Entity Resolution in the Web of Data

Part II

Kostas Stefanidis\textsuperscript{1}, Vasilis Efthymiou\textsuperscript{1,2},
Melanie Herschel\textsuperscript{3,4}, Vassilis Christophides\textsuperscript{5}

kstef@ics.forth.gr, vefthym@ics.forth.gr, melanie.herschel@lri.fr
vassilis.christophides@technicolor.com

\textsuperscript{1}FORTH, \textsuperscript{2}University of Crete, \textsuperscript{3}Université Paris Sud, \textsuperscript{4}Inria Saclay,
\textsuperscript{5}Paris R\&I Center, Technicolor
Iterative Approaches
Iterative Entity Resolution

**Basic algorithm** for entity resolution in one source $E$ (dirty)

- Compare each entity description $e_i \in S$ with all other entity descriptions in $E$, i.e., with all $e_j \in E \setminus \{e_i\}$
- For comparison, use a match function to classify each pair $(e_i, e_j)$ as a match/non-match
  - Based on similarity measures
  - Based on domain-specific rules
  - Based on a combination of both
- **Complexity**: $O(N^2)$, with $N$ being the number of entity descriptions in $E$

Algorithm easily extends to entity resolution among two sources (clean-clean or dirty-dirty)
Iterative Entity Resolution

Partial results of the entity resolution process can be propagated to generate new results.

Iterative approaches can be grouped into:

- **Matching-based**: Exploit relationships between entity descriptions
  - *If descriptions related to* $e_i$ *are similar to descriptions related to* $e_j$, *this is an evidence that* $e_i$ *and* $e_j$ *are also similar*

- **Merging-based**: Exploit the partial results of merging descriptions
Iterative Entity Resolution on Complex Data

• **Tabular data**
  – Homogeneous structure
  – Similarity measures focus on variations in the values, not the structure

• **Tree data**
  – Structure of entity descriptions (of same and different types) varies
  – Similarity measures consider values, structure, and parent-child relationships

• **Graph data**
  – Structure of entity descriptions varies
  – Similarity functions consider values, structure, and neighbor relationships
Iterative Entity Resolution – Tabular Data

Table example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Eifel Tower</td>
<td>1889</td>
<td>NULL</td>
<td>France</td>
</tr>
</tbody>
</table>

- **Input:**
  - A relation with $N$ tuples
  - A similarity measure
- **Output:**
  - Classes (clusters) of equivalent tuples (= matches)
- **Problem:** a large number of tuples
  - Comparing each pair is too costly

=> Effectiveness strongly depends on good choice

=> Avoid comparisons that (most likely) yield no match
A generic approach for entity resolution in tabular data

Black-boxes:
- A match function $M$
- A merge function $\mu$

The goal:
*Minimize the number of invocations to the these expensive black-boxes*

Merged entity descriptions are considered as new entity descriptions
- Possible match candidates to other, already examined descriptions
A generic approach for entity resolution in tabular data

Black-boxes:
- A **match** function $M$
- A **merge** function $\mu$

The goal:
*Minimize the number of invocations to the these expensive black-boxes*

Merged entity descriptions are considered as new entity descriptions
- Possible match candidates to other, already examined descriptions
Swoosh

Properties that can be exploited to enhance efficiency

- **Idempotence:**
  \[ M(e_1, e_1) = true \text{ and } \mu(e_1, e_1) = e_1 \]

- **Commutativity:**
  \[ M(e_1, e_2) = M(e_2, e_1) \text{ and } \mu(e_1, e_2) = \mu(e_2, e_1) \]

- **Associativity:**
  \[ \mu(e_1, \mu(e_2, e_3)) = \mu(\mu(e_1, e_2), e_3) \]

- **Representativity:**
  if \( \mu(e_1, e_2) = e_3 \) and \( M(e_1, e_4) = true \), then \( M(e_3, e_4) = true \)
Recall-Maintaining Similarity Functions

Matching pairs of entity descriptions

Pairs of entity descriptions satisfying a strict similarity function

Set of all pairs of entity descriptions
Recall-Maintaining Similarity Functions

Matching pairs of entity descriptions

Pairs of entity descriptions satisfying a recall-preserving similarity function

Set of all pairs of entity descriptions
Recall-Maintaining Similarity Functions

- Matching pairs of entity descriptions
- Pairs of entity descriptions not satisfying a recall-preserving similarity function
- Set of all pairs of entity descriptions
Iterative Entity Resolution – Tree Data

- Examples of hierarchically organized data
  - Relational star / snowflake schema [Ananthakrishna et al. 2002]

<table>
<thead>
<tr>
<th>ID</th>
<th>Actor</th>
<th>Film</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Al Pacino</td>
<td>F1</td>
</tr>
<tr>
<td>S2</td>
<td>Al Pacino</td>
<td>F2</td>
</tr>
<tr>
<td>S3</td>
<td>Marlon Brando</td>
<td>F2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Year</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>The Godfather</td>
<td>1972</td>
<td>9.2</td>
</tr>
<tr>
<td>F2</td>
<td>Gottvatter, The</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>

- Hierarchical XML data [CHL10]

=> Specialized similarity measures

=> Specialized algorithms that traverse the tree structure
DELPHI Containment Metric [ACG02]

- **Hybrid similarity measure** [Ananthakrishna et al. 2002] considering
  - Similarity of attribute values \( (tcm) \)
  - Similarity of children sets reached by following foreign keys \( (fkcm) \)
- Similarity of **attribute values**
  - Divide tuples into tokens \( \rightarrow \) token sets \( TS \)
  - Compute the edit distance between token sets
  - Determine weight of each token using IDF [Baeza-Yates & Ribeiro-Neto 1999]
  - The **token similarity metric \( tcm \)** measures which fraction of one tuple \( T \) is covered by the other tuple \( T' \)

\[
tcm(T, T') = \frac{\sum \text{idf} (TS(T) \cap TS(T'))}{\sum \text{idf} (TS(T))}
\]
DELPHI Containment Metric [ACG02]

• 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:**
    \[
    pos(x) = 1 \quad \text{if } x > 0, \\
    -1 \quad \text{otherwise}
    \]
  - Threshold for \( tcm \): \( s1 \)
  - **Threshold for \( fkcm \): \( s2 \)**
  - Classification of pairwise comparison between \( T \) and \( T' \) using
    \[
    pos(IDF(TS) \cdot pos(tcm(T, T') - s1) + IDF(CS) \cdot pos(fkcm(T, T') - s2))
    \]
  - If final result equals 1, then match, otherwise non-match
Containment Metric - Example

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

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

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

4. Token containment metric
   $tcm(F1,F2) = \frac{3}{4}$, $tcm(F2,F1) = 1$

5. Children co-occurrence
   $fkcm(F1,F2) = 1$, $fkcm(F2,F1) = \frac{1}{2}$

6. Combination of both metrics
   ($s1 = s2 = 0.5$, weights = 1)
   pos(pos(3/4 - 0.5) + pos(1 - 0.5) = 1 
   $\rightarrow F1$ and $F2$ match
Iterative Entity Resolution - 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 [Dong et al. 2005, Weis & Naumann 2006, Bhattacharya & Getoor 2007, ...]
  – Based on an entity graph
    (1 node = 1 entity, 1 edge = relationship between 2 entities)
  – Based on reference graph
    (1 node = 2 entities, 1 edge = relationship to another entity pair)
• Many of them conform to a general framework [Herschel et al. 2012]
Iterative Entity Resolution – Graph Framework

- **Domain knowledge specification**
- **Entity-pair queue initialization**

**Initialization**

**Iterative phase**

- **Retrieval**
- **Classification**
- **Update**

- Get next pair in queue
- Apply similarity measure
- Update pair queue
Domain Expert Knowledge Specification

- Domain expert specifies
  - Duplicate **candidate entities** (e.g., movie, actor, title)
  - (Additional) **relationships** between candidates (e.g., title → movie)
Domain Expert Knowledge Specification

- Domain expert specifies
  - Duplicate **candidate entities** (e.g., movie, actor, title)
  - (Additional) **relationships** between candidates (e.g., title → movie)
For pairwise similarity computation, domain expert also selects what information is relevant for comparisons

- **Entity description** (attribute values)
- **Influencing neighbor** candidates
Duplicate detection through structure optimization

Duplicate detection through structure optimization

ACM Conference on Information and Knowledge Management,

Luis Leitao

L. Leitao

Pavel Calado

P. Calado

7 comparisons (3 re-comparisons)
Entity Pair Queue

Duplicate detection through structure optimization

Duplicate detection through structure optimization

Duplicate detection through structure optimization

Duplicate detection through structure optimization

Duplicate detection through structure optimization

4 comparisons (0 re-comparisons)
Entity Pair Queue

• Queue **maintenance** necessary whenever a **match is found**
  – Manage order in which pairs are compared to reduce re-comparisons
  – Merge matches:
    • Let $m = \text{merge} (e1, e2)$
    • Replace all occurrences of $e1$ and $e2$ in pair queue by $m$
    • Add additional pairs to queue that compare $m$ with entities already compared to either $e1$ or $e2$

• In general, goal of maintaining the priority queue is to **reduce the number of re-comparisons while maximizing effectiveness**
Iterative blocking

*Entity resolution interleaved with blocking*
Iterative Blocking [Whang et al. 2009]

Blocking is not just a simple preprocessing step of entity resolution.

Perform entity resolution on each block, and propagate results to other blocks.

Entity resolution results of a processed block, may help identifying more matches in another block.

- Newly created entity descriptions, i.e. merges of descriptions found matching, are distributed to other blocks, replacing the found matches.

Blocks are processed multiple times, until no new matches are found.
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:

\[
\begin{array}{cccc}
1889 & 1886 & 1885 \\
|1|1|1| & |1|1| & |1|1| \\
\end{array}
\]

\[
\begin{array}{cccc}
e_1, e_4 & e_2, e_5 & e_3  \\
|1|0|1| & |1|0| & |0|1| \\
\end{array}
\]

\[
\begin{array}{cccc}
P & N & L \\
|1|0|0| & |0|1|0| & |0|0|1| \\
\end{array}
\]

\[
e_1, e_4 \text{ match! they are merged as } e_{14}
\]
## Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

**Blocks generated if blocking keys are the year and the 1st letter of the location:**

- **1889**: $e_1$, $e_4$, $e_{14}$
- **1886**: $e_2$, $e_5$
- **1885**: $e_3$
- **P**: $e_1$, $e_4$, $e_{14}$
- **N**: $e_2$
- **L**: $e_3$, $e_5$
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:

1889
  e₁, e₄, e₁₄

1886
  e₂, e₅

1885
  e₃

P
  e₁, e₄, e₁₄

N
  e₂

L
  e₃, e₅

e₂, e₅ match! they are merged as e₂₅
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:

- **1889**: $e_1, e_4, e_{14}$
- **1886**: $e_2, e_5, e_{25}$
- **1885**: $e_3$
- **P**: $e_1, e_4, e_{14}$
- **N**: $e_2, e_{25}$
- **L**: $e_3, e_5, e_{25}$

$e_2, e_5$ match! they are merged as $e_{25}$
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:
### Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

**Blocks generated if blocking keys are the year and the 1st letter of the location:**

- **1889**: $e_1$, $e_4$, $e_{14}$
- **1886**: $e_2$, $e_5$, $e_{25}$
- **1885**: $e_3$

- **P**: $e_{14}$
- **N**: $e_2$, $e_{25}$
- **L**: $e_3$, $e_5$, $e_{25}$
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1\textsuperscript{st} letter of the location:
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1st letter of the location:

- e₁, e₄, e₁₄
- e₂, e₅, e₂₅
- e₃

e₃, e₂₅ match! they are merged as e₂₃₅
### Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Eiffel</td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

**Blocks generated if blocking keys are the year and the 1st letter of the location:**

- 1889: e₁, e₄, e₁₄
- 1886: e₂, e₅, e₂₅, e₂₃₅
- 1885: e₃, e₂₃₅

- e₃, e₂₅ match! they are merged as e₂₃₅
Iterative Blocking - Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Architects</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td>Sauvestre</td>
<td>Paris</td>
</tr>
<tr>
<td>Statue of Liberty</td>
<td>1886</td>
<td>Bartholdi, Eiffel</td>
<td>NY</td>
</tr>
<tr>
<td>Lady Liberty</td>
<td>1885</td>
<td>Eiffel</td>
<td>Liberty Island, NY</td>
</tr>
<tr>
<td>Eiffel Tower</td>
<td>1889</td>
<td></td>
<td>Paris</td>
</tr>
<tr>
<td>Miss Liberty</td>
<td>1886</td>
<td>Gustave Eiffel</td>
<td>Liberty Island</td>
</tr>
</tbody>
</table>

Blocks generated if blocking keys are the year and the 1\text{st} letter of the location:

- **1889**: $e_{14}$, $e_{14}$
- **1886**: $e_2$, $e_5$, $e_{25}$, $e_{235}$
- **1885**: $e_3$, $e_{235}$
- **P**: $e_{25}$, $e_{235}$
- **N**: $e_{235}$
- **L**: $e_3$, $e_{25}$, $e_{235}$

Process continues iteratively, until no new matches are found.
Extend iterative blocking by using MinHash
MinHash

Assume an entity description is a set of tokens:

\[ \text{e}_1 = \{ \text{the, statue, of, liberty} \} \]
\[ \text{e}_2 = \{ \text{lady, liberty, statue} \} \]
\[ \text{e}_3 = \{ \text{the, eiffel, tower} \} \]

<table>
<thead>
<tr>
<th>Token/Description</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>statue</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>of</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>liberty</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lady</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>eiffel</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>tower</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

MinHash:

- Pick a permutation of the tokens
- \( \text{minHash(e)} \): the first token of \( \text{e} \) in the permuted order of tokens

\( \text{minHash(e}_1 \) = ‘of’
\( \text{minHash(e}_2 \) = ‘lady’
\( \text{minHash(e}_3 \) = ‘tower’
MinHash as a Jaccard Approximation

\[ P[\text{minHash}(e_i) = \text{minHash}(e_j)] = \text{Jaccard}(e_i, e_j) \]

- Type A rows: both descriptions have 1
- Type B rows: one has 1 and the other has 0
- Type C rows: both have 0

\[ P[\text{minHash}(e_i) = \text{minHash}(e_j)] = \frac{\#A}{\#A + \#B} = \text{Jaccard}(e_i, e_j) \]

- Create several random permutations \((h_1, \ldots, h_n)\) of the tokens, \(n << \#\text{tokens}\)
  - Permutations are expensive; hash functions can simulate this functionality
- For each permutation, find the minHash of each description
  - These, concatenated, constitute the minHash signature of each description
- Compare the minHash signatures of the descriptions
  - Using Locality-Sensitive Hashing (LSH)
Extends iterative blocking by employing MinHash (for Jaccard approximation)

**Scalability:** A single hash table is used
- Before placing a description in a block, the description is compared to the contents of the block
HARRA - Example

e_6 should be placed in the blue bucket

Hash Table:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>e_1, e_2</td>
</tr>
<tr>
<td>Red</td>
<td>e_3, e_5</td>
</tr>
<tr>
<td>Black</td>
<td>e_4</td>
</tr>
</tbody>
</table>
Before placing it there, we check if it matches $e_1$ or $e_2$

$$e_6 = e_1 \ ? \ NO$$
$$e_6 = e_2 \ ? \ YES$$

**Hash Table:**

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>$e_1$ $e_2$</td>
</tr>
<tr>
<td>Red</td>
<td>$e_3$ $e_5$</td>
</tr>
<tr>
<td>Black</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>
HARRA - Example

Before placing it there, we check if it matches $e_1$ or $e_2$

$e_6 = e_1$ ? NO
$e_6 = e_2$ ? YES

$e_26$ is the result of merging $e_6$ and $e_2$

$e_26 = e_1$ ? NO

Hash Table:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>$e_1$  $e_{26}$</td>
</tr>
<tr>
<td>Red</td>
<td>$e_3$  $e_5$</td>
</tr>
<tr>
<td>Black</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>
HARRA - Example

Continue until:
• no merge occurs, OR
• saved comparisons > threshold, OR
• # iterations > constant

Re-initialize the input:

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td></td>
</tr>
</tbody>
</table>

Hash Table:
Blocking vs Iterative Blocking

- Matching pairs of entity descriptions
- Blocking
- Iterative Blocking
- Set of all pairs of entity descriptions

**Iterative Blocking**

(Pros) Lead to more identified matches
(Cons) Lead to more comparisons
Discussion on Iterative Approaches

Each iteration is based on new knowledge
- Identified matches
- Merged descriptions of identified matches

Hybrid methods, i.e. iterative blocking, benefit from:
- The efficiency of blocking approaches
- The effectiveness of iterative approaches

Iterative approaches seem to fit well to similarity functions using relationships
- Relationships between descriptions are an important part of the available semantics
# A Classification of Iterative Approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Matching-based</th>
<th>Merging-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bhattacharya &amp; Getoor 2004, 2007</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>Rastogi et al. 2011</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>Dong et al. 2005</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>Herschel et al. 2012</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>Weis &amp; Naumann 2006</td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>Weis &amp; Naumann 2004</td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>Leitão et al. 2007, 2013</td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>Puhlmann et al. 2006</td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>Böhm et al. 2012</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Benjelloun et al. 2009</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Benjelloun et al. 2007</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Whang et al. 2009</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Kim &amp; Lee 2010</td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

• : tabular data
□ : tree data
+ : graph data
Scalability Limitations

Computations are sequential
• High time requirements
  – E.g., 66 hours, just to create the clusters of attributes for attribute clustering blocking for a dataset of 3M entities

Data reside in the memory of a single machine
• High memory and space requirements
  – I.e., “out of memory” errors, when datasets get bigger and in some cases “no space left on device” errors

Parallel algorithms can be used to overcome these limitations
For handling huge volumes of data

MapReduce
MapReduce

Input data are partitioned

Input data partitions are sent to different nodes (mappers) in the cluster

- **Map phase**: distribute the current partition to multiple nodes (reducers)
  - Emit (key, value) pairs
  - Pairs with the same key are processed by the same reducer

- **Reduce phase**: process the pairs having the same key
  - Emit (key, value) pairs – the output of the program
For handling huge volumes of data:

*Proceed entity resolution in partitions!*

The **map phase** reflects **blocking** (re-distribute descriptions)

The **reduce phase** reflects **entity resolution** (check for matches)
MapReduce – Input Data

<table>
<thead>
<tr>
<th>e1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MapReduce – Input Data Partitioning
MapReduce – Mapper Input

Mapper 1
 Mapper 2
 Mapper 3
MapReduce – Mapper Example

**Input:**
e1={(name, Auguste Bartholdi),(year,1834)}
e2={(about, Auguste Bartholdi)}
e3={(architects, Bartholdi Eiffel)}

**Output:**
Bartholdi e1 Auguste e1 1834 e1
Bartholdi e2 Auguste e2
Eiffel e3 Bartholdi e3
MapReduce – Mapper Output

Mapper 1

Mapper 2

Mapper 3
MapReduce – Shuffling & Sorting
MapReduce – Merging

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5
MapReduce – Reducer

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5
MapReduce – Reducer Example

Input:
Bartholdi e1 e2 e3 e4

Output:
e1-e2 match
e3-e4 match

*Dedoop performs standard blocking using MapReduce*

**Map function**
- Input: an entity description
- Output: a (key, value) pair
  - key: the BKV of the description
  - value: the description having this BKV

The partitioning operates on the BKVs and distributes (key, value) pairs among reduce tasks
- All entities sharing the same BKV are assigned to the same reduce task

**Reduce function**: Computes in each block the similarities between all description pairs within the block
- Input: A BKV along with descriptions with this BKV
- Output: (key, value) pairs
  - key: a pair of descriptions
  - value: match/non-match
Dedoop – Mapper: BKVs as intermediate keys
Dedoop – Mappers: Build Blocks

Block 1
- k1 e1
- k2 e2
- k4 e3

Block 2
- k1 e4
- k2 e2
- k2 e6

Block 3
- k3 e5
- k3 e5
- k3 e7

Block 4
- k4 e3
- k4 e8
Dedoop –Reducers: Compare Block Contents

- **Block 1**
  - **k1 e1**
  - **k2 e2**
  - **k4 e3**
  - **k1 e4**
- **e1-e4 match**

- **Block 2**
  - **k2 e2**
  - **k2 e6**
  - **k3 e5**
  - **k3 e5**
- **e2-e6 non-match**

- **Block 3**
  - **k2 e6**
  - **k3 e5**
  - **k3 e7**
- **e5-e7 non-match**

- **Block 4**
  - **k4 e3**
  - **k4 e3**
  - **k4 e8**
  - **k4 e8**
- **e3-e8 match**
The output of a MapReduce Job can be the input of another Job.

Chaining MapReduce reflects iterative entity resolution.
Dedoop – Sorted Neighborhood [Kolb et al. 2011]

composite key = (partitionID, BKV)
partitionID(BKV) = 1, if BKV < “k3”
partitionID(BKV) = 2, else

(we know that we have two reducers available)
Dedoop SN: Sorting the Keys
Dedoop SN: Reducers Apply the Sliding Window
Dedoop SN: Reducers Apply the Sliding Window

Reducer 1

Window 1

e1-e4 match
e1-e2 match
e4-e2 non-match

Reducer 2

Window 1

e5-e7 match
e5-e3 non-match
e7-e3 match

\[ w = 3 \]
Dedoop SN: Reducers Apply the Sliding Window

Reducer 1

1.k1 e1
1.k2 e2
1.k1 e4
1.k2 e2
1.k2 e6

Window 2

e4-e2 non-match
e4-e6 match
e2-e6 non-match

Reducer 2

1.k1 e1
2.k4 e3
1.k2 e2
1.k2 e6
2.k3 e5
2.k3 e7
2.k3 e5
2.k4 e3
2.k4 e8
2.k4 e8

Window 2

e7-e3 match
e7-e8 non-match
e3-e8 match

w = 3
Dedoop SN: We Also Need To Compare The Boundary Entities

Reducer 1

Reducer 2

\[ w = 3 \]
**Dedoop SN: Reducers Also Output the Boundary Descriptions**

Add a **boundary number prefix** to the output composite keys

**Boundary number:**
The last w-1 descriptions of reducer i are assigned the boundary number i

The first w-1 descriptions of reducer i+1 are also assigned the boundary number i

The actual blocking key of e5 is k3, it was assigned to reducer 2 and it is associated with boundary number 1

Reduction:

- **Reducer 1**
  - 1.k1 e1
  - 1.k1 e4
  - 1.k2 e2
  - 1.k2 e6
  - 2.k3 e5
  - 2.k3 e7

- **Reducer 2**
  - 2.k4 e3
  - 2.k4 e8

w = 3
Dedoop SN: New MapReduce Job for the Boundary Pairs

Reducer 1
- 1.k1 e1
- 1.k1 e4
- 1.k2 e2
- 1.k2 e6
- 2.k3 e5
- 2.k3 e7

Reducer 2
- 2.k4 e3
- 2.k4 e8

Identical map
- 1.1.k2 e2
- 1.1.k2 e6
- 1.2.k3 e5
- 1.2.k3 e7
Dedoop SN: Partition by Boundary Number

Reducer 1

1.k1 e1
1.k1 e4
1.k2 e2
1.k2 e6
2.k3 e5
2.k3 e7

Reducer 2

2.k4 e3
2.k4 e8

Identical map

Reducer applies sliding window

Window 1

1.1.k2 e2
1.1.k2 e6
1.1.k2 e5
1.2.k3 e5
1.2.k3 e7

Window 2

e2-e6 non-match
e2-e5 match
e6-e5 non-match
e6-e5 non-match
e6-e7 match
e5-e7 match
Still, there are repeated comparisons
Dedoop SN: Skipping Repeated Comparisons

These comparisons are not performed again:
They have been performed in the previous
MapReduce job (they come from the same reducer)
Don’t match twice [Kolb et al. 2013]

Overlapping blocks lead to repeated comparisons

Adopt Comparison Propagation [Papadakis et al. 2012] to MapReduce:
• Descriptions need to be compared only within their least common block
Overlapping Blocks Lead to Repeated Comparisons

<table>
<thead>
<tr>
<th></th>
<th>k1</th>
<th>e1</th>
<th>k2</th>
<th>e2</th>
<th>k3</th>
<th>e1</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>k1</td>
<td>e1</td>
<td>k2</td>
<td>e1</td>
<td>k3</td>
<td>e1</td>
</tr>
<tr>
<td>e1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>k4</td>
<td>e3</td>
<td>k1</td>
<td>e3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e3</td>
<td>k4</td>
<td>e3</td>
<td>k1</td>
<td>e3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e4</td>
<td>k2</td>
<td>e4</td>
<td>k1</td>
<td>e4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e4</td>
<td>k2</td>
<td>e4</td>
<td>k1</td>
<td>e4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e5</td>
<td>k5</td>
<td>e5</td>
<td>k3</td>
<td>e5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e5</td>
<td>k5</td>
<td>e5</td>
<td>k3</td>
<td>e5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e6</td>
<td>k2</td>
<td>e6</td>
<td>k4</td>
<td>e6</td>
<td>k5</td>
<td>e6</td>
</tr>
<tr>
<td>e6</td>
<td>k2</td>
<td>e6</td>
<td>k4</td>
<td>e6</td>
<td>k5</td>
<td>e6</td>
</tr>
<tr>
<td>e7</td>
<td>k5</td>
<td>e7</td>
<td>k3</td>
<td>e7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e7</td>
<td>k5</td>
<td>e7</td>
<td>k3</td>
<td>e7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e8</td>
<td>k4</td>
<td>e8</td>
<td>k5</td>
<td>e8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e8</td>
<td>k4</td>
<td>e8</td>
<td>k5</td>
<td>e8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reducer 1
Reducer 3
Reducer 4
Reducer 5

- e1-e2 match
- e1-e3 match
- e1-e2 match
- e4-e6 match
- e5-e7 match
- e3-e8 match
- e5-e7 match
- e6-e7 match

80
### Map: Append the Subset of Smaller Keys for the Same Description

<table>
<thead>
<tr>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
<th>e5</th>
<th>e6</th>
<th>e7</th>
<th>e8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k1</td>
<td>e1,{}</td>
<td>k2</td>
<td>e1,{k1}</td>
<td>k3</td>
<td>e1,{k1,k2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k1</td>
<td>e2,{}</td>
<td>k2</td>
<td>e2,{k1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k1</td>
<td>e3,{}</td>
<td>k4</td>
<td>e3,{k1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k1</td>
<td>e4,{}</td>
<td>k2</td>
<td>e4,{k1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k3</td>
<td>e5,{}</td>
<td>k5</td>
<td>e5,{k3}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k2</td>
<td>e6,{}</td>
<td>k4</td>
<td>e6,{k2}</td>
<td>k5</td>
<td>e6,{k2,k4}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k3</td>
<td>e7,{}</td>
<td>k5</td>
<td>e7,{k3}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k4</td>
<td>e8,{}</td>
<td>k5</td>
<td>e8,{k4}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Map: Append the Subset of Smaller Keys for the Same Description

- e1
- e2
- e3
- e4
- e5
- e6
- e7
- e8
Resulting Comparisons

- k1: e1, {}  
  e2, {}  
  e3, {}  
  e4, {}  

- k2: e1, {k1}  
  e2, {k1}  
  e4, {}  

- k3: e1, {k1, k2}  
  e3, {}  
  e4, {}  

- k4: e3, {k1}  
  e6, {k2}  
  e8, {}  

- k5: e6, {k2, k4}  
  e7, {k3}  
  e8, {}  

Comparisons:
- e1-e2  
- e1-e3  
- e1-e4  
- e2-e3  
- e2-e4  
- e3-e4  
- e1-e5  
- e1-e6  
- e1-e7  
- e2-e6  
- e4-e6  
- e3-e6  
- e3-e8  
- e6-e8  
- e5-e6  
- e5-e7  
- e5-e8  
- e6-e7  
- e7-e8  

Arrows indicate the direction of the comparisons.
A More Efficient Approach [Vernica et al. 2010]

Reducer’s output...
A More Efficient Approach [Vernica et al. 2010]

...becomes a new job’s input
Large-Scale Collective Entity Matching
[Rastogi et al. 2011]

Assume that there is a rule $R: \text{Match}(e_1, e_2) \Rightarrow \text{Match}(e_4, e_5)$
and that we have inferred: $\text{Match}(e_1, e_2)$
In $C_2$, we cannot infer $\text{Match}(e_4, e_5)$

We should somehow inform $C_2$ that $e_1$ matches $e_2$
• Then we could infer that $e_4$ matches $e_5$, according to rule $R$

Solution: message passing
• After matching in $C_1$ finishes, send a message “$\text{Match}(e_1, e_2)$”
• In the next MapReduce round, entity resolution runs with the new evidence and infers $\text{Match}(e_4, e_5)$
Linda [Böhm et al. 2012]

- Works on an **entity graph** constructed from RDF triples having URIs as subject, predicate and object
  - Literals are stored for each entity e as L(e)

- Matches are identified using **two kinds of similarities**:
  - String similarity (token-based) of their literal values L(e)
    - Checked once
  - Contextual similarity (based on neighbors in the entity graph)
    - Checked iteratively
Contextual Similarity

What is context?

- Let node \( n \) in an entity graph correspond to an RDF subject or object, identified by a URI
- The context \( C(n) \) of \( n \) is a set of tuples \( (p_i, z_i, w_i) \), where
  - \( z_i \) is a neighboring node of \( n \)
  - \( p_i \) is the predicate associated with an edge connecting \( n \) with \( z_i \)
  - \( w_i \) is a numeric weight (how discriminative this information is)

That is, the context of \( n \) includes objects \( z_i \) of triples with \( n \) as subject and subjects \( z_i \) of triples with \( n \) as object

\[
C(\text{Statue of liberty}) = \{(\text{location, Liberty Island, w1}), (\text{is work of, Bartholdi, w2})\}
\]
Contextual Similarity

The contextual similarity of nodes $n$ and $m$ is:

$$
\text{context}_\text{sim}(n, m) =
\begin{cases}
\sum \max (p_i, z_i, w_i) \in C(n) (p_j, z_j, w_j) \in C(m) \quad w_i \cdot x_{z_i, z_j} \cdot \text{sim}(p_i, p_j), & \text{if } |C(n)| \leq |C(m)| \\
\sum \max (p_j, z_j, w_j) \in C(m) (p_i, z_i, w_i) \in C(n) \quad w_j \cdot x_{z_i, z_j} \cdot \text{sim}(p_i, p_j), & \text{else}
\end{cases}
$$

where

$x_{n,m}$ is 1, if $n$, $m$ are identified as matches, and 0, else

$\text{sim}(p_i, p_j)$ is the string similarity of the predicates of $n$, $m$

Intuitively, the contextual similarity finds matching neighbors and sums up their similarity values.
Contextual Similarity

Overall similarity: combine sim and context_sim

The similarity score for descriptions n and m is:
\[ sim(n, m) + \beta \cdot \text{context}_\text{sim}(C(n), C(m)) - \theta \]

\( \beta \) controls the contextual influence
\( \theta \) is used for re-normalization to values around 0
- positive scores reflect likely mappings
- negative scores imply dissimilarities

Experiments have shown \( \beta = 1 \) to perform well
LINDA Algorithm

Two square matrices (|E|x|E|) are used:
- X captures the identified matches (binary values)
- Y captures the pair-wise similarities (real values)
  - Initialization: common neighbors and string similarity of literals
  - Updates: Use the new identified matches of X

Until a priority queue of pairs (extracted from Y) becomes empty:
- Get the pair (e_i, e_j) with the highest similarity
  - (e_i, e_j) match by default!
    - Update X: matches of e_i are also matches of e_j
- Update the queue wrt. the new matches
LINDA – Distributed Entity Resolution Using MapReduce

Distribute across a cluster the input entity graph
  • A node $i$ holds a portion $Q_i$ of the priority queue and the respective part $G_i$ of the graph

**Map phase**
  • Mapper $i$ reads $Q_i$ and forwards messages to reducers for similarities re-computations
    – Matrix $X$ of identified matches is updated

**Reduce phase**
  • Similarities re-computations (Matrix $Y$)
  • Updates on priority queues
dbpedia:Statue_of_Liberty

dbpedia:Liber
ty_Island

dbpedia:Liberty_Island

dbpedia:Statue_of_Liberty

Priority Queue:

(dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)

(dbpedia:Statue_of_Liberty, yago:Liberty_Island)

(dbpedia:Liberty_Island,yago:Upper_NY_Bay)

(dbpedia:Liberty_Island, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Statue_of_Liberty)

(dbpedia:Bartholdi, fb:m.0jph6)

(dbpedia:Bartholdi, yago:Statue_of_Liberty)

(dbpedia:Bartholdi, fb:m.072p8)
Priority Queue 1 (machine 1):

- (dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
- (dbpedia:Statue_of_Liberty, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Upper_NY_Bay)
- (dbpedia:Liberty_Island, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2 (machine 2):

- (dbpedia:Bartholdi, fb:m.0jph6)
- (dbpedia:Bartholdi, yago:Statue_of_Liberty)
- (dbpedia:Bartholdi, fb:m.072p8)

The priority queue is partitioned and partitions are sent to the MapReduce nodes
The priority queue is partitioned and partitions are sent to the MapReduce nodes.
Priority Queue 1:

- (dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
- (dbpedia:Statue_of_Liberty, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Upper_NY_Bay)
- (dbpedia:Liberty_Island, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2:

- (dbpedia:Bartholdi, fb:m.0jph6)
- (dbpedia:Bartholdi, yago:Statue_of_Liberty)
- (dbpedia:Bartholdi, fb:m.072p8)

The head of each queue is a match by default
This triggers update messages
Priority Queue 1:

- (dbpedia:Statue_of_Liberty, yago:Statue_of_Liberty)
- (dbpedia:Statue_of_Liberty, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Upper_NY_Bay)
- (dbpedia:Liberty_Island, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2:

- (dbpedia:Bartholdi, fb:m.0jph6)
- (dbpedia:Bartholdi, yago:Statue_of_Liberty)
- (dbpedia:Bartholdi, fb:m.072p8)
- (dbpedia:Bartholdi, yago:Liberty_Island)

Dequeue these pairs, as each entity can be mapped to at most one entity per data source.
Priority Queue 1:

2. (dbpedia:Statue_of_Liberty, yago:Liberty_Island)
3. (dbpedia:Liberty_Island, yago:Upper_NY_Bay)
4. (dbpedia:Liberty_Island, yago:Liberty_Island)
5. (dbpedia:Liberty_Island, yago:Statue_of_Liberty)

Priority Queue 2:

1. (dbpedia:Bartholdi, fb:m.0jph6)
2. (dbpedia:Bartholdi, yago:Statue_of_Liberty)
3. (dbpedia:Bartholdi, fb:m.072p8)

Send messages to the other nodes and check this constraint again.
Priority Queue 1:

(dbpedia:Liberty_Island,yago:Upper_NY_Bay)
(dbpedia:Liberty_Island, yago:Liberty_Island)

Priority Queue 2:

Contextual similarity re-computations
Property names are also taken into account
Priority Queue 1:

(dbpedia:Liberty_Island, yago:Liberty_Island)  
(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

Priority Queue 2:

Priority queues are updated based on the new similarities
Priority Queue 1:

(dbpedia:Liberty_Island, yago:Liberty_Island)

(dbpedia:Liberty_Island, yago:Upper_NY_Bay)

Priority Queue 2:

The head of each queue is a match by default
This triggers update messages
Priority Queue 1:

- (dbpedia:Liberty_Island, yago:Liberty_Island)
- (dbpedia:Liberty_Island, yago:Upper_NY_Bay)

Dequeue this pair, as each entity can be mapped to at most one entity per data source.
Using Neighbors for Computing Similarities

- Matching pairs of entity descriptions
- Without neighbors (a loose similarity function is used)
- With neighbors (a strict similarity function is used)
- Set of all pairs of entity descriptions

With neighbors (pros) Lead to more identified matches
(cons) Lead to more comparisons
Entity Resolution in the Web of Data

So far…
Rely on the values of the descriptions
• *A good way to handle data heterogeneity and low structuredness*

=> Deal with loosely structured entities
=> Deal with various vocabularies (*side effect*)
=> Deal with large volumes of data

Still, many redundant comparisons are performed!
• Can we also use the structural type of the descriptions?

A brief overview of our ongoing work on blocking using MapReduce follows…
Methods and Datasets

MapReduce implementations of:
- Token Blocking (ToB)
- Attribute Clustering Blocking (AtC)
- Prefix-Infix(-Suffix) Blocking (PIS)

Datasets: Billion Triples Challenge 2012 (BTC12)
- D1: BTC12 DBpedia (~148M triples) and DBpedia 3.5 (~32M triples)
- D2: BTC12 DBpedia and BTC12 Rest (~918K triples)
- D3: BTC12 DBpedia and BTC12 Freebase (~29M triples)

On a cluster of 15 nodes (each with 8 CPUs, 8GB RAM, and 60GB HDD)

For ground-truth, we used the owl:sameAs links provided by DBpedia

For each dataset, we performed both dirty and clean-clean ER
Scalability

D1 (10.6M entities)

D2 (9M entities)

D3 (10.8M entities)
# Quality Results

<table>
<thead>
<tr>
<th></th>
<th>ToB (Dirty)</th>
<th>AtC (Clean-Clean)</th>
<th>PIS (Dirty)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>$3.598 \times 10^{-7}$</td>
<td>$8.064 \times 10^{-6}$</td>
<td>$9.955 \times 10^{-7}$</td>
</tr>
<tr>
<td>D2</td>
<td>$2.540 \times 10^{-8}$</td>
<td>$6.103 \times 10^{-6}$</td>
<td>$5.964 \times 10^{-8}$</td>
</tr>
<tr>
<td>D3</td>
<td>$4.376 \times 10^{-7}$</td>
<td>$9.759 \times 10^{-6}$</td>
<td>$3.729 \times 10^{-6}$</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>90.5%</td>
<td>88%</td>
<td>91.7%</td>
</tr>
<tr>
<td>D2</td>
<td>86.3%</td>
<td>70.6%</td>
<td>87.6%</td>
</tr>
<tr>
<td>D3</td>
<td>86.5%</td>
<td>84.3%</td>
<td>84.4%</td>
</tr>
<tr>
<td><strong>RR</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>90.1%</td>
<td>97.8%</td>
<td>92.5%</td>
</tr>
<tr>
<td>D2</td>
<td>90.9%</td>
<td>98.6%</td>
<td>91.7%</td>
</tr>
<tr>
<td>D3</td>
<td>92.7%</td>
<td>98.9%</td>
<td>90.8%</td>
</tr>
<tr>
<td><strong>#comparisons</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>5,588 B</td>
<td>322 B</td>
<td>4,394 B</td>
</tr>
<tr>
<td>D2</td>
<td>3,681 B</td>
<td>4 B</td>
<td>3,449 B</td>
</tr>
<tr>
<td>D3</td>
<td>4,272 B</td>
<td>184 B</td>
<td>5,339 B</td>
</tr>
</tbody>
</table>
Room for Improvement

Ideally:
• Similar entity descriptions in the same block
• Dissimilar entity descriptions in different blocks?

Entity Descriptions → Blocking → Entity Resolution → Resolved Entities
Current Trends
Temporal Entity Resolution

Entity resolution should account for changes over time

- Entities evolve over time
- Entities have a lifetime
## Linked Datasets Evolve Over Time

### Current version of DBpedia

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>Statue of Liberty, Freiheitsstatue, …</td>
</tr>
<tr>
<td>dbpprop:location</td>
<td>New York City, New York, U.S., dbpedia:Liberty_Island</td>
</tr>
<tr>
<td>dbpprop:sculptor</td>
<td>dbpedia:Frédéric_Auguste_Bartholdi</td>
</tr>
<tr>
<td>dcterms:subject</td>
<td>dbpedia_category:1886_sculptures, …</td>
</tr>
<tr>
<td>dbpprop:beginningDate</td>
<td>1886-10-28 (xsd:date)</td>
</tr>
<tr>
<td>dbpprop:restored</td>
<td>19381984 (xsd:integer)</td>
</tr>
<tr>
<td>dbpprop:visitationNum</td>
<td>3200000 (xsd:integer)</td>
</tr>
<tr>
<td>dbpprop:visitationYear</td>
<td>2009 (xsd:integer)</td>
</tr>
<tr>
<td><a href="http://www.w3.org/ns/prov#wasDerivedFrom">http://www.w3.org/ns/prov#wasDerivedFrom</a></td>
<td><a href="http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330">http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330</a></td>
</tr>
</tbody>
</table>

### Previous version of DBpedia

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>Statue of Liberty, Freiheitsstatue, …</td>
</tr>
<tr>
<td>dbpprop:location</td>
<td>New York City, New York, U.S., dbpedia:Liberty_Island</td>
</tr>
<tr>
<td>dbpprop:sculptor</td>
<td>dbpedia:Frédéric_Auguste_Bartholdi</td>
</tr>
<tr>
<td>dcterms:subject</td>
<td>dbpedia_category:1886_sculptures, …</td>
</tr>
<tr>
<td>dbpprop:built</td>
<td>1886-10-28 (xsd:date)</td>
</tr>
<tr>
<td>dbpprop:restored</td>
<td>19381984 (xsd:integer)</td>
</tr>
<tr>
<td>dbpprop:hasHeight</td>
<td>151 (xsd:integer)</td>
</tr>
<tr>
<td><a href="http://www.w3.org/ns/prov#wasDerivedFrom">http://www.w3.org/ns/prov#wasDerivedFrom</a></td>
<td><a href="http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330">http://en.wikipedia.org/wiki/Statue_of_Liberty?oldid=494328330</a></td>
</tr>
</tbody>
</table>
Entities Have a Lifetime

Example: Matching a description of Ronald Reagan, stating that he was the US President to a description of Barack Obama, stating that he is the US President.

Yago2 [Hoffart et al. 2012]: A temporal knowledge base, built with data from Wikipedia, GeoNames and Wordnet.

- Entities are assigned a time span to denote their existence in time:
  - e.g. Ronald Reagan is associated with 1911-02-06 (birthdate) and 2004-06-05 (time of death).

- Facts are assigned a time point, or a time span:
  - e.g. the fact “Ronald Reagan is US President” is associated with the time span from 1981-01-20 to 1989-01-20.
Privacy Protection

Example ([PleaseRobMe.com](http://PleaseRobMe.com)): Exploit Foursquare to detect the current location of Twitter users and identify houses easy to burgle, as the inhabitants are traveling at the time.

*Measure one’s ability to link two matching descriptions and then, try to minimize it.*

Privacy breach can be measured in terms of how much of the complete information about a person is available to an adversary [Whang et al. 2011, 2013]

- Use *disinformation* to make entity resolution for an adversary difficult:
  - Link a description to irrelevant descriptions
  - Add “incorrect but believable” information to a description
Crowdsourcing

Instead of machine-only approaches, use hybrid human-machine approaches
- Pass the critical (expensive/difficult) decisions to the users

Machines do a first, coarse pass over all the data (e.g. blocking)
- When algorithms fail to reach a match decision, ask humans [Demartini et al. 2013]
- Humans are used to verify only the most likely matches [Wang et al. 2012]

Humans can make mistakes too! (spam, low attention, etc.)
- Given a set of descriptions, find which questions between pairs of descriptions will reveal the most about the underlying entities [Verroios et al. 2014]
Thank You!

Other points for future work?

Questions?
References
References

References

• Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. TKDD 1(1) (2007)
References

References

References

- Kolb, L., Thor, A., Rahm, E.: Don’t match twice: redundancy-free similarity computation with MapReduce. In: Data Analytics in the Cloud (2013)
References

• Weis, M., Naumann, F.: DogmatiX Tracks down Duplicates in XML. In: SIGMOD, pp. 431-442 (2005)
Acknowledgements

We are thankful to the support provided by the following projects:


- GSRT ARISTEIA (LODGOV) Data Governance in the era of the Web of Data

- FP7-PEOPLE- 2013-IRSES SemData (Semantic Data Management)
License

These slides are made available under a Creative Commons Attribution-ShareAlike license (CC BY-SA 3.0):
http://creativecommons.org/licenses/by-sa/3.0/

You can share and remix this work, provided that you keep the attribution to the original authors intact, and that, if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.