Progressive Query-driven Entity Resolution

SISAP 2021 Doctoral Symposium Paper

Luca Zecchini
University of Modena and Reggio Emilia
luca.zecchini@unimore.it

Advisors: **Sonia Bergamaschi** and **Giovanni Simonini**, University of Modena and Reggio Emilia

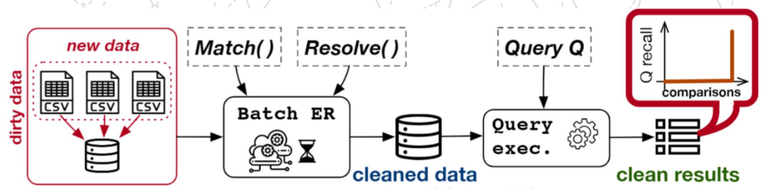
SISAP 2021
Sept 29 – Oct 01
Dortmund, Germany

Entity Resolution (ER)

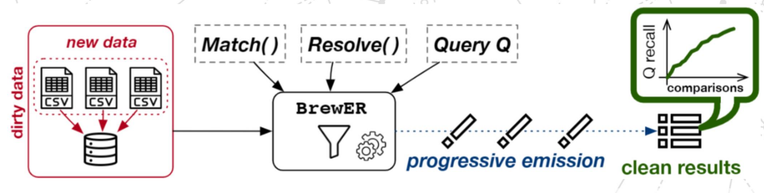
Individuating inside a **dataset** the **records** that refer to the **same real-world entity** (*duplicates*).

Hard task, due to **dirty and ambiguous data**:
- words written in different ways (or even misspelled);
- cases of homonymy and synonymy;
- missing or wrong values.

Brand	Model	Megapixels	Price ($)
canon	eos 400d	10.0	185.00
cannon	rebel xti	10.1	150.00
nikon	d200	10.2	130.00
cannon nikon	kiss x3	null	90.00
canon	eos 400d	10.1	110.00
nikon	d-200	10.2	175.00



Traditional (batch) approach to ER



Progressive query-driven approach to ER (BrewER)

A data scientist can be only interested in just a portion of the dataset (e.g., for *data exploration*) and this interest can be expressed by using a **query**.

In **traditional (*batch*) approach**, first we need to perform **ER on the whole dataset**, then we can run the query on its cleaned version.

This implies a lot of **useless comparisons**, needed to produce entities that will surely not appear in the result of the query, **wasting time, resources, and money** (e.g., *pay-as-you-go* contracts in the cloud), which are always limited.

Thus, we need a new approach, with two characteristics:

- **Query-driven**: it performs ER only on the portion of dataset useful to answer the query (according to its **WHERE** clauses);
- **Progressive**: it returns the resulting entities in the right order (defined by the **ORDER BY** clause) as soon as they are obtained.

This is exactly the aim of ***BrewER***, our algorithm designed to run ***clean queries on dirty data***.

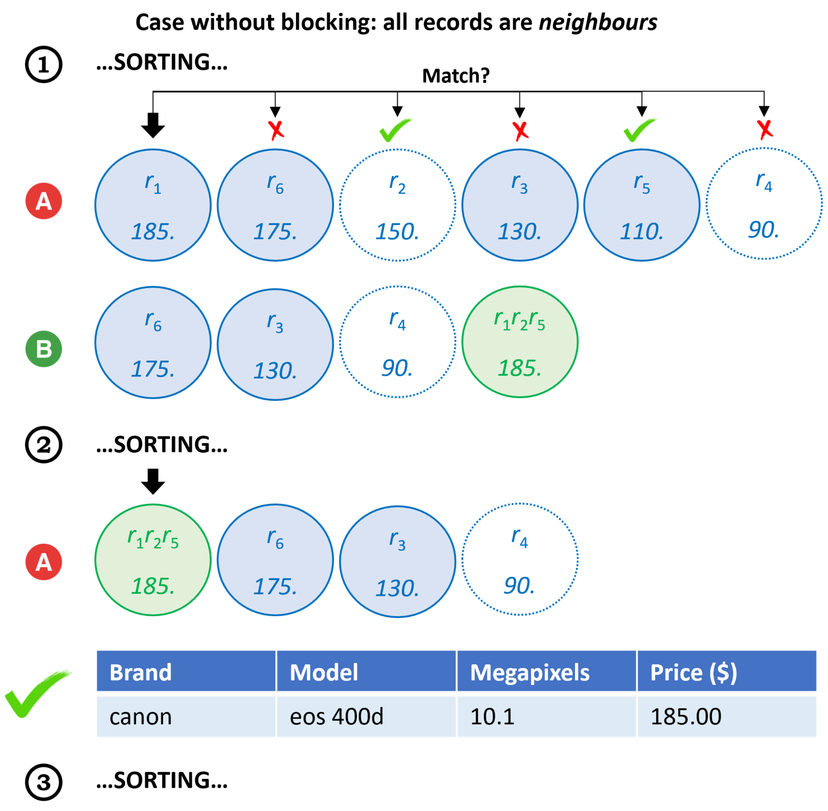
BrewER: Progressive Query-driven Entity Resolution

BrewER adopts an **agnostic approach to blocking and matching functions**.

	Brand	Model	Megapixels	Price ($)	
r_1	canon	eos 400d	10.0	185.00	SEED
r_2	cannon	rebel xti	10.1	150.00	
r_3	nikon	d200	10.2	130.00	SEED
r_4	olympus	om-1	null	90.00	
r_5	canon	eos 400d	10.1	110.00	SEED
r_6	nikon	d-200	10.2	175.00	SEED

Q1 - SQL on clean data
SELECT Brand, Model, Megapixels, Price
FROM Camera
WHERE Brand **LIKE** '%canon%'
AND Model **LIKE** 'd'
ORDER BY Price **DESC**

Q2 - SQL on dirty data
SELECT VOTE(Brand), VOTE(Model), VOTE(Megapixels), MAX(Price)
FROM Camera
GROUP BY ENTITY
HAVING VOTE(Brand) **LIKE** '%canon%'
AND VOTE(Model) **LIKE** 'd'
ORDER BY MAX(Price) **DESC**



Brand	Model	Megapixels	Price ($)
canon	eos 400d	10.1	185.00

BrewER in action

- First, we perform a **preliminary filtering of the blocks**: if a block contains at least a ***seed* record**, i.e., a record that satisfies one of the WHERE clauses, it can produce a useful entity and passes the filtering (if the clauses are in AND, the block must contain at least a seed record for each clause to pass the filtering);
- All the records appearing in the blocks that pass the filtering are inserted in an **Ordering List (OL)**, each one with a list of its **neighbours** (records appearing in the same blocks);
- Then, we iterate on OL:
  - OL is **sorted** according to the **Ordering Key (OK)**, i.e., the attribute used in the ORDER BY clause, defining the emission priority
  - We **check the first element**:
    - If it is one of the original records, we look for its duplicates among its neighbours and replace all the matching records with a single representative record presenting the aggregated value for OK;
    - If it is one of these representative records, we perform ER and check for its emission.

Early Results

Tested on real-world datasets, BrewER clearly shows its **progressive** nature. The **Query Recall**, computed on batches of queries, is the number of emitted resulting entities out of the total number of resulting entities to be emitted. An adaptation of **QDA** (query-driven, but not progressive) is used as batch baseline.

Optimization for Discordant Ordering

An **optimization** can be used for the frequent case of **discordant ordering (MAX/ASC or MIN/DESC)**. In this case, it is possible to insert **in OL only the seed records**, while the other records appear just as neighbours, reducing the number of comparisons. This is possible since in this case updating OK for the first element postpones its emission, guaranteeing the correctness of the emission order.

Conclusions

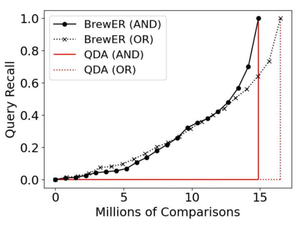
Early results confirm the **benefits** in terms both of reduction of performed comparisons and of progressive emission of the results, paving the way for **new and more comprehensive solutions to ER tasks**.

Many **challenges** are open: we want to find other cases to be optimized, study the impact of blocking and missing values, and the benefits for TOP(K) queries, integrate BrewER with other data preparation/cleaning steps, and analyze its possible impact on other classification tasks.

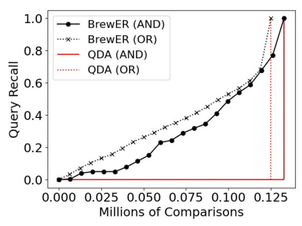
The formalized algorithm, many other results and a further exploration will be presented in a **dedicated research paper**.

Name	Records	Duplicates	Entities (Mean Size)	Attributes	Ordering Key	
SIGMOD20 [7, 14]	13.58k	12.01k	3.06k (4.439)	5	Megapixels	Cameras
SIGMOD21	1.12k	1.08k	190 (5.879)	5	Price	USB sticks
Altosight	12.47k	12.44k	453 (27.534)	5	Price	USB sticks
Funding [8]	17.46k	16.70k	3.11k (5.609)	18	Amount	Organizations

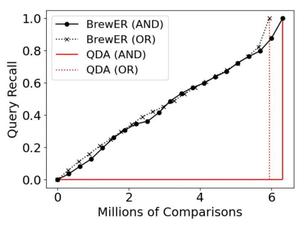
BrewER vs Batch ER (QDA)



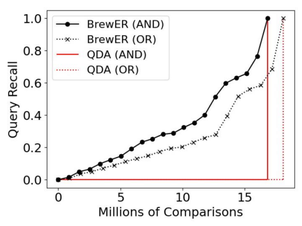
(a) SIGMOD20
Brand AND Model
Brand OR Brand



(b) SIGMOD21
Brand AND Size
Brand OR Brand

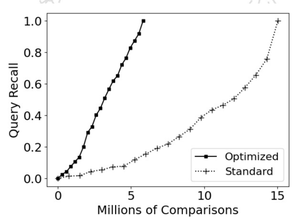


(c) Altosight
Brand AND Size
Brand OR Brand

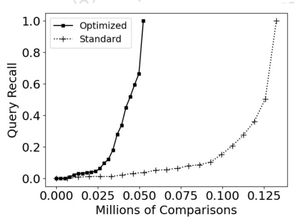


(d) Funding
Source AND Legal_Name
Source OR Source

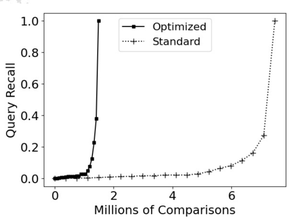
Optimization for discordant ordering



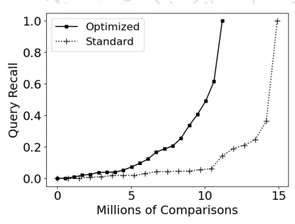
(a) SIGMOD20
Brand AND Model



(b) SIGMOD21
Brand AND Size



(c) Altosight
Brand AND Size



(d) Funding
Source AND Legal_Name