Generic Entity Resolution with Swoosh

20.6.2013
Felix Naumann

With slides from Johannes Dyck and Steven Whang
The Stanford SERF Project

- Stanford Entity Resolution Framework (SERF)
  - Generic infrastructure for *Entity Resolution*
- Idea: "match" and "merge" are black-boxes
  - Makes ER resemble a database self-join operation (of the initial set of records with itself),
  - But: No knowledge about which records may match, so all pairs of records need to be compared
  - But: Merged records may lead us to discover new matches,
- Protagonists
  - Omar Benjelloun
  - Steven Euijong Whang
  - Hector Garcia-Molina
  - And more
- [http://infolab.stanford.edu/serf/](http://infolab.stanford.edu/serf/)
Overview

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh
Taxonomy of Deduplication Algorithms

- Pairwise decisions vs. clustering
  - Easier to write pairwise decisions
- Schema differences vs. same schema
  - Bag of tokens approach for unaligned schemata
- Relationships vs. individual records
  - Joint entity resolution
- Exact vs. approximate
  - Binary decision, no probability for match
  - No confidence values
- Generic vs. application specific
  - Decisions through similarity measure are abstracted
  - Black box
Overview

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh
Intuitive example

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>{John Doe}</td>
<td>{235-2635}</td>
</tr>
<tr>
<td>r₂</td>
<td>{J. Doe}</td>
<td>{234-4358}</td>
</tr>
<tr>
<td>r₃</td>
<td>{John D.}</td>
<td>{234-4358}</td>
</tr>
</tbody>
</table>

- Similarity function
  - Match if similar Name OR same Phone and E-Mail
  - Name is "feature" and Phone + E-Mail is "feature"
- Step 1: r₁ and r₂ match
- Step 2: Merge r₁ and r₂ to new r₄
  
  \[
  r₄ = \{\text{John Doe}\} \quad \{234-4358, 235-2635\} \quad \{jdoe@yahoo\}
  \]
- Step 3: Now r₃ and r₄ match
- Each merged record must be re-compared to all other records
- Swoosh is an exhaustive approach: No partitioning
Notation

- Domain \( R \)
- Instance \( I = \{r_1,...,r_n\} \) finite set of records from \( R \)
- Match function \( M: R \times R \rightarrow \text{Boolean} \)
  - \( M(r,s) = \text{true} \) iff \( r \) and \( s \) represent same real-world entity
  - No confidence
  - No dependency on data outside of \( r \) and \( s \)
  - Notation: \( r \approx s \) iff \( M(r,s) = \text{true} \)
- Merge function \( m: R \times R \rightarrow R \)
  - Defined only for matching records
  - Notation \( m(r,s) = \langle r,s \rangle \)
Given instance I, the merge closure of I, denoted \( \hat{I} \), is the smallest set of records \( S \), such that

- \( I \subseteq S \)
- For any \( r, s \): If \( r \approx s \) then \( <r,s> \in S \)

Intuition: Extend I with all records that can be created by matching and merging

Properties

- \( \hat{I} \) exists and is unique
- \( \hat{I} \) can be infinite
  - Unrealistic in practice
Domination

- Record $r$ is dominated by $s$ if $r \approx s$ and $s$ holds more information
  - $r \preceq s$
  - Any partial order on records
    - Reflexive, transitive
    - Antisymmetric: if $r \preceq s$ and $s \preceq r$, then $r = s$
- Examples: $r_1 \preceq r_4$ and $r_2 \preceq r_4$
- Application-dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>{JohnDoe}</td>
<td>{jdoe@yahoo}</td>
</tr>
<tr>
<td>$r_2$</td>
<td>{J.Doe}</td>
<td>{234-4358}</td>
</tr>
<tr>
<td>$r_3$</td>
<td>{JohnD.}</td>
<td>{jdoe@yahoo}</td>
</tr>
<tr>
<td>$r_4$</td>
<td>{John Doe}</td>
<td>{234-4358, 235-2635, jdoe@yahoo}</td>
</tr>
</tbody>
</table>
Instance domination

- Given instances $I_1$ and $I_2$, $I_1$ is dominated by $I_2$ ($I_1 \preceq I_2$) if for all $r_1 \in I_1$ there exists an $r_2 \in I_2$ such that $r_1 \preceq r_2$.
  - Reflexive
  - Transitive
  - Not antisymmetric: If $r_1 \preceq r_2$ then
    - $\{r_2\} \preceq \{r_1, r_2\}$ and $\{r_1, r_2\} \preceq \{r_2\}$
Given an instance $I$, an entity resolution of $I$ (ER($I$)) is a set of records $I'$ that satisfies the following conditions:

1. $I' \subseteq \hat{I}$
2. $\hat{I} \preceq I'$
3. No strict subset of $I'$ satisfies conditions 1 and 2.

Reminder: $\hat{I}$ is merge closure

- Condition 1: Cannot produce more than $\hat{I}$
- Condition 2: Produce at least all information of $\hat{I}$
- Condition 3: Minimal solution
What is best sequence of match, merge calls that give us right answer?
Brute Force Algorithm

- **Input R:**
  - r1 = [a:1, b:2]
  - r2 = [a:1, c:4, e:5]
  - r3 = [b:2, c:4, f:6]
  - r4 = [a:7, e:5, f:6]
  - r12 = [a:1, b:2, c:4, e:5]

- **Match all pairs:**
  - r1 = [a:1, b:2]
  - r2 = [a:1, c:4, e:5]
  - r3 = [b:2, c:4, f:6]
  - r4 = [a:7, e:5, f:6]
  - r12 = [a:1, b:2, c:4, e:5]
  - r123 = [a:1, b:2, c:4, e:5, f:6]

**Note:** Redundant comparisons, such as M(r3, r4)

**Note:** Redundant records, such as r1 and r2
ICAR properties

- **Idempotence**: ∀r, r ≈ r and ⟨r, r⟩ = r.
  - A record always matches itself, and merging it with itself still yields the same record.

- **Commutativity**: ∀r, s: r ≈ s iff s ≈ r,
  - and if r ≈ s, then ⟨r, s⟩ = ⟨s, r⟩.
  - Direction of match and merge is irrelevant

- **Associativity**: ∀r₁, r₂, r₃ such that ⟨r₁, ⟨r₂, r₃⟩⟩ and ⟨⟨r₁, r₂⟩, r₃⟩ exist, then ⟨r₁, ⟨r₂, r₃⟩⟩ = ⟨⟨r₁, r₂⟩, r₃⟩.
  - Order of merge is irrelevant

- **Representativity**: If r₃ = ⟨r₁, r₂⟩ then for any r₄ such that r₁ ≈ r₄, we also have r₃ ≈ r₄.
  - r₃ “represents” r₁ and r₂.
  - Merging does not lose matches; no “negative evidence”

- **Transitivity** is **not** assumed: r ≈ s and s ≈ t does not imply r ≈ t.
Merge domination

- When the match and merge functions satisfy the ICAR properties, there is a natural domination order.
  - Before “domination” was only informal.

- Given two records, \( r_1 \) and \( r_2 \), we say that \( r_1 \) is merge dominated by \( r_2 \), denoted \( r_1 \leq r_2 \), if \( r_1 \approx r_2 \) and \( <r_1, r_2> = r_2 \).
  - \( r_1 \) does not add information.
Monotonicity

- For any records \( r_1, r_2 \) such that \( r_1 \approx r_2 \), it holds that \( r_1 \leq <r_1, r_2> \) and \( r_2 \leq <r_1, r_2> \)
  - Merge record always dominates the records it was derived from
- If \( r_1 \leq r_2 \) and \( r_1 \approx r \), then \( r_2 \approx r \)
  - Match function is monotonic
- If \( r_1 \leq r_2 \) and \( r_1 \approx r \), then \( <r_1, r> \leq <r_2, r> \)
  - Merge function is monotonic
- If \( r_1 \leq s, r_2 \leq s \) and \( r_1 \approx r_2 \), then \( <r_1, r_2> \leq s \).
ER with ICAR properties

- ER process is guaranteed to be finite
- Records can be matched and merged in any order
- Dominated records can be discarded anytime

Union match and merge

- Union-merge: All values are kept in merged record
  - Keeps data lineage, ensures that we do not miss future matches
  - Presentation to user or app my do some actual fusion
  - Alternative for numbers: Keep range
- Union-match: At least one values is in common
- ICAR properties hold
  - Idempotence, Commutativity, Associativity, Representativity
Overview

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh
Naive Breadth First

1: **input**: a set \( I \) of records
2: **output**: a set \( I' \) of records, \( I' = ER(I) \)
3: \( I' \leftarrow I; N \leftarrow \emptyset \)
4: **repeat**
5: \( I' \leftarrow I' \cup N; N \leftarrow \emptyset \)
6: **for all** pairs \( (r, r') \) of records in \( I' \) **do**
7: \( \text{if } r \approx r' \text{ then} \)
8: \( \text{merged } \leftarrow \langle r, r' \rangle \)
9: \( \text{if } \text{merged} \notin I' \text{ then} \)
10: \( \text{add } \text{merged} \text{ to } N \)
11: **end if**
12: **end if**
13: **end for**
14: **until** \( N = \emptyset \)
15: **for all** pairs \( (r, r') \) of records in \( I' \) where \( r \neq r' \) **do**
16: \( \text{if } r' \leq r \text{ then} \)
17: \( \text{Remove } r' \text{ from } I' \)
18: **end if**
19: **end for**

Continue as long as duplicates are found
Naive Breadth First

- 4 rounds
- Last round finds nothing
- 3rd round on 8 records
- Many unnecessary comparisons
  - M(r4,r5) computed four times
- G-Swoosh avoids this redundancy
All records in \( I' \) have been compared with one another.

Iteratively move records from \( I \) to \( I' \). If matched place merged record into \( I \).
G-Swoosh Example

1. \( I = 1,2,3,4,5 \) \( I' = {} \)
2. Compare 1 with each \( I' \)
   \( I = 2,3,4,5 \) \( I' = 1 \)
3. Compare 2 with each \( I' \)
   \( I = 3,4,5,12 \) \( I' = 1,2 \)
4. \( I = 4,5,12,23 \) \( I' = 1,2,3 \)
5. \( I = 5,12,23 \) \( I' = 1,2,3,4 \)
6. \( I = 12,23 \) \( I' = 1,2,3,4,5 \)
7. \( I = 23 \) \( I' = 1,2,3,4,5,12 \)
8. \( I = 123 \) \( I' = 1,2,3,4,5,12,23 \)
9. \( I = 1235 \) \( I' = 1,2,3,4,5,12,23,123 \)
10. \( I = {} \) \( I' = 1,2,3,4,5,12,23,123,1235 \)
G-Swoosh discussion

1: input: a set $I$ of records
2: output: a set $I'$ of records, $I' = ER(I)$
3: $I' \leftarrow \emptyset$
4: while $I \neq \emptyset$ do
5: $r \leftarrow$ a record from $I$
6: remove $r$ from $I$
7: for all records $r'$ in $I' \cup \{r\}$ do
8: if $r \approx r'$ (resp. $r' \approx r$) then
9: merged $\leftarrow \langle r, r' \rangle$ (resp. $\langle r', r \rangle$)
10: if merged $\notin I \cup I' \cup \{r\}$ then
11: add merged to $I$
12: end if
13: end if
14: end for
15: add $r$ to $I'$
16: end while
17: Remove dominated records from $I'$ (See lines 15–18 in BFA)
18: return $I'$

Idempotency: $\cup \{r\}$ not needed
Commutativity: $r' \approx r$ not needed
Commutativity: $\langle r', r \rangle$ not needed
Without ICAR properties, G-Swoosh is optimal in number of match-calls.
Overview

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh
Assumes ICAR and merge domination

- Reminder: $r_1$ is merge dominated by $r_2$, denoted $r_1 \leq r_2$, if $r_1 \approx r_2$ and $<r_1, r_2> = r_2$

Idea 1: If $r_1 \approx r_2$ we can remove $r_1$ and $r_2$

- Whatever would match $r_1$ or $r_2$ now also matches $<r_1, r_2>$
- Representativity and associativity

Idea 2: Removal of dominated records (last step in algorithm) not necessary.

- Assume $r_1$ and $r_2$ appear in final answer and $r_1 \leq r_2$. Then $r_1 \approx r_2$ and $<r_1, r_2> = r_2$.
- Thus comparison of $r_1$ and $r_2$ should have generated merged record $r_2$, and $r_1$ should have been eliminated.
1: **input**: a set $I$ of records /* Initialization */
2: **output**: a set $I'$ of records, $I' = ER(I)$
3: $I' \leftarrow \emptyset$
4: **while** $I \neq \emptyset$ /* Main loop */
5: \text{currentRecord} \leftarrow \text{a record from } I$
6: remove \text{currentRecord} from $I$
7: \text{buddy} \leftarrow \text{null}$
8: \text{for all} records $r'$ in $I'$ \text{do}
9: \text{if } $M(\text{currentRecord}, r') = \text{true}$ \text{then}
10: \text{buddy} \leftarrow r'$
11: \text{exitfor}$
12: \text{end if}$
13: \text{end for}$
14: \text{if } \text{buddy} = \text{null} \text{then}
15: \text{add } \text{currentRecord} \text{ to } I'$
16: \text{else}$
17: \text{remove } \text{buddy} \text{ from } I'$
18: \text{add } r'' \text{ to } I$
19: \text{end if}$
20: \text{end while}$
21: \text{return } I'$

In case of match, no further comparisons
As before
Add merged record to $I$ and remove both original records
R-Swoosh Example

1. $I = 1,2,3,4,5$  \( \Gamma' = \{\} \)
2. $I = 2,3,4,5$  \( \Gamma' = 1 \)
3. $I = 3,4,5,12$  \( \Gamma' = \{\} \)
4. $I = 4,5,12$  \( \Gamma' = 3 \)
5. $I = 5,12$  \( \Gamma' = 3,4 \)
6. $I = 12$  \( \Gamma' = 3,4,5 \)
7. $I = 123$  \( \Gamma' = 4,5 \)
8. $I = \{\}$  \( \Gamma' = 4,1235 \)

- Fewer iterations
- Fewer comparisons per iteration

- Further improvement: Order records intelligently, if possible
  - Achieve early matches
If ICAR Properties Do Not Hold?

r1: [Joe Sr., 123 Main, DL:X]

r2: [Joe, 123 Main, Ph:123]

r3: [Joe Jr., 123 Main, DL:Y]

r12: [Joe Sr., 123 Main, Ph: 123, DL:X]

r23: [Joe Jr., 123 Main, Ph: 123, DL:Y]

Full Answer: \( \text{ER}(R) = \{r12, r23, r1, r2, r3\} \)

Minus Dominated: \( \text{ER}(R) = \{r12, r23\} \)

R-Swoosh Yields: \( \text{ER}(R) = \{r12, r3\} \) or \( \{r1, r23\} \)
Overview

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh
F-Swoosh – Idea

- R-Swoosh saves record comparisons
- F-Swoosh saves feature comparisons
  - $M(r_1,r_3)$: Compare "JohnDoe" with "JohnD."
  - $<r_1,r_3> = r_4$
  - $M(r_3,r_4)$: Compare "JohnDoe" with "JohnD." again.
- Different records may have common values
  - (Expensive) comparisons are performed redundantly

<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Phone</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>{JohnDoe}</td>
<td>{235-2635}</td>
<td>{jdoe@yahoo}</td>
</tr>
<tr>
<td>$r_2$</td>
<td>{J.Doe}</td>
<td>{234-4358}</td>
<td></td>
</tr>
<tr>
<td>$r_3$</td>
<td>{JohnD.}</td>
<td>{234-4358}</td>
<td>{jdoe@yahoo}</td>
</tr>
<tr>
<td>$r_4$</td>
<td>{John Doe}</td>
<td>{234-4358, 235-2635}</td>
<td>{jdoe@yahoo}</td>
</tr>
</tbody>
</table>
Preliminaries

- Positive comparisons: Sufficiently similar
- Negative comparisons: Not sufficiently similar
- Avoid repeating both kinds

Idea

- Break down match function into multiple feature comparisons
  - Feature can be one or multiple attribute values
  - Two records match if one or more features map: Disjunction of feature matches
    - This makes keeping track easy!
- Keep track of encountered values and avoid comparing them twice
F-Swoosh Algorithm

- Same pattern as R-Swoosh: Iteratively build I'.
- Hash tables for previously seen features
  - Hash table Pf: For each value store pointer to the record r that currently "represents" the value.
    - Either first record where feature value appeared for feature f
    - Or record that was derived from it through a sequence of merge steps
    - Can be only one record, otherwise records would have been merged
    - Update on each encounter of value
  - Hash table Nf: For each feature the set of values that were compared against all of I' and did not match
    - Representativity: If feature value of current record is in Nf, then no comparison is necessary.
- Size: Linear in num values
  - Not quadratic to store all comparisons
Further Swooshs

- Incremental F-Swoosh
  - Idea: Keep around hash tables. No old data will be re-compared

- D-Swoosh
  - Distributed ER
Summary

- ER Classification
- Fundamentals
- Naive Algorithms
- R-Swoosh
- F-Swoosh