Information Systems 102 (2021) 101830

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Reproducible experiments on Three-Dimensional Entity Resolution with JedAI

George Mandilaras [a], George Papadakis [a,*], Luca Gagliardelli [b], Giovanni Simonini [b], Emmanouil Thanos [c], George Giannakopoulos [d], Sonia Bergamaschi [b], Themis Palpanas [e], Manolis Koubarakis [a], Alicia Lara-Clares [f,1], Antonio Fariña [g,1]

[a] National and Kapodistrian University of Athens, Greece
[b] University of Modena and Reggio Emilia, Italy
[c] KU Leuven, Belgium
[d] NCSR "Demokritos", Greece
[e] University of Paris & French University Institute (IUF), France
[f] NLP & IR Research Group, Universidad Nacional de Educación a Distancia (UNED), Spain
[g] University of A Coruña, CITIC, Database Lab, Spain

ARTICLE INFO

Article history:
Received 30 October 2020
Received in revised form 3 June 2021
Accepted 9 June 2021
Available online 17 June 2021
Recommended by Juan J. Lastra-Díaz

Keywords:
Entity Resolution
Batch methods
Progressive methods
Reproducibility

ABSTRACT

In Papadakis et al. (2020), we presented the latest release of JedAI, an open-source Entity Resolution (ER) system that allows for building a large variety of end-to-end ER pipelines. Through a thorough experimental evaluation, we compared a schema-agnostic ER pipeline based on blocks with another schema-based ER pipeline based on similarity joins. We applied them to 10 established, real-world datasets and assessed them with respect to effectiveness and time efficiency. Special care was taken to juxtapose their scalability, too, using seven established, synthetic datasets. Moreover, we experimentally compared the effectiveness of the batch schema-agnostic ER pipeline with its progressive counterpart. In this companion paper, we describe how to reproduce the entire experimental study that pertains to JedAI's serial execution through its intuitive user interface. We also explain how to examine the robustness of the parameter configurations we have selected.

1. Introduction

Entity Resolution (ER) is the task of identifying *matches* or *duplicates*, i.e., different entity profiles that describe the same real-world object. For example, ER should match the entity profiles https://www.wikidata.org/wiki/Q30 and https://en.wikipedia.org/wiki/United_States, which refer to the United States of America in two different data sources, Wikidata[2] and Wikipedia[3] respectively. ER constitutes a core data integration task and, thus, numerous approaches for tackling it have been proposed in the literature. Overviews of the main methods can be found in recent books [1–4], surveys [5–7] and tutorials [8–11].

To facilitate the use of the main ER methods, we created JedAI [12], an open-source system that allows for building end-to-end pipelines. JedAI enables users to effectively address the ER problem by categorizing the main methods into three orthogonal dimensions:

1. *Schema-awareness* categorizes ER methods into *schema-based* and *schema-agnostic* ones, depending on whether they rely on schema knowledge or not.
2. *Budget-awareness* categorizes ER methods into *budget-agnostic* ones, which operate as batch processes, and *budget-aware* ones, which operate in a pay-as-you-go manner that produces results progressively — they maximize the detected matches within a specific budget of temporal or computational resources.
3. *Execution mode* categorizes ER methods into serial and massively parallelized ones, e.g., over Apache Spark.[4]

DOI of original article: https://doi.org/10.1016/j.is.2020.101565.

[*] Corresponding author.

E-mail addresses: gmandi@di.uoa.gr (G. Mandilaras), gpapadis@di.uoa.gr (G. Papadakis), luca.gagliardelli@unimore.it (L. Gagliardelli), giovanni.simonini@unimore.it (G. Simonini), emmanouil.thanos@kuleuven.be (E. Thanos), ggianna@iit.demokritos.gr (G. Giannakopoulos), sonia.bergamaschi@unimore.it (S. Bergamaschi), themis@mi.parisdescartes.fr (T. Palpanas), koubarak@di.uoa.gr (M. Koubarakis), alara@lsi.uned.es (A. Lara-Clares), antonio.farina@udc.es (A. Fariña).

[1] Reviewer.

[2] https://www.wikidata.org.

[3] https://www.wikipedia.org.

[4] https://spark.apache.org

https://doi.org/10.1016/j.is.2021.101830
0306-4379/© 2021 Elsevier Ltd. All rights reserved.

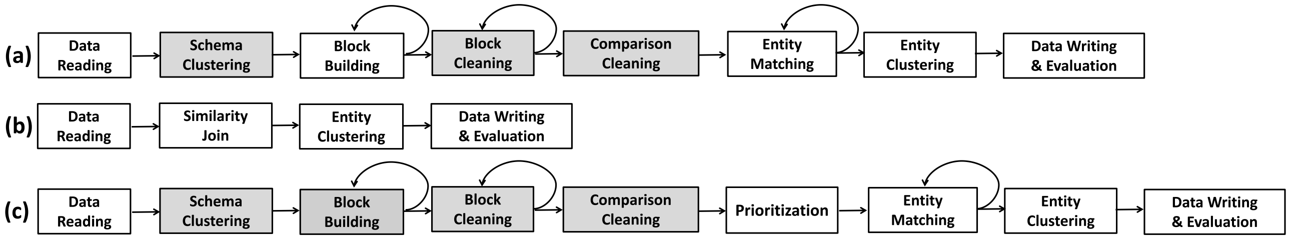


Fig. 1. The three main end-to-end ER pipelines implemented by JedAI: (a) the budget- & schema-agnostic one, (b) the budget-agnostic, schema-based one, and (c) the budget-aware, schema-agnostic one. Shaded rectangles indicate optional steps.

Table 1
Technical characteristics of the dirty ER datasets. $|E|$ stands for the number of entity profiles, NVP for the total number of name-value pairs in the dataset, $|N|$ for the number of distinct attributes, $|\bar{p}|$ for the average profile size (in terms of name-value pairs), $|D(E)|$ for the number of duplicate pairs, and $||E||$ for the comparisons executed by the brute-force approach.

	D_{cora}	D_{cddb}	D_{10k}	D_{50k}	D_{100k}	D_{200k}	D_{300k}	D_{1M}	D_{2M}
$\|E\|$	1295	9763	10,000	50,000	100,000	200,000	300,000	1,000,000	2,000,000
NVP	7166	183,072	106,108	530,854	1,061,421	2,123,728	3,184,885	10,617,729	21,238,252
$\|N\|$	12	106	12	12	12	12	12	12	12
$\|\bar{p}\|$	5.53	18.75	10.61	10.62	10.61	10.62	10.62	10.62	10.62
$\|D(E)\|$	17,184	299	8705	43,071	85,497	172,403	257,034	857,538	1,716,102
$\|\|E\|\|$	$8.38 \cdot 10^5$	$4.77 \cdot 10^7$	$5.00 \cdot 10^7$	$1.25 \cdot 10^9$	$5.00 \cdot 10^9$	$2.00 \cdot 10^{10}$	$4.50 \cdot 10^{10}$	$5.00 \cdot 10^{11}$	$2.00 \cdot 10^{12}$

Table 2
Technical characteristics of the Clean-Clean ER datasets.

	D_{c1}	D_{c2}	D_{c3}	D_{c4}	D_{c5}	D_{c6}	D_{c7}	D_{c8}
$Dataset_1$	Rest.1	Abt	Amazon	DBLP	Walmart	DBLP	DBPedia	DBPedia 3.0rc
$Dataset_2$	Rest.2	Buy	Google Pr.	ACM	Amazon	Scholar	IMDB	DBPedia 3.4
$\|E_1\|/\|E_2\|$	339/2,256	1076/1,076	1354/3,039	2616/2,294	2554/22,074	2516/61,353	27,615/23,182	$1.19 \cdot 10^6/2.16 \cdot 10^6$
NVP_1/NVP_2	1130/7,519	2568/2,308	5302/9,110	10,464/9,162	$14{,}143/1.1 \cdot 10^5$	$10{,}064/2 \cdot 10^5$	$1.6 \cdot 10^5/8.2 \cdot 10^5$	$1.69 \cdot 10^7/3.50 \cdot 10^7$
$\|N_1\|/\|N_2\|$	7/7	3/3	4/4	4/4	6/6	4/4	4/7	30,688/52,489
$\|\bar{p}_1\|/\|\bar{p}_2\|$	3.33/3.33	2.39/2.14	3.92/3.00	3.99/4.00	5.54/5.18	3.23/3.26	5.63/35.20	14.19/16.18
$\|D(E_1 \cap E_2)\|$	89	1076	1104	2224	853	2308	22,863	892,579
$\|\|E_1 \times E_2\|\|$	$7.65 \cdot 10^5$	$1.16 \cdot 10^6$	$4.11 \cdot 10^6$	$6.00 \cdot 10^6$	$5.64 \cdot 10^7$	$1.54 \cdot 10^8$	$6.40 \cdot 10^8$	$2.58 \cdot 10^{12}$

Using JedAI, we experimentally evaluated in [12] the relative performance of the main end-to-end ER pipelines that are defined by the three aforementioned dimensions. In this work, we focus on serially executed pipelines of any type.

Regarding schema-awareness, the **schema-agnostic pipeline** consists of the following steps, as shown in Fig. 1(a):

- *Data Reading* loads the data to be processed into main memory.
- *Schema Clustering* is an optional step that groups together different attributes that share syntactically similar values so as to improve the performance of the subsequent steps. Note that this task differs from Schema Matching, which tries to identify the semantically matching attributes.
- *Block Building* aims to reduce the computational cost of the brute-force approach, by limiting the search space to similar entity profiles. To this end, it clusters together entity profiles that share identical or similar signatures.
- *Block Cleaning* is an optional step that further curtails the computational cost of ER by refining the output of Block Building. Its goal is actually to discard those blocks that are dominated by *redundant* and *superfluous comparisons*; the former involve pairs of entities co-occurring in multiple blocks, while the latter compare pairs of entities that do not match.
- *Comparison Cleaning* is another optional step that serves the same purpose as Block Cleaning. It offers a more time-consuming, but more precise functionality that operates at the level of individual comparisons.
- *Entity Matching* estimates the matching likelihood for all entity pairs in the final set of blocks, using string similarity measures.
- *Entity Clustering* models the estimated similarities as a weighted, undirected graph and then partitions it into *equivalence clusters*, i.e., disjoint sets of entity profiles that are considered as matches.
- *Data Writing & Evaluation* allows for storing the final results and for assessing the performance of the selected ER pipeline with respect to the main effectiveness and time efficiency measures.

The **schema-based end-to-end pipeline** also starts with Data Reading and ends with Entity Clustering and Data Writing & Evaluation, as shown in Fig. 1(b). In between, it applies a single step, called *Similarity Join*, which rapidly estimates the pairs of entity profiles that satisfy a given *matching rule*, which consists of:

1. a similarity measure,
2. the attribute on which the measure is applied, and
3. a threshold designating the minimum acceptable similarity for two entity profiles that are considered as matching.

As an example, consider the following matching rule for bibliographic entities: $JaccardSim(title_1, title_2) > 0.8$.

In [12], we also compare the batch, schema-agnostic pipeline with its progressive counterpart, i.e., the **budget-aware, schema-agnostic pipeline**, which is shown in Fig. 1(c). The only difference from the batch pipeline is the *Prioritization* step, which intervenes between Comparison Cleaning and Entity Matching. Its goal is to define the optimal processing order of the entity pairs in the final set of blocks so that the matching ones are detected as early as possible.

A video demonstrating JedAI in action is available at: https://www.youtube.com/watch?v=OJY1DUrUAe8

Table 3
Core information about each dataset: its reference work, its type (i.e., whether it involves real or synthetic data), the corresponding ER task (Clean-Clean or Dirty ER), the paths of its entity profiles and its golden standard files in the data repository of [13] and the original data source. We have categorized the 17 datasets in three groups according to their type and task, following [13], which contains a different folder for each group. Note that in [13], all parts of D_{c8} are provided through a single zipped file, `newDBPedia.tar.xz`, to minimize their large size.

Dataset	Type	Task	Path to the entity profiles file in [13]	Path to the golden standard file in [13]	Source
D_{c1} [14]	Real	Clean-Clean ER	Real Clean-Clean ER data/restaurant1Profiles Real Clean-Clean ER data/restaurant2Profiles	Real Clean-Clean ER data/restaurant1IdDuplicates	[15]
D_{c2} [16]	Real	Clean-Clean ER	Real Clean-Clean ER data/abtProfiles Real Clean-Clean ER data/buyProfiles	Real Clean-Clean ER data/abtBuyIdDuplicates	[17]
D_{c3} [16]	Real	Clean-Clean ER	Real Clean-Clean ER data/amazonProfiles Real Clean-Clean ER data/gpProfiles	Real Clean-Clean ER data/amazonGpIdDuplicates	[17]
D_{c4} [16]	Real	Clean-Clean ER	Real Clean-Clean ER data/dblpProfiles Real Clean-Clean ER data/acmProfiles	Real Clean-Clean ER data/dblpAcmProfiles	[17]
D_{c5} [18]	Real	Clean-Clean ER	Real Clean-Clean ER data/walmartProfiles Real Clean-Clean ER data/amazonProfiles2	Real Clean-Clean ER data/amazonWalmartIdDuplicates	[19]
D_{c6} [16]	Real	Clean-Clean ER	Real Clean-Clean ER data/dblpProfiles2 Real Clean-Clean ER data/scholarProfiles	Clean-Clean ER data/dblpScholarIdDuplicates	[17]
D_{c7} [20]	Real	Clean-Clean ER	Real Clean-Clean ER data/imdbProfiles Real Clean-Clean ER data/dbpediaProfiles	Clean-Clean ER data/moviesIdDuplicates	[21]
D_{c8} [20]	Real	Clean-Clean ER	Real Clean-Clean ER data/cleanDBPedia1 Real Clean-Clean ER data/cleanDBPedia2	Clean-Clean ER data/newDBPediaMatches	[21]
D_{cora} [22]	Real	Dirty ER	Real Dirty ER data/coraProfiles	Real Dirty ER data/coraIdDuplicates	[23]
D_{cddb} [24]	Real	Dirty ER	Real Dirty ER data/cddbProfiles	Real Dirty ER data/cddbIdDuplicates	[23]
D_{10k} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/10Kprofiles	Synthetic Dirty ER data/10KIdDuplicates	[21]
D_{50k} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/50Kprofiles	Synthetic Dirty ER data/50KIdDuplicates	[21]
D_{100k} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/100Kprofiles	Synthetic Dirty ER data/100KIdDuplicates	[21]
D_{200k} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/200Kprofiles	Synthetic Dirty ER data/200KIdDuplicates	[21]
D_{300k} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/300Kprofiles	Synthetic Dirty ER data/300KIdDuplicates	[21]
D_{1M} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/1Mprofiles	Synthetic Dirty ER data/1MIdDuplicates	[21]
D_{2M} [25]	Synthetic	Dirty ER	Synthetic Dirty ER data/2Mprofiles	Synthetic Dirty ER data/2MIdDuplicates	[21]

Table 4
Detailed instructions for installing and running JedAI's Docker image on Ubuntu. The steps 1–7 install the latest version of Docker Community Edition. For more details, please refer to the official Docker setup page at: https://docs.docker.com/engine/install/ubuntu. The remaining steps download JedAI's Docker image from the Docker Hub (step 8) or from JedAI's Mendeley data repository (step 8') and execute it (step 9).

Step	Setup instructions
(1)	*Update the apt package index.* $ sudo apt-get update
(2)	*Install packages to allow apt to use a repository over HTTPS.* $ sudo apt-get -y install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
(3)	*Add Docker's official GPG key.* $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
(4)	*Set up the stable repository.* $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
(5)	*Update the apt package index.* $ sudo apt-get update
(6)	*Install the latest version of Docker Engine.* $ sudo apt-get -y install docker-ce docker-ce-cli containerd.io
(7)	*Verify that Docker Engine is installed correctly.* $ sudo docker run hello-world
(8)	*Download the latest JedAI Docker image from Docker Hub.* $ sudo docker pull gmandi/jedai-webapp:latest
(8')	*Alternatively, download JedAI's Docker image from the Mendeley dataset.* wget -O jedai.tar https://data.mendeley.com/public-files/datasets/4whpm32y47/files/79f5ccdd-e60a-4f9c-99cb-8f2d7ef0fc25/file_downloaded $ sudo docker load < jedai.tar
(9)	*Launch the JedAI Web application.* *Note that parameter -Xmx4g allows JedAI to use up to 4 Gb RAM. Higher values can be used if more main memory is available.* *Note also that parameter -v should point to a directory, e.g., `/home/user/jedai`, with user-write permissions.* $ sudo docker run -e 'JAVA_OPTIONS=-Xmx4g' -p 8080:8080 -v /absolute/path gmandi/jedai-webapp

2. The reproducible experiments on Entity Resolution

2.1. Preliminaries

Depending on the input data, Entity Resolution is categorized into two main categories:

1. *Clean-Clean ER* receives as input two datasets, which are individually duplicate-free (e.g., Wikipedia and Wikidata), and its goal is to identify the matches they share.
2. *Dirty ER* receives as input one or more datasets, with at least one of them containing duplicates in itself. Its goal is to partition all entity profiles into equivalence clusters.

In both cases, the end-result of any end-to-end pipeline is evaluated with respect to three **effectiveness** measures:

- *Recall* assesses the portion of existing duplicates that are actually identified as such.
- *Precision* estimates the portion of entity pairs that are marked as matches and are indeed duplicates.

Table 5
The testing platforms that were successfully used to reproduce our experiments. Note that $Ubuntu - base1$ was used in [12] for performing the experiments reported in Tables 8 and 9, while $Ubuntu - base1'$ was only used for the experiments in Table 10.

Testing platform	Type	Software configuration	Hardware configuration	Tested by
$Ubuntu - base1$	Server	Ubuntu 14.04.5 LTS OpenJDK 1.8.0	1 Intel Xeon E5-4603 v2 @2.20 GHz, 128 Gb DDR3 RAM, 1.6 Tb mechanical disk	Authors
$Ubuntu - base1'$	Server	Ubuntu 14.04 LTS Java 1.8.0	1 Intel Xeon E5-2670 v2 @2.50 GHz, 80 GB DDR3 RAM, 1 Tb mechanical disk	Authors
$Ubuntu - base2$	Server	Ubuntu 14.04.6 LTS Docker 19.03.13, Java 1.8.0	1 AMD Opteron 6320 @2.80 GHz, 128 Gb DDR3 RAM, 1.6 Tb mechanical disk	Authors
$Ubuntu - base3$	Laptop	Ubuntu 18.04.5 LTS Docker 20.10.5, Java 1.8.0	1 Intel Core i7-4710MQ @2.50 GHz, 16 Gb DDR3 RAM, 120 Gb SSD	Authors
$Ubuntu - base4$	Laptop	Ubuntu 20.04 LTS Docker 19.03.8, OpenJDK 1.8.0	1 Intel Core i5-1035G1 @1.00 GHz, 4 Gb DDR4 RAM, 32 Gb flash drive	Authors
$Ubuntu - base5$	Laptop	Linux Mint 19.1 Tessa Docker 19.03.8, Java 1.8.0	1 Intel Core i7-3770 @3.40 GHz, 16 Gb DDR3 RAM, 1 Tb mechanical disk	Authors
$Ubuntu - base6$	Laptop	Ubuntu 20.04.2 LTS Docker 19.03.14, OpenJDK 1.8.0	Intel Core i7-9750H @2.60 GHz, 32 GB RAM, 2.5 Tb mechanical disk	Reviewer
$Ubuntu - base7$	Server	Ubuntu 20.04.2 LTS	1 Intel Xeon Bronze 3204 @1.9 GHz, 512 Gb DDR4 RAM,120 Gb mechanical disk	Reviewer
$Ubuntu - base8$	Server	Ubuntu 16.04.7 LTS	1 Intel Core i7 8700k @3.7 GHz, 64 Gb swap, 64 Gb DDR4 RAM, 3 Tb mechanical disk	Reviewer
$Ubuntu - base9$	Laptop	Ubuntu 20.04.1 LTS	1 Intel Core i5 8265u @1.6 GHz, 16 DDR4 RAM, no swap, 34 Gb virtual disk over SSD	Reviewer
$Windows - base1$	Laptop	Windows 10 Pro v. 20H2, Docker 20.10.5, Java 15.0.1	1 Intel Core i5-1035G1 @1.00 GHz, 6 Gb DDR4 RAM, 240 Gb SSD	Authors

Table 6
The aggregate time required to run all the experiments included in Tables 8–10 (that could be completed in less than 40 h) for each testing platform, while reproducing most experiments from [12]. The testing platforms $Ubuntu - base3$, $Ubuntu - base4$, $Ubuntu - base5$, $Ubuntu - base6$, $Ubuntu - base9$ and $Windows - base1$ were limited in some experiments by the available main memory, thus exhibiting lower aggregate running times.

Run	Testing platform	Running time	Tested by
1	$Ubuntu - base1$	5526 min ≈ 92.1 h	Authors
2	$Ubuntu - base2$	6832 min ≈ 113.9 h	Authors
3	$Ubuntu - base3$	2678 min ≈ 44.6 h	Authors
4	$Ubuntu - base4$	187 min ≈ 3.1 h	Authors
5	$Ubuntu - base5$	2198 min ≈ 36.6 h	Authors
6	$Ubuntu - base6$	1428 min ≈ 23.8 h	Reviewer
7	$Ubuntu - base7$	6393 min ≈ 106.5 h	Reviewer
8	$Ubuntu - base8$	3212 min ≈ 53.5 h	Reviewer
9	$Ubuntu - base9$	1731 min ≈ 28.8 h	Reviewer
10	$Windows - base1$	1743 min ≈ 29.1 h	Authors

- *F-Measure* is the harmonic mean of Recall and Precision.

The progressive pipelines are additionally assessed through *Progressive Recall*, which quantifies the evolution of recall as more entity pairs are compared. We actually consider the area under its curve (AUC), which is derived from a two-dimensional diagram, where horizontal axis corresponds to the number of executed comparisons and the vertical one to the number of detected duplicates. The larger (the area under the curve of) Progressive Recall is, the earlier are the matches identified and the better is the progressive pipeline.

All effectiveness measures are defined in the interval [0, 1], with higher values corresponding to higher effectiveness.

The **time efficiency** of an end-to-end pipeline is measured through its *run-time*, i.e., the time that intervenes between receiving the input entity profiles and producing the end result.

Note that we also provide the minimum amount of main memory that is required to successfully run each test in a way that approximates the lowest possible running time by minimizing the impact of the garbage collector. The reported values correspond to the $-Xmx$ parameter when running each experiment as a Java process, independently of Docker and the browser, which raise additional memory requirements.

2.2. Sets of experiments

The experimental analysis of [12] used 17 datasets. Each of them consists of one or two sets of entity profiles, in the case of Dirty and Clean-Clean ER, respectively, as well as a golden standard, i.e., the complete ground-truth of the actual duplicate entity profiles. They are all publicly available in the form of Java serialized objects as a Mendeley dataset [13] and through JedAI's repository.[5] Their technical characteristics are reported in Tables 1 and 2, which are the same as Tables 1 and 2 in [10], but are repeated here for convenience. Additional information about all datasets is provided in Table 3.

Our experiments are divided into three sets as follows:

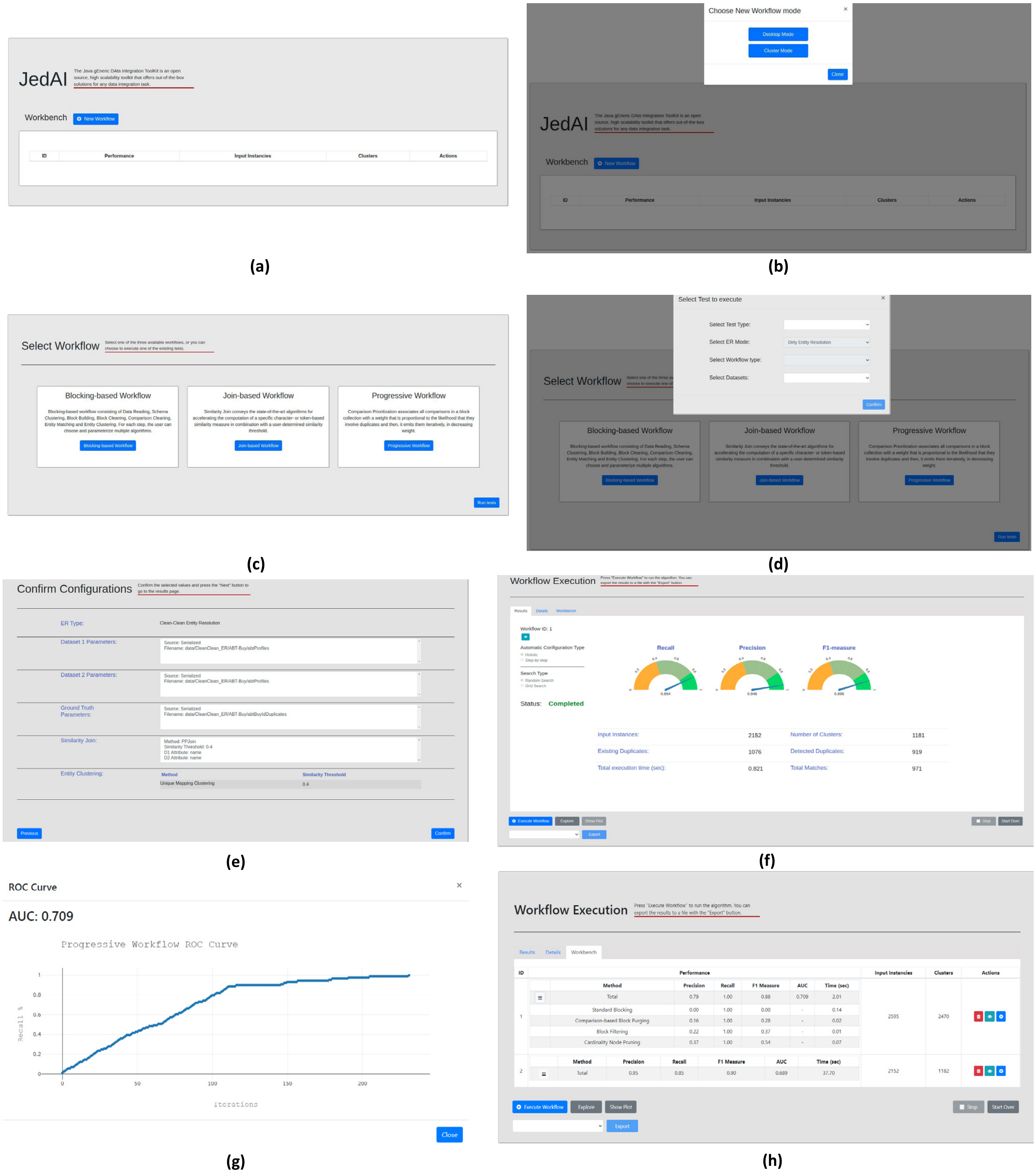
1. The *Performance Tests* examine the relative performance of the two budget-agnostic pipelines - the schema-based and the schema-agnostic one.
2. The *Scalability Tests* examine how the performance of the two budget-agnostic pipelines evolves as the size of the input data increases.
3. The *Budget-awareness Tests* examine the relative performance of the two forms of the schema-agnostic pipeline: the budget-agnostic and the budget-aware.

Below, we describe every set of experiments in more detail.

Performance tests. These experiments, which are reported in Table 4 of [12], compare the schema- and budget-agnostic pipeline with its schema-based counterpart over 10 real-world datasets. Two of them pertain to Dirty ER (D_{cora} and D_{cddb}) and the rest to Clean-Clean ER (D_{c1}-D_{c8}). The goal of these experiments is to evaluate both the relative effectiveness and the relative time efficiency of these pipelines. For the schema-agnostic pipeline, we consider two configurations:

1. the *best* one, which uses the parameters that maximize the F-Measure per dataset, and
2. the *default* one, which uses the default parameters for each method in the pipeline, thus being the same for all datasets.

[5] https://github.com/scify/JedAIToolkit.



JedAI The Java gEneric DAta Integration ToolKit is an open source, high scalability toolkit that offers out-of-the-box solutions for any data integration task.

Workbench ⊕ New Workflow

ID	Performance	Input Instancies	Clusters	Actions

(a)

Choose New Workflow mode ×

Desktop Mode

Cluster Mode

Close

(b)

Select Workflow Select one of the three available workflows, or you can choose to execute one of the existing tests.

Blocking-based Workflow

Blocking-based workflow consisting of Data Reading, Schema Clustering, Block Building, Block Cleaning, Comparison Cleaning, Entity Matching and Entity Clustering. For each step, the user can choose and parameterize multiple algorithms.

Blocking-based Workflow

Join-based Workflow

Similarity Join conveys the state-of-the-art algorithms for accelerating the computation of a specific character- or token-based similarity measure in combination with a user-determined similarity threshold.

Join-based Workflow

Progressive Workflow

Comparison Prioritization associates all comparisons in a block collection with a weight that is proportional to the likelihood that they involve duplicates and then, it emits them iteratively, in decreasing weight.

Progressive Workflow

Run tests

(c)

Select Test to execute ×

Select Test Type:

Select ER Mode: Dirty Entity Resolution

Select Workflow type:

Select Datasets:

Confirm

(d)

Confirm Configurations Confirm the selected values and press the "Next" button to go to the results page.

ER Type: Clean-Clean Entity Resolution

Dataset 1 Parameters:
Source: Serialized
Filename: data/CleanClean_ER/ABT-Buy/abtProfiles

Dataset 2 Parameters:
Source: Serialized
Filename: data/CleanClean_ER/ABT-Buy/abtProfiles

Ground Truth Parameters:
Source: Serialized
Filename: data/CleanClean_ER/ABT-Buy/abtBuyIdDuplicates

Similarity Join:
Method: PPJoin
Similarity Threshold: 0.4
D1 Attribute: name
D2 Attribute: name

Entity Clustering:

Method	Similarity Threshold
Unique Mapping Clustering	0.4

Previous   Confirm

(e)

Workflow Execution Press "Execute Workflow" to run the algorithm. You can export the results to a file with the "Export" button.

Results Details Workbench

Workflow ID: 1

Automatic Configuration Type
● Holistic
○ Step-by-step

Search Type
● Random Search
○ Grid Search

Status: Completed

Input Instances:	2152	Number of Clusters:	1181
Existing Duplicates:	1076	Detected Duplicates:	919
Total execution time (sec):	0.821	Total Matches:	971

Execute Workflow   Explore   Show Plot   Export   Stop   Start Over

(f)

ROC Curve ×

AUC: 0.709

Close

(g)

Workflow Execution Press "Execute Workflow" to run the algorithm. You can export the results to a file with the "Export" button.

Results Details Workbench

ID	Method	Precision	Recall	F1 Measure	AUC	Time (sec)	Input Instancies	Clusters	Actions
1	Total	0.79	1.00	0.88	0.709	2.01	2595	2470	
	Standard Blocking	0.00	1.00	0.00	-	0.14			
	Comparison-based Block Purging	0.16	1.00	0.28	-	0.02			
	Block Filtering	0.22	1.00	0.37	-	0.01			
	Cardinality Node Pruning	0.37	1.00	0.54	-	0.07			
2	Total	0.95	0.85	0.90	0.689	37.70	2152	1182	

Execute Workflow   Explore   Show Plot   Export   Stop   Start Over

(h)

Fig. 2. The screens of JedAI's Web application for reproducing all single core experiments in [12]: (a) The initial screen of JedAI's Web application. The button '`New Workflow`' should be pressed. (b) The second screen, which defines the execution mode. The button '`Desktop Mode`' should be pressed for the single-core experiments. (c) The third screen, which defines the type of the end-to-end pipeline. The button '`Run tests`' should be pressed to start the reproduction of the experiments. (d) The fourth screen, which defines the experimental settings we want to reproduce with respect to the type of experiments, the type of ER, the type of end-to-end pipeline and the dataset. (e) The '`Confirm Configuration`' screen that summarizes the experimental settings we have selected. (f) The final screen, '`Workflow Execution`', which presents the performance of the selected end-to-end pipeline. (g) The screen showing the area under the curve of Progressive Recall (AUC) in case of Budget-awareness Tests. (h) The benchmark screen summarizing the performance of all pipelines executed so far with respect to Precision, Recall, F-Measure, Run-time and Progressive Recall (AUC), in case of Budget-awareness Tests.

For the schema-based pipeline, we exclusively consider the best configuration per dataset, which maximizes F-Measure.

Note that these tests involve two baseline systems that have been developed by other research groups, Magellan [26] and DeepMatcher [27]. Due to their human-in-the-loop approach and the lack of necessary details, we could not test their performance ourselves. Instead, we reported their top F-measure per dataset

Table 7
Detailed instructions for reproducing all single-core experiments in [12] using the graphical user interface of JedAI's Web application.

Step	Reproduction instructions
	After launching JedAI's Docker image with the last command in Table 4:
(1)	Open a browser at http://localhost:8080. *If Docker runs on a server, replace 'localhost with its URL. The host port 8080 was arbitrarily specified by the last command in Table 4 and can be changed at will. JedAI's homepage, depicted in Fig. 2(a), shows up.*
(2)	Press the button 'New Workflow'. *The window 'Choose New Workflow mode' in Fig. 2(b) pops up.*
(3)	Press the button 'Desktop Mode'. *Because we are interested in the serial execution of JedAI's experiments.* *The Web page 'Select Workflow' in Fig. 2(c) shows up.*
(4)	Press the button 'Run tests' at the bottom right corner. *The window 'Select Test to execute' in Fig. 2(d) shows up.* *The web application is already equipped with the parameters of all experiments.* *Thus, any experiment in [12] can be reproduced simply by selecting it from the menus of Fig. 2(d).*
(5)	In 'Test Type', select 'Performance Test', 'Scalability Test' or 'Budget-awareness Test'. *The options for the rest of the selection criteria in the same window are activated.*
(6)	In 'ER Mode', select 'Clean-Clean ER' or 'Dirty ER'. *For Scalability Tests, only 'Dirty ER' is available.*
(7)	In 'Workflow Type', select 'Best Schema-agnostic', 'Default Schema-agnostic' or 'Schema-aware' pipelines. *For Scalability Tests, only the last two options are available.*
(8)	In 'Datasets', select one among the available datasets in Tables 1–3.
(9)	Press the button 'Confirm'. *JedAI loads the selected pipeline with the parameter configuration corresponding to the selected dataset.* *One Web page for each step in the selected pipeline (see Fig. 1) shows up.*
(10)	Press the button 'Next' in the window of each pipeline step to proceed to the next one. *After going through all pipeline steps, the Web page 'Confirm Configurations' in Fig. 2(e) shows up.*
(11)	Press the button 'Confirm'. *The Web page 'Workflow Execution' shows up.*
(12)	Press the button 'Execute Workflow'. *The selected experiment is carried out. Upon completion, the respective performance is reported in the same window with respect to recall, Precision, F-Measure and running time, as in Fig. 2(f).*
(13)	In case of Budget-awareness Tests, press the button '*Show Plot*' at the bottom left corner. *A window similar to the one in Fig. 2(g) shows up, depicting Progressive recall along with the area under its curve.*
(14)	Press JedAI logo at the top of the window to return to the first screen and proceed with the next test.

in [27], among all configurations and dataset versions. For this reason, we disregard both systems in the following.

Scalability tests. These experiments are described in the diagrams of Figure 7 in [12], comparing again the two budget-agnostic end-to-end pipelines. In this case, though, the goal is to assess how their time efficiency and effectiveness evolve as the size of the data increase from several thousand to few million entity profiles. To this end, we use seven datasets that pertain exclusively to Dirty ER; their names indicate their size, i.e., the number of their entity profiles: D_{10K}, D_{50K}, D_{100K}, D_{200K}, D_{300K}, D_{1M} and D_{2M}. These datasets contain synthetic census data, i.e., information about individuals that has been enriched with various forms of artificial noise (see [12] for more details). For both pipelines, we consider a single configuration that is applied to all datasets: the default configuration for the schema-agnostic pipeline and the matching rule that consistently achieves reasonable performance across all datasets for the schema-based one, i.e., $JaccarSim(all\_tokens\_1, all\_tokens\_2) > 0.4$, executed by PPJoin and followed by Connected Components with the same similarity threshold.

Budget-awareness tests. These experiments are reported in the diagrams of Figure 8 in [12]. They compare the budget- and schema-agnostic pipeline with its budget-aware counterpart across the same datasets as the Performance Tests - except the largest one, D_{c8}. For each dataset, the parameter configuration that corresponds to the optimal performance of the budget- and schema-agnostic pipeline is also used for the common methods of its budget-aware version. In this way, these tests assess the impact of the Prioritization step, which constitutes the sole difference between the two pipelines. We evaluate the time efficiency of the two workflows through their running times and the effectiveness through the area under their Progressive Recall.

2.3. Experimental setup in our primary paper

All single-core experiments in [12] were implemented in Java 8 and can be reproduced through JedAI's Docker image, which is publicly available.[6] The only requirement is to have Docker[7] installed. Table 4 provides detailed instructions for installing the latest version of Docker on Ubuntu. A similar procedure is required for other Linux distributions, like Debian,[8] Fedora[9] and CentOS.[10] JedAI's Docker image is expected to run seamlessly in all these cases. Upon successful completion of these commands, JedAI's Web application appears in a browser at: http://localhost:8080.

Note that the option `-e JAVA_OPTIONS='-Xmx4g'` determines that 4 Gigabytes (GB) of RAM memory is allocated to Java to run JedAI's Web application. This is an optional parameter, as the vast majority of our experiments can be run with much fewer memory, as indicated by the memory requirements that are reported in Tables 8–10 for each experiment. In our tests, though,

[6] https://hub.docker.com/repository/docker/gmandi/jedai-webapp.

[7] https://www.docker.com.

[8] See https://docs.docker.com/engine/install/debian for detailed instructions.

[9] See https://docs.docker.com/engine/install/fedora for detailed instructions.

[10] See https://docs.docker.com/engine/install/centos for detailed instructions.

Table 8
The results of the performance tests over all real datasets across all testing platforms. For each pipeline, the effectiveness measures per dataset are common among all testing platforms. Only the running times differ among them. *IM* indicates a test that was not carried out due to insufficient memory. Note that Precision, Recall and F-Measure are rounded to three decimal places, memory requirements to two decimal places and running times to one decimal place.

	Clean-Clean ER								Dirty ER	
	Restaurants D_{c1}	Abt Buy D_{c2}	Amazon GP D_{c3}	DBLP ACM D_{c4}	Walmart Amazon D_{c5}	DBLP Scholar D_{c6}	IMDB DBPedia D_{c7}	DBP-3.0rc DBP-3.4 D_{c8}	D_{cora}	D_{cddb}
(a) Default configuration of the budget- and schema-agnostic pipeline										
Precision	0.473	0.902	0.544	0.975	0.310	0.887	0.908	0.806	0.876	0.874
Recall	1.000	0.836	0.653	0.988	0.878	0.952	0.834	0.819	0.816	0.856
F-Measure	0.643	0.867	0.594	0.981	0.459	0.919	0.869	0.813	0.845	0.865
Memory (Gb)	0.02	0.04	0.19	0.09	0.32	0.75	0.99	105.00	0.17	1.45
*Ubuntu − base*1	1.1 s	1.3 s	12.0 s	2.0 s	8.3 s	23.5 s	91.0 s	14.5 h	5.5 s	65.0 s
*Ubuntu − base*2	0.6 s	1.3 s	15.1 s	1.3 s	6.2 s	28.9 s	113.0 s	22.1 h	2.7 s	61.8 s
*Ubuntu − base*3	0.5 s	1.0 s	11.2 s	0.9 s	4.4 s	10.2 s	68.0 s	*IM*	1.8 s	30.6 s
*Ubuntu − base*4	0.2 s	0.6 s	8.6 s	0.7 s	3.5 s	9.2 s	53.4 s	*IM*	1.8 s	23.4 s
*Ubuntu − base*5	0.3 s	0.6 s	8.2 s	0.8 s	3.7 s	9.1 s	48.5 s	*IM*	1.3 s	23.9 s
*Ubuntu − base*6	0.1 s	0.6 s	8.3 s	0.7 s	3.0 s	7.7 s	51.9 s	*IM*	1.3 s	21.7 s
*Ubuntu − base*7	0.2 s	1.3 s	15.9 s	1.2 s	5.3 s	15.3 s	98.4 s	16.8 h	2.4 s	49.0 s
*Ubuntu − base*8	0.2 s	0.6 s	7.9 s	0.6 s	2.5 s	6.5 s	39.5 s	*IM*	1.0 s	18.8 s
*Ubuntu − base*9	0.4 s	1.1 s	16.1 s	1.0 s	5.1 s	11.8 s	75.5 s	*IM*	1.8 s	30.8 s
*Windows − base*1	0.3 s	1.0 s	8.7 s	0.9 s	4.2 s	18.2 s	98.6 s	*IM*	1.7 s	26.7 s
(b) Best configuration of the budget- and schema-agnostic pipeline										
Precision	0.788	0.946	0.576	0.993	0.590	0.946	0.905	0.841	0.912	0.869
Recall	1.000	0.854	0.646	0.992	0.753	0.949	0.876	0.821	0.819	0.886
F-Measure	0.881	0.898	0.609	0.992	0.662	0.948	0.890	0.831	0.863	0.877
Memory (Gb)	0.03	0.04	0.07	0.04	0.12	0.98	0.80	64.00	0.02	1.47
*Ubuntu − base*1	1.0 s	1.1 s	4.5 s	1.3 s	5.3 s	30.0 s	46.0 s	12.7 h	0.9 s	65.7 s
*Ubuntu − base*2	0.7 s	1.1 s	6.1 s	0.8 s	12.9 s	45.1 s	49.5 s	21.9 h	0.8 s	70.0 s
*Ubuntu − base*3	0.5 s	0.8 s	5.0 s	0.6 s	2.4 s	16.4 s	29.0 s	*IM*	0.6 s	32.4 s
*Ubuntu − base*4	0.5 s	0.7 s	4.1 s	0.4 s	1.8 s	12.6 s	23.9 s	*IM*	0.4 s	25.6 s
*Ubuntu − base*5	0.4 s	0.6 s	3.2 s	0.7 s	2.0 s	12.2 s	24.5 s	*IM*	0.5 s	30.8 s
*Ubuntu − base*6	0.1 s	0.5 s	3.1 s	0.5 s	1.7 s	13.1 s	21.8 s	*IM*	0.3 s	23.3 s
*Ubuntu − base*7	0.1 s	0.6 s	7.3 s	0.8 s	2.8 s	23.9 s	41.7 s	16.5 h	0.7 s	51.3 s
*Ubuntu − base*8	0.1 s	0.4 s	3.5 s	0.4 s	1.2 s	11.2 s	18.6 s	12.9 h	0.3 s	24.5 s
*Ubuntu − base*9	0.2 s	0.4 s	5.9 s	0.6 s	2.2 s	16.8 s	34.8 s	*IM*	0.3 s	34.9 s
*Windows − base*1	0.2 s	0.6 s	4.8 s	0.5 s	2.3 s	18.7 s	28.9 s	*IM*	0.5 s	32.5 s
(c) Best configuration of the budget-agnostic, schema-based pipeline										
Precision	0.755	0.884	0.663	0.978	0.829	0.953	0.931	0.833	0.751	0.278
Recall	0.933	0.438	0.423	0.932	0.552	0.775	0.499	0.370	0.859	0.719
F-Measure	0.834	0.585	0.517	0.954	0.663	0.855	0.649	0.512	0.802	0.401
Memory (Gb)	0.01	0.02	0.02	0.02	0.06	0.11	0.42	30.00	0.02	0.06
*Ubuntu − base*1	0.2 s	0.4 s	0.5 s	0.6 s	0.5 s	14.0 s	7.7 s	15.2 min	0.3 s	0.6 s
*Ubuntu − base*2	0.2 s	0.2 s	0.2 s	0.5 s	0.2 s	13.8 s	6.9 s	12.4 min	0.3 s	0.3 s
*Ubuntu − base*3	0.2 s	0.3 s	0.3 s	0.2 s	0.2 s	10.6 s	5.2 s	*IM*	0.2 s	0.3 s
*Ubuntu − base*4	0.1 s	0.1 s	0.1 s	0.1 s	0.1 s	10.2 s	3.5 s	*IM*	0.2 s	0.3 s
*Ubuntu − base*5	0.1 s	0.1 s	0.2 s	0.2 s	0.3 s	7.4 s	3.4 s	*IM*	0.2 s	0.3 s
*Ubuntu − base*6	0.1 s	0.1 s	0.2 s	0.3 s	0.1 s	6.3 s	3.3 s	*IM*	0.1 s	0.3 s
*Ubuntu − base*7	0.1 s	0.2 s	0.2 s	0.2 s	0.2 s	14.2 s	7.7 s	11.0 min	0.1 s	0.2 s
*Ubuntu − base*8	0.1 s	0.1 s	0.1 s	0.1 s	0.1 s	5.9 s	3.2 s	5.2 min	0.1 s	0.1 s
*Ubuntu − base*9	0.1 s	0.1 s	0.2 s	0.2 s	0.1 s	16.5 s	5.6 s	*IM*	0.1 s	0.2 s
*Windows − base*1	0.1 s	0.1 s	0.2 s	0.2 s	0.1 s	16.5 s	5.6 s	*IM*	0.1 s	0.2 s

we noticed that 4GB are more suitable for ensuring Docker's stability. Otherwise, it needs restarting after some tests. When experimenting with larger datasets, it is actually recommended to devote all or most of the available memory to Docker so as to avoid out-of-memory exceptions or excessively large running times, due to the overuse of the garbage collector.

Note also that the option `-v /absolute/path` is necessary because JedAI's Docker starts by downloading all datasets from the Mendeley data repository [13]. Thus, this option determines the directory on the host system (e.g., `/home/user/jedai`), where Docker will store and unpack the dataset files as long as it has user-write permissions.

It is also worth noting that in the option `-p 8080:8080`, the first 8080 refers to the host port, and could be replaced by any other free port in the host. Docker will map the first port 8080 to the http port (second 8080) from the docker container.

Finally, it is worth noting that it is also possible to use Docker on Windows 10. The installation is a straightforward procedure[11] that merely needs some additional steps.[12] After the successful installation, all experiments can be seamlessly run, without any performance issue. Indeed, one of our testing platforms runs on Windows 10 Pro (*Windows − base*1 in Table 5).

2.4. System requirements and performance evaluation

All single-core experiments in [12] can be reproduced on any Java 8 compliant platform, which practically includes all major

[11] See https://docs.docker.com/docker-for-windows/install for detailed instructions.

[12] See https://docs.docker.com/docker-for-windows/wsl for more details.

Table 9
The results of the scalability tests over the seven synthetic datasets across all testing platforms. For each pipeline, the effectiveness measures per dataset are common among all testing platforms. Only the running times differ among them. *IM* indicates a test that was not carried out due to insufficient memory. Note that Precision, Recall and F-Measure are rounded to three decimal places, memory requirements to two decimal places and running times to one decimal place.

	D_{10k}	D_{50k}	D_{100k}	D_{200k}	D_{300k}	D_{1M}	D_{2M}
(a) Default configuration of the budget- and schema-agnostic pipeline							
Precision	0.948	0.899	0.887	0.844	0.866	0.868	0.836
Recall	0.994	0.989	0.983	0.978	0.973	0.960	0.954
F-Measure	0.970	0.942	0.933	0.906	0.916	0.911	0.891
Memory (Gb)	0.12	0.80	3.10	6.20	7.20	15.00	25.00
$Ubuntu - base1$	1.8 s	12.8 s	35.1 s	120.2 s	193.1 s	32.3 min	147.1 min
$Ubuntu - base2$	1.6 s	11.4 s	37.3 s	130.8 s	199.3 s	33.4 min	145.4 min
$Ubuntu - base3$	1.4 s	5.2 s	19.8 s	51.9 s	141.9 s	22.1 min	*IM*
$Ubuntu - base4$	0.9 s	4.3 s	10.7 s	54.3 s	*IM*	*IM*	*IM*
$Ubuntu - base5$	1.0 s	3.7 s	11.5 s	37.8 s	77.0 s	16.9 min	*IM*
$Ubuntu - base6$	0.7 s	4.2 s	10.8 s	36.7 s	114.6 s	–	–
$Ubuntu - base7$	0.8 s	6.3 s	19.9 s	63.4 s	148.0 s	24.2 min	93.5 min
$Ubuntu - base8$	0.5 s	3.6 s	9.0 s	27.1 s	71.3 s	12.9 min	51.4 min
$Ubuntu - base9$	1.5 s	8.2 s	15.7 s	46.9 s	118.4 s	22.6 min	*IM*
$Windows - base1$	1.4 s	5.0 s	10.8 s	47.2 s	232.5 s	*IM*	*IM*
(b) Best configuration of the budget-agnostic, schema-based pipeline							
Precision	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Recall	0.593	0.598	0.602	0.600	0.602	0.603	0.602
F-Measure	0.744	0.749	0.752	0.750	0.751	0.752	0.752
Memory (Gb)	0.03	0.10	0.30	1.15	1.75	11.00	16.00
$Ubuntu - base1$	7.0 s	137.2 s	695.3 s	55.6 min	140.3 min	17.8 h	> 40 h
$Ubuntu - base2$	5.3 s	120.3 s	534.8 s	49.6 min	96.9 min	19.3 h	> 40 h
$Ubuntu - base3$	4.0 s	89.4 s	367.8 s	26.3 min	69.4 min	13.0 h	> 40 h
$Ubuntu - base4$	3.9 s	74.6 s	316.5 s	24.0 min	48.4 min	*IM*	*IM*
$Ubuntu - base5$	3.6 s	67.9 s	298.2 s	21.3 min	55.9 min	10.3 h	> 40 h
$Ubuntu - base6$	3.8 s	78.6 s	341.5 s	23.6 min	57.1 min	–	> 40 h
$Ubuntu - base7$	7.9 s	172.2 s	704.1 s	49.5 min	111.9 min	19.8 h	> 40 h
$Ubuntu - base8$	3.1 s	140.1 s	375.2 s	22.3 min	49.7 min	10.2 h	39.8 h
$Ubuntu - base9$	5.3 s	96.8 s	376.4 s	28.7 min	64.5 min	10.8 h	> 40 h
$Windows - base1$	4.3 s	87.3 s	376.7 s	26.7 min	56.8 min	*IM*	*IM*

Linux distributions. Our experiments have been successfully reproduced on all testing platforms reported in Table 5, with the aggregate running times that are reported in Table 6. Note that in all systems, a single CPU core was used for each experiment.

Our original configuration corresponds to $Ubuntu - base1$ for the Performance and Scalability Tests and to $Ubuntu - base1'$ for the Budget-awareness Tests. $Ubuntu-base2$ is a similar server but with a different CPU that accounts for significant diversity in the running times. A more important difference is that in $Ubuntu - base1$ and $Ubuntu - base1'$, all experiments were run through script files,[13] whereas in $Ubuntu - base2$, the experiments were carried out through the user interface of JedAI's Web application. The same applies to all other systems.

Among the other platforms, it is worth stressing that $Ubuntu-base4$ consists of a bootable USB stick that runs a live Ubuntu instance on top of a Windows 10 laptop. The only implication was that it required a different approach for installing Docker.[14] No performance issue arose. In fact, $Ubuntu - base4$ is often one of the fastest testing platforms, due to the newer generation of CPU and RAM technology.

Regarding the minimum system specifications required by our experiments, the size of the hard disk plays a minor role. Given that all experiments are executed in main memory and produce no output files, the hard disk requirements are determined by the space occupied by the Java JDK and the Docker installation as well as the size of JedAI's Docker image, which also includes all datasets. In total, this amounts to around 4 GB, assuming an underlying blank Ubuntu installation. Note, though, that this space is occupied whenever command 9 in Table 4 is executed. To recover the space occupied after multiple runs, we can:

- Remove the existing Docker containers:
  `sudo docker container ls -a | grep gmandi`
  obtains the IDs of JedAI's containers, and
  `sudo docker rm -f containerID`
  removes a given container.
- Remove JedAI's Docker image:
  `sudo docker rmi gmandi/jedai-webapp`.[15]
- Finally, recover disk space for unused volumes:
  `sudo docker volume prune`.

Regarding the size of main memory (RAM), the vast majority of experiments require less than 2 Gb, as reported in Tables 8–10, but 4 Gb are suggested to ensure Docker's stability, as explained above. However, the experiments with the two largest synthetic datasets, D_{1M} and D_{2M}, require up to 25 Gb, whereas the largest real dataset, D_{c8}, requires up to 105 Gb. The corresponding experiments cannot be run on most testing platforms that are equipped with 16 Gb RAM or less, namely $Ubuntu - base3$, $Ubuntu - base4$, $Ubuntu - base5$, $Ubuntu - base6$, $Ubuntu - base9$ and $Windows - base1$. Below, we report in detail the memory requirements of every experiment, highlighting the experiments that were not feasible, due to insufficient main memory in the testing platforms.

Finally, it is worth noting that the times reported in Table 6 merely correspond to the time taken by each system to run all experiments. Given that each experiment is carried out through

[13] The source code of all tests is available at: https://github.com/scify/JedAIToolkit/tree/master/src/test/java/org/scify/jedai/version3.

[14] For more details, please refer to https://stackoverflow.com/questions/30248794/run-docker-in-ubuntu-live-disk.

[15] Alternatively, run `sudo docker images` to obtain the IDs of the images, and then use `sudo docker rmi imageID` to remove them.

Table 10
The results of the Budget-awareness Tests over all real datasets across all testing platforms. For each pipeline, effectiveness is measured through the area under the curve of Progressive Recall, which is common among all testing platforms in each dataset only for the budget-aware pipeline. Its budget-agnostic counterpart arranges all pairwise comparisons in a random order, thus yielding a Progressive Recall that differs in each run and, thus, among the testing platforms. Note that Precision, Recall and F-Measure are rounded to three decimal places, memory requirements to two decimal places and running times to one decimal place. Note also that D_{c8} is omitted, as in [12], due to the excessively large running time and the very high memory requirements of the corresponding experiment.

	Clean-Clean ER							Dirty ER	
	Restau rants	Abt Buy	Amazon GP	DBLP ACM	Walmart Amazon	DBLP Scholar	IMDB DBPedia	D_{cora}	D_{cddb}
	D_{c1}	D_{c2}	D_{c3}	D_{c4}	D_{c5}	D_{c6}	D_{c7}		
(a) Budget-aware, schema-agnostic pipeline									
Progressive recall	0.709	0.689	0.573	0.866	0.635	0.930	0.616	0.416	0.585
Memory (Gb)	0.06	0.08	0.30	0.16	0.65	4.00	6.00	0.30	3.50
*Ubuntu − base*1′	0.5 s	13.7 s	1.8 min	32.9 s	5.2 min	46.3 min	18.4 h	16.9 s	79.3 s
*Ubuntu − base*2	0.6 s	19.0 s	3.4 min	49.8 s	7.8 min	68.2 min	20.1 h	15.8 s	96.5 s
*Ubuntu − base*3	0.4 s	14.6 s	2.2 min	48.5 s	7.4 min	54.1 min	12.7 h	12.2 s	73.5 s
*Ubuntu − base*4	0.3 s	12.5 s	1.5 min	35.7 s	6.1 min	40.7 min	*IM*	9.9 s	55.6 s
*Ubuntu − base*5	0.6 s	12.2 s	2.0 min	31.5 s	5.1 min	37.2 min	10.8 h	9.4 s	49.9 s
*Ubuntu − base*6	0.3 s	11.9 s	1.4 min	33.5 s	4.9 min	34.5 min	9.6 h	10.1 s	53.8 s
*Ubuntu − base*7	0.4 s	20.9 s	2.7 min	63.1 s	9.4 min	64.7 min	22.0 h	19.6 s	107.8 s
*Ubuntu − base*8	0.3 s	10.2 s	1.2 min	30.0 s	4.2 min	28.7 min	9.5 h	8.8 s	45.8 s
*Ubuntu − base*9	0.8 s	16.6 s	2.2 min	48.8 s	6.9 min	38.8 min	13.9 h	14.3 s	66.9 s
*Windows − base*1	0.5 s	14.9 s	1.9 min	20.4 s	8.3 min	52.4 min	11.3 h	15.3 s	80.0 s
(b) Budget- and schema-agnostic pipeline									
Memory (Gb)	0.09	0.20	0.20	0.35	0.65	4.00	6.00	0.30	3.50
Progressive recall	0.489	0.418	0.337	0.491	0.386	0.478	0.435	0.661	0.451
*Ubuntu − base*1′	1.6 s	19.2 s	2.4 min	34.4 s	11.6 min	51.0 min	20.8 h	30.4 s	13.5 min
Progressive recall	0.491	0.400	0.341	0.489	0.383	0.479	0.436	0.665	0.446
*Ubuntu − base*2	0.5 s	15.5 s	2.6 min	40.4 s	13.6 min	61.5 min	21.8 h	18.5 s	13.6 min
Progressive recall	0.481	0.403	0.328	0.488	0.397	0.474	0.437	0.659	0.466
*Ubuntu − base*3	0.4 s	12.4 s	1.9 min	31.4 s	12.6 min	49.5 min	14.5 h	15.4 s	11.9 min
Progressive recall	0.521	0.405	0.335	0.495	0.371	0.488	*IM*	0.668	0.464
*Ubuntu − base*4	0.5 s	10.2 s	1.6 min	25.9 s	9.7 min	36.7 min	*IM*	10.4 s	7.8 min
Progressive recall	0.487	0.402	0.322	0.488	0.383	0.475	0.435	0.678	0.482
*Ubuntu − base*5	0.5 s	11.1 s	1.4 min	25.8 s	8.8 min	37.3 min	12.2 h	10.8 s	8.0 min
Progressive recall	0.483	0.399	0.326	0.498	0.374	0.476	0.436	0.666	0.460
*Ubuntu − base*6	0.2 s	10.7 s	1.6 min	28.9 s	9.0 min	35.6 min	11.1 h	10.6 s	8.0 min
Progressive recall	0.457	0.406	0.344	0.501	0.381	0.463	0.436	0.661	0.510
*Ubuntu − base*7	0.4 s	21.9 s	2.9 min	57.4 s	19.0 min	72.7 min	23.3 h	21.8 s	15.9 min
Progressive recall	0.528	0.416	0.319	0.502	0.376	0.464	0.435	0.668	0.488
*Ubuntu − base*8	0.2 s	9.0 s	1.4 min	25.1 s	7.5 min	29.8 min	10.8 h	9.9 s	6.9 min
Progressive recall	0.453	0.414	0.330	0.490	0.379	0.476	0.436	0.669	0.463
*Ubuntu − base*9	0.3 s	14.4 s	2.4 min	39.7 s	12.4 min	46.4 min	14.5 h	18.0 s	11.9 min
Progressive recall	0.520	0.396	0.334	0.498	0.381	0.469	0.434	0.659	0.453
*Windows − base*1	0.7 s	14.5 s	2.2 min	37.9 s	13.8 min	50.8 min	13.5 h	15.1 s	24.3 min

the user interface of JedAI's Web application (i.e., they are not executed through a script), significant time is taken to manually navigate through all menus. Among them, the Entity Matching step requires additional time to transform the selected dataset into the textual representation that is suitable for assessing entity similarity (e.g., by tokenizing all attribute values into character n-grams). This time, which is negligible only for the smallest datasets, is not added to the overall running times in Table 6, which disregard completely the navigation time.

2.5. Obtaining and compiling our source code

The source code for JedAI version 3.0, which is used in [12] and in the present experimental study, has been publicly released at: https://github.com/scify/JedAIToolkit. Any development kit and/or IDE for Java 8 or higher can be used for compiling it, but this is not necessary. JedAI's Docker image contains an executable jar file with the entire source code and its dependencies. When executed, it deploys JedAI's Web application, allowing users to reproduce all experiments by following the instructions below, in Section 2.6.

2.6. Running the experiments

Table 7 provides detailed guidelines for reproducing all experiments. In essence, the user merely needs to navigate through the windows of JedAI's user interface, which are illustrated in Fig. 2. This means that minimal human intervention is required. For example, all datasets in Tables 1–3 are already included in JedAI's Docker image; the one selected in Step 8 is automatically loaded after the Data Reading step, which follows Step 9 in all pipelines. Similarly, there is a separate window with all available methods for each pipeline step, but no particular action is required from the user: the method used in the chosen experiment is already marked as selected and its parameters are appropriately configured. The user simply needs to press 'Next' in each step to proceed with the next one.

It is worth stressing at this point the wealth of information that is provided by the final window, called 'Workflow Execution', after completing an experiment:

1. The button 'Explore' presents the entity profiles that form each equivalence cluster.

2. The tab ‘Details’ contains the output of each step in the latest pipeline so as understand its operation and contribution to the overall performance.
3. The tab ‘Workbench’ summarizes the performance of all pipelines executed so far, as shown in Fig. 2(h). This allows for juxtaposing the performance of different pipelines over the same dataset, even at the level of individual steps: pressing the button ≡ in the leftmost column displays a performance breakdown among all steps.

The outcomes of the Performance, the Scalability and the Budget-awareness tests over all testing platforms are reported in Tables 8–10, respectively. In all cases, the effectiveness measures are common among all platforms, with the only differences corresponding to the running times. Compared to the experiments reported in [12], the effectiveness results of Budget-awareness tests are practically identical in most cases. The only significant exceptions pertain to the best schema-agnostic pipeline over D_{c2}, D_{c3} and D_{cddb}, whose F-Measure has now changed from 0.900, 0.607 and 0.872 to 0.898, 0.609 and 0.877, respectively, after some bug fixes. The F-Measure of the default schema-agnostic pipeline over D_{c3} has also increased from 0.586 to 0.594. The effectiveness results of the Scalability and the Budget-awareness tests are also identical with those reported in [12]; only their format has changed from diagrams to tables. In all cases, the running times in [12] are reproduced here, corresponding to $Ubuntu - base1$ in Tables 8 and 9 and to $Ubuntu - base1'$ in Table 10.

Finally, it is worth stressing that there is a delay when pressing the ‘Next’ button in the window ‘Entity Matching’ of the schema-agnostic pipelines. For small datasets, the delay is hardly observable, but it increases for larger datasets, raising up to few minutes for D_{1M}, D_{2M} and D_{c8}. This delay is caused by a process that converts all entity profiles into the representation model of the selected Entity Matching method. This is included in the running times of $Ubuntu - base1$, where all experiments were run through script files, but is not considered by any other testing platform, where all experiments were executed through JedAI’s user interface. This is one of the reasons for the significantly higher running times of $Ubuntu - base1$ even in comparison to similar testing platforms, like $Ubuntu - base2$.

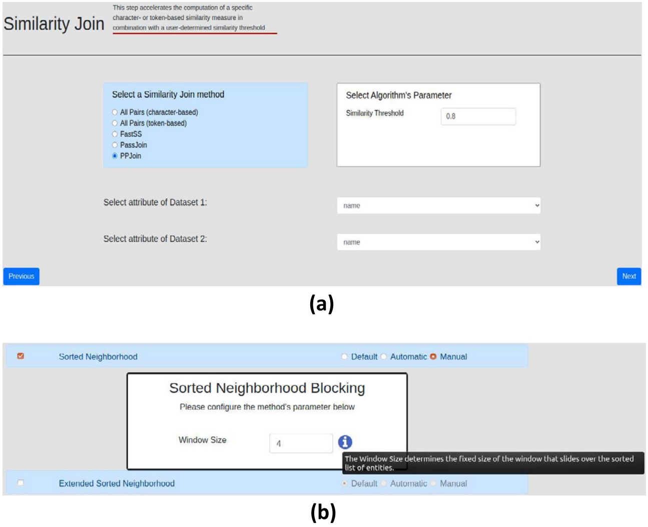
3. Reconfiguring and extending our experiments

3.1. Evaluating different experimental setups

To test the robustness of our experimental study, the configuration of a particular experiment can be adjusted in two different ways as follows:

1. by enriching or modifying the methods of at least one pipeline step, and/or
2. by altering the value of at least one parameter in one of the selected methods.

This is possible by repeating the procedure in Table 7 up to the first window of Step 10, namely ‘Data Reading’. Subsequently, in the separate window of each step, the pre-selected options can be modified as described below, in Sections 3.1.1–3.1.3, for each type of experiments.



Similarity Join
This step accelerates the computation of a specific character- or token-based similarity measure in combination with a user-determined similarity threshold

Select a Similarity Join method
○ All Pairs (character-based)
○ All Pairs (token-based)
○ FastSS
○ PassJoin
● PPJoin

Select Algorithm's Parameter
Similarity Threshold: 0.8

Select attribute of Dataset 1: name
Select attribute of Dataset 2: name

☑ Sorted Neighborhood ○ Default ○ Automatic ● Manual

Sorted Neighborhood Blocking
Please configure the method's parameter below
Window Size: 4

The Window Size determines the fixed size of the window that slides over the sorted list of entities.

☐ Extended Sorted Neighborhood ● Default ○ Automatic ○ Manual

Fig. 3. (a) The screen showing the configuration for a particular pipeline step. (b) The tooltip that explains the role of a particular parameter during the manual configuration of a method.

Note that every method in every pipeline step is associated with three configuration approaches: ‘Default’, ‘Automatic’, ‘Manual’. The ‘Default’ configuration is already widely used in the experimental analysis of [12]. The ‘Automatic’ configuration applies grid or random search over numerous iterations so as to identify the settings that maximize F-Measure. The random search involves 100 iterations, while the grid search might yield an exponential number of iterations in case multiple parameters are simultaneously fine-tuned. As both options might lead to long running times, the preferred approach is the ‘Manual’ configuration. After selecting it, JedAI presents all parameters of the current pipeline along with their default values, as in Fig. 3(a). The user can alter these values at will and store them by pressing ‘Next’ to proceed to the next window.

Note also that every method in JedAI implements the IDocumentation interface, which conveys all necessary information for its manual configuration. When configuring a specific parameter, the information image ⓘ is shown. When leaving the mouse cursor over it, a tooltip appears that describes the role of this parameter. An example is shown in Fig. 3(b).

Below, we explain the restrictions that apply to each pipeline step with respect to the methods that can be selected.

3.1.1. Schema-agnostic end-to-end pipeline

As explained above, this pipeline involves six steps:

1. Schema Clustering. At most one method can be selected, but this step is not used in the considered experiments.
2. Block Building. One or more of the nine available methods can be selected. All experiments exclusively employ Token Blocking, which is a parameter-free approach.
3. Block Cleaning. Any combination of the three available methods is possible. All experiments apply Comparison-based Block Purging and Block Filtering with their default parameter values.
4. Comparison Cleaning. At most one of the nine available methods can be selected. In our experiments, we exclusively use Cardinality Node Pruning (CNP) with its default configuration. All methods are configured simply by selecting one of the six weighting schemes.
5. Entity Matching. One of the two available methods can be applied. All experiments employ the Profile Matcher. Both methods are configured by selecting a similarity measure and a compatible representation model, which transforms the set of textual attribute values in each entity profile into a suitable format. These two parameters give rise to numerous configurations.

6. Entity Clustering. At most one method can be selected in this step. There are three methods available for Clean-Clean ER, but all experiments employ the Unique Mapping Clustering approach. For Dirty ER, there are seven methods for Dirty ER, but all experiments use the Connected Components Clustering. All methods are configured by setting their similarity threshold, below which all pairwise comparisons are discarded.

3.1.2. Schema-based end-to-end pipeline

This pipeline consists of two steps:

1. The Similarity Join step offers five similarity join algorithms. Among them, PPJoin is used in all experiments. All methods are configured by setting their similarity threshold along with the attribute(s), to which they are applied.
2. The Entity Clustering step is the same as the schema-agnostic pipeline. In most cases, it uses the same similarity threshold as the previous step.

3.1.3. Budget-aware schema-agnostic pipeline

This pipeline differs from its budget-agnostic counterpart (see Section 3.1.1) only in the Prioritization step that intervenes between Comparison Cleaning and Entity Matching. There are different options for this step, depending on the preceding pipeline steps: if no Block Building method is employed, two methods are available, otherwise one of five different methods can be used. The latter approach was used in all Budget-awareness tests. In both cases, at most one approach can be selected and it is configured by setting its budget (i.e., number of executed comparisons) and the weighting scheme that lies at its core.

Note that for all tests, the next configuration experiment is performed by pressing the '`Start Over`' button at the bottom right corner of Fig. 2(f) to return to the Data Reading step of the current experiment.

3.2. Extending our experiments

Our experimental study can be extended in two ways. First, by adding new datasets through the '`Data Reading`' step. The window of this step allows users to select any dataset in any of the supported formats (CSV, relational DB, XML or RDF) that is stored either locally or is available through a server with a public URL. Note that each dataset should be accompanied by the golden standard comprising all duplicates.

Second, it is possible to extent our experimental analysis with new methods in any of the considered pipeline steps by leveraging JedAI's extensible architecture. The only requirement is that every new method is available through a Java class that implements the interface of the corresponding pipeline step - as explained in [12], every step is associated with a simple Java interface that determines its input and output. In this way, new methods can be seamlessly integrated into JedAI's code and be treated like the already available methods. Ideally, the new methods should also implement the `IDocumentation` interface, which exposes the following functions that return textual descriptions about the core characteristics of an algorithm:

- `getMethodName()` returns the name of the method.
- `getParameterName(int parameterId)` returns the name of a particular configuration parameter.
- `getParameterDescription(int parameterId)` returns a short description for a particular configuration parameter.
- `getMethodParameters()` returns a description for all configuration parameters of the method, using the above functions.
- `getMethodInfo()` returns a short description of the method's internal functionality.
- `getMethodConfiguration()` returns the parameter configuration of the current instance of a method. It is called by logger.
- `getParameterConfiguration()` returns a `JsonArray` object with a `JsonObject` for every configuration parameter that comprises the following information: the class of the parameter (e.g., `java.lang. Integer`), its name, determined by the function `getParameterName`, its default, minimum and maximum values along with the step one, and its description, determined by the function `getParameterDescription`. This information is used for the manual configuration through JedAI's interface.

This documentation, which is also leveraged by JedAI's user interface, ensures that new methods can be easily employed by users other than their creators. For more details on extending JedAI please refer to [12].

4. Conclusions

We have presented an analytical user guide for JedAI's Web application, which is available through a Docker image. Our instructions allow a user with limited or no familiarity with Entity Resolution to repeat all single-core experiments in [12] so as to evaluate the relative performance of the main end-to-end pipelines. Our instructions also facilitate the reconfiguration of these experiments, by constructing and evaluating pipelines of arbitrary complexity.

All these experiments involve learning-free methods. In the future, we plan to extend JedAI with learning-based methods, paying particular attention to the integration of Deep Learning technologies.

5. Revision comments

This reproducibility manuscript is a valuable complement to the parent paper [12], where the last release of JedAI software was presented. JedAI includes a web-based user interface and a complete library of techniques needed to create end-to-end Entity Resolution (ER) pipelines. The authors compared different ER techniques by considering three different dimensions that included: (a) Schema-awareness, (b) Budget-awareness, and (c) Execution mode. The wide set of experiments provided included the evaluation of 17 datasets and considered the performance, scalability, and budget awareness of the ER pipelines. This paper provides the actual configuration used for those ER pipelines, and gives some ideas regarding how they can be personalized. Furthermore, some guidelines showing how JedAI can be extended are also devised.

Apart from creating a permanent repository in Mendeley with the necessary software and datasets, the authors provide a Docker-based system to reproduce those experiments. Using the web-based interface of JedAI, any researcher can easily use the default configuration parameters provided for each experiment, execute it, and finally see the results of that execution. Besides, JedAI also allows to configure and personalize those default parameters, as well as the addition of new methods for the comparison with existing methods, adding extra value to the current work.

While reviewing this manuscript, a few issues around reproducibility were brought into the discussion, which show how difficult it can be to provide a complete reproducible framework. We dealt with some experiments where the provided default parameters were wrong, which led to unexpected results. Another minor issue was related to yielding slightly different values than those reported in the parent paper or figures showing the results in a rather different shape. We also found some mismatches concerning the memory requirements needed to run some experiments, which would not end or report higher execution times than expected. All those issues were successfully fixed during the revision process. The authors satisfactorily took all our comments into account and improved their software library and web application. Finally, the JedAI reproduction framework does not provide a mechanism to automatically run all the experiments, gather all the results, and create the same tables and figures of the parent paper, which would be extremely interesting to reproduce the original work easily. However, the workflow included in JedAI still allows any researcher to effortlessly reproduce each experiment. The process consists of choosing the experiment to perform, going through the screens that display the default parameters, starting the execution, waiting for it to complete, and finally gathering the results.

We would like to thank the authors for their considerable effort to provide a valuable software library to the research community. This library allows new researchers to understand and reproduce state-of-the-art experiments with minimal effort and guarantees long-term software support, following a sequence of precise and straightforward instructions.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: (1) Dr. Dimitrios Skoutas, Athena Research Center, Greece (2) Prof. Ekaterini Ioannou, Tilburg University, The Netherlands (3) Prof. Dr. Erhard Rahm, University of Leipzig, Germany (4) Prof. Nikolaus Augsten, University of Salzburg, Austria (5) Prof. Vassilis Christophides, University of Crete, Greece (6) Prof. Kostas Stefanidis, University of Tampere, Finland

Acknowledgment

This work was partially funded by the EU H2020 project ExtremeEarth (Grant No. 825258).

References

[1] G. Papadakis, E. Ioannou, E. Thanos, T. Palpanas, The Four Generations of Entity Resolution, in: Synth. Lect. Data Manage., Morgan & Claypool Publishers, 2021.

[2] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, in: Data-Centric Systems and Applications, Springer, 2012.

[3] X.L. Dong, D. Srivastava, Big Data Integration, in: Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2015.

[4] V. Christophides, V. Efthymiou, K. Stefanidis, Entity Resolution in the Web of Data, in: Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2015.

[5] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: A survey, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 1–16.

[6] G. Papadakis, D. Skoutas, E. Thanos, T. Palpanas, Blocking and filtering techniques for entity resolution: A survey, ACM Comput. Surv. 53 (2) (2020) 31:1–31:42.

[7] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, K. Stefanidis, An overview of end-to-end entity resolution for big data, ACM Comput. Surv. 53 (6) (2020).

[8] L. Getoor, A. Machanavajjhala, Entity resolution: Theory, practice & open challenges, Proc. VLDB Endow. 5 (12) (2012) 2018–2019.

[9] K. Stefanidis, V. Efthymiou, M. Herschel, V. Christophides, Entity resolution in the web of data, in: The Web Conference (WWW), 2014, pp. 203–204.

[10] G. Papadakis, T. Palpanas, Web-scale, schema-agnostic, end-to-end entity resolution, in: The Web Conference (WWW), Lyon, France, 2018.

[11] G. Papadakis, E. Ioannou, T. Palpanas, Entity resolution: Past, present and yet-to-come, in: EDBT, 2020, pp. 647–650.

[12] G. Papadakis, G. Mandilaras, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, M. Koubarakis, Three-dimensional entity resolution with JedAI, Inf. Syst. 93 (2020) 101565.

[13] G. Papadakis, Entity resolution benchmark dataset, 2020, https://data.mendeley.com/datasets/4whpm32y47.

[14] J. Euzenat, A. Ferrara, C. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, V. Svátek, C.T. dos Santos, Results of the Ontology Alignment Evaluation Initiative 2010, in: Proceedings of the 5th International Workshop on Ontology Matching (OM-2010), 2010.

[15] Ontology alignment evaluation initiative, 2010, http://oaei.ontologymatching.org/2010.

[16] H. Köpcke, A. Thor, E. Rahm, Evaluation of entity resolution approaches on real-world match problems, Proc. VLDB Endow. 3 (1) (2010) 484–493.

[17] Benchmark datasets for entity resolution, 2010, https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution.

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

[19] S. Das, A. Doan, G.C.P. Suganthan, C. Gokhale, P. Konda, Y. Govind, D. Paulsen, The Magellan Data Repository, https://sites.google.com/site/anhaidgroup/projects/data.

[20] G. Papadakis, E. Ioannou, C. Niederée, P. Fankhauser, Efficient entity resolution for large heterogeneous information spaces, in: WSDM, 2011, pp. 535–544.

[21] G. Papadakis, Blocking Framework, https://sourceforge.net/projects/erframework/.

[22] A. McCallum, K. Nigam, L.H. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, in: KDD, 2000, pp. 169–178.

[23] Repeatability Datasets, https://hpi.de/naumann/projects/repeatability/datasets.html.

[24] U. Draisbach, F. Naumann, A comparison and generalization of blocking and windowing algorithms for duplicate detection, in: Proceedings of the International Workshop on Quality in Databases (QDB), 2009, pp. 51–56.

[25] B. Kenig, A. Gal, Mfiblocks: An effective blocking algorithm for entity resolution, Inf. Syst. 38 (6) (2013) 908–926.

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

[27] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Deep learning for entity matching: A design space exploration, in: SIGMOD, 2018, pp. 19–34.