JedAI^3: beyond batch, blocking-based Entity Resolution

George Papadakis[1], Leonidas Tsekouras[2], Manos Thanos[3], Nikiforos Pittaras[2], Giovanni Simonini[5], Dimitrios Skoutas[6], Paul Isaris[7], George Giannakopoulos[2,7], Themis Palpanas[4], Manolis Koubarakis[1]

[1]University of Athens, Greece {gpapadis,koubarak}@di.uoa.gr
[2]NCSR "Demokritos", Greece {ltsekouras, pittarasnikif, ggianna}@iit.demokritos.gr
[3]KU Leuven, Belgium emmanouil.thanos@kuleuven.be
[4]Paris Descartes University, France themis@mi.parisdescartes.fr
[5]University of Modena and Reggio Emilia, Italy simonini@unimore.it
[6]IMSI, Athena Research Center, Greece dskoutas@imis.athena-innovation.gr
[7]SciFY, Greece paul@scify.org

ABSTRACT

JedAI is an open-source toolkit that allows for building and benchmarking thousands of schema-agnostic Entity Resolution (ER) pipelines through a non-learning, blocking-based end-to-end workflow. In this paper, we present its latest release, JedAI^3, which conveys two new end-to-end workflows: one for budget-agnostic ER that is based on similarity joins, and one for budget-aware (i.e., progressive) ER. This version also adds support for pre-trained word or character embeddings and connects JedAI to the Python data analysis ecosystem. Overall, these enhancements provide JedAI with features offered by no other ER tool, especially in the schema- and domain-agnostic context.

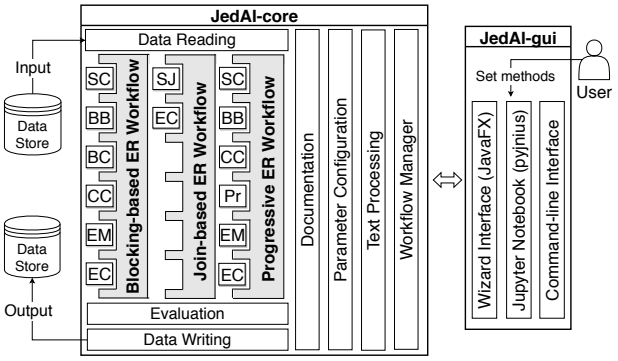


Figure 1: JedAI's architecture.

1 INTRODUCTION

Entity Resolution (ER) aims to detect *duplicates*, i.e., different entity profiles that describe the same real-world objects. It is a core data integration task, with many applications that range from knowledge bases to question answering [8]. Yet, the available systems focus exclusively on *batch ER*, which is carried out in a budget-agnostic, offline way that imposes no strict constraints on temporal or computational resources. This means that they do not support *progressive ER*, which is carried out in a budget-aware manner that determines specific time frames or resources (e.g., by executing a fixed number of comparisons).

Even for batch ER, though, the existing tools have significant limitations: they cover the end-to-end pipeline partially, they constitute stand-alone systems with a limited variety of methods (usually the methods proposed by their creators), they apply only to a specific data type (i.e., structured or semi-structured data), or they require power users, providing insufficient guidelines on how to perform ER efficiently and effectively [8]. Another major disadvantage of existing tools is that they typically disregard the bulk of similarity join algorithms that allow for detecting pairs of duplicates in a rather efficient way.

Magellan resolves most of these issues, but is restricted to *Clean-Clean ER*, i.e., the task of detecting duplicate profiles across two overlapping, but individually duplicate-free datasets. This means that Magellan cannot tackle *Dirty ER*, i.e., the task of resolving the entities of a single dataset that contains duplicates in itself. Moreover, Magellan applies exclusively to relational data, it lacks a GUI, merely offering a command-line interface, and requires heavy user involvement. Its goal is actually to facilitate the development of tailor-made methods for the data at hand [8].

Java gEneric DAta Integration (JedAI) Toolkit [15] goes beyond existing ER tools by focusing on non-learning, schema- and structure-agnostic methods, which apply seamlessly to both structured and semi-structured data. At the same time, JedAI requires minimal human intervention, as neither domain knowledge nor training sets are needed. At its core lies a blocking-based end-to-end ER workflow, which is implemented by JedAI-core[1], conveying numerous state-of-the-art methods in each step. These methods can be mixed and matched to form thousands of ER pipelines that can be easily benchmarked through the wizard-like interface of its desktop application, JedAI-gui[2]. Thus, JedAI fulfills the main challenges arising in data integration [5]: the development of extensible, open-source[3] tools and the provision of solutions that apply to data of any structuredness - even unstructured (free-text) data. These new capabilities are exhibited through a live demonstration that involves user interaction.

In more detail, this demonstration presents the latest release of JedAI^3, which significantly enhances the core blocking-based workflow: it connects it with the Python ecosystem (see Section 5.1) and enriches it with pre-trained embeddings and with new techniques and capabilities per workflow step (see Section 2.1). Most importantly, JedAI now supports two new workflows: one based on similarity joins (Section 3) and one implementing budget-aware ER pipelines (Section 4). These enhancements equip JedAI with unique features that are offered by no other ER tool, especially in a schema- and domain-agnostic context.

[1]https://github.com/scify/JedAIToolkit
[2]https://github.com/scify/jedai-ui
[3]All code is available under Apache License V2.0, which supports both academic and industrial applications.

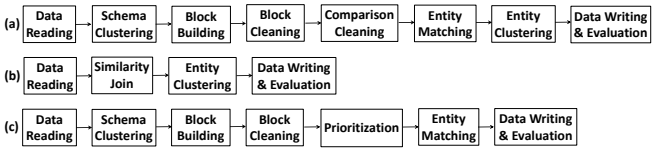


Figure 2: ER workflows in JedAI: (a) blocking-based; (b) join-based (*new*); (c) progressive (*new*).

Id	Rule	Dataset	Recall	Precision	F1
R1	0.59 < JaccardSim (title1, title2) & 0.26 < JaccardSim (authors1, authors2)	DBLP-ACM	0.926	0.930	0.928
R2	0.53 < JaccardSim (title1, title2)	Cora	0.855	0.749	0.799
R3	0.25 < JaccardSim (all_tokens1, all_tokens2)	1OK census	0.969	0.995	0.982

Figure 3: Three matching rules along with their performance over established datasets.

2 BATCH, BLOCKING-BASED ER WORKFLOW

Figure 2(a) depicts the budget-agnostic, blocking-based end-to-end ER workflow of JedAI. It consists of the following steps.

1) *Data Reading* loads from disk the dataset(s) to be processed and the golden standard. The JedAI data model accommodates both structured (relational databases, CSV) and semi-structured (SPARQL endpoints, CSV, XML, OWL and RDF) data as well as any mixture of those.

2) *Schema Clustering* groups together attributes that share similar names and/or values, but are not necessarily semantically equivalent. It is an optional step that is suitable for highly heterogeneous datasets with hundreds of different attribute names. In these settings, it significantly improves the overall precision with no impact on recall [15, 18].

3) *Block Building* clusters similar entities into blocks so as to drastically reduce the candidate match space. It includes most of the state-of-the-art methods, using their schema-agnostic adaptation [13], which extracts multiple blocking keys from each entity and places it into multiple blocks. The resulting *overlapping* blocks contain high levels of redundancy, achieving high recall at the cost of low precision [13], due to two types of unnecessary comparisons [13]: (i) the *redundant* ones, which are repeated across different blocks, and (ii) the *superfluous* ones, which involve non-matching entities.

4) *Block Cleaning* is an optional step that cleans the original blocks from both types of unnecessary comparisons, improving their precision at a negligible cost in recall. All available methods are complementary and can be combined.

5) *Comparison Cleaning* is another optional step that targets both types of unnecessary comparisons. It operates at the level of individual comparisons, achieving higher accuracy at the cost of a higher time complexity. Several methods are included, primarily Meta-blocking techniques [14, 18].

6) *Entity Matching* implements schema-agnostic methods for assessing the value similarity of all entity pairs in the final set of blocks. These methods can be combined with various similarity measures and graph or bag representation models from the Text Processing component (see Figure 1). The end result is a *similarity graph*, where the nodes correspond to entities, and the weighted edges connect compared entities.

7) *Entity Clustering* includes the main methods that are typically used for partitioning the nodes of the similarity graph into equivalence clusters, such that every cluster corresponds to a distinct real-world object [6].

8) *Evaluation* uses the golden standard of the selected dataset in order to compute several measures for effectiveness and time efficiency. The user may store intermediate or end results through the *Data Writing* functionality.

2.1 New features

This workflow has been enriched with support for embeddings, which lie at the core of the latest ER works that are based on deep learning [4, 12]. JedAI supports any pre-trained character- (e.g., fastText [1]) and word-level embeddings (e.g., Glove [17], word2vec [11]). The user is only required to provide a path to an embeddings file in CSV form. These embeddings can be used in two steps: (i) in Block Building, combining them with LSH as in [4], and (ii) in Entity Matching, providing external, contextual information for the computation of value similarities, which is particularly useful for noisy entity profiles [12].

Another major improvement is the combination of multiple Entity Matching methods. It is now possible to select any (reasonable) number of algorithms, similarity measures and representation models for comparing the candidate matches that are contained in a set of blocks. For example, it is possible to combine the traditional bag-of-words model with the popular word2vec embeddings, each coupled with a different similarity measure. JedAI combines the similarity scores produced by the individual methods and normalizes them into the $[0, 1]$ interval.

Finally, several steps of this workflow have been enriched with new techniques. For example, Data Reading and Data Writing now support HDT and JSON files, while Entity Clustering conveys two new algorithms for Clean-Clean ER that are designed for solving the assignment problem: an efficient approximation called *Row-Column Proxy Clustering* [9], and a heuristic algorithm called *Assignment Problem Heuristic Clustering* [2].

3 JOIN-BASED ER WORKFLOW (NEW)

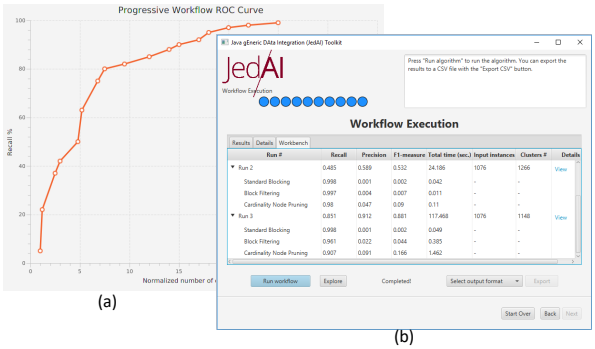
Similarity joins [7, 10] constitute a rather efficient alternative to blocking-based ER, especially for structured data that conforms to a schema of known quality. These joins accelerate the execution of matching rules and are combined with an Entity Clustering algorithm for high effectiveness [6], as shown in Figure 2(b). For example, consider the atomic (R1, R2) and composite (R3) matching rules in Figure 3, which exhibit very high effectiveness in combination with Connected Components clustering over established benchmark datasets [13].

JedAI-core now implements the workflow in Figure 2(b), conveying a library with the state-of-the-art string [7] and set [10] similarity join techniques. Most of them require that the user is familiar with the schema describing the entity profiles so as to select the most suitable attribute names. JedAI-gui facilitates this process through the *data exploration* feature, which allows for observing the schema and the values of entity profiles. Most importantly, JedAI goes beyond this schema-aware approach, applying similarity joins to semi-structured data through a schema-agnostic transformation that considers all tokens (or q-grams) in all attribute values (e.g., R2 in Figure 3). The only caveat comes from the resulting low similarity thresholds that render inapplicable character-based and token-based methods that are crafted for much larger thresholds [7, 10]. JedAI covers such cases by incorporating novel join techniques that inherently support low similarity thresholds, like SilkMoth [3] and Atlas [21].

The list of the character- and token-based similarity joins that are currently supported by JedAI appears in Figure 4. It entails

Similarity Join Methods		Prioritization Methods
Token-based	Character-based	1) Local Progressive Sorted Neighborhood
1) AllPairs	1) AllPairs	2) Global Progressive Sorted Neighborhood
2) PPJoin	2) FastSS	3) Progressive Block Scheduling
3) SilkMoth	3) PassJoin	4) Progressive Entity Scheduling
	4) EdJoin	5) Progressive Global Top Comparisons
		6) Progressive Local Top Comparisons

Figure 4: The methods available for the workflow steps Similarity Join (Section 3) and Prioritization (Section 4).



(a) (b)

Figure 5: JedAI-gui reporting the performance of a Progressive ER workflow (a) and its benchmark screen (b).

most state-of-the-art approaches, as determined by recent experimental analyses [7, 10]. Any combination of matching rules and similarity join techniques is allowed to form atomic and composite schema-based or schema-agnostic pipelines. These can be readily juxtaposed to more complex blocking-based ones, providing a unique feature that is offered by no other relevant tool.

4 PROGRESSIVE ER WORKFLOW (NEW)

Progressive ER applies to *budget-aware applications*, which have limited computational or time resources. These limitations can only be addressed in a *pay-as-you-go* way that provides the best possible *partial solution* in the context of the available resources. For example, the Google dataset search system has indexed ~26 billion datasets [5], which can only be resolved progressively.

Schema-based progressive methods have been proposed [16, 20], but JedAI exclusively considers domain-agnostic ones [19], implementing the workflow in Figure 2(c). The first four steps are common with the blocking-based ER workflow, which is depicted in Figure 2(a). The only difference is that Data Reading also receives as input the user-specified *budget*, either in terms of the maximum running time or the maximum number of executed comparisons. Next, *Prioritization* is applied, assigning a weight to all entities, comparisons or blocks in order to schedule their processing in decreasing likelihood that they involve duplicates. Then, the top-weighted entity pairs are iteratively emitted, one at a time, in order to compare the corresponding entity descriptions (Entity Matching). Finally, the Evaluation estimates the rate of detected duplicates per comparison, i.e., the evolution of recall as more comparisons are executed. The resulting diagram is used for estimating the area under curve, which is analogous to performance - see Figure 5(a).

JedAI implements the Prioritization methods in Figure 4, which can be distinguished into two types:

i) Inspired by Sorted Neighborhood (SN), the *sort-based methods* rely on the similarity of blocking keys. They produce a list of entities by sorting all descriptions alphabetically, according to the corresponding blocking keys. In the schema-agnostic context, every token forms a blocking key and thus, every entity appears in the list as many times as the number of its distinct attribute value tokens. To avoid the large number of redundant

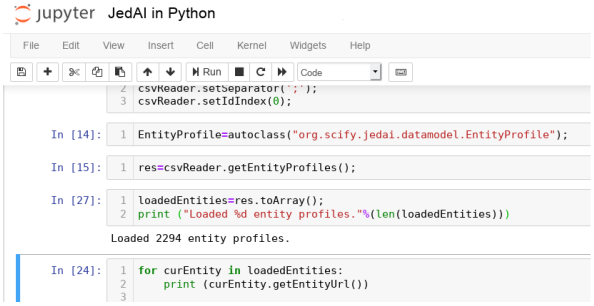


Figure 6: JedAI in Python.

comparisons and the arbitrary ordering of entities with identical keys, *Local Schema-agnostic Progressive SN* [19] weights all comparisons within the current window size via a schema-agnostic function that considers the frequency of appearance of every entity in the list along with the co-occurrence frequency of entity pairs for the current window size. The *Global Schema-agnostic Progressive SN* [19] does the same, but for a predetermined range of windows, eliminating their redundant comparisons.

ii) The *hash-based methods* are based on identical, schema-agnostic blocking keys and the resulting overlapping blocks. Similar to Meta-blocking, they assign a weight to every pair of candidate matches, assuming that their similarity is proportional to the number of blocks they share [14]. *Progressive Block Scheduling* [19] orders the blocks in ascending number of comparisons and then prioritizes all comparisons in the current block, by ordering them in decreasing weight. *Progressive Profile Scheduling* [19] orders entities in decreasing average weight of the corresponding candidate matches and then prioritizes all comparisons involving the current entity by ordering them in decreasing weight. *Progressive Global (Local) Top Comparisons* considers the top-K weights over the entire blocking graph (per entity), where K is derived from the given budget.

5 USER INTERFACE

Up to v2.1, JedAI offered two interfaces for user interaction [15]: (i) the desktop application, JedAI-gui, which offers a graphical interface, and (ii) the command-line interface implemented by JedAI-core. Both of them allow for constructing any combination of the available methods in the context of the blocking-based end-to-end ER workflow. Especially, JedAI-core conveys the Documentation module (see Figure 1), which facilitates the use and configuration of ER methods. It also enables the benchmarking of different workflows or configurations over a particular dataset through the workbench window, which summarizes the outcome of all runs and maintains details about the performance and the configuration of every step [15] (see Figure 5(b)).

5.1 New features

JedAI[3] extends both interfaces so that they cover all new features discussed above. The command-line interface has also been enriched with documentation support: at any step, the user is able to retrieve information about individual methods or specific parameters, thus facilitating their use. JedAI-core has also been augmented with a Python wrapper based on `pyjnius`, thus facilitating its adoption by the large user base of Python data analytics (see Figure 6). For example, the new wrapper allows for integrating JedAI with popular frameworks like `scikit`[4] for machine learning and `NLTK`[5] for natural language processing.

[4]https://scikit-learn.org
[5]https://www.nltk.org

6 DEMONSTRATION SCENARIOS

In this demo, we will present JedAI through a live interaction with users so as to highlight all new features discussed above. First, the user is asked to choose among the available interfaces: command-line, JedAI-gui or a Jupyter notebook. Then, she is asked to select one dataset among a set of carefully selected ones, which are easy to comprehend, yet interesting for an ER application, while involving both a structured and a semi-structured part (e.g., CSV-XML). These settings lay the ground for the following demo scenarios that showcase JedAI's functionalities.

1. Versatility. In this scenario, we compare the new ER workflows supported by JedAI with the blocking-based one that lies at the core of the previous version. Four different workflows are involved in this process:

(1) The user builds a traditional blocking-based workflow to be applied to the selected dataset.
(2) The same workflow is enhanced with word embeddings so as to examine the effect of deep learning techniques on improving accuracy.
(3) The user is asked to form a join-based end-to-end ER workflow with matching rules of arbitrary complexity; this might seem a complex procedure, but in reality, JedAI's data visualization feature simplifies it to a large extent.
(4) The user forms a Progressive ER workflow and applies it to the same dataset.

In all cases, the user can manually configure all methods, or use the recommended default parameter values. JedAI's workbench functionality allows to easily compare the performance of these fundamentally different workflows, assessing their pros and cons.

2. Automatic Configuration. In this scenario, we fine-tune the configuration parameters of the above four workflows in one of the available automatic ways, i.e., through random or grid search in a step-by-step or a holistic approach. Thus, the goal of this scenario is three-fold: (i) to test how well users can manually tune the parameters, (ii) to evaluate how close the default parameter values are to the "optimal" ones, and (iii) to compare the four workflows in terms of their best possible performance. For example, the performance of the fine-tuned progressive workflow demonstrates the minimum number of comparisons that are required for achieving sufficiently high recall. How close are the other three workflows to this ideal case? Again, JedAI's workbench functionality, the first for such a tool, renders these complex comparative process into a simple procedure.

3. Scalability. This last scenario focuses on the time efficiency of the selected workflows. The results of the previous scenarios advise us beforehand about the relative running time of the selected workflows, as JedAI's workbench feature reports both effectiveness and efficiency measures. Here, though, we examine how well each workflow scales to datasets of increasing size (10K, 50K, 100K, 500K, 1M and 2M entities), which have been derived from the selected dataset using artificial noise. This scenario demonstrates how workflows with similar running times over small datasets might end up differing by orders of magnitude. Most importantly, JedAI's workbench reports the running time per workflow step, thus facilitating the detection of bottlenecks.

Finally, it is worth stressing that during our demonstration, we will take special care to explain to users how to make the most of JedAI's functionalities, emphasizing the new features. Note also that most of the features listed above are demonstrated for the first time, and are not supported by any other ER system.

7 CONCLUSIONS

JedAI^3 is an open-source ER tool with 4 unique characteristics: (i) Based on blocking and similarity joins, it implements two different end-to-end workflows for batch ER that allow for composing millions of pipelines. (ii) It supports pre-trained embeddings of any kind. (iii) It supports budget-aware ER, enabling thousands of schema-agnostic progressive workflows. (iv) It can be integrated with Python's data analysis ecosystem. These capabilities will be exhibited through a live demonstration with user interaction.

Acknowledgements. This work was partially funded by the EU H2020 projects ExtremeEarth (825258) and SmartDataLake (825041).

REFERENCES

[1] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.
[2] Y. Chuang, H. Lee, and S. Tan. A robust heuristic method on the clustering-assignment problem model. *Computers & Industrial Engineering*, 98:63–67, 2016.
[3] D. Deng, A. Kim, S. Madden, and M. Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *PVLDB*, 10(10):1082–1093, 2017.
[4] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.
[5] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *PODS*, 2017.
[6] O. Hassanzadeh, F. Chiang, R. Miller, and H. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1), 2009.
[7] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.
[8] P. Konda, S. Das, et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.
[9] J. M. Kurtzberg. On approximation methods for the assignment problem. *J. ACM*, 9(4):419–439, 1962.
[10] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *PVLDB*, 9(9):636–647, 2016.
[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.
[12] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *SIGMOD*, pages 19–34, 2018.
[13] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *PVLDB*, 9(4):312–323, 2015.
[14] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.
[15] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.
[16] T. Papenbrock, A. Heise, and F. Naumann. Progressive duplicate detection. *IEEE TKDE.*, 27(5):1316–1329, 2015.
[17] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.
[18] G. Simonini, S. Bergamaschi, and H. V. Jagadish. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12):1173–1184, 2016.
[19] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi. Schema-agnostic progressive entity resolution. In *ICDE*, pages 53–64, 2018.
[20] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *TKDE*, 25(5):1111–1124, 2013.
[21] J. Zhai, Y. Lou, and J. Gehrke. ATLAS: a probabilistic algorithm for high dimensional similarity search. In *SIGMOD*, 2011.