# The return of JedAI: End-to-End Entity Resolution for Structured and Semi-Structured Data

George Papadakis[1], Leonidas Tsekouras[2], Emmanouil Thanos[3],
George Giannakopoulos[2], Themis Palpanas[4], Manolis Koubarakis[1]
[1]University of Athens, Greece {gpapadis,koubarak}@di.uoa.gr
[2]NCSR "Demokritos", Greece {ltsekouras, ggianna}@iit.demokritos.gr
[3]KU Leuven, Belgium emmanouil.thanos@kuleuven.be
[4]Paris Descartes University, France themis@mi.parisdescartes.fr

## ABSTRACT

JedAI is an Entity Resolution toolkit that can be used in three ways: *(i)* as an open-source library that combines state-of-the-art methods into a plethora of end-to-end workflows, *(ii)* as a user-friendly desktop application with a wizard-like interface that provides complex, out-of-the-box solutions even to lay users, and *(iii)* as a workbench for comparing the performance of numerous workflows over both structured and semi-structured data. Here, we present its significant upgrade, JedAI 2.0, which enhances the original version in three important respects: *(i) time efficiency*, as the running time has been drastically reduced with the use of high performance data structures and multi-core processing, *(ii) effectiveness*, since we enriched its library with more established methods, a new layer that exploits loose schema binding as well as the automatic, data-driven configuration of individual methods or entire workflows, and *(iii) usability*, as the GUI now enables users to manually configure any method based on concrete guidelines, to store the matching results into any of the supported data formats and to visually explore both input and output data.

**PVLDB Reference Format:**
George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, Manolis Koubarakis. The return of JedAI. *PVLDB*, 11 (12): xxxx-yyyy, 2018.
DOI: https://doi.org/10.14778/3229863.3236232

## 1. INTRODUCTION

Entity Resolution (ER) constitutes a core task for data integration, identifying entity profiles that correspond to the same real-world objects, but are located in different data collections. Yet, the functionality of the available ER systems is significantly restricted by the format of the various data collections. We can actually distinguish the existing systems into two major categories: one crafted for *structured data*, which are described by a well-defined schema and reside in relational databases or CSV files, and one applying exclusively to *semi-structured data*, which are associated with loose, diverse schemata and are located in XML/RDF repositories or SPARQL endpoints. The former category is mainly represented by Magellan[1], Febrl[2] and Dedoop[3], while the latter is dominated by LIMES[4] and Silk[5]. A detailed overview of each category can be found in the extended version of [8] and in [10], respectively.


*Proceedings of the VLDB Endowment,* Vol. 11, No. 12

DOI: https://doi.org/10.14778/3229863.3236232

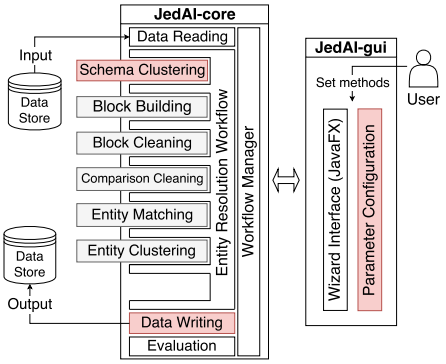


**Figure 1: JedAI's architecture, with the extensions highlighted in red.**

To facilitate researchers, practitioners and simple users in applying ER to *any type of data*, we presented the **Java gEneric DAta Integration** (JedAI)[6] toolkit in [14]. At its core lies a novel *end-to-end ER workflow* that applies uniformly to structured and semi-structured data. As shown in Figure 1, this workflow is implemented by JedAI's *back end*, called **JedAI-core**[7], which is an open-source library that includes several state-of-the-art ER methods for each step. In more detail, these steps are the following:

i) *Data Reading* loads from the disk into main memory the data collection(s) to be processed along with the respective golden standard. It supports the following data formats: CSV, XML, OWL, RDF and relational databases. It accommodates all of them via a simple name-value pairs model.

[1]https://sites.google.com/site/anhaidgroup/projects/magellan
[2]https://sourceforge.net/projects/febrl
[3]https://dbs.uni-leipzig.de/dedoop
[4]http://aksw.org/Projects/LIMES.html
[5]http://silkframework.org
[6]http://jedai.scify.org
[7]Code available under Apache License V2.0 at: https://github.com/scify/JedAIToolkit.

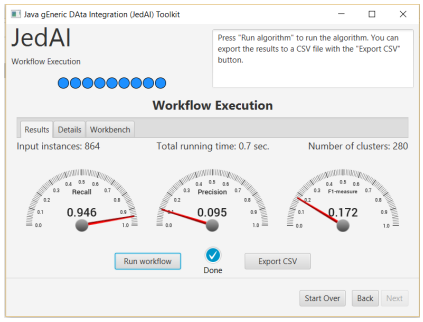


**Figure 2: Evaluation screen.**

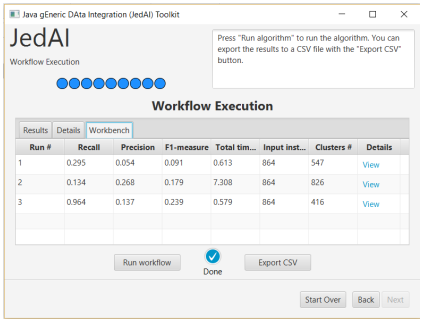


**Figure 3: Workbench screen.**

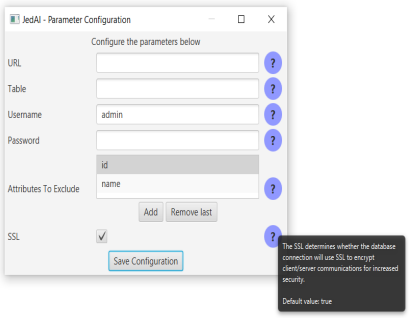


**Figure 4: Manual configuration.**

ii) *Block Building* clusters very similar entities into blocks so as to drastically reduce the candidate match space and to cut down on the running time. It includes 7 established methods that achieve high recall through *redundancy* [11], i.e., they extract several schema-agnostic signatures from every entity, placing it into multiple blocks.

iii) *Block Cleaning* further improves time efficiency by cleaning the original set of *overlapping* blocks from those dominated by *redundant* or *superfluous* comparisons (the former are repeated across different blocks, while the latter involve non-matching entities) [11]. This step includes 3 methods that are complementary to each other.

iv) *Comparison Cleaning* includes 7 competitive methods that serve the same purpose as Block Cleaning, but operate at the level of individual comparisons. They offer a more accurate functionality at the cost of lower time efficiency.

v) *Entity Matching* conveys several methods for carrying out all comparisons in the final set of blocks. Then, it creates a *similarity graph*, with one node for every entity and a weighted edge for every pair of compared entities.

vi) *Entity Clustering* involves 7 established methods [6] that partition the nodes of the similarity graph into *equivalence clusters* such that every cluster contains all entities corresponding to the same real-world object.

vii) *Evaluation* estimates the performance of the identified equivalence clusters with respect to the golden standard that was specified in Step 1. To this end, it employs a series of measures for effectiveness and time efficiency.

Regarding the *front end*, it consists of **JedAI-gui**[8], an open-source desktop application with an intuitive GUI that is suitable for both expert and lay users. It is based on a user-friendly wizard that allows to build ER workflows in a straightforward way, simply by selecting among the available methods for every workflow step. No manual fine-tuning is required from the user, since every method in JedAI-core involves an unsupervised functionality, independent from domain knowledge, and is associated with a default parameter configuration that consistently achieves high performance [13] - the only exception is the Data Reading step, where the user has to specify the format of the input data.

JedAI-gui can also be used as a *workbench*: the available methods result in more than 4,000 different combinations, i.e., ER workflows, whose performance can be easily compared through the GUI. Figure 2 illustrates the performance report for an individual workflow, while Figure 3 depicts the workbench functionality, with every line reflecting the performance of a different workflow run.

[8]Code available Apache License V2.0 at `https://github.com/scify/jedai-ui`.

## 2. NEW FEATURES IN JEDAI 2.0

In this demonstration, we will present **JedAI 2.0**, which extends the original version in three respects. First of all, we have significantly enhanced the *time efficiency* in two ways:

1) We have replaced the inverted indexes of Lucene[9] as well as the native data structures of Java with the high performance structures of **GNU Trove**[10]. The new library operates on primitive data types instead of objects, reducing the memory footprint of JedAI-core's data structures by up to 75%. For example, collections of integer values are handled through the 4-byte `int` type instead of the 16-byte `Integer` objects. Most importantly, Trove conveys significant gains in running time, as demonstrated in Figure 5: compared to Lucene, the running time of the Block Building step has been reduced by almost an order of magnitude (by 85% on average, across all methods), whereas compared to native Java, it has been reduced at least to the half (by 58% on average). Similar gains apply to the methods of all other workflow steps.

2) To further curtail the running time of JedAI, we have enabled the **in-memory multi-core execution** of every method supported by JedAI. This was accomplished by generalizing the parallelization approach that achieved the highest speedup in [12]. Thus, JedAI 2.0 is able to fully exploit the processing power of any multi-core CPU simply by specifying the number of available cores.

Second, we upgraded the *effectiveness* of JedAI as follows:

3) We have enriched every workflow step with **new methods**. Data Reading now supports SPARQL endpoints, while Block Building has been extended with a schema-agnostic version of LSH blocking. Block Cleaning now incorporates a clustering-based approach for controlling block sizes [4]. In Comparison Cleaning, we have added the state-of-the-art meta-blocking method BLAST [15] along with schema-agnostic Canopy Clustering [11]. The latter is typically considered as a block building method that depends on the blocks provided by Q-Grams Blocking [2]. JedAI 2.0 decouples Canopy Clustering from Q-Grams Blocking to render it compatible with any Block Building method, essentially treating it as a generic Comparison Cleaning method. Finally, Entity Matching is extended with an adapted version of the greedy algorithm SiGMa [9].

4) A new step has been added at the start of JedAI's end-to-end workflow, called **Schema Clustering** (see Figure 1). Its functionality is similar to Schema Matching in the sense that it yields a mapping between attributes based

[9]`https://lucene.apache.org/core`

[10] `https://bitbucket.org/trove4j/trove`

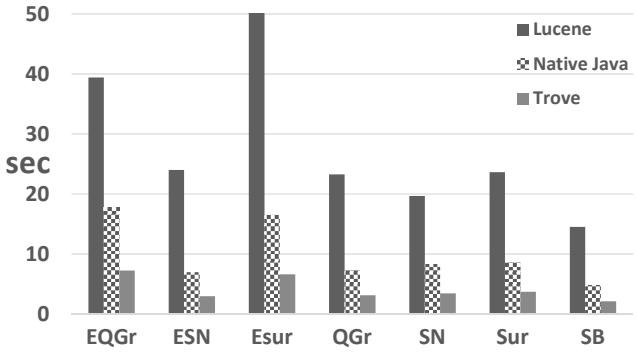


**Figure 5: The overall time in seconds that is required on average, after 10 repetitions, for applying every Block Building method supported by JedAI 2.0* to all datasets provided in our data repository**. The experiments were performed on a laptop with an Intel i7-4710MQ (2.50GHz) and 16GB RAM, running Windows 10.**

***These methods are (Extended) Q-Grams Blocking - (E)QGr, (Extended) Sorted Neighborhood - (E)SN, (Extended) Suffix Arrays - (E)Sur and Standard Blocking - SB.**

****https://github.com/scify/JedAIToolkit/tree/mavenizedVersion/jedai-core/data**

on their relatedness, as inferred from the similarity of their structure, name and values. Yet, its purpose is fundamentally different: instead of seeking semantically identical attributes (e.g., "profession" and "job"), its goal is to improve the creation and processing of schema-agnostic blocks. This is accomplished by splitting large blocks into smaller ones according to the schema clusters that are associated with every signature. For example, consider a signature "Washington" that has been extracted from the attribute "name" of entities $e_1$ and $e_2$ and from the attribute "location" of entities $e_3$ and $e_4$; in a completely schema-agnostic functionality, all these entities will be placed in the same block, yielding (4-1)·4/2=6 comparisons, while in JedAI 2.0, there will be two different blocks, $\{e_1\text{-}e_2\}$ and $\{e_3\text{-}e_4\}$, with each one containing a single comparison; the only requirement is that "name" and "location" are placed in different clusters, irrespective of the semantics of the other attributes that are associated with each one of them. In this way, precision is significantly enhanced as long as there is a limited (if any) impact on recall. This idea has been successfully applied to blocking via Attribute Clustering [13] and to meta-blocking via BLAST [15]. JedAI 2.0 generalizes it to cover all workflow steps. This new layer includes some state-of-the-art methods from the literature [3] plus Attribute Clustering.

5) We have added the **automatic parameter configuration** of individual methods and entire workflows, enabling users to investigate the impact of fine-tuning on the quality of results through JedAI's workbench functionality. The automatic configuration applies *grid search* [1]; yet, instead of an exhaustive trial of all valid values, every parameter is associated with a limited set of reasonable settings that are typically used by experts in practice [13]. For methods with multiple parameters, *random search* can be used to find the best balance between effectiveness and time efficiency [1]. The same procedure is extended to cover entire workflows, allowing users to examine the interplay among the configuration of consecutive methods. This approach is scalable, thanks to JedAI's enhanced time efficiency (Trove & multi-core processing).

Finally, JedAI 2.0 goes beyond the original version in terms of *usability*. The following new features have been added:

6) Both the back and the front end have been extended to support the **manual fine-tuning** of every available method. JedAI-core has enriched every method with a JSON file that provides information about its parameter configuration; this information includes the name and a short description of each parameter, the type of values it receives (e.g., an integer or real number), the range of acceptable values as well as the default value that was found to consistently achieve high performance through an extensive experimental study [13]. JedAI-gui presents this information to the user in the form of tooltips that pop-up in the configuration windows. An example of these new windows is shown in Figure 4. Through the workbench functionality, the user can then observe the effect of every parameter value to the overall performance.

7) A new step has been added at the end of JedAI's ER workflow, called **Data Writing** (see Figure 1). Its goal is to store all pairs of identified matches into any of the supported data formats. In case a structured format is selected (CSV or relational database), the output retains the original entity ids. When selecting a semi-structured format (XML, RDF or SPARQL endpoint), the user has to specify the URI prefix in case it is not available, i.e., when the original data were structured. To store the output to relational databases or SPARQL endpoints, the user should also provide the necessary credentials, if applicable, along with the table and the dataset namespace, respectively.

8) JedAI-gui now offers a **data exploration** functionality. After selecting the data to be processed, the user is able to go through the entity profiles that have been loaded in memory, observing their schema and attribute values as well as the level of noise and heterogeneity they contain. Similarly, by the end of the workflow execution, the user can examine the equivalence clusters that have been formed, assessing the quality of the results.

## 3. DEMONSTRATION SCENARIO

This demonstration presents JedAI through a live interaction with users, highlighting most new features.

Using JedAI-gui, the user is asked to select among several pairs of data collections that are overlapping, containing duplicate entities. Each pair consists of a structured data collection (in CSV or SQL), a semi-structured one (in RDF, OWL, XML or SPARQL) and the corresponding golden standard. After loading the entity profiles in memory, the user examines them through the data exploration functionality, and forms an ER workflow of arbitrary complexity to process them. This is done in four iterations:

1) In the first execution, the default configuration parameters for all methods of the user-defined workflow is employed. The execution uses the multi-core setting.

2) In the second iteration, the automatic parameter configuration is applied independently to every selected method. Again, the multi-core setting is employed.

3) In the third round, the automatic parameter configuration of the entire ER workflow is applied, investigating whether it yields significantly better effectiveness. Due to the multi-core processing and the high efficiency provided by Trove, this iteration is completed within a few minutes.

4) The workflow with the optimal performance is formed and fine-tuned via the manual configuration provided by JedAI-gui and then is carried out using serial processing.

Finally, the identified matches are seen through the data exploration functionality, and are stored in the data format that is chosen by the user.

In a nutshell, our demonstration scenario aims to teach users how to make the most of JedAI's functionalities, emphasizing the new features of version 2.0 that take JedAI to the next level. Indeed, users will be able to evaluate the effect of parameter configuration in individual methods or entire workflows on the overall ER performance and to assess the speedup achieved by end-to-end multi-core processing. Through the workbench functionality, they will also examine the relative performance of different state-of-the-art workflows and will investigate the role played by any individual method in a particular ER workflow.

# 4. RELATION TO OTHER TOOLS

As explained above, two types of Entity Resolution systems have been developed so far. The first one includes Link Discovery frameworks, which are crafted for semi-structured data. The most prominent representatives are LIMES and Silk, while 8 more tools are surveyed in [10]. Unlike JedAI, none of them is applicable to structured data, while half of them lack a GUI. Most importantly, though, these tools typically implement only the method(s) introduced by their creators and are suitable only for experts: they require the manual configuration of matching rules, or a labeled dataset for learning such rules in a supervised way [7].

The second category involves ER tools that apply exclusively to structured data. The extended version of [8] provides a thorough list of 18 non-commercial and 15 commercial systems. Most of them suffer from one or more of the following problems [8]: they cover the Entity Resolution pipeline partially, they constitute stand-alone systems that are hard to extend with new functionality, even though they offer a limited variety of methods, or they are exclusively meant for expert users, providing insufficient guidelines to lay users on how to perform ER efficiently and effectively. Magellan, which lies at the core of BigGorilla's Data Matching process (`https://www.biggorilla.org`), resolves these issues [8], but is restricted to relational data, lacks a GUI (it offers a command-line interface) and requires heavy user involvement, as its goal is actually to facilitate the development of tailor-made methods for the data at hand.

On the whole, JedAI is the only system that applies uniformly to structured and semi-structured data, while conveying one of the largest libraries with state-of-the-art ER methods. No other toolkit exploits the benefits of schema-agnostic blocking, which minimizes user involvement, while maximizing recall [11], nor does it include the Block and Comparison Cleaning steps, which are indispensable for enhancing the time efficiency by orders of magnitude [13]. JedAI is also one of the few ER systems that are suitable for lay users, providing an intuitive GUI, and one of the few systems to support hands-off ER, as the default configuration of every method saves the cost of parameter fine-tuning.

# 5. CONCLUSIONS

We presented JedAI 2.0, the enhanced version of a user-friendly toolkit that bridges the gap between ER methods for structured and semi-structured data. We have improved its time efficiency by at least an order of magnitude, we extended its effectiveness with more methods, layers and configuration options and we enriched its GUI with advanced functionalities. All these new capabilities will be exhibited through a live demonstration that involves user interaction. Our demonstration will also stress how JedAI fulfills the two main challenges that arise in data integration [5], i.e., the development of extensible, open-source tools and the provision of solutions that apply not only to structured, but also to semi- or even un-structured data.

In the future, we plan to extend JedAI with more methods per workflow step (e.g., support for NoSQL databases and JSON files in Data Reading) and to exploit Apache Spark for massive parallelization. We will also examine the possibility to include more specialized methods, e.g., for product deduplication [16, 17, 18].

**Acknowledgements.** This work has been supported by the project Optique, which is funded by the Seventh Framework Program of the European Commission under Grant Agreement 318338.

# References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9):1537–1555, 2012.

[3] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, and et al. Instance-based matching of large ontologies using locality-sensitive hashing. In *ISWC*, pages 49–64, 2012.

[4] J. Fisher, P. Christen, Q. Wang, and E. Rahm. A clustering-based framework to control block sizes for entity resolution. In *KDD*, pages 279–288, 2015.

[5] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. Tan. Data integration: After the teenage years. In *ACM PODS*, pages 101–106, 2017.

[6] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[7] R. Isele and C. Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.

[8] P. Konda and S. D. et. al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[9] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, and Z. Ghahramani. Sigma: simple greedy matching for aligning large knowledge bases. In *KDD*, pages 572–580, 2013.

[10] M. Nentwig, M. Hartung, A. Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.

[11] 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.

[12] G. Papadakis, K. Bereta, T. Palpanas, and M. Koubarakis. Multi-core meta-blocking for big linked data. In *SEMANTICS*, pages 33–40, 2017.

[13] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9):684–695, 2016.

[14] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. Jedai: The force behind entity resolution. In *ESWC (demo paper)*, 2017.

[15] 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.

[16] R. van Bezu, S. Borst, R. Rijkse, J. Verhagen, D. Vandic, and F. Frasincar. Multi-component similarity method for web product duplicate detection. In *SAC*, pages 761–768, 2015.

[17] I. van Dam, G. van Ginkel, and et al. Duplicate detection in web shops using LSH to reduce the number of computations. In *SAC*, pages 772–779, 2016.

[18] G. van Rooij, R. Sewnarain, M. Skogholt, T. van der Zaan, F. Frasincar, and K. Schouten. A data type-driven property alignment framework for product duplicate detection on the web. In *WISE*, pages 380–395, 2016.