Big Data Entity Resolution
TIETS44

Kostas Stefanidis
konstantinos.stefanidis@uta.fi

Period III, 2018 – 2019

https://coursepages.uta.fi/tiets44/
Matching and Resolving Entities: Iterative Resolution Techniques
Central vs. Peripheral KBs

Zooming into the center of the LOD cloud, we can find KBs, such as DBpedia, YAGO and Freebase, each containing millions of entity descriptions of thousands of different types, that are heavily interlinked. On the other hand, peripheral KBs are sparsely interlinked and they typically describe entities of very specific types.
In Search of Similarity Evidence in KBs

- **Attribute-based comparisons**
  - unique attributes (e.g., rdfs:label) provide strong evidence
    - >90% of matching pairs have >80% overlap similarity in the values of rdfs:label

- **Content-based comparisons**
  - central KBs: 3-4 common tokens in entity values
  - peripheral KBs: 1-2 common tokens in entity values
    - blocking algorithms miss up to 30% matches in peripheral KBs

- **Relationships-based comparisons**
  - matching neighbors provide positive evidence
    - >92% of pairs with at least one matching neighbor, are matches in most KBs
  - some types of relationships provide strong negative evidence
    - dissimilar values for wasBornIn indicate a non-matching pair
Types of Missed Matches (FNs)

- **Type A**: a third, matching description (transitivity)
  - applicable to identify matches within a KB

- **Type B**: matches of their neighbors
  - can identify matches both within a KB and across different KBs
Iterative ER: identify new matches based on partial results either of matches or of merges

- Good for high Variety
Generate new candidate pairs of descriptions not considered in a previous step.
Clearly, several passes increase the number of comparisons and reduce the Reduction Ratio.
Iterative ER Approaches

- **Merging-based**: new matches can be found by exploiting merged (more complete) descriptions of previously identified matches
  - **Idea**: ER resembles a *database self-join operation* (of the initial set of descriptions with itself)
    - But: No knowledge about which descriptions may match, so all pairs of descriptions need to be compared

- **Matching-based**: If descriptions related to entity \( e_i \) are matching to descriptions related to \( e_j \), then \( e_i \) and \( e_j \) are likely to match
  - **Idea**: ER resembles to a *graph traversal problem* in which similarity is propagated until a fixed point is reached
    - Use positive or negative evidence for prioritize similarity re-computation
Matching and Resolving Entities: Merging-based Iterative Resolution
Merging-based Iterative Resolution

In merging-based iterative entity resolution:

- The matching decision between two descriptions triggers a merge operation, which transforms the initial entity collection by adding the new, merged description and potentially removing the two initial descriptions.

- This change also triggers more updates in the matching decisions, since the new, merged description needs to be compared to the other descriptions of the collection.

*Intuitively, the final result of merging-based iterative entity resolution is a new set of descriptions which are the results of merging all the matches found in the initial entity collection.*

- Each real-world entity described in the input entity collection is represented by a single description in the resolution results and each description in the resolution results represents a distinct real-world entity from the input entity collection.
Merging-based ER– Formal Definition

Let $E = \{e_1, \ldots, e_m\}$ a set of entity descriptions and

- $M : E \times E \rightarrow \{\text{true, false}\}$ is a match function
- $\mu : E \times E \rightarrow E$ is a partial merge function

The merging-based resolution of entities in $E$ is a set of descriptions $E'$, such that:

1. $\forall e_i, e_j \in E : M(e_i, e_j) = \text{true}, \exists e_k \in E' : \mu(e_i, e_j) \leq e_k$

2. $\forall e_k \in E', \forall e_l \in E, e_l \leq e_k$

3. no strict subset of $E'$ satisfies conditions 1 and 2

where $e_l \leq e_k$ means that $e_k$ is at least as informative than $e_l$, regarding the same real-world entity

Note: $E' \subseteq E$ (the merge closure)

- Condition 1: $E'$ cannot produce more than $E$
- Condition 2: $E'$ produce at least all information of $E$
Match and Merge Functions: ICAR Properties [Benjelloun et al., 2009]

- **Idempotence**: a description always matches itself, and merging it with itself still yields the same description
  \[ \forall e_i \in E \text{ if } M(e_i, e_i) = \text{true then } \mu(e_i, e_i) = e_i \]

- **Commutativity**: direction of match and merge is irrelevant
  \[ \forall e_i, e_j \in E \text{ if } M(e_i, e_j) = M(e_j, e_i) = \text{true then } \mu(e_i, e_j) = \mu(e_j, e_i) \]

- **Associativity**: order of merge is irrelevant
  \[ \forall e_i, e_j, e_k \in E \text{ if } \mu(\mu(e_i, e_j), e_k) \text{ and } \mu(e_i, \mu(e_j, e_k)) \text{ exist then } \mu(\mu(e_i, e_j), e_k) = \mu(e_i, \mu(e_j, e_k)) \]

- **Representativity**: merging does not lose matches; no “negative evidence”
  \[ \text{if } e_k = \mu(e_i, e_j) \text{ then } \forall e_l \in E \text{ such that } M(e_i, e_l) = \text{true also } M(e_k, e_l) = \text{true} \]

- **Transitivity is not assumed!**
Merge Domination & Monotonicity

- When the match and merge functions satisfy the ICAR properties, there is a natural domination (partial) order of descriptions.
- Given two descriptions, $e_1$ and $e_2$, we say that $e_1$ is merge dominated by $e_2$, denoted $e_1 \leq e_2$, if $M(e_1, e_2) = \text{true}$ and $\mu(e_1, e_2) = e_2$.
  - $e_1$ does not add information.
- Merged descriptions always dominates the ones they have derived from.
  - $\forall e_1, e_2 \in E$ such that $M(e_1, e_2) = \text{true}$, it holds that $e_1 \leq \mu(e_1, e_2)$ and $e_2 \leq \mu(e_1, e_2)$.
- Match function is monotonic.
  - $\forall e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $M(e_1, e_3) = \text{true}$, then $M(e_2, e_3) = \text{true}$.
- Merge function is monotonic.
  - $\forall e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $M(e_1, e_3)$, then $\mu(e_1, e_3) \leq \mu(e_2, e_3)$.
  - $\forall e_1, e_2, e_3 \in E$ if $e_1 \leq e_3$, $e_2 \leq e_3$ and $M(e_1, e_2) = \text{true}$, then $\mu(e_1, e_2) \leq e_3$. 
A Sequence of Match & Merge Calls

$M(e_1, e_{2,3}) = \text{true}$

$e_1$

$e_2$

$e_3$

$M(e_2, e_3) = \text{true}$

$e_2, e_3 = \mu(e_2, e_3)$

$M(e_1, e_{2,3}) = \text{true}$
Assumes ICAR and merge domination

- **Idea 1**: if $M(e_1,e_2) = true$ we can remove $e_1$ and $e_2$
  - Whatever would match $e_1$ or $e_2$ now also matches $\mu(e_1,e_2)$
  - Representatativity and associativity

- **Idea 2**: Removal of dominated descriptions is not necessary as a last step in the algorithm
  - Assume $e_1$ and $e_2$ appear in final answer where $e_1 \leq e_2$
    - Then $M(e_1,e_2) = true$ and $\mu(e_1,e_2) = e_2$
  - Thus comparison of $e_1$ and $e_2$ should have generated merged description $e_2$, and $e_1$ should have been eliminated
R-Swoosh operates as follows:

- A set $E$ of entity descriptions is initialized to contain all the input descriptions.
- At each iteration, a description $e$ is removed from $E$ and compared to each description $e'$ of the, initially empty, set $E'$.
- If $e$ and $e'$ are found to match, then they are removed from $E$ and $E'$, respectively, and the result of their merging is placed into $E$ (exploiting representatively).
- If there is no description $e'$ matching with $e$, then $e$ is placed in $E'$.
- This process continues until $E$ becomes empty, i.e., there are no more matches to be found.
R-Swoosh Example

\[ E \]
- \( e_1 \)
- \( e_2 \)
- \( e_3 \)

\[ E' \]
- \( e_1 \)
- \( e_2 \)
- \( e_3 \)

(a) \( e_1 \)
(b) \( e_2 \)
(c) \( e_3 \)

(d) \( e_{23} \)
(e) \( e_{123} \)
(f) \( e_{123} \)
ER with ICAR properties

- General ER process is guaranteed to be **finite**
- Entity descriptions can be **matched** and **merged** in any order

- **Dominated** entity descriptions can be discarded anytime

- Union class of match and merge
  - **Union-merge**: All values are kept in merged descriptions
  - **Union-match**: At least one of the values in common

- Commutativity of **match** and **merge** functions for highly heterogeneous descriptions does **not always hold**
The key idea is that each record maintains all the values seen in its base records.

- For example, if a record with name \{John Doe\} is merged with a record with name \{J. Doe\}, the result would have the name \{John Doe, J. Doe\}.

Unioning values is convenient in practice since we record all the variants seen for a person’s name, a hotel’s name, a company’s phone number, and so on.

Keeping the “lineage” of our records is important in many applications, and ensures we do not miss future potential matches.
Matching and Resolving Entities: Matching-based Iterative Resolution
Positive and Negative Match Evidence

Hard constraints are those which we definitely want to be true.

- These might relate to the successful assembly of a mechanism.

Soft constraints are those we would like to be true - but not at the expense of the others.

- These might say that a mechanism must follow a given path.
# Positive and Negative Match Evidence

<table>
<thead>
<tr>
<th></th>
<th>Hard Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td><strong>FD</strong>: if $M(e_i, e_j) = \text{true}$ then $M(e_k, e_l) = \text{true}$</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong>: if two movies match then their director also match</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Soft Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>FD</strong>: if $M(e_i, e_j) = \text{true}$ then most likely $M(e_k, e_l) = \text{true}$</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong>: if two movies match then their actors are most likely to match</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Hard Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td><strong>FD</strong>: if $M(e_i, e_j) = \text{false}$ then $M(e_k, e_l) = \text{false}$</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong>: if two directors don’t match then movies directed by them don’t also match</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Soft Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>FD</strong>: if $M(e_i, e_j) = \text{false}$ then most likely $M(e_k, e_l) = \text{false}$</td>
</tr>
<tr>
<td></td>
<td><strong>Example</strong>: if two movies don’t match then their actors are less likely to match</td>
</tr>
</tbody>
</table>

Negative constraints are usually stated by domain experts.
## Additional Constraints [Shen et al, 2005]

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>No director has produced more than four movies per year</td>
</tr>
<tr>
<td>Subsumption</td>
<td>If a movie X from Yago matches a movie Y from IMBD then each studio cited by Y matches some studio cited by X</td>
</tr>
<tr>
<td>Neighborhood</td>
<td>If actors X and Y share similar names and some co-actors in a movie, they are likely to match</td>
</tr>
<tr>
<td>Incompatible</td>
<td>No director has filmed a movie in studios both in Africa and Japan</td>
</tr>
<tr>
<td>Layout</td>
<td>If two movies with similar names are mentioned by the same review they are likely to match</td>
</tr>
<tr>
<td>Key/Uniqueness</td>
<td>Actors of the same movie must refer to distinct persons</td>
</tr>
<tr>
<td>Ordering</td>
<td>If two paper citations match, then their authors will be matched in order</td>
</tr>
<tr>
<td>Individual</td>
<td>‘Victorina Mérida Rojas’ matches with ‘Victoria Abril’</td>
</tr>
</tbody>
</table>
Matching-based Iterative ER

- **Pair-wise ER**: matching decisions are made independently
  - Deduplication, Linkage

- **Collective ER**: matching decisions depend on other matching decisions according to positive and negative evidence (general constraints)
  - Similarity propagation approaches
    - Dependency graphs of matching decisions
    - Collective relational clustering
  - Probabilistic approaches
  - Hybrids of constraints and probabilistic models
Matching-based Iterative ER

- [Dong et al]: consider a dependency graph whose nodes represent similarities between record pairs, and edges represent dependencies between matching decisions.
  - The graph indicates that, e.g., if two publications match (maybe based on the fact that all their authors match) then their venues must match as well.

- [Relational clustering for entity resolution (RC-ER)]: iteratively create clusters of matched records. In each iteration, clusters that are deemed most similar are merged. The similarity between two clusters is measured based on matching (strings) of the cluster labels, in addition to matching attributes of their related records.
  - Cluster labels and their set of related records can change.
Similarity Propagation Approaches

- A graph structure for encoding the similarity between entity descriptions and matching decisions, and iteratively assess matching of entities by propagating similarity values
  - Details of how the graph is constructed and traversed and how (content and context) similarity is computed vary

  **Similarity-propagation ER**: the match function is re-computed at each iteration step by considering previous matching decisions:
  - \( M^n(e_i, e_j) = \text{true}, \) if \( \text{sim}^{n-1}(e_i, e_j) \geq \vartheta \)
  - \( M^n(e_i, e_j) = \text{false}, \) if \( \text{sim}^{n-1}(e_i, e_j) \leq \vartheta' \)
  - \( M^n(e_i, e_j) = \text{undecided}, \) otherwise

- **Total similarity**:
  \[
  \text{sim}(e_i, e_j) = a \times \text{sim}_{\text{nbr}}(e_i, e_j) + (1-a) \times \text{sim}_{\text{nbr}}(\text{nbr}(e_i), \text{nbr}(e_j))
  \]

  where \( \text{nbr}(e) \) denotes the neighborhood (in, out) nodes of \( e \)
Maintaining the Order of Comparisons

- In similarity-propagation approaches the order of comparisons is dynamic.
- Graph traversal usually supported by a priority queue (PQ) on the similarity score of nodes.
  - As entities are resolved, the PQ is updated for maximizing effectiveness & reducing re-comparisons.
- Different strategies of order maintenance:
  - Based on heuristics
    • type of nodes and edge direction [Dong et al. 2005]
    • degree of nodes [Weis & Naumann 2006]
    • edge weights [Kalashnikov & Mehrotra 2006]
  - Triggered by recent matches [Böhm et al. 2012, Lacoste-Julien et al. 2013]
Dependency Graph [Dong et al., 2005]

- Works on an **entity graph** constructed from the relational records
  - nodes represent similarity comparisons between pairs of records and their attribute values (**real-valued**)
  - edges represent match decisions based on the matching of associated nodes (**boolean-valued**)
- **A matching decision** is taken when the real-valued similarity score (between 0 and 1) of a node is above a threshold $\theta$
  - If it exceeds the threshold, it is marked as **match**, otherwise as **undecided**,
  - if no more neighbors are undecided, it is marked as **non-match**
- Idea 1: consider **richer matching evidence**
- Idea 2: **propagate similarity** between matching decisions
- Idea 3: Gradually **enrich references** by merging attribute values (Swoosh-style)
Let $E$ be a set of entity descriptions
- A node $v = \{e_i, e_j\}$, where $e_i, e_j \in E$, $i \neq j$
- An edge $e = (v_a, v_b)$ from $v_a = \{e_{ai}, e_{aj}\}$ to $v_b = \{e_{bi}, e_{bj}\}$ implies $e_{bi}, e_{bj} \in values(e_{ai}) \cup values(e_{aj})$

Directed edge when dependency is only in one direction
- can be stated by domain experts
- inferred by the data semantics (e.g., keys/foreign keys, rdf properties)

Include only nodes whose two entities have the potential to be similar
Consider Richer Matching Evidence

- **Positive evidence** (i.e., constraints for match nodes) is captured by the **Boolean similarity** of neighborhood nodes
  - **Strong-boolean**: Resolution implies resolution of neighbour
    - E.g., if two movies are matched then director must also be matched
  - **Weak-boolean**: No direct implication
    - E.g., similarity of two movies increases as their rdf:labels are highly similar

- **Negative evidence** (i.e., constraints for non-match nodes) is verified after similarity propagation is performed, and inconsistencies are fixed
Similarity Propagation

- **Similarity function** for node $u$: $sim(u) = S_{rv} + S_{sb} + S_{wb}$
  - $S_{rv}$: from **real-valued** neighbors
  - $S_{sb}$: from **strong-boolean-valued** neighbors
  - $S_{wb}$: from **weak-boolean-valued** neighbors

- When a node is matched, the similarity score of its neighbors is **re-computed**

- Compute neighbor similarities only if similarity increase is **not too small**
Traversing the ER Graph

Initially all nodes are active and placed in the PQ
A node is processed before its out-neighbors
Traversing the ER Graph

- a
- b
- c
- d
- e
- f
- g
- h

merged node
inactive node
Traversing the ER Graph

a ➔ b ➔ c ➔ d ➔ e

f ➔ g ➔ h ➔ i

PQ
- c
- h
- d
- e
- g
Traversing the ER Graph

[ER Graph Diagram with nodes labeled a, b, c, d, e, h, g, f, and PQ column with h, d, e, g]
Traversing the ER Graph

a → b → c → d → h → g → f

PQ
- d
- e
- g
Traversing the ER Graph
Traversing the ER Graph

PQ
Traversing the ER Graph

- a
- b
- c
- d
- e
- f
- g
- h
- PQ
  - c

Correct paths:
- a → b → c → d
- a → b → e
- a → b → h
- a → h
- g → f

Incorrect paths:
- b → e
- c → e
- d → e
- a → g
- h → d
Traversing the ER Graph
Linda [Böhm et al. 2012]  
- Works on an entity graph constructed from the RDF descriptions
- **Key Idea**: the more matching neighbors via similar relationships two descriptions have, the more likely it is that they match
  - **String similarity** of the literal values of entities: checked once
  - **Contextual similarity** of the graph neighbors: checked iteratively
- Two square matrices (|E| \times |E|) are used:
  - X captures the identified matches (binary values)
  - Y captures the pair-wise similarities (real values)
    - Initialization: common neighbors & string similarity of literals
    - Updates: Use the new identified matches of X
- Until the priority queue (extracted from Y) becomes empty:
  - Get the pair \((e_i, e_j)\) with the **highest similarity**: match by default!
    - Update X: matches of \(e_i\) are also matches of \(e_j\)
    - Update the similarity of nodes influenced by the new matches
Linda Algorithm Example

A priority queue, derived by an initial similarity computation between all pairs, based on their attribute values.
### Linda Algorithm Example

#### Matches

<table>
<thead>
<tr>
<th></th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
<th>e5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>e5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### PQ

- e1 – e4
- e2 – e4
- e1 – e3
- e5 – e3
- e2 – e3
- ...

The head of PQ is a match by default.
**Linda Algorithm Example**

<table>
<thead>
<tr>
<th>Matches</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
<th>e5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e2</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e3</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**PQ**

| e2 → e4  |
| e1 → e3  |
| e2 → e3  |
| e5 → e3  |
| ...      |

Unique mapping constraint (1-1 Assumption)

Similarity recomputation, based on the matching neighbors and the names of the links to them.
Linda Algorithm Example

<table>
<thead>
<tr>
<th>Matches</th>
<th>e1</th>
<th>e2</th>
<th>e3</th>
<th>e4</th>
<th>e5</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>e2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PQ
- e2 – e3
- e5 – e3
- ...

The diagram represents the relationships between the elements e2, e3, e4, and e5, showing the `directs` relationships.
Linda Algorithm Example

Matches | e1 | e2 | e3 | e4 | e5
---|---|---|---|---|---
e1 | 1 | 0 | 0 | 1 | 0
e2 | 1 | 1 | 0 | 0 | 0
e3 | 1 | 0 | 0 | 0 | 0
e4 | 1 | 0 | 0 | 0 | 0
e5 | 1 | 0 | 0 | 0 | 0

PQ
- e5 — e3
- ...

unique mapping constraint (1-1 Assumption)

directs

e3 directs e4

e2 directs e1

e5 stops when PQ is empty
Matching and Resolving Entities: Progressive Resolution Techniques
ER traditionally seen as a pre-processing step prior to data analysis.

Such an offline strategy is not suitable for many emerging applications that require (near) real-time analysis, especially if it involves big, fast, or streaming data.
Progressive ER

Extend the typical ER workflow with a planning phase, which is responsible for selecting which pairs of descriptions, that have resulted from blocking, will be compared in the entity matching phase and in what order.

The goal of this new phase is to favour more promising comparisons, i.e., those that are more likely to result in matches.

This way, those comparisons are executed before less promising ones and thus, more matches are identified early on in the process.

An (optional) update phase propagates the results of matching, such that a new scheduling phase will promote the comparison of pairs that were influenced by the previous matches.

This iterative process continues until the pre-defined computing budget is consumed.
Progressive ER

Optimization: maximize benefit (number or type of matches) for a given cost (number of comparisons, disk/cloud access)

- Blocking
- Planning
- Matching
- Update

Progressive ER: estimates which part of the data to resolve next and adapts this decision in a pay as you go fashion

- Good for high Velocity
Progressive Approach to Relational Entity Resolution [Altowim et al. 2014]

- **Key Idea**: divides the ER process into several *windows* and generates a *resolution plan* for each window
  - specifies *which blocks* and *entity pairs* within these blocks will be resolved during the plan execution phase of a window
  - associates with each identified pair the *order* in which to apply the similarity functions on the attributes of the two entities

- **Lazy resolution strategy** to resolve pairs with the smallest cost
  - Unlike single entity type resolution a *block based prioritization* is significantly more important when resolving *multiple types*. 
Progressive Relational ER

- Blocks with entities of the same type

- Nodes: Pairs of entity descriptions of the same type (relation)

- Edges: Dependency between pairs (foreign keys) - an edge indicates that the resolution of a node influences the resolution of another node.
Progressive Relational ER

Black-box blocking phase

- Avoid building a dependency graph with all the description pairs

Scheduling phase: divide the total cost budget into several windows of equal cost

- For each window, a comparison schedule is generated
- Choose among the schedules whose cost does not exceed the current window, the one with the highest expected benefit
- The cost of a schedule is computed by considering the cost of finding the description pairs in a block according to the available storage policy (in memory/disk/cloud), and the cost of resolving every description pair
Progressive Relational ER

Schedule benefit:

○ How many matches are expected to be found by this schedule – *direct benefit*

○ How useful it will be to declare those nodes as matches, in identifying more matches within the cost budget – *indirect benefit*

*A node is more likely to be a match, when it is influenced by more matching nodes, and it is more influential, when it is expected to be a match and it has many direct dependent nodes*
Progressive Relational ER

Update phase

- After schedule execution: matching decisions are propagated to all influenced nodes, whose expected benefit now increases and have, thus, higher chances of being chosen by the next schedule

The algorithm *terminates* when the cost budget has been reached

- All unresolved pairs are considered non-matches – statistically, matches are significantly fewer than non-matches
Relational progressive ER algorithms assume that the more entity pairs are correctly identified, the higher the quality of the result is expected to be.

We are interested in characterizing the quality of resolved pairs:
- # of real-world entities resolved (*shallow strategy*)
  - entity-centric search (*entity coverage*)
- # of real-world entity graphs resolved (*deep strategy*)
  - entity-centric recommendation (*relationship completeness*)
- # descriptions resolved for the same real-world entity (*tail strategy*)
  - web-scale knowledge curation (*attribute completeness*)
Questions ?