Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

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/



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

	Name	Phone	E-mail
r_1	{JohnDoe}	{235-2635}	{jdoe@yahoo}
r_2	{J.Doe}	{234-4358}	
r_3	{JohnD.}	{234-4358}	{jdoe@yahoo}

- 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

r_4	{John Doe}	{234-4358, 235-2635}	{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 = {r1,…rn} finite set of records from R
- Match function M: R x R ->Boolean
  - M(r,s) = 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) = true
- Merge function m: R x R -> R
  - Defined only for matching records
  - Notation m(r,s) = <r,s>

Merge closure

- Given instance I, the merge closure of I, denoted Î, is the smallest set of records S, such that
  - $I \subseteq S$
  - For any r, s: If $r \approx s$ then $\langle r,s \rangle \in S$

- Intuition: Extend I with all records that can be created by matching and merging
- Properties
  - Î exists and is unique
  - Î can be infinite
    - Unrealistic in practice

Domination

- Record r is dominated by s if r ≈ s and s holds more information
  - r $\preccurlyeq$ s
  - Any partial order on records
    - Reflexive, transitive
    - Antisymmetric: if r $\preccurlyeq$ s and s $\preccurlyeq$ r, then $r = s$,
- Examples: r1 $\preccurlyeq$ r4 and r2 $\preccurlyeq$ r4
  - Application-dependent

	Name	Phone	E-mail
r_1	{JohnDoe}	{235-2635}	{jdoe@yahoo}
r_2	{J.Doe}	{234-4358}	
r_3	{JohnD.}	{234-4358}	{jdoe@yahoo}
r_4	{John Doe}	{234-4358, 235-2635}	{jdoe@yahoo}

Instance domination

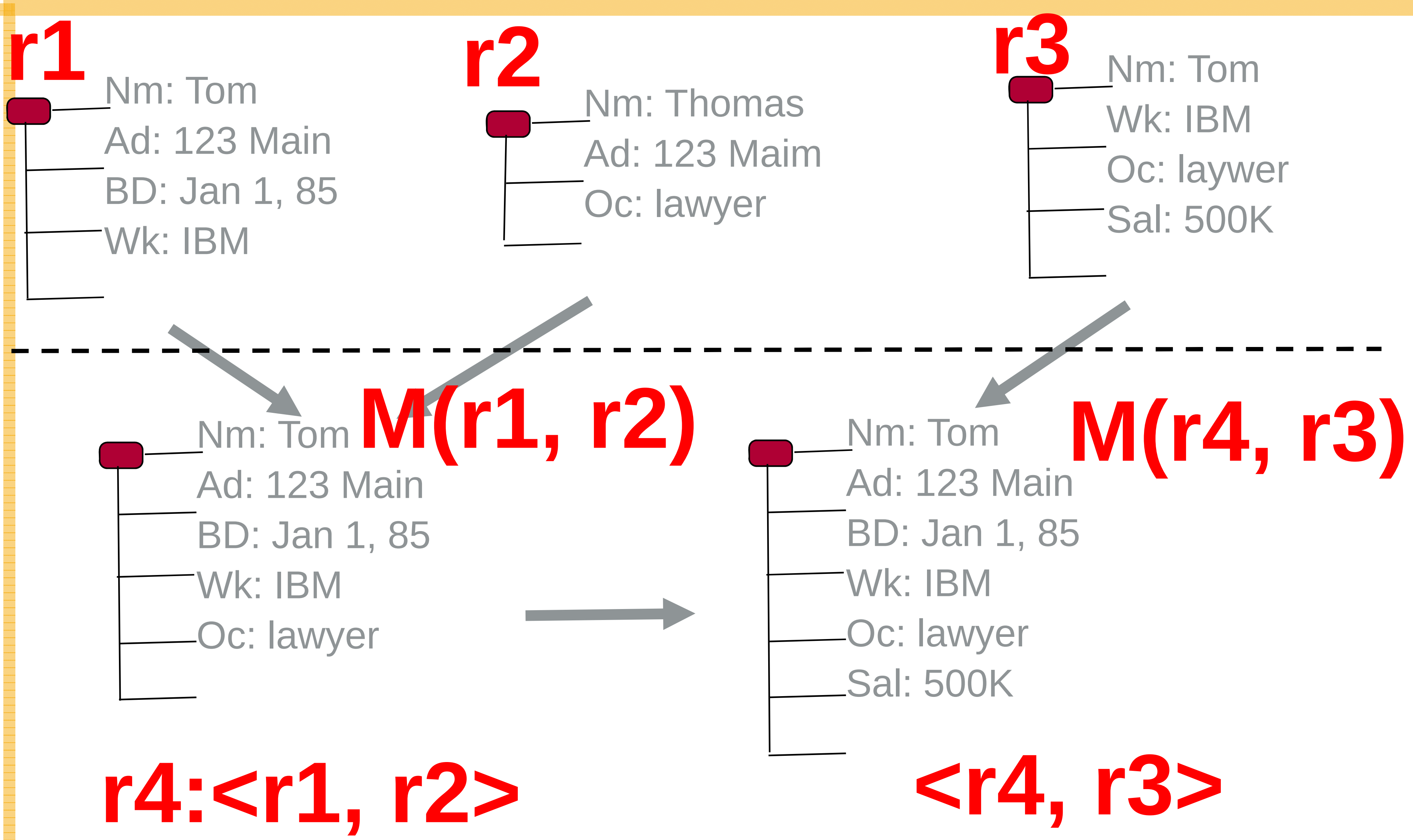
- Given instances I1 and I2, I1 is dominated by I2 (I1 $\preccurlyeq$ I2) if for all r1$\in$I1 there exists an r2$\in$I2 such that r1 $\preccurlyeq$ r2.
  - Reflexive
  - Transitive
  - Not antisymmetric: If r1 $\preccurlyeq$ r2 then
    - {r2} $\preccurlyeq$ {r1,r2} and {r1, r2} $\preccurlyeq$ {r2}

Entity resolution

- 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. $\hat{I} \preccurlyeq I'$
  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

Model



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*: $\forall r,\ r \approx r$ and $<r, r> = r$.
  - A record always matches itself, and merging it with itself still yields the same record.
- *Commutativity*: $\forall r, s$: $r \approx s$ iff $s \approx r$,
  - and if $r \approx s$, then $<r, s> = <s, r>$.
  - Direction of match and merge is irrelevant
- *Associativity*: $\forall r1, r2, r3$ 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 $r1 \approx r4$, we also have $r3 \approx r4$.
  - r3 “represents” r1 and r2.
  - Merging does not lose matches; no “negative evidence”
- *Transitivity* is <u>not</u> assumed: $r \approx s$ and $s \approx t$ does not imply $r \approx t$.

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, $r1$ and $r2$, we say that $r1$ *is merge dominated by* $r2$, denoted $r1 \leq r2$, if $r1 \approx r2$ and $<r1, r2> = r2$.
  - r1 does not add information.

Monotonicity

- For any records $r1$, $r2$ such that $r1 \approx r2$, it holds that $r1 \leq \langle r1, r2 \rangle$ and $r2 \leq \langle r1, r2 \rangle$
  - Merge record always dominates the records it was derived from
- If $r1 \leq r2$ and $r1 \approx r$, then $r2 \approx r$
  - Match function is monotonic
- If $r1 \leq r2$ and $r1 \approx r$, then $\langle r1, r \rangle \leq \langle r2, r \rangle$
  - Merge function is monotonic
- If $r1 \leq s$, $r2 \leq s$ and $r1 \approx r2$, then $\langle r1, r2 \rangle \leq s$.

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

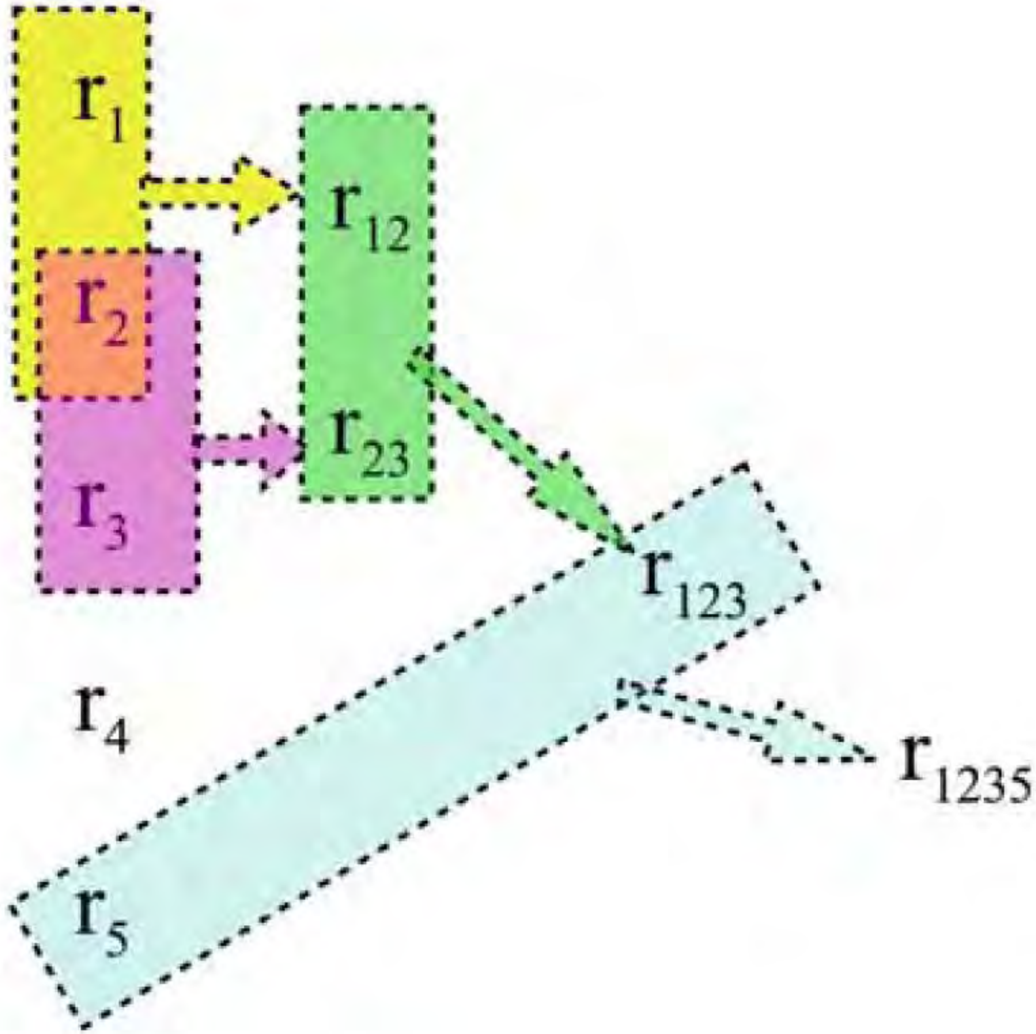
Naive Breadth First

input: a set I of records
output: a set I' of records, $I' = ER(I)$
$I' \leftarrow I$; $N \leftarrow \emptyset$
repeat
  $I' \leftarrow I' \cup N$; $N \leftarrow \emptyset$
  for all pairs (r, r') of records in I' **do**
    if $r \approx r'$ **then**
      $merged \leftarrow \langle r, r' \rangle$
      if $merged \notin I'$ **then**
        add $merged$ to N
      end if
    end if
  end for
until $N = \emptyset$
for all pairs (r, r') of records in I' where $r \neq r'$ **do**
  if $r' \preceq r$ **then**
    Remove r' from I'
  end if
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



G-Swoosh

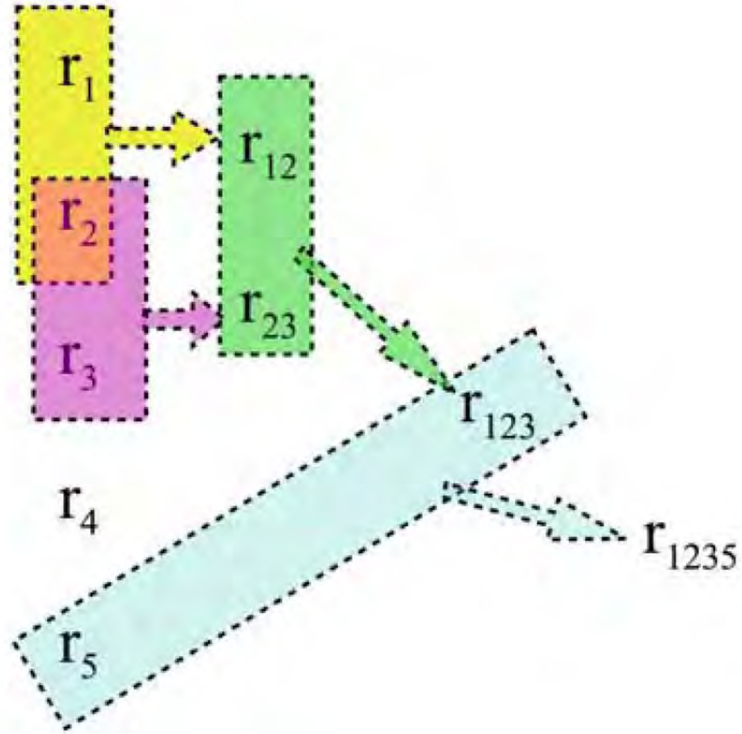
input: a set I of records
output: a set I' of records, $I' = ER(I)$
$I' \leftarrow \emptyset$
while $I \neq \emptyset$ **do**
    $r \leftarrow$ a record from I
    remove r from I
    for all records r' in $I' \cup \{r\}$ **do**
        if $r \approx r'$ (resp. $r' \approx r$) **then**
            $merged \leftarrow \langle r, r' \rangle$ (resp. $\langle r', r \rangle$)
            if $merged \notin I \cup I' \cup \{r\}$ **then**
                add $merged$ to I
            end if
        end if
    end for
    add r to I'
end while
Remove dominated records from I' (See lines 15–18 in BFA)
return I'

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

input: a set I of records
output: a set I' of records, $I' = ER(I)$
$I' \leftarrow \emptyset$
while $I \neq \emptyset$ **do**
  $r \leftarrow$ a record from I
  remove r from I
  for all records r' in $I' \cup \{r\}$ **do**
    if $r \approx r'$ (resp. $r' \approx r$) **then**
      $merged \leftarrow \langle r, r' \rangle$ (resp. $\langle r', r \rangle$)
      if $merged \notin I \cup I' \cup \{r\}$ **then**
        add $merged$ to I
      end if
    end if
  end for
  add r to I'
end while
Remove dominated records from I' (See lines 15–18 in BFA)
return I'

Idempotency: ∪ {r} not needed

Commutativity: r‘≈r not needed

Commutativity: <r‘,r> not needed

Without ICAR properties, G-Swoosh is optimal in number of match-calls.

Overview

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

R-Swoosh – Ideas

- Assumes ICAR and merge domination
  - Reminder: $r1$ *is merge dominated by* $r2$, denoted $r1 \leq r2$, if $r1 \approx r2$ and $\langle r1, r2\rangle = r2$
- Idea 1: If $r1 \approx r2$ we can remove r1 and r2
  - Whatever would match r1 or r2 now also matches <r1,r2>
  - Representativity and associativity
- Idea 2: Removal of dominated records (last step in algorithm) not necessary.
  - Assume r1 and r2 appear in final answer and $r1 \leq r2$.
    Then $r1 \approx r2$ and $\langle r1,r2\rangle = r2$.
  - Thus comparison of *r1* and *r2* should have generated merged record r2, and r1 should have been eliminated.

R-Swoosh

```
input: a set I of records   /* Initialization */
output: a set I' of records, I' = ER(I)
I' ← ∅
while I ≠ ∅ do   /* Main loop */
  currentRecord ← a record from I
  remove currentRecord from I
  buddy ← null
  for all records r' in I' do
    if M(currentRecord, r') = true then
      buddy ← r'
      exitfor
    end if
  end for
  if buddy = null then
    add currentRecord to I'
  else
    r'' ← < currentRecord, buddy >
    remove buddy from I'
    add r'' to I
  end if
end while
return I'
```

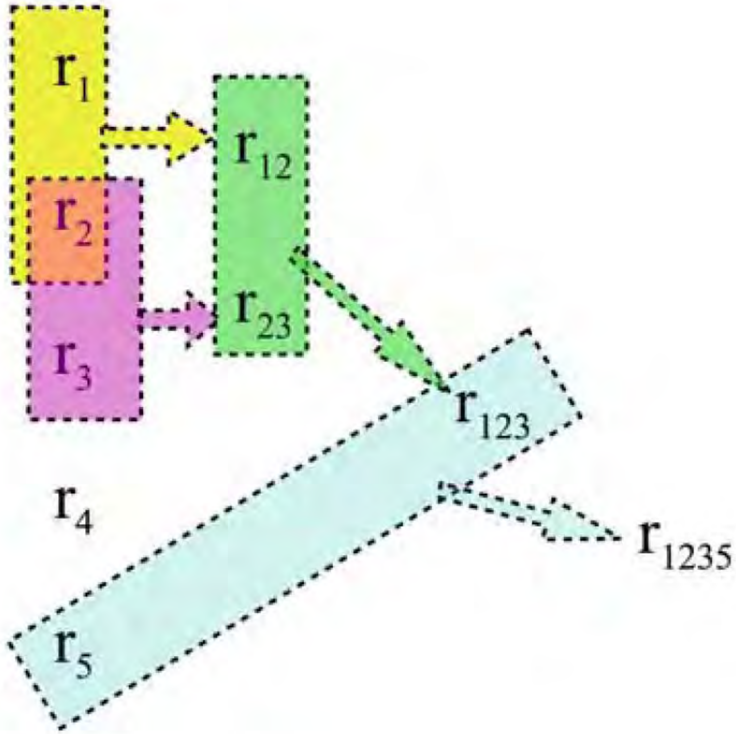
In case of match, no further comparisons

As before

Add merged record to I and remove both original records

R-Swoosh Example

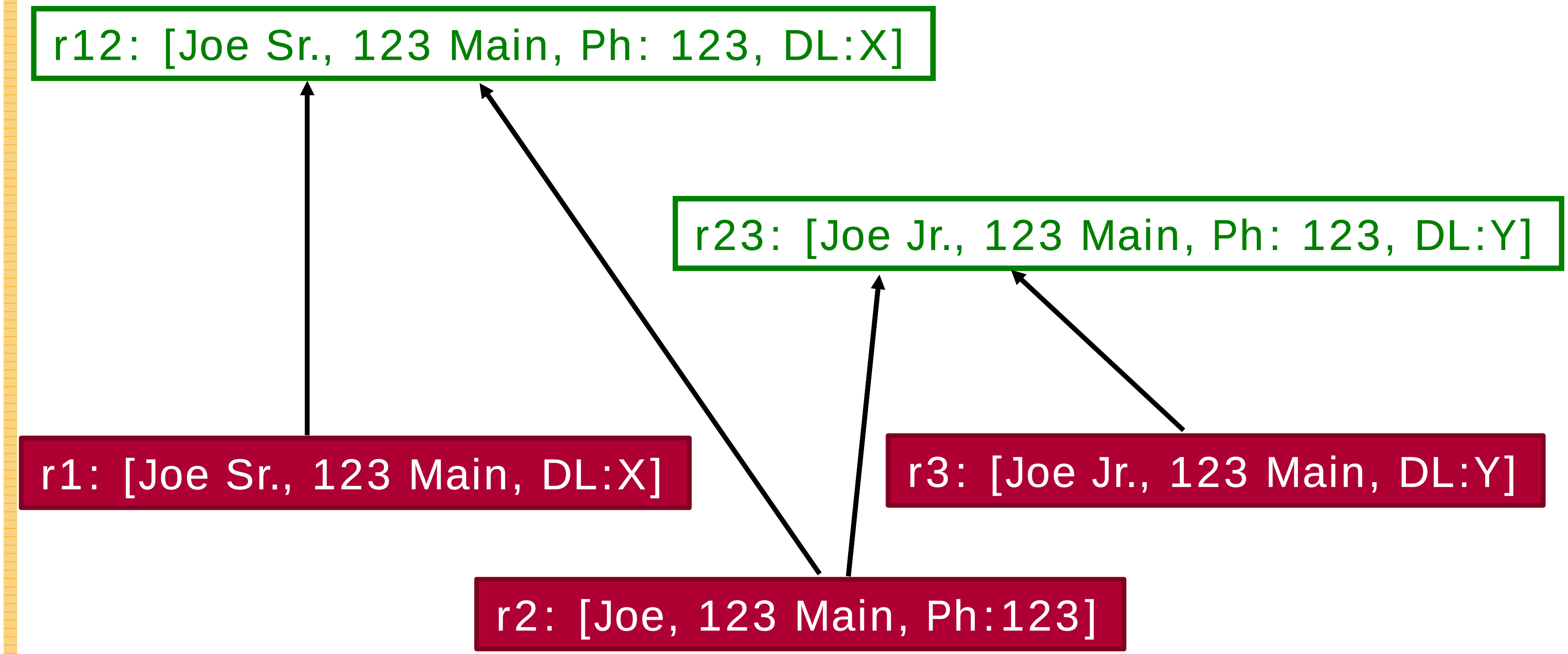
1. I = 1,2,3,4,5 I‘ = {}
2. I = 2,3,4,5 I‘ = 1
3. I = 3,4,5,12 I‘ = {}
4. I = 4,5,12 I‘ = 3
5. I = 5,12 I‘ = 3,4
6. I = 12 I‘ = 3,4,5
7. I = 123 I‘ = 4,5
8. I = {} I‘ = 4,1235



- Fewer iterations
- Fewer comparisons per iteration

- Further improvement: Order records intelligently, if possible
  - Achieve early matches

If ICAR Properties Do Not Hold?



Full Answer:	ER(R) = {r12, r23, r1, r2, r3}
Minus Dominated:	ER(R) = {r12, r23}
R-Swoosh Yields:	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(r1,r3): Compare „JohnDoe“ with „JohnD.“
  - <r1,r3> = r4
  - M(r3,r4): Compare „JohnDoe“ with „JohnD.“ again.
- Different records may have common values
  - (Expensive) comparisons are performed redundantly

	Name	**Phone**	**E-mail**
r_1	{JohnDoe}	{235-2635}	{jdoe@yahoo}
r_2	{J.Doe}	{234-4358}	
r_3	{JohnD.}	{234-4358}	{jdoe@yahoo}
r_4	{John Doe}	{234-4358, 235-2635}	{jdoe@yahoo}

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