BEER: Blocking for Effective Entity Resolution

Sainyam Galhotra
UMass Amherst
sainyam@cs.umass.edu

Donatella Firmani
Roma Tre University
donatella.firmani@uniroma3.it

Barna Saha
UC Berkeley
barnas@berkeley.edu

Divesh Srivastava
AT&T Chief Data Office
divesh@att.com

ABSTRACT

Blocking is a key component of Entity Resolution (ER) that aims to improve efficiency by quickly pruning out non-matching record pairs. However, depending on the noise in the dataset and the distribution of entity cluster sizes, existing techniques can be either (a) too aggressive, such that they help scale but can adversely affect the ER effectiveness, or (b) too permissive, potentially harming ER efficiency. We propose a new methodology of *progressive blocking* that enables both efficient and effective ER and works across different entity cluster size distributions without manual fine tuning. In this paper, we demonstrate BEER (**B**locking for **E**ffective **E**ntity **R**esolution), the first end-to-end system that leverages intermediate ER output in a feedback loop to refine the blocking result in a data-driven fashion, thereby enabling effective entity resolution.

BEER allows the user to explore the different components of the ER pipeline, analyze the effectiveness of alternative blocking techniques and understand the interaction between blocking and ER. BEER supports visualization of the different entities present in a block, explains the change in blocking output with every round of feedback and allows the end-user to interactively compare different techniques. BEER has been developed as open-source software; the code and the demonstration video are available at beer-system.github.io.

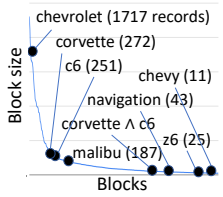
ACM Reference Format:
Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. 2021. BEER: Blocking for Effective Entity Resolution. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 18–27, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452747

1 INTRODUCTION

Data integration pipelines are useful for complex tasks like knowledge graph construction and de-duplication. One of the core components of data integration is Entity Resolution (ER), which aims to identify clusters of records that refer to the same real-world entity [5]. ER inherently has quadratic complexity since it requires comparing all record pairs to identify matches. This is not scalable for large datasets, a problem that is exacerbated by the increased availability of data from a variety of heterogeneous big data sources. For this reason, blocking is performed as a pre-processing step to select a sub-quadratic number of record pairs which are then compared in subsequent steps. There are a plethora of blocking techniques that aim to group similar records into a collection of overlapping blocks so that it suffices to compare record pairs that belong to the same block [20]. However, depending on the distribution of entity cluster sizes in the input data set, different techniques either prune out too many matching pairs leading to poor effectiveness, or include too many non-matching pairs leading to low efficiency. Selecting and fine-tuning blocking is an iterative process where the blocking quality can only be evaluated on completion of ER. We demonstrate these limitations with an example below.

Table 1: Sample records where r_i^e represents the i-th record referring to entity e. The different entities in the dataset are Chevrolet Corvette C6 ($c6$), Chevrolet Corvette Z6 ($z6$), Chevrolet Malibu (ma) and Citröen C6 (ci) (same model name as Corvette C6 but different car). The figure shows the size distribution of different blocks in the dataset [8].

r_1^{c6}: 'chevy corvette c6'
r_2^{c6}: 'chevy corvette c6 navigation'
r_3^{c6}: 'chevrolet corvette c6'
r_1^{z6}: 'corvette z6 navigation'
r_1^{ma}: 'chevy malibu navigation'
r_3^{ma}: 'chevrolet malibu'
r_1^{ci}: 'citroen c6 navigation'



Example 1.1. Consider the records in Table 1 from a `cars` dataset corresponding to car models and their textual descriptions. A standard blocking strategy generally consists of three sub-tasks [20]. First, during *block building* it creates a separate block for each text token t such that all records containing t are assigned to that block. Second, *block cleaning* uses a threshold to prune out all blocks of large size. Using a low size threshold (say < 100) prunes out all blocks that contain r_3^{c6} and r_1^{c6}), missing matching pairs like (r_3^{c6}, r_1^{c6}) while using a high size threshold captures many non-matching record pairs. Third, *comparison cleaning* techniques like meta-blocking [17] also do not yield a small set of blocked candidates that guarantees to capture all matching record pairs.

To cope with these shortcomings, we proposed a new methodology that performs blocking and ER in tandem in our previous work [8]. Our method starts with an aggressive blocking step, which is efficient but not necessarily effective. Then it computes a limited amount of ER results on a subset of pairs selected by the aggressive blocking, and sends these partial (matching and non-matching) results from the ER phase back to the blocking phase, creating a "loop", to improve blocking effectiveness. Blocking components use this feedback to evaluate their quality and generate new blocks that are expected to capture more matching record pairs. In this way, blocking self-regulates progressively and adapts to the properties of each dataset, with no manual configuration effort. In this paper, we demonstrate BEER, the first end-to-end progressive ER system that performs blocking and entity resolution in tandem. We now demonstrate its effectiveness on the example presented in Table 1.

SIGMOD '21, June 18–27, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452747

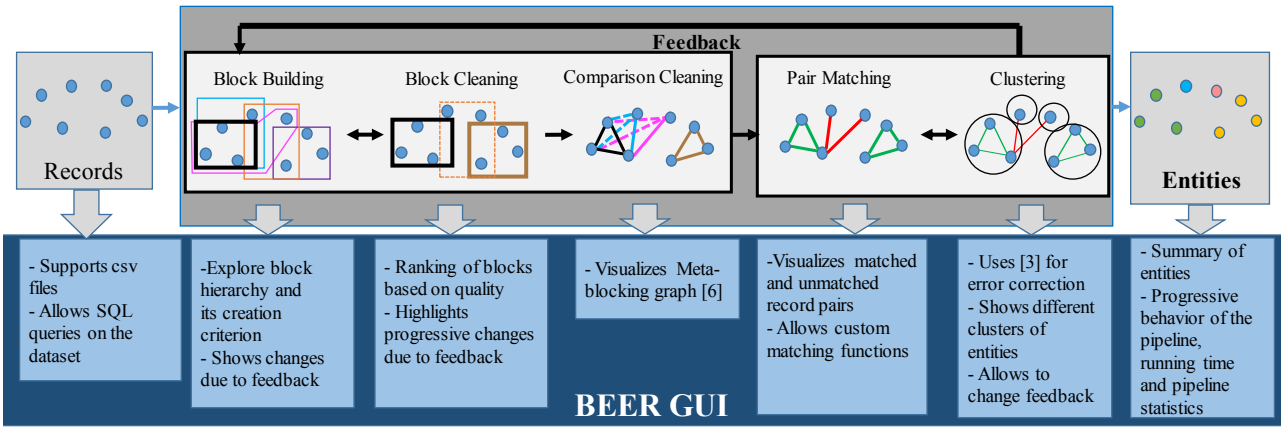


Figure 1: System Architecture and key highlights of BEER's graphical user interface.

Example 1.2. The feedback from the ER phase is used by BEER to create a *new* block 'corvette ∧ c6' containing records present in both blocks 'corvette' and 'c6'. This block is much smaller than its two constituents and has only Corvette C6 cars. Additionally, BEER estimates block quality and discourages the pruning of the block 'malibu' as it contains the majority of the Chevrolet Malibu cars and prunes out smaller blocks like 'navigation' that do not contain any matching pair.

In this demo, the attendees will first-hand understand the impact of blocking and ER on each other and observe the effectiveness of BEER to resolve entities efficiently. The participants will be able to customize different blocking techniques, pairwise matching functions and the feedback set in each iteration.

Related work. Blocking techniques have been studied for many decades. However, all prior techniques [1, 3, 10, 11, 13–15, 17, 17, 19, 20, 23] (including the advanced meta-blocking based techniques [1, 3, 17, 19, 20, 23]) have considered blocking as a pre-processing step and suffered from the effectiveness and efficiency trade-off [20]. Recent studies have also shown that all these methods require careful fine-tuning of parameters to yield high performance [20]. BEER presents the first progressive ER technique that performs blocking and pairwise matching in tandem and demonstrates superior performance as compared to prior techniques. Among existing systems, Magellan [12] presents a framework to define blocking and ER rules but does not consider the interaction between them. JedAI [18, 21] and SystemER [22] explain different pairwise matching algorithms but do not focus on using feedback to improve the blocking components of the ER pipeline.

We now provide a solution sketch (Section 2) and a detailed outline of our demonstration (Section 3).

2 SYSTEM OVERVIEW

Figure 1 presents the overall system architecture along with the corresponding visualization of BEER's graphical user interface. BEER supports csv files containing textual description of the records. We first describe the blocking component of BEER and then discuss the pair matching and clustering algorithms.

Block Building takes the records as input and returns a collection of blocks, by assigning each record to multiple blocks. One of the most commonly used techniques, *standard blocking* [15, 20] creates a separate block for each token t in the records such that all records that contain t are assigned to it. In order to tolerate spelling errors, *q-gram blocking* [10] considers character-level q-grams instead of entire tokens. Sorted neighborhood [11] sorts all the records according to multiple sort orders, for example, a sorting of tokens for each attribute in the dataset. It then slides a window of tokens over each ordering and creates a new block corresponding to all tokens present in the window. This new block contains all records that contain any of the tokens in the window.

All these strategies construct many large blocks that contain a lot of non-matching record pairs. To ensure effectiveness, BEER constructs an intersection block hierarchy comprising of multiple levels of blocks. The first level of the hierarchy contains blocks generated by one of the prior block building techniques. Blocks in the ith level ($i > 1$) consist of the intersection of i distinct blocks in the first level. These blocks are referred to as *refined* blocks that are subsets of the intersecting *parent* blocks. Note that not all refined blocks may be useful and BEER tests a correlation condition between parent blocks to decide if the refined block should be generated.

Example 2.1. Consider our `cars` example and the blocks corresponding to tokens 'corvette' and 'c6', namely $B_{\texttt{corvette}}$, and $B_{\texttt{c6}}$. A sample block in the second level of the hierarchy is $B_{corvette} \cap B_{\texttt{c6}}$. This block contains the records having the two tokens 'corvette' and 'c6', thus obtaining a cleaner block than the original ones.

Intuitively, the intersection blocks are more refined and more likely to contain matching record pairs than their parents. The correlation criterion captures the co-occurrence of blocks in the dataset to generate refined blocks. BEER employs efficient data structures to generate these blocks in $O(n)$ running time [8] (where n denotes the number of records in the dataset) as opposed to $O(n^2)$ running time for naive techniques.

Block cleaning takes the block collection as input and returns a subset by pruning blocks that may contain too many non-matching record pairs. Block cleaning is typically performed by assigning each block a score and then pruning the ones with low score. Traditional strategies include TF-IDF [5, 16] based scoring which assigns scores inversely proportional to the block size, prioritizing smaller blocks over larger ones. However, such scoring techniques do not capture the block quality and are suited only for datasets

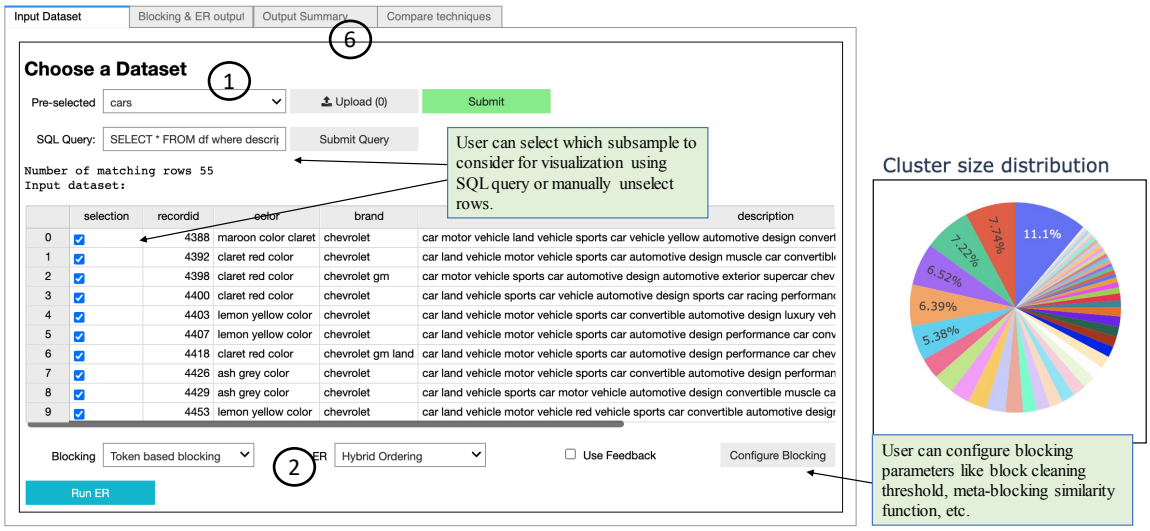


Figure 2: Input screen for BEER with dataset statistics, distribution of clusters and SQL query option.

that contain very few matching pairs. BEER's block scoring method distinguishes informative blocks based on their ability to capture records from a single cluster.

Specifically, the scoring algorithm of BEER prioritizes blocks having the following properties. (a) High fraction of matching pairs estimated from the matching probability of record pairs within the block; and (b) Fewer number of clusters (especially larger clusters) measured as entropy of the cluster distribution within the block. Lower entropy values indicate the representativeness towards a particular cluster as opposed to higher entropy values which imply the presence of many fragmented clusters. The block scores are estimated from similarity of record pairs which converges to the ground truth scores with feedback.

Comparison Cleaning takes the set of all intra-block record pairs as input and identifies a subset as the final set of candidates using meta-blocking techniques [17]. It constructs a graph over the records such that any pair of records that shares a block are connected by an edge. Each edge is then assigned a weight (that captures the likelihood of being a match) using strategies like number of common blocks or weighted jaccard similarity of the record pair. All low weight edges are pruned and the remaining edges are considered as candidates for pair matching and clustering components of the pipeline. In addition to the blocking graph, it outputs similarity of every record pair that is used by the subsequent stages to prioritize pairwise comparisons.

Pair Matching and Clustering takes the similarity of record pairs as input and generates a prioritization for the matching algorithm. BEER allows the user to select one of the recent progressive ER techniques [6, 24, 25] which prioritize record pairs corresponding to entities that contain a large number of records, thereby boosting the progressiveness of the ER procedure. In the pre-loaded datasets, BEER uses a random forest classifier trained using active learning to label every pair as matching or non-matching. In the absence of training data, BEER provides the functionality to use pre-defined similarity based rules or manually input new matching rules in first order logic. BEER employs the random graph toolkit [7] to ensure robustness of the pipeline and convert the output of pairwise comparisons into a set of clusters.

Feedback. As soon as the pair matching component of the pipeline processes 1% of the newly blocked pairs, the feedback set of identified matching and non-matching pairs is sent back to blocking. This feedback is then incorporated by blocking to update its intersection hierarchy and quality based scores which in turn change the set of blocked pairs. To ensure efficiency of block building and scoring components, BEER leverages the union-find data structure to maintain the clusters of records referring to different entities and generates a random sample of records from each block to quickly estimate block scores.

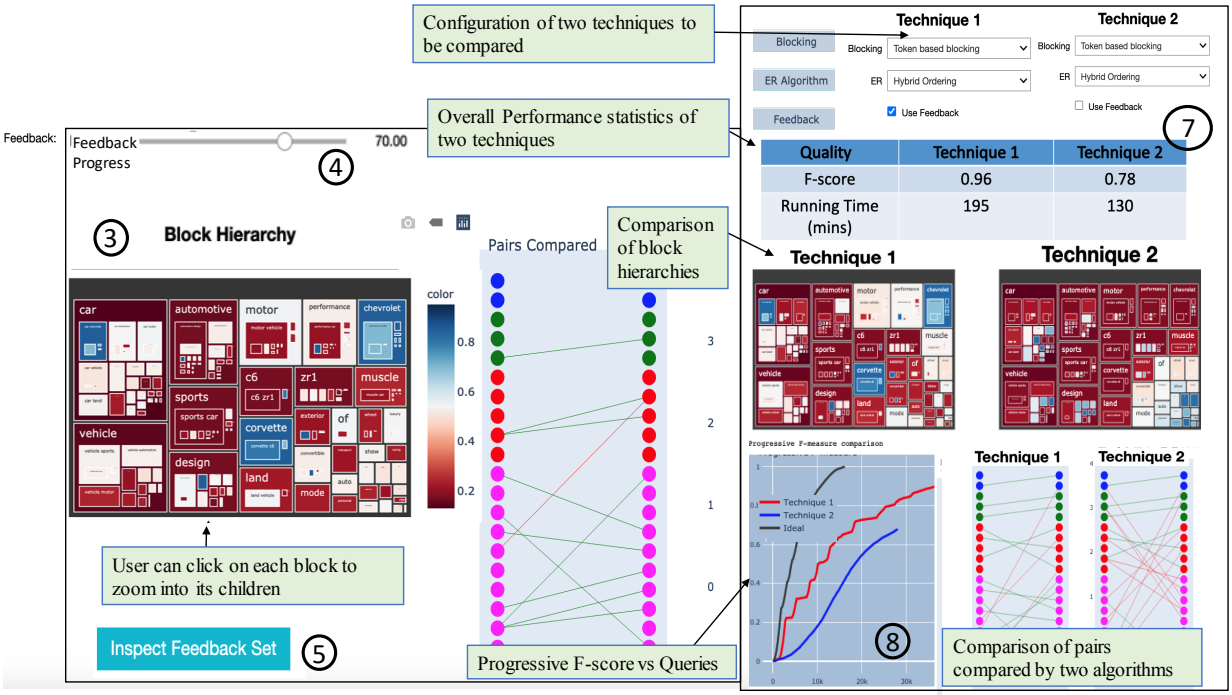
3 DEMONSTRATION OVERVIEW

We demonstrate BEER on the `cars` dataset where the user can change the blocking configuration to simulate 60 pipelines (comprising 5 block building methods, 2 block cleaning, 2 comparison cleaning and 3 pair matching and clustering methods).

Step (1) (Select input dataset): First, the user selects an input dataset used to perform entity resolution. We present a SQL query option, where the user can input a query to sub-sample the data for subsequent analysis. Figure 2 shows an example SQL output for records that contain the token 'corvette'.

Step (2) (Choose blocking functions and ER algorithm): The user selects the blocking and pair matching methods from the presented suite of techniques. We allow the user to configure the different components of the pipeline. In this scenario, the user picks standard token based blocking [20] and hybrid ER algorithm [6].

Step (3) (Generate blocks and corresponding set of pairs compared): The user clicks 'Run ER' and BEER visualizes the different blocks and corresponding ER output. Figure 3(a) shows the intersection block hierarchy where each block size is proportional to the number of records present and the color of a block indicates its estimated block score (red denotes 0 and blue denotes 1). The subset blocks are shown inside the parent block, for example, the



(a) ER and blocking comparison (b) Technique Comparison

Figure 3: (a) Comparison of blocking and corresponding set of pairs compared. The left panel shows the block hierarchy where block size is proportional to the number of records present and color is proportional to its score. The right panel shows the set of compared pairs with the green and red edges corresponding to matching and non-matching pairs, respectively. The feedback slider allows the user to observe the change in block hierarchy and scores for different iterations of feedback. (b) Visual comparison of different blocking techniques where 'Technique 1' uses feedback and 'Technique 2' does not use feedback.

block corresponding to 'corvette c6' is placed inside the block corresponding to 'corvette'. The user can interact by zooming into the hierarchy and clicking on individual blocks, BEER shows a word cloud of the tokens shared by records in the block. The right panel of Figure 3(a) shows the compared record pairs where green edges denote matching and red edges denote non-matching pairs (Records with the same color refer to the same entity). In this dataset, the block scores show that many large blocks have high score while many small blocks have low score.

Step (4) (Feedback set selection): The user can change the feedback progress slider (Figure 3(a)) to inspect the change in blocks and their scores at any iteration. This helps to demonstrate the impact of feedback at any iteration through the process.

Step (5) (Blocking Updates): For any feedback iteration, BEER shows the newly constructed blocks, change in scores and blocking graph (shown as a separate window on clicking 'Inspect feedback').

Step (6) (Summarization): The output summary tab (Figure 2) shows the overall statistics of the pipeline (running time, F-score, number of blocks, progressive behavior).

Step (7) (Pipeline comparison): BEER presents a comparative analysis between two different parameter settings for the user to compare effectiveness, efficiency, blocking graphs and ER progressiveness. Figure 3(b) compares the benefit of using feedback over naive blocking, where Technique 1 uses feedback but Technique 2 does not. This comparison visualizes the difference in blocking and ER components of the two pipelines along with a summarization of overall performance. The block hierarchy visual shows that feedback based scoring assigns higher scores to some of the larger blocks while Technique 2 assigns scores inversely proportional to size. Similarly, the two different graphs for pairs compared show that the second technique requires more comparisons and has explored more non-matching edges than Technique 1. The plot labelled (8) compares the progressive behavior of these techniques.

Demonstration Engagement After our guided demonstration, participants will be able to plug their own datasets into BEER. We will also provide four pre-loaded datasets (i) `camera` [2] (ii) `citations` [9] (iii) `songs` [4], and (iv) `cars` [8] as these datasets have varied levels of noise and cluster size distributions. Through this demonstration, we will showcase how BEER can effectively leverage feedback from the ER phase to boost the effectiveness of the end-to-end pipeline.

ACKNOWLEDGEMENTS

This work is supported partly by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

REFERENCES

[1] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, 2006.

[2] Valter Crescenzi, Andrea De Angelis, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. Alaska: A flexible benchmark for data integration tasks, 2021.

[3] Guilherme dal Bianco, Marcos André Gonçalves, and Denio Duarte. Bloss: Effective meta-blocking with almost no effort. *Information Systems*, 75, 2018.

[4] Sanjib Das, Paul Suganthan GC, AnHai Doan, Jeffrey F Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, 2017.

[5] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1), 2007.

[6] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5), 2016.

[7] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, 2018.

[8] Sainyam Galhotra, Donatella Firmani, Barna Saha, and Divesh Srivastava. Efficient and effective er with progressive blocking. *Accepted VLDB journal*, 2021.

[9] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[10] Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *PVLDB*, pages 491–500, 2001.

[11] Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. In *ACM Sigmod Record*, volume 24, pages 127–138. ACM, 1995.

[12] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, , et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[13] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[14] N McNeill, Hakan Kardes, and Andrew Borthwick. Dynamic record blocking: efficient linking of massive databases in mapreduce. Citeseer, 2012.

[15] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB*, 9(4):312–323, 2015.

[16] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederee, and Wolfgang Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE TKDE*, 25(12):2665–2682, 2012.

[17] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. Meta-blocking: Taking entity resolutionto the next level. *IEEE TKDE*, 26(8):1946–1960, 2013.

[18] George Papadakis, George Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. Three-dimensional entity resolution with jedai. *Information Systems*, 93:101565, 2020.

[19] George Papadakis, George Papastefanatos, and Georgia Koutrika. Supervised meta-blocking. *PVLDB*, 7(14):1929–1940, 2014.

[20] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.

[21] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. The return of jedai: end-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.

[22] Kun Qian, Lucian Popa, and Prithviraj Sen. Systemer: a human-in-the-loop system for explainable entity resolution. *PVLDB*, 12(12):1794–1797, 2019.

[23] Giovanni Simonini, Sonia Bergamaschi, and HV Jagadish. Blast: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12), 2016.

[24] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, 2014.

[25] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.