Domain- and Structure-Agnostic End-to-End Entity Resolution with JedAI

George Papadakis[1], Leonidas Tsekouras[2], Emmanouil Thanos[3], George Giannakopoulos[2], Themis Palpanas[4], Manolis Koubarakis[1]
[1]National and Kapodistrian 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

We present JedAI, a new open-source toolkit for end-to-end Entity Resolution. JedAI is *domain-agnostic* in the sense that it does not depend on background expert knowledge, applying seamlessly to data of any domain with minimal human intervention. JedAI is also *structure-agnostic*, as it can process any type of data, ranging from structured (relational) to semi-structured (RDF) and un-structured (free-text) entity descriptions. JedAI consists of two parts: *(i)* JedAI-core is a library of numerous state-of-the-art methods that can be mixed and matched to form (thousands of) end-to-end workflows, allowing for easily benchmarking their relative performance. *(ii)* JedAI-gui is a user-friendly desktop application that facilitates the composition of complex workflows via a wizard-like interface. It is suitable for both lay and power users, offering concrete guidelines and automatic configuration, as well as manual configuration options, visual exploration, and detailed statistics for each method's performance. In this paper, we also delve into the new features of JedAI's latest version (2.1), and demonstrate its performance experimentally.

1. INTRODUCTION

Entity Resolution (ER) aims to detect different entity profiles that describe the same real-world objects [4]. It is a core task for data integration, with many applications that range from knowledge bases to question answering [6]. 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 those crafted for *structured* (relational) data that is described by a well-defined schema, and those applying exclusively to *semi-structured data* that is associated with loose, diverse schemata and resides in XML/RDF repositories or SPARQL endpoints.

The latter category encompasses Link Discovery frameworks, which are surveyed in [20]. LIMES[1] and Silk[2] are the most prominent representatives. However, most of these tools implement only the method(s) introduced by their creators, and/or are suitable for power users, requiring the manual configuration of matching rules, or a labeled dataset for learning such rules in a supervised way [14]. Another drawback is that none of them is applicable to structured data, while half of them lack a GUI [20].

A larger variety of tools is available for structured data. A thorough list of 15 commercial and 18 non-commercial systems (such as Febrl[3] and Dedoop[4]) is analyzed in [15]. Most of them, though, suffer from one or more of the following problems: they cover the ER pipeline partially, they constitute stand-alone systems with a limited variety of methods, or they are exclusively meant for power users, providing insufficient guidelines on how to perform ER efficiently and effectively [15]. Magellan [16] resolves these issues, offering various blocking and matching methods. However, it is restricted to Record Linkage over relational data, lacks a GUI (it merely offers 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. Similarly, heavy user involvement is required by the crowd-sourcing systems Corleone [11] and Falcon [5], which also address ER in an end-to-end manner through various efficiency techniques.

To overcome these drawbacks, we developed the **Java gEneric DAta Integration**[5] toolkit (JedAI for short), aiming to facilitate researchers, practitioners and lay users in applying ER solutions to any type of data. At its core lies the *end-to-end ER workflow* of Figure 1, which covers both Dirty ER (Deduplication) and Clean-Clean ER (Record Linkage). JedAI conveys one of the largest libraries

[1]http://aksw.org/Projects/LIMES.html

[2]http://silkframework.org
[3]https://sourceforge.net/projects/febrl
[4]https://dbs.uni-leipzig.de/dedoop
[5]http://jedai.scify.org

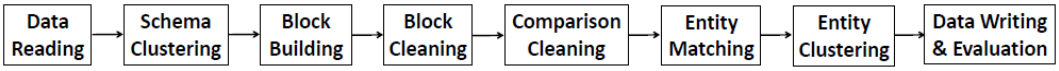

Figure 1: The end-to-end workflow for Entity Resolution implemented by JedAI.

with state-of-the-art ER methods, while being one of the few ER systems that is suitable even for lay users, providing an intuitive GUI. In more detail, JedAI has the following advantages:

1) *Structure-agnostic functionality.* JedAI applies uniformly to structured, semi-structured and unstructured (free-text) data.

2) *Domain-agnostic functionality.* All methods implemented by JedAI apply to data from any domain, ranging from homogeneous census, customer, product and bibliographic data to heterogeneous Knowledge Bases. The only requirement is that their entities contain string-dominated values.

3) *High time efficiency.* JedAI offers the largest variety of blocking and block processing methods. No other toolkit exploits the benefits of schema-agnostic blocking, which minimizes user involvement, while maximizing recall [22]. Also, no other toolkit includes the Block and Comparison Cleaning steps, which are indispensable for enhancing the time efficiency of ER by orders of magnitude [26]. JedAI also uses GNU Trove[6] for minimizing its memory footprint. This is done by operating on primitive data types instead of objects. E.g., collections of integer values are handled through the 4-byte `int` type instead of the 16-byte `Integer` objects, thus occupying up to 75% less memory. This also reduces the running time by more than 50% when compared to native Java [28].

4) *Hands-off functionality.* JedAI couples every implemented method with a default configuration of its internal parameters. Thus, neither manual parameter fine-tuning nor expert knowledge are required for building an end-to-end ER workflow; users simply select one or more methods per step.

5) *Learning-free functionality.* None of the implemented methods requires a labelled dataset for its training. Their default configuration renders them directly applicable to any data. Labelled data in the form of all true matches in a dataset (i.e., positive instances) are only required for evaluating the performance of a method or workflow and for fine-tuning its configuration parameters. In contrast, the learning-based methods require a labelled dataset for their operation, i.e., in order to learn their blocking or matching model. This labelled dataset includes not only the positive instances considered by JedAI, but also a carefully selected sample of non-matches (i.e., negative instances). The relative number of positive and negative instances as well as their representativity affects significantly the performance of the learned model. No such restrictions apply to JedAI's learning-free methods.

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

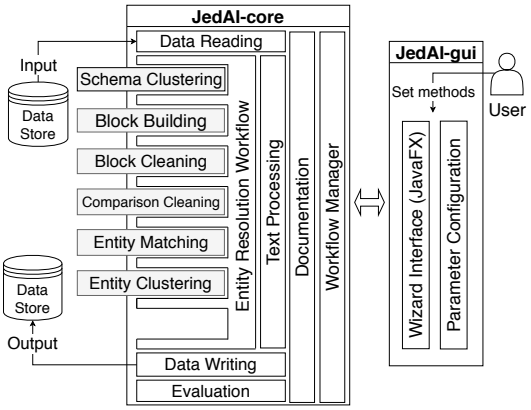

Figure 2: JedAI's architecture.

Most importantly, JedAI's learning-free methods optimize their performance by fine-tuning their internal parameters, which are generic in the sense that they are independent of the data at hand. In contrast, the features of learning-based methods are domain or dataset-specific, typically requiring heavy human intervention for their definition. Note also that JedAI's learning-free methods are inherently crafted for highly noisy and heterogeneous data [24]: to address possible errors in attribute values, their domain-agnostic functionality considers all values in each entity profile rather than relying on a particular (set of) attribute(s).

JedAI has been presented as a demo to two different communities, namely Semantic Web [27] and databases [28]. In this work, we present its structure and characteristics in more detail, introduce the new features of version 2.1, and provide experimental evidence of its performance over real data.

The rest of the paper is structured as follows: Section 2 delves into JedAI's architecture, Section 3 elaborates on the new features in version 2.1, Section 4 presents experiments that highlight the potential of JedAI, and Section 5 concludes the paper along with directions for future work.

2. ARCHITECTURE

JedAI's architecture appears in Figure 2. It is *modular* so that it can be easily extended by expert users in the future. Every component implements a simple (Java) interface such that every new class (algorithm) implementing it can be seamlessly added.

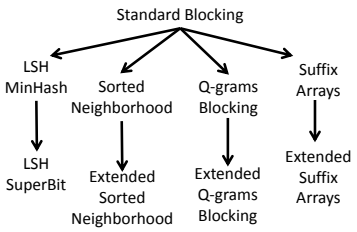

Figure 3: The Block Building methods.

JedAI consists of two parts: *(i)* JedAI-core[7], the back-end that essentially constitutes a library of state-of-the-art methods, and *(ii)* JedAI-gui[8], the front-end that facilitates the use of the library. Below, we describe each part in detail.

2.1 JedAI-core

Figure 1 depicts the workflow implemented by JedAI-core, which consists of the following eight steps:

1) *Data Reading* loads from the disk into main memory the data collection(s) to be processed along with the respective golden standard. The following data formats are supported: CSV, XML, OWL and RDF files as well as relational databases and SPARQL endpoints. Any mixture of these formats is possible in case of Clean-Clean ER. This is made feasible by transparently converting entities of any format into a flat name-value pairs model.

2) *Schema Clustering* is an optional step that groups together syntactically similar attributes, which share similar names and/or values. Unlike Schema Matching, this step does not seek semantically identical attributes (e.g., "place" and "location"). Instead, it aims to improve the performance of the next steps by raising precision significantly with no impact on recall [28]. Three methods are currently supported, Attribute Value Clustering, Attribute Name Clustering, Attribute Holistic Clustering [23], but at most one of them can be added in a workflow. They can be combined with any technique offered by the Text Processing component (see Figure 2) for comparing aggregations of strings. All pairs of attributes with a similarity above a% of the maximum similarity for either attribute are placed into the same cluster [29].

3) *Block Building* clusters similar entities into blocks so as to drastically reduce the candidate match space and to cut down on the running time. JedAI includes the nine methods in Figure 3, where every edge $A \rightarrow B$ denotes that method B is built on top of A, using the same core structures in a different way. For more details on the internal functionality of each method, please refer to [22]. All methods operate in a schema-agnostic fashion, extracting several signatures from every entity to place it into multiple blocks. The resulting redundancy yields high recall at the cost of low precision [4, 22].

[7]Code available under Apache License V2.0 at: `https://github.com/scify/JedAIToolkit`

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

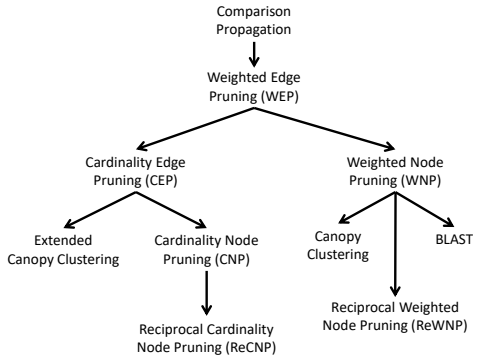

Figure 4: Comparison Cleaning methods.

4) *Block Cleaning* is an optional step that improves time efficiency by cleaning the original set of *overlapping* blocks from *redundant* and *superfluous* comparisons; the former are repeated across different blocks, while the latter involve non-matches [6, 22]. This step includes three complementary methods that operate at the level of entire blocks: Block Purging, Block Filtering, Block Clustering [8, 25].

5) *Comparison Cleaning* is another optional step that targets superfluous and redundant comparisons. Unlike Block Cleaning, though, it operates at the finer level of individual comparisons, offering a more accurate functionality at a higher computational cost. This step includes the ten methods that are depicted in Figure 4, where every edge $A \rightarrow B$ denotes that method B extends method A. Most of them are Meta-blocking techniques, described in [6, 26]. Only one of them can be selected, as they are competitive with each other.

6) *Entity Matching* conveys two schema-agnostic methods that carry out all comparisons in the final set of blocks: Group Linkage [21] and Profile Matcher, a custom approach that aggregates all attribute values of an entity into a common representation. Both methods can be combined with a large variety of graph and bag representation models and several associated similarity metrics [10] that are implemented by the Text Processing component (see Figure 2). This step yields a *similarity graph*, where every node corresponds to an entity and a weighted edge connects every pair of compared entities.

7) *Entity Clustering* partitions the nodes of the similarity graph into equivalence clusters such that every cluster contains all entity profiles corresponding to the same real-world object. For Dirty ER, this step implements the seven most efficient state-of-the-art methods evaluated in [13]. For Clean-

Clean ER, it offers the prevalent method in the literature, namely Unique Mapping Clustering [18].

8) *Evaluation* estimates the performance of the end result with respect to the golden standard that was specified in Step 1. To this end, it reports a series of measures for effectiveness and time efficiency. This step also includes *Data Writing*, which stores intermediate or end results 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 databases or SPARQL endpoints, the user should also provide the necessary credentials, if applicable, along with the table and the dataset namespace, respectively.

Regarding the use of optional steps, Schema Clustering is indispensable for heterogeneous data sources that involve a large number of noisy attributes. In these settings, it reduces the computational cost of Block Building and provides useful information for Comparison Cleaning and Entity Matching. Block Cleaning is indispensable for block collections that exhibit a Zipf distribution, where the larger a block size is, the less blocks correspond to it. Comparison Cleaning should be used in all cases, at least for eliminating all redundant comparisons at no cost in recall via Comparison Propagation. In case of redundancy-positive blocks, a Meta-blocking approach should be used to drastically reduce the computational cost. The exact method that should be selected for every optional step depends on the data at hand, as there is no clear winner among them.

2.2 JedAI-gui

This desktop application conveys a user-friendly wizard that allows for building ER workflows in a straightforward way, simply by selecting among the available methods per workflow step. A unique characteristic is that it offers three configuration options that are suitable for both expert and lay users:

1) The *default configuration* associates every available method with recommended parameter values that consistently achieve high performance, as verified through an extensive experimental study [26].

2) The *manual configuration* leverages the Documentation component of JedAI-core (see Figure 2), which enriches every method with a JSON file that provides information about its parameter configuration: 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 its recommended default value. JedAI-gui presents this information to the user in the form of tooltips that pop-up in the configuration windows.

3) The *automatic configuration* currently accommodates two established approaches [1]: (i) *grid search* exhaustively applies to each parameter a set of reasonable settings that have been determined experimentally [26], or are typically used by experts in practice. (ii) *random search* iteratively tries arbitrary configurations that lay within the range of acceptable values for each parameter. Both approaches apply to individual methods and entire workflows. In the latter case, two operations are supported: (i) *holistic configuration*, where all parameters of all methods in a workflow are simultaneously optimized, and (ii) *step-by-step configuration*, where the parameters of each method are gradually optimized, independently of the others, following the workflow execution order.

To make the most of these options, JedAI-gui supports a *workbench* functionality. The evaluation window summarizes the performance of all experiments, enabling users to investigate the impact of parameter fine-tuning on the quality of results. This applies to all possible levels of granularity – from one or more parameters in a particular method to one or more methods in an entire ER workflow. The workbench functionality also facilitates the performance evaluation of the >10,000 different workflows that can be derived from the combination of the available methods per workflow step.

Another major characteristic of JedAI-gui is the *data exploration* functionality. After specifying the data to be processed, the user is able to go through the golden standard and the corresponding entity profiles, observing their properties as well as the level of noise and heterogeneity they contain. By the end of a workflow execution, the user can also examine the equivalence clusters that have been formed, assessing the quality of the results.

3. NEW FEATURES IN VERSION 2.1

The new JedAI version, 2.1, extends both JedAI-core and JedAI-gui. The latter is enriched with a hierarchically-structured benchmark screen, which is a crucial feature for the workbench functionality, as it allows users to review all aspects of performance per method. In this way, users can identify the weak link in an end-to-end workflow and assess whether a better parameter configuration is required or it should be substituted by another method.

Another important new feature is the command-line interface that has been added to JedAI-core. This allows developers to easily test changes made in the implementation of a method and to evalu-

ate new methods in the context of an end-to-end workflow. It also facilitates users to test JedAI on a server, exploiting much higher computational power than commodity hardware.

Additionally, we altered the way Block Building is handled by both the command-line interface and JedAI-gui. Instead of selecting a single method, they are now able to combine the results of multiple approaches. In this way, we satisfy a user requirement, which was articulated by businesses that applied JedAI to incomplete or noisy data. As an example, consider a customer database that abounds in profiles with missing or erroneous information. Applying a single block building technique, such as bigram blocking, yields a set of blocks with low levels of redundancy. This means that the block co-occurrence patterns for matching entities are scarce, downgrading the performance of Block and Comparison Cleaning methods, which are crucial for deriving high quality candidate matches [26]. To leverage their performance, we can apply them to the union of blocks formed by two or more block building methods (e.g., bigram and trigram blocking), which provides much denser co-occurrence patterns.

Another user requirement was to update JedAI's output. In version 2, it simply comprised equivalence clusters of matching entities, without providing any evidence for the degree of similarity. As a result, the end result of JedAI could not be refined by expert users or domain-specific applications that incorporate additional, contextual information. This is now resolved in version 2.1, as each pair of matching entities is associated with a confidence score that indicates their profile similarity.

Finally, several new methods have been added in JedAI-core, such as Correlation Clustering and chi-squared weighting scheme for Comparison Cleaning. We also integrated the results of Schema Clustering into Comparison Cleaning and Entity Matching. As indicated in [29], evidence from attribute clusters, such as entropy, enhances significantly the weights assigned to entity pairs in the sense that it facilitates the distinction between matching and non-matching ones. The same principle has been incorporated into Entity Matching, weighting the contribution of n-grams to the overall pair similarity according to the corresponding attribute clusters.

4. EXPERIMENTS

We now evaluate the performance of JedAI with respect to the state-of-the-art in the literature. To this end, we use the four real-world, structured, Clean-Clean ER datasets that were introduced in [17]. Their technical characteristics are listed in Table 1. Note that D_1 and D_2 entail product data, while D_3 and D_4 involve bibliographic data.

Dataset	D_1	D_2	D_3	D_4
$Source_1$	Abt	Amazon	DBLP	DBLP
$Source_2$	Buy	Google Pr.	ACM	Scholar
$Entities_1$	1,076	1,354	2,616	2,516
$Entities_2$	1,076	3,039	2,294	61,353
NVP_1	2,568	5,302	10,464	10,064
NVP_2	2,308	9,110	9,162	198,001
Duplicates	1,076	1,104	2,224	2,308
Cartesian Pr.	$1.16{\cdot}10^6$	$4.11{\cdot}10^6$	$6.00{\cdot}10^6$	$1.54{\cdot}10^8$

Table 1: Dataset technical characteristics. NVP stands for attribute name-value pairs.

We applied the following workflow to all of them: Token Blocking for Block Building, Block Purging with size constraints along with Block Filtering for Block Cleaning, Cardinality Node Pruning (CNP) for Comparison Cleaning, Profile Matcher for Entity Matching and Unique Mapping Clustering (UMC) for Entity Clustering. This workflow involves five parameters: (i) the maximum block size (Block Purging), (ii) the ratio of retained blocks (Block Filtering), (iii) the weighting scheme (CNP), (iv) the representation model in combination with the similarity metric (Profile Matcher), and (v) the minimum similarity for a pair of matches (UMC).

To explore the potential of this workflow, we fine-tuned these parameters in three ways: (i) step-by-step random configuration, where we used the methodology of [26] for independently optimizing each method until CNP[9] and the F-Measure for optimizing the last two methods, (ii) holistic random configuration, whose goal is to maximize the overall F-Measure, and (iii) step-by-step grid configuration, where we used the same criteria as the first case. We compare these configurations against three state-of-the-art domain-specific, learning-based systems: (i) COSY, which is a commercial system[10] that achieves the top performance in [17], (ii) DeepMatcher [19], and (iii) Magellan [16]. For the last two, we consider the top performance that is reported in [19] among all configurations and dataset versions.

The F-Measure of all systems is reported in Figure 5(a). We observe that JedAI outperforms all baseline systems over D_1 by 15% to 45%, depending on its configuration. For D_3, the differences between all approaches are negligible (±1.5%), as they all achieve practically perfect performance. For D_2 and D_4, DeepMatcher achieves the top performance

[9]This methodology aims to maximize for each method the measure $PC(B){\cdot}RR(B,B')$, where B stands for the input blocks, B' for the output ones, $PC(B)$ for the pairs completeness, i.e., the recall of blocks B, and $RR(B,B')$=1-$\frac{||B'||}{||B||}$ for the reduction ratio - the decrease in pairwise comparisons when transforming B into B'.

[10]Due to license restrictions, its name is not disclosed.

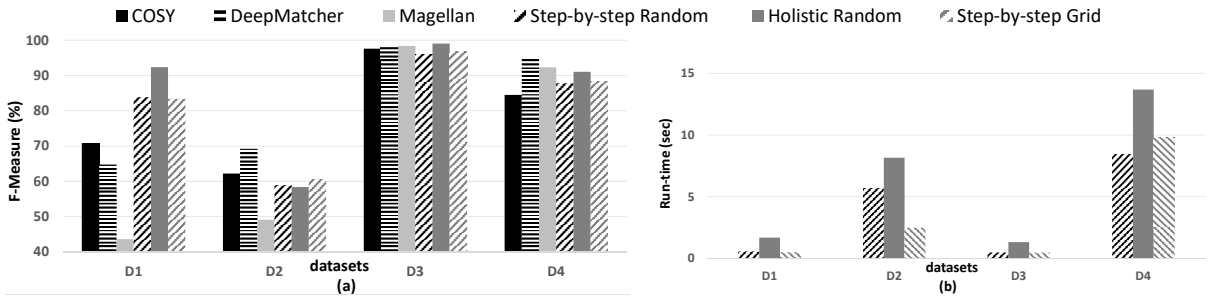

Figure 5: (a) Effectiveness of 3 different configurations of a JedAI workflow in comparison with 3 state-of-the-art domain-specific systems, and (b) the corresponding running times of JedAI.

to a significant extent, due to the external contextual information that is encapsulated in its features (i.e., word embeddings). JedAI is very close to COSY and outperforms Magellan to a significant extent over D_2, and vice versa over D_4.

It is worth associating these measurements with the corresponding time efficiency of each system. To measure JedAI's running time, we used a laptop with an Intel i7-4710MQ @ 2.50GHz, running Ubuntu 18.04.3 LTS and Java 8. We applied each configuration to every dataset[11], allocating 1 GB of RAM and performing 10 iterations with a clear Java cache. The average running times appear in Figure 5(b). We do not report the time performance of the baseline systems, as they all rely on manually-defined blocks of high performance, which are not reproducible, due to lack of details. We observe the high efficiency of JedAI's configurations, as they process every dataset within few seconds - less than 2 seconds for D_1 and D_3, from 3 to 8 seconds for D_2 and from 8 to 14 seconds for D_4. D_2 and D_4 are the most time consuming datasets, because they involve higher levels of noise and, thus, a larger number of candidate matches is processed. D_4 also involves the largest number of entities by far. Similar running times are reported in [17] for COSY, while Magellan and DeepMatcher require few seconds and few hours, respectively, for training their matching models, after having performed blocking with the help of an expert and labelling a considerable number of comparisons [19], operations that are very expensive in time. Therefore, we conclude that JedAI is much faster than the last two systems.

Another advantage of JedAI over Magellan is its ability to process Dirty ER datasets that Magellan cannot handle. This is illustrated in Table 2, which reports JedAI's performance over two established dirty datasets [3, 26]: Cora (1,295 entities, 17,184 pairs of duplicates) and CdDb (9,763 entities, 299 pairs of duplicates). We used the same system and approach for the time measurements and the same end-to-end workflow, except that UMC is replaced by Connected Components Clustering (recall that UMC does not apply to Dirty ER). We observe that JedAI achieves very high effectiveness, combined with very high time efficiency. The only exception is the high running time that is required for CdDb (between 2 and 4.5 minutes). This is caused by the large entity profiles, which involve 17.75 name-value pairs, on average (against 5.5 for Cora), and, thus, yield high levels of redundancy and a large number of candidate matches after CNP.

	Cora		CdDb	
	F-Measure	Run-time	F-Measure	Run-time
HRC	85.55%	514 msec	89.45%	266 sec
SRC	82.16%	294 msec	89.66%	155 sec
SGC	77.29%	3,752 msec	89.23%	127 sec

Table 2: Performance of JedAI over two Dirty ER datasets in combination with holistic random configuration (HRC) and step-by-step random and grid configuration (SRC and SGC, respectively).

Combined with the results in Figure 5, we observe a trade-off between the holistic and the step-by-step configurations. The former optimizes the parameters of all methods in an end-to-end workflow simultaneously: in every iteration, a new random, but valid value is assigned to each internal parameter and the iteration achieving the highest F-Measure is selected as optimal. As a result, holistic random configuration is crafted for identifying the global maximum, unlike the step-by-step configurations, which optimize every workflow step independently of the subsequent ones and, thus, are prone to confining themselves in local maxima. Yet, step-by-step configurations yield very low running times, because every method is fine-tuned to minimize its computational cost for the best possible effectiveness. In contrast, holistic random configuration consistently exhibits a significantly higher computational cost, since it exclusively considers the F-Measure in its optimization. As a result, its

[11]The corresponding code is available here: `https://github.com/scify/JedAIToolkit/tree/mavenizedVersion/jedai-core/src/test/java/org/scify/jedai/configuration/version2_1`.

blocking methods are configured to return a relatively high number of candidate matches.

On the whole, these results indicate that with proper configuration, JedAI produces learning-free, domain-agnostic workflows with an effectiveness that is comparable to, or better than, high-end, learning-based, domain-specific ER solutions, while exhibiting very low running times and limited memory consumption across various domains.

5. CONCLUSIONS

We presented JedAI, a user-friendly ER toolkit that fulfills the two main challenges arising in data integration [12]: the development of extensible, open-source tools, and the provision of solutions that apply not only to structured, but also to semi- or even unstructured data. We described JedAI's unique characteristics and elaborated on its main components, highlighting the new features in version 2.1. We demonstrated the high performance of its workflows and verified that it is ideal for the development phase of ER solutions, as it facilitates the identification of the best end-to-end workflows for a particular dataset and use case. Yet, the resulting workflow should be optimized for production systems.

In the future, JedAI will support pre-trained embeddings and will also allow for combining multiple algorithms and/or representations models for the pairwise comparisons during Entity Matching. Special care will be taken to progressively evaluate the performance of end-to-end workflows, a feature that is particularly useful when processing very large datasets. In later versions, we intend to enrich JedAI with support for supervised learning techniques. To comply with its domain-agnostic foundations, we will begin with approaches that leverage generic, schema-agnostic features, like those in [2]. Upon successful completion of this extension, we will also consider schema-based features and constraints, which call for fundamental changes for their incorporation. Special care will be taken to add active learning techniques from top crowd-sourced ER approaches [7, 9], thus addressing the sensitivity to cluster size for some of JedAI's workflows.

Acknowledgements. This work was partially funded by the EU project ExtremeEarth (825258).

References

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

[2] G. D. Bianco, M. A. Gonçalves, and D. Duarte. BLOSS: effective meta-blocking with almost no effort. *Inf. Syst.*, 75:75–89, 2018.

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

[4] V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Morgan & Claypool Publishers, 2015.

[5] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.

[6] X. L. Dong and D. Srivastava. *Big Data Integration*. Morgan & Claypool Publishers, 2015.

[7] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.

[8] 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.

[9] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, pages 3–18, 2018.

[10] G. Giannakopoulos, P. Mavridi, G. Paliouras, G. Papadakis, and K. Tserpes. Representation models for text classification: a comparative analysis over three web document types. In *WIMS*, pages 13:1–13:12, 2012.

[11] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

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

[13] 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.

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

[15] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[16] P. Konda and S. D. et. al. Technical perspective: : Toward building entity matching management systems. *SIGMOD Record*, 47(1):33–40, 2018.

[17] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.

[18] 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*, 2013.

[19] 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.

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

[21] B. On, N. Koudas, D. Lee, and D. Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007.

[22] 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.

[23] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Trans. Knowl. Data Eng.*, 25(12):2665–2682, 2013.

[24] G. Papadakis and W. Nejdl. Efficient entity resolution methods for heterogeneous information spaces. In *ICDE Workshops*, pages 304–307, 2011.

[25] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.

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

[27] G. Papadakis, L. Tsekouras, E. Thanos, G. Giannakopoulos, T. Palpanas, and M. Koubarakis. Jedai: The force behind entity resolution. In *ESWC*, pages 161–166, 2017.

[28] 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.

[29] 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.