *independently* of Entity Matching results
  – Dynamic Methods [3]:
    • If \( c_{i,j} \) is a duplicate, then check \( c_{i+1,j} \) and \( c_{i,j+1} \) as well.
    • Assumption:
      – Oracle for Entity Matching
Taxonomy of Static Prioritization Methods

- Sorted Neighborhood Blocking (SN)
- Standard Blocking
- Token Blocking
- Meta-blocking (Blocking Graph)

Schema-based
- Progressive SN (PSN) [1]
- Hierarchy of Record Partitions (HRP) [1,4]
- Ordered List of Records (OLR) [1]

Schema-agnostic
- Progressive Suffix Arrays Blocking (SA-PSAB) [2]
- Progressive Block Scheduling (PBS) [2]
- Progressive Profile Scheduling (PPS) [2]

Naive
- Local SA-PSN (LS-PSN) [2]

Advanced
- Global SA-PSN (GS-PSN) [2]
- Progressive Block Scheduling (PBS) [2]

Comparison-based
- Block-based
- Profile-based
- Hybrid

- Meta-blocking (Blocking Graph)
- Progressive Suffix Arrays Blocking (SA-PSAB) [2]
- Progressive Block Scheduling (PBS) [2]
- Progressive Profile Scheduling (PPS) [2]
Real-time Entity Resolution

**Same** workflow as Generations 1 and 2:

![Diagram showing workflow: Schema Alignment, Blocking, Matching, Clustering]

**Same** scope (so far):
- Structured data

**Different** input:
- Stream of query entity profiles

**Different** goal:
- Resolve each query over a large dataset in the shortest possible time (& with the minimum memory footprint)
Techniques per workflow step

Incremental Blocking
- **DySimII** [1] - extends Standard Blocking
- **F-DySNI** [2,3] - extends Sorted Neighborhood
- **(S)BlockSketch** [4] - bounded matching time, constant memory footprint

Incremental Matching
- **QDA** [5] - SQL-like selection queries over a single dataset
- **QuERy** [6] - complex join queries over multiple, overlapping, dirty DSs
- **EAQP** [7] - queries under data
- Evolving matching rules [8]

Incremental Clustering
- Incremental Correlation Clustering [9]
Progressive ER References

Incremental ER References


• Introduction
• The First Four Generations

Part C – The Fifth Generation: Leveraging External Knowledge

– Types of External Knowledge
– Blocking
– Matching
– Clustering

• Challenges and Final Remarks
Crowd-sourcing for Entity Resolution

• Key Idea: tasks complex for computers, but simple for human intelligence are divided among many people, called workers (e.g., from Amazon Mechanical Turk)

• Adaptation to ER: Delegate the entity matching decisions to the workers, i.e., transform pairwise comparisons into Human Intelligence Tasks (HITS)

• Challenges:
  1. Generating HITs
  2. Formulating HITs
  3. Balancing accuracy and monetary cost
  4. Restricting the labor cost
Language Models

• Based on the **distributional hypothesis**
  i.e., *words appearing in the same context share meaning*

• Each word is represented as a distribution of weights (positive or negative) across specific dimensions

• Goal: capture **semantic** string similarities based on the **contextual information** from huge textual corpora

• Note: it applies to both **blocking** and **matching**, unlike crowd-sourcing
Performance of Language Models

- Experimental analyses for:
  1. Blocking
  2. Matching

using the 10 established real-world datasets from various domains.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>D_1</th>
<th>D_2</th>
<th>D_3</th>
<th>D_4</th>
<th>D_5</th>
<th>D_6</th>
<th>D_7</th>
<th>D_8</th>
<th>D_9</th>
<th>D_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rest.1</td>
<td>339</td>
<td>1,076</td>
<td>1,354</td>
<td>2,616</td>
<td>5,118</td>
<td>5,118</td>
<td>6,056</td>
<td>2,554</td>
<td>2,516</td>
<td>27,615</td>
</tr>
<tr>
<td>Rest.2</td>
<td>2,256</td>
<td>1,076</td>
<td>3,039</td>
<td>2,294</td>
<td>6,056</td>
<td>7,810</td>
<td>7,810</td>
<td>22,074</td>
<td>61,353</td>
<td>23,182</td>
</tr>
<tr>
<td>NVP_1</td>
<td>1,130</td>
<td>2,568</td>
<td>5,302</td>
<td>10,464</td>
<td>21,294</td>
<td>21,294</td>
<td>23,761</td>
<td>14,143</td>
<td>10,064</td>
<td>1.6-10^5</td>
</tr>
<tr>
<td>NVP_2</td>
<td>7,519</td>
<td>2,308</td>
<td>9,110</td>
<td>9,162</td>
<td>23,761</td>
<td>20,902</td>
<td>20,902</td>
<td>1.14-10^5</td>
<td>1.98-10^5</td>
<td>8.2-10^5</td>
</tr>
<tr>
<td></td>
<td>A_1</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>13</td>
<td>13</td>
<td>30</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>A_2</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>30</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>p_1</td>
<td>3.33</td>
<td>2.39</td>
<td>3.92</td>
<td>4.00</td>
<td>4.16</td>
<td>4.16</td>
<td>3.92</td>
<td>5.54</td>
<td>4.00</td>
</tr>
<tr>
<td></td>
<td>p_2</td>
<td>3.33</td>
<td>2.14</td>
<td>3.00</td>
<td>3.99</td>
<td>3.92</td>
<td>2.68</td>
<td>2.68</td>
<td>5.18</td>
<td>3.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Filtering techniques [1]

- **Blocking workflow**

- **Nearest Neighbor workflow**
  - i) **Sparse** Vectors → Similarity of token sets (Jaccard, Cosine, etc)
  - ii) **Dense** Vectors → Similarity of Vector Embeddings

Evaluated approaches

Non-trivial comparison → Solution:

**Configuration Optimization**, i.e., maximize precision for a recall $> \tau$, where $\tau = 0.85, 0.90, 0.95, \ldots$

FastText embeddings

<table>
<thead>
<tr>
<th>Blocking Methods</th>
<th>Number of Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Blocking</td>
<td>3,440</td>
</tr>
<tr>
<td>Q-Grams Blocking</td>
<td>17,200</td>
</tr>
<tr>
<td>Extended Q-Grams Blocking</td>
<td>68,800</td>
</tr>
<tr>
<td>(Ex.) Suffix Arrays Blocking</td>
<td>21,285</td>
</tr>
<tr>
<td>Sparse NN Methods</td>
<td></td>
</tr>
<tr>
<td>$\varepsilon$-Join</td>
<td>6,000</td>
</tr>
<tr>
<td>kNN-Join</td>
<td>12,000</td>
</tr>
<tr>
<td>Dense NN Methods</td>
<td></td>
</tr>
<tr>
<td>MH-LSH</td>
<td>168</td>
</tr>
<tr>
<td>HP-LSH</td>
<td>400</td>
</tr>
<tr>
<td>CP-LSH</td>
<td>2,000</td>
</tr>
<tr>
<td>FAISS</td>
<td>2,720</td>
</tr>
<tr>
<td>SCANN</td>
<td>10,880</td>
</tr>
<tr>
<td>DeepBlocker</td>
<td>2,720</td>
</tr>
</tbody>
</table>
Conclusions:

- **Blocking workflows:** Attribute value tokens yield better results than substrings of tokens.
- **Sparse NN:** Cardinality thresholds are more effective than similarity thresholds.
- **Dense NN:** Similarity-based methods achieve high PC only with low precision. Learning-based tuple embedding raises the PQ, but does not scale.
Scalability

Blocking Workflows

Sparse NN Methods

Dense NN Methods

RT (msec)

D10K  D50K  D100K  D200K  D300K  D1M  D2M

SBW  Q8W  ESBW  SABW  ESABW  PBW  DBW

ε-Join  kNN-Join  DkNN-Join

10K  50K  100K  200K  300K  1M  2M

MH-LSH  CP-LSH  HP-LSH  FAISS  SCANN  DeepBlocker  DDB
Conclusions

We used 10 real, established datasets for Record Linkage. We compared 13 methods, all fine-tuned to each dataset. We included 4 baseline methods with default configurations.

- PQ of all methods is highly correlated → performance heavily depends on dataset characteristics
- Parameter fine-tuning significantly increases blocking performance
- Schema-agnostic settings are preferable
- Cardinality thresholds are preferable
- Syntactic representations are preferable

Conclusion verified by Sparkly [2].

More language models for blocking [2]

Three main types:
1. Static models $\rightarrow$ each word is transformed into a fixed, context-agnostic vector word2vec
2. BERT-based models $\rightarrow$ token and sentence embeddings that capture context via the multi-headed attention in their transformer design
3. SentenceBERT models $\rightarrow$ effective, dynamic sentence embeddings efficiently via their Siamese architecture

<table>
<thead>
<tr>
<th>Static</th>
<th>BERT-based</th>
<th>SentenceBERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec (WC)</td>
<td>BERT (BT)</td>
<td>S-MPNet (ST)</td>
</tr>
<tr>
<td>GloVe (GE)</td>
<td>ALBERT (AT)</td>
<td>S-GTR-T5 (S5)</td>
</tr>
<tr>
<td>FastText (FT)</td>
<td>RoBERTa (RA)</td>
<td>S-DistilRoBERTa (SA)</td>
</tr>
<tr>
<td></td>
<td>DistilBERT (DT)</td>
<td>S-MiniLM (SM)</td>
</tr>
<tr>
<td></td>
<td>XLNet (XT)</td>
<td></td>
</tr>
</tbody>
</table>

Performance Results

- Plots show blocking recall with kNN search for k = 10.
- SentenceBERT based models (especially S-GTR-T5) perform very well.
- BERT based models (especially XLNet and ALBERT) perform poorly, even worse than static.
- Compared to DeepBlocker, S-GTR-T5 achieves about 15% higher recall on average, despite its lack of fine-tuning.
Overall Performance

- X-axis Blocking Recall, Y-axis normalized time (over fastest model).
- SentenceBERT models clearly outperform all others.
Comparison to SotA

<table>
<thead>
<tr>
<th>k</th>
<th>S-GTR-T5</th>
<th>DeepBlocker</th>
<th>Sparkly</th>
<th>TokenJoin</th>
<th>kNN-Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.5%</td>
<td>19.5%</td>
<td>6.1%</td>
<td>9.2%</td>
<td>8.9%</td>
</tr>
<tr>
<td>5</td>
<td>3.0%</td>
<td>15.3%</td>
<td>4.2%</td>
<td>6.2%</td>
<td>4.4%</td>
</tr>
<tr>
<td>10</td>
<td>2.6%</td>
<td>13.5%</td>
<td>3.3%</td>
<td>4.9%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

Average recall distance per method and number of candidates (k) over the 10 datasets.

Unsupervised Matching [4][5]

We consider algorithms that:

1. Are crafted for bipartite similarity graphs (Clean-Clean ER)
   - Dirty ER was examined in [6], Multi-source ER by FAMER
2. Have a learning-free functionality
   - We perform fine-tuning based on the ground-truth
3. Time complexity ≤ $O(n^2)$
   - $n$ stands for the number of input entities
   - E.g., the Hungarian algorithm is excluded, due to its cubic complexity, $O(n^3)$
4. Space complexity is $O(n+m)$
   - $m$ denotes the number of edges

Selected Algorithms

1. Connected components (CNC) – O(m)
2. Ricochet Sequential Rippling Clustering (RSR) – O(n m)
3. Row Column Assignment Clustering (RCA) – O(|V_1| |V_2|)
4. Best Assignment Heuristic (BAH)
   - Additional configuration parameter: 10,000 search steps or 2 min. run-time
5. Best Match Clustering (BMC) – O(m)
   - Additional configuration parameter: the node partition used as basis
6. Exact Clustering (EXC) – O(n m)
7. Király’s Clustering (KRC) – O(n + m log m)
8. Unique Mapping Clustering (UMC) – O(m log m)

Common configuration parameter: similarity threshold t
# Weighting functions for similarity graphs

## Scope

<table>
<thead>
<tr>
<th>Form</th>
<th>Representation model</th>
<th>Similarity Measure</th>
<th>Representation model</th>
<th>Similarity Measure</th>
</tr>
</thead>
</table>
| **Schema-agnostic** | character n-grams (n=2,3,4) and token n-grams (n=1,2,3) | 1) Arcs Similarity  
2) Cosine Similarity with TF Weights  
3) Cosine Similarity with TF-IDF Weights  
4) Jaccard Similarity  
5) Generalized Jaccard Similarity with TF Weights  
6) Generalized Jaccard Similarity with TF-IDF Weights | Schema-based | Character-level |
| **Character-level** | Character-level | 1) Damerau-Levenshtein  
2) Levenshtein Distance  
3) q-grams Distance  
4) Jaro Similarity  
5) Needleman Wunch  
6) Longest Common Subsequence  
7) Longest Common Substring | 1) Cosine Similarity  
2) Monge-Elkan  
3) Block Distance  
4) Dice Similarity  
5) Overlap Coefficient  
6) Euclidean Distance  
7) Jaccard Similarity  
8) Generalized Jaccard Similarity  
9) Euclidean Distance | 1) Cosine Similarity  
2) Euclidean Similarity  
3) World Mover’s Similarity |
| **Schema-based** | Token-level | 1) Cosine Similarity  
2) Monge-Elkan  
3) Block Distance  
4) Dice Similarity  
5) Overlap Coefficient  
6) Euclidean Distance  
7) Jaccard Similarity  
8) Generalized Jaccard Similarity  
9) Euclidean Distance | Token-level | Token-level |

## Syntactic Similarity

- **Character n-grams (n=2,3,4)** and **token n-grams (n=1,2,3)**

## Semantic Similarity

- **fastText and S-GTR-T5**

## Representation model

### Character-level
- fastText and S-GTR-T5

### Token-level
- fastText and S-GTR-T5
Comparison to SotA

• Baseline methods:
  – ZeroER [7], the state-of-the-art unsupervised matching algorithm
  – DITTO [8], the state-of-the-art deep learning-based matching algorithm

• Best clustering algorithm:
  – Unique Mapping Clustering coupled with:
    • schema-agnostic TF-IDF weights
    • cosine similarity

• Results w.r.t. to F-measure:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ZeroER</th>
<th>DITTO</th>
<th>UMC</th>
<th>configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>0.52</td>
<td>0.89</td>
<td>0.95</td>
<td>character bi-grams, $t=0.35$</td>
</tr>
<tr>
<td>D3</td>
<td>0.48</td>
<td>0.76</td>
<td>0.60</td>
<td>token bi-grams, $t=0.05$</td>
</tr>
<tr>
<td>D4</td>
<td>0.96</td>
<td>0.99</td>
<td>0.99</td>
<td>token uni-grams, $t=0.40$</td>
</tr>
<tr>
<td>D5</td>
<td>0.86</td>
<td>0.96</td>
<td>0.94</td>
<td>character four-grams, $t=0.35$</td>
</tr>
</tbody>
</table>

More language models for matching [2]

- Plots show maximum F1 score per case.
- SentenceBERT based models (especially S-GTR-T5) perform very well.
- BERT based models (especially XLNet and ALBERT) perform poorly, even worse than static.
- Compared to ZeroER, S-GTR-T5 (end-to-end, k = 10) is clearly better.
- ZeroER even fails to terminate in most cases.
Overall Performance

- X-axis F1 Measure, Y-axis normalized time (over fastest model).
- SBERT models clearly outperform all others.
Part D – Final Remarks and Challenges

- Introduction
- The First Four Generations
- The Fifth Generation: Leveraging External Knowledge
Conclusions

• The five generations scheme allows for easily categorizing works on ER.

• The 5\textsuperscript{th} generation dominates most recent state-of-the-art works.

• Language models, especially \texttt{SentenceBERT} ones, achieve top performance in both blocking and matching.
Challenge 1 – Large Language Models

• Increasingly replacing language models, e.g., [1][2][3]
• So far: straightforward applications based on prompt engineering
  – Zero, one, few shot prompting
  – Hand-written matching rules
  – Performance comparable to DL-based matching algorithms
• Challenge: leverage LLMs in more complex ways, e.g., end-to-end workflows

Challenge 2 – ER systems

• Literature focuses on stand-alone methods
• More emphasis on end-to-end systems
  – E.g., [https://github.com/AI-team-UoA/pyJedAI](https://github.com/AI-team-UoA/pyJedAI)
  – Library that partially covers the five generations
  – Processes data of any structuredness
  – Open-source, extensible based on the Python ecosystem
• Challenge: *how to integrate learning-based algorithms into end-to-end pipelines?*
Challenge 3 – Automatic Configuration

Facts:

• Several parameters in every method
  – Applies to all generations and workflow steps
• Performance sensitive to internal configuration
• Fine-tuning required, but huge configuration space for end-to-end pipelines

Challenge [4]:

optimize parameter configuration when:

1. Known ground-truth and pipeline
2. Known ground-truth, but unknown pipeline
3. Unknown ground-truth and pipeline

Thank You!