Generic Entity Resolution: Identifying Real-World Entities in Large Data Sets

Steven Whang
Stanford University

Work with: Hector Garcia-Molina, David Menestrina, Omar Benjelloun, Qi Su, Jennifer Widom, Tyson Condie, Nicolas Pombourcq, Tait Larson, Johnson Gong, Makoto Tachibana, Suithipong Thavisomboon, Georgia Koutrika, Martin Theobald

Applications
- comparison shopping
- mailing lists
- classified ads
- customer files
- counter-terrorism

Outline
- Why is ER challenging?
- How is ER done?
- Some ER work at Stanford
  - Generic ER
  - Distributed ER
  - Iterative Blocking

Challenges (1)
- No keys!
- Value matching
  - “Kaddafi”, “Qaddafi”, “Kadafi”, “Kaddaffi”...
- Record matching

Challenges (2)
- Merging records

Entity Resolution

Applications

Outline

Challenges (1)

Challenges (2)
Challenges (3)

- Chaining
  - Nm: Tom
  - Ad: 123 Main
  - Bd: Jan 1, 85
  - Wk: IBM
  - Oc: lawyer

- Nm: Thomas
  - Ad: 123 Main
  - Bd: Jan 1, 85
  - Wk: IBM
  - Oc: lawyer
  - Sal: 500K

- Nm: Tom
  - Ad: 123 Main
  - Bd: Jan 1, 85
  - Wk: IBM
  - Oc: lawyer

Challenges (4)

- Un-merging
  - Nm: Tom
  - Ad: 123 Main
  - Bd: Jan 1, 85
  - Wk: IBM
  - Oc: lawyer
  - Sal: 500K

  too young to make 500K at IBM!!

Challenges (5)

- Confidences in data
  - Nm: Tom (0.9)
  - Ad: 123 Main St (1.0)
  - Ph: (650) 555-1212 (0.6)
  - Ph: (650) 777-7777 (0.8)

- In value matching, match rules, merge:
  - conf = ?

Taxonomy

- Pairwise snaps vs. clustering
- De-duplication vs. fidelity enhancement
- Schema differences
- Relationships
- Exact vs. approximate
- Generic vs application specific
- Confidences

Schema Differences

Name: Tom
- Address: 123 Main St
- Ph: (650) 555-1212

FirstName: Tom
- StreetName: Main St
- StreetNumber: 123
- Tel: (650) 777-7777

Pair-Wise Snaps vs. Clustering

1  2  3  4  5  6  7  8  9  10

1  2  3  4  5  6  7  8  9  10
De-Duplication vs. Fidelity Enhancement

Relationships

Exact vs Approximate ER

Generic vs Application Specific

Taxonomy

Outline

Why is ER challenging?
• How is ER done?
• Some ER work at Stanford
  • Generic ER
  • Distributed ER
  • Iterative Blocking

Match function $M(r, s)$
Merge function $<r, s> \Rightarrow t$
**Taxonomy**

- Pairwise snaps vs. clustering
- De-duplication vs. fidelity enhancement
- Schema differences No
- Relationships No
- Exact vs. approximate
- Generic vs application specific
- Confidences No

---

**Model**

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nm: Tom</td>
<td>Nm: Thomas</td>
<td>Nm: Tom</td>
</tr>
<tr>
<td>Ad: IBM</td>
<td>Ad: IBM</td>
<td>Ad: IBM</td>
</tr>
<tr>
<td>Oc: Laywer</td>
<td>Oc: Laywer</td>
<td>Oc: Laywer</td>
</tr>
<tr>
<td>BD: Jan 1, 85</td>
<td>BD: Jan 1, 85</td>
<td>BD: Jan 1, 85</td>
</tr>
</tbody>
</table>

**Correct Answer**

- ER(R) = All derivable records....
- Minus “dominated” records

---

**Question**

- 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, c:5, f:6]

---

**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, c:5, f:6]

- 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, c:5, f:6]
  - r12 = [a:1, b:2, c:4, e:5]
**ICAR Properties**

- **Idempotence:**
  \[-M(r_1, r_1) = true; <r_1, r_1> = r_1\]
- **Commutativity:**
  \[-M(r_1, r_2) = M(r_2, r_1)\]
  \[-<r_1, r_2> = <r_2, r_1>\]
- **Associativity**
  \[-<r_1, <r_2, r_3>> = <<r_1, r_2>, r_3>\]

**ICAR Properties \(\Rightarrow\) Efficiency**

- **Commutativity**
- **Idempotence**
- **Associativity**
- **Representativity**

- Can discard records
- ER result independent of processing order
Swoosh Algorithms

- **R-Swoosh**
  - Merges records as soon as they match
  - Optimal in terms of record comparisons

- **F-Swoosh**
  - Remembers values seen for each feature
  - Avoids redundant value comparisons

---

**R-Swoosh**

<table>
<thead>
<tr>
<th>R</th>
<th>R'</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>r1</td>
</tr>
<tr>
<td>r3</td>
<td>r2</td>
</tr>
<tr>
<td>r4</td>
<td>r3</td>
</tr>
</tbody>
</table>

M(r1, r2) = true
M(r12, r3) = true

---

**R-Swoosh**

<table>
<thead>
<tr>
<th>R</th>
<th>R'</th>
</tr>
</thead>
<tbody>
<tr>
<td>r12</td>
<td>r1</td>
</tr>
<tr>
<td>r3</td>
<td>r12</td>
</tr>
<tr>
<td>r4</td>
<td>r3</td>
</tr>
</tbody>
</table>

M(r1, r2) = true
M(r12, r3) = true

---

**R-Swoosh**

<table>
<thead>
<tr>
<th>R</th>
<th>R'</th>
</tr>
</thead>
<tbody>
<tr>
<td>r3</td>
<td>r12</td>
</tr>
<tr>
<td>r4</td>
<td>r12</td>
</tr>
</tbody>
</table>

M(r1, r2) = true
M(r12, r3) = true

---

**R-Swoosh**

<table>
<thead>
<tr>
<th>R</th>
<th>R'</th>
</tr>
</thead>
<tbody>
<tr>
<td>r123</td>
<td>r1</td>
</tr>
<tr>
<td>r4</td>
<td>r123</td>
</tr>
<tr>
<td>r4</td>
<td>r4</td>
</tr>
</tbody>
</table>

M(r1, r2) = true
M(r12, r3) = true
R-Swoosh

![Image 1](image1)

Scalability

![Image 2](image2)

If ICAR Properties Do Not Hold?

![Image 3](image3)

Full Answer: \( ER(R) = \{r_{12}, r_{23}, r_1, r_2, r_3\} \)

Minus Dominated: \( ER(R) = \{r_{12}, r_{23}\} \)

R-Swoosh Yields: \( ER(R) = \{r_{12}, r_3\} \) or \( \{r_1, r_{23}\} \)
Outline

- Why is ER challenging?
- How is ER done?
- Some ER work at Stanford
  - Generic ER
  - Distributed ER
  - Iterative Blocking

Partition Input Data

Generic ER Summary

- Entity resolution is critical
- Generic approach yields reusable techniques
- Efficient resolution is important

Partition Input Data
Partition Input Data

R_1  R'_1
r1 r4
r2 r5
r3 r6

R_2  R'_2
r1 r4
r2 r5
r3 r6

R_3  R'_3
r1 r4
r2 r5
r3 r6

Redundant work!

Scope and Responsible

- **scope(r)** – Returns list of processors that receive r
- **resp(p_k, r, r_j)** – true if processor p_k should perform comparison of r_i and r_j

**Coverage property:**
For any pair of matching records r, r', there exists at least one processor P_k such that P_k \in scope(r) \cap scope(r') and resp(P_k, r, r') = true.

D-Swoosh

R_1  R'_1
r1 r3
r4 r6

R_2  R'_2
r1 r2
r4 r5

R_3  R'_3
r2 r3
r5 r6

D-Swoosh

R_1  R'_1
r1 r3
r4 r6

R_2  R'_2
r1 r2
r4 r5

R_3  R'_3
r2 r3
r5 r6
Defining *scope and resp*

- What are good *scope* and *resp* functions?
- How do different functions compare and scale?
- How can we exploit semantic knowledge?
Distributed ER Summary

- Entity Resolution is fundamentally expensive
- Reduce processing time by:
  - Distributing data
  - Eliminating redundant work
- ER benefits greatly from distributed computing

Outline

- Why is ER challenging?
- How is ER done?
- Some ER work at Stanford
  - Generic ER
  - Distributed ER
  - Iterative Blocking

Entity Resolution

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Zip</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>John Doe</td>
<td>12345</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₂</td>
<td>John Doe</td>
<td>94305</td>
<td></td>
</tr>
<tr>
<td>r₃</td>
<td>J. Foe</td>
<td>94305</td>
<td>jdoe@yahoo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Zip</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>John Doe</td>
<td>12345</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₂</td>
<td>John Doe</td>
<td>94305</td>
<td></td>
</tr>
<tr>
<td>r₃</td>
<td>J. Foe</td>
<td>94305</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₁₂</td>
<td>John Doe</td>
<td>[12345, 94305]</td>
<td>jdoe@yahoo</td>
</tr>
</tbody>
</table>
Entity Resolution

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Zip</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>John Doe</td>
<td>12345</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₂</td>
<td>John Doe</td>
<td>94305</td>
<td></td>
</tr>
<tr>
<td>r₃</td>
<td>J. Foe</td>
<td>94305</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₁₂</td>
<td>John Doe</td>
<td>{12345, 94305}</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₁₂₃</td>
<td>John Doe, J. Foe</td>
<td>{12345, 94305}</td>
<td>jdoe@yahoo</td>
</tr>
</tbody>
</table>

ER Solution: {r₁₂₃}

Blocking

<table>
<thead>
<tr>
<th>Record</th>
<th>Name</th>
<th>Zip</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁</td>
<td>John Doe</td>
<td>12345</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₂</td>
<td>John Doe</td>
<td>94305</td>
<td></td>
</tr>
<tr>
<td>r₃</td>
<td>J. Foe</td>
<td>94305</td>
<td>jdoe@yahoo</td>
</tr>
<tr>
<td>r₁₂</td>
<td>John Doe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r₁₂₃</td>
<td>John Doe, J. Foe</td>
<td>{12345, 94305}</td>
<td>jdoe@yahoo</td>
</tr>
</tbody>
</table>

Blocking Solution: {r₁₂, r₃}

Partition by Block#1 Block#2
zip       r₁     r₂, r₃
1st char last name r₁₂    r₃

Partition by Block#1 Block#2
zip       r₁     r₂, r₃
1st char last name r₁₂    r₃
### Iterative Blocking

#### Record | Name    | Zip    | Email
---|---------|--------|--------
$r_1$ | John Doe | 12345  | jdoe@yahoo
$r_2$ | John Doe | 94305  | 
$r_3$ | J. Foe   | 94305  | jdoe@yahoo

**Partition by**: 
- zip: $r_1, r_2, r_3$
- 1st char last name: $r_1, r_2, r_3$

### Iterative Blocking

#### Record | Name    | Zip    | Email
---|---------|--------|--------
$r_1$ | John Doe | 12345  | jdoe@yahoo
$r_2$ | John Doe | 94305  | 
$r_3$ | J. Foe   | 94305  | jdoe@yahoo

**Partition by**: 
- zip: $r_1, r_2, r_3$
- 1st char last name: $r_1, r_2, r_3$

### Iterative Blocking

#### Record | Name    | Zip    | Email
---|---------|--------|--------
$r_1$ | John Doe | 12345  | jdoe@yahoo
$r_2$ | John Doe | 94305  | 
$r_3$ | J. Foe   | 94305  | jdoe@yahoo

**Partition by**: 
- zip: $r_1, r_2, r_3$
- 1st char last name: $r_1, r_2, r_3$

### Iterative Blocking

#### Record | Name    | Zip    | Email
---|---------|--------|--------
$r_1$ | John Doe | 12345  | jdoe@yahoo
$r_2$ | John Doe | 94305  | 
$r_3$ | J. Foe   | 94305  | jdoe@yahoo

**Partition by**: 
- zip: $r_1, r_2, r_3$
- 1st char last name: $r_1, r_2, r_3$

### Iterative Blocking

#### Record | Name    | Zip    | Email
---|---------|--------|--------
$r_1$ | John Doe | 12345  | jdoe@yahoo
$r_2$ | John Doe | 94305  | 
$r_3$ | J. Foe   | 94305  | jdoe@yahoo

**Partition by**: 
- zip: $r_1, r_2, r_3$
- 1st char last name: $r_1, r_2, r_3$

**Iterative Blocking Solution**: \{r\_{123}\}
Overview

- Model
- Algorithms
- Experimental Results

Iterative Blocking Model

- Result is the fixed-point state of applying a "core" ER algorithm on blocks and redistributing new records
- Can plug in any core ER algorithm that partitions records

In-memory Algorithm (Lego)

- Maintains “maximal records” for efficient block updates
- Maintains a queue of blocks to process

Disk-based Algorithm (Duplo)

- Reads into memory and processes “N blocks at a time” using segments
- Maintains a queue of segments to process
- Updates segments by scanning a merge log on disk
- Uses timestamps to avoid a full scan for each segment read

Accuracy

Performance
Scalability

Iterative Blocking Summary
- Proposed model & efficient algorithms (in-memory, disk) for iterative blocking
- Showed that iterative blocking can be more accurate and scalable than simple blocking

ER in the InfoLab
- Generic ER [VLDB J. 09]
- Distributed ER [ICDCS 07]
- Iterative Blocking [SIGMOD 09]
- Negative Rules [VLDB J. 09]
- Confidences [VLDB CleanDB 06]
- Evolving Rules
- Joint ER
- ER Measures
- ER in Probabilistic Databases
- Privacy and Information Leakage