This is a pre print version of the following article:

Reproducible experiments on Three-Dimensional Entity Resolution with JedAI / Mandilaras, George; Papadakis, George; Gagliardelli, Luca; Simonini, Giovanni; Thanos, Emmanouil; Giannakopoulos, George; Bergamaschi, Sonia; Palpanas, Themis; Koubarakis, Manolis; Lara-Clares, Alicia; Farina, Antonio. - In: INFORMATION SYSTEMS. - ISSN 0306-4379. - 102:(2021), pp. 101830-101830. [10.1016/j.is.2021.101830]

Terms of use:
The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher’s website.

21/08/2024 12:53

(Article begins on next page)
Reproducible experiments on Three-Dimensional Entity Resolution with JedAI

George Mandilas1, George Papadakis1∗, Luca Gagliardelli2, Giovanni Simonini2, Emmanouil Thanos3, George Giannakopoulos4, Sonia Bergamaschi5, Themis Palpanas5, Manolis Koubarakis1, Alicia Lara-Clare6∗∗, Antonio Fariña7∗∗∗
1National and Kapodistrian University of Athens, Greece {gmandi,gpapadis,koubarakis}@di.uoa.gr
2University of Modena and Reggio Emilia, Italy {name.surname}@unimore.it
3KU Leuven, Belgium emmanouil.thanos@kuleuven.be
4NCSR “Demokritos”, Greece ggianna@iit.demokritos.gr
5University of Paris & French University Institute (IUF), France themis@mi.parisdescartes.fr
6NLP&IR Research Group, Universidad Nacional de Educaci´on a Distancia (UNED), Spain alara@ls.i.uned.es
7University of A Coruña, CITIC, Database Lab, Spain antonio.farina@uad.es

Abstract
In Papadakis et al. [1], 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.

Keywords: Entity Resolution, Batch Methods, Progressive Methods, Reproducibility

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 and Wikipedia 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 [2,3,4,5], surveys [6,7,8] and tutorials [9,10,11,12]. To facilitate the use of the main ER methods, we created JedAI [1], 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 [3,4,5].

Using JedAI, we experimentally evaluated in [1] 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 Figure 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.
 Entity Matching

Data Reading
Block Cleaning
Comparison Cleaning
Entity Clustering
Data Writing & Evaluation

Figure 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.

- **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 Figure 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: $\text{JaccardSim(title_1, title_2)} > 0.8$.

In [1], we also compare the batch, schema-agnostic pipeline with its progressive counterpart, i.e., the **budget-aware** schema-agnostic pipeline, which is shown in Figure 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=OJY1DUrU AeB

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.

- **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.
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, $|\bar{p}|$ for the average profile size (in terms of name-value pairs), $|D(E)|$ for the number of duplicate pairs, and $||E||$ for the combinations executed by the brute-force approach.

| Dataset | $|E|$ | NVP | $|\bar{p}|$ | $|D(E)|$ | $||E||$ |
|---------|------|-----|---------|--------|-------|
| cora    | 1.16 | 1.69 | 6.40    | 6.00   | 3.92  |
| D       | 12   | 106  | 12      | 12     | 12    |
| t1      | 5.53 | 18.75| 10.62   | 10.62  | 10.62 |
| t2      | 17.18| 299  | 8.705   | 172.403| 57.538|
| t3      | 8.38 | 4.77 | 5.00    | 2.50   | 2.50  |

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

| Dataset | $|E_1|/|E_2|$ | NVP | $|\bar{p}_1|/|\bar{p}_2|$ | $|E_1 \bigcap E_2|$ | $||E_1 \times E_2||$ |
|---------|-------------|-----|------------------|-----------------|-----------------|
| D1      | 339/2,256   | 1,076/1,076| 1.354/3,039 | 2,616/2,294 | 2,554/22,074 |
| D2      | 1,130/7,519| 2,568/2,308| 5,302/9,162 | 10,456/9,162| 14,143/1.10|
| D3      | 7/7         | 3/3    | 4/4           | 4/4            | 6/6          |
| D4      | 3.33/3.33  | 2.39/2.14| 3.92/3.00    | 3.99/4.00     | 5.54/5.18   |
| D5      | 89          | 1.076  | 1.104        | 2.224         | 853           |
| D6      | 7.65/106    | 1.16/106| 4.11/106     | 6.00/106      | 5.64/107    |

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 $\text{Xmx}$ 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 $\text{[1]}$ 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 a Mendeley dataset [13] and through entity profiles. They are all publicly available in the form of a standard, i.e., the complete ground-truth of the actual duplicate entity profiles. They are all publicly available in the form of a Mendeley dataset [13] and through entity profiles.

Performance Tests. These experiments, which are reported in Table 4 of $\text{[1]}$, 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 $D_{dabh}$) and the rest to Clean-Clean ER ($D_{1-d, D_{3-b}}$). 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.

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 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 $\text{[1]}$, 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 input data increases.

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

https://github.com/scify/JedAIToolkit
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 $D_{\wedge}$ 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 |
|---------|----------|-----------------|------------------------------------------|------------------------------------------|--------|
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/restaurants1Profiles | Real Clean-Clean ER data/restaurants1IdDuplicates | [15]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/restaurants2Profiles | Real Clean-Clean ER data/restaurants2IdDuplicates | [15]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/abtProfiles | Real Clean-Clean ER data/abtBuyIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/buyProfiles | Real Clean-Clean ER data/buyIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/amazonProfiles | Real Clean-Clean ER data/amazonGplIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/gpProfiles | Real Clean-Clean ER data/gpIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/acmProfiles | Real Clean-Clean ER data/acmIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/walmartProfiles | Real Clean-Clean ER data/walmartIdDuplicates | [19]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/dblpProfiles2 | Real Clean-Clean ER data/dblpScholarIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/scholarProfiles | Real Clean-Clean ER data/scholarIdDuplicates | [17]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/dblpProfiles | Real Clean-Clean ER data/dblpIdDuplicates | [21]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/cleanDBPedia1 | Real Clean-Clean ER data/cleanDBPedia1IdDuplicates | [21]   |
| $D_{\wedge}$ | Real     | Clean-Clean ER | Real Clean-Clean ER data/cleanDBPedia2 | Real Clean-Clean ER data/cleanDBPedia2IdDuplicates | [21]   |
| $D_{\wedge}$ | Real     | Dirty ER       | Real Dirty ER data/coraProfiles | Real Dirty ER data/coraIdDuplicates | [23]   |
| $D_{\wedge}$ | Real     | Dirty ER       | Real Dirty ER data/cdbProfiles | Real Dirty ER data/cdbIdDuplicates | [23]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/10Kprofiles | Synthetic Dirty ER data/10KIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/50Kprofiles | Synthetic Dirty ER data/50KIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/100Kprofiles | Synthetic Dirty ER data/100KIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/200Kprofiles | Synthetic Dirty ER data/200KIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/300Kprofiles | Synthetic Dirty ER data/300KIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/1Mprofiles | Synthetic Dirty ER data/1MIdDuplicates | [21]   |
| $D_{\wedge}$ | Synthetic | Dirty ER       | Synthetic Dirty ER data/2Mprofiles | Synthetic Dirty ER data/2MIdDuplicates | [21]   |

noise (see [11] 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., $Jaccard_{\text{sim}}(all\_tokens_1, all\_tokens_2) > 0.4$, 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 [11]. They compare the budget-awareness and schema-agnostic pipeline with its budget-aware counterpart across the same datasets as the Performance Tests - except the largest one, $D_{\wedge}$. For each dataset, the parameter configuration that corresponds to the optimal performance of the budget-aware version costs much fewer memory, as indicated by the memory requirements that are reported in Tables 8, 9 and 10 for each experiment. In our tests, though, 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 that the option `-e JAVA_OPTS=-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, 9 and 10 for each experiment. In our tests, though, 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. Thus, this option determines the directory on the host system (`/home/user/jedai` for Debian

7See [https://docs.docker.com/engine/install/debian](https://docs.docker.com/engine/install/debian) for detailed instructions.

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

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

2.3. Experimental setup in our primary paper

All single-core experiments in [11] were implemented in Java 8 and can be reproduced through JedAI’s Docker image, which is publicly available. The only requirement is to have Docker 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 and CentOS. 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_OPTS=-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, 9 and 10 for each experiment. In our tests, though, 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. Thus, this option determines the directory on the host system (`/home/user/jedai`),
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](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  
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 |
| 2    | Add Docker’s official GPG key.  
$ sudo apt-key add - 
Set up the stable repository.  
$ sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu/gpg” |
| 3    | $ sudo apt-get update  
Install the latest version of Docker Engine.  
$ sudo apt-get -y install docker-ce docker-ce-cli containerd.io  
Verify that Docker Engine is installed correctly.  
$ sudo docker run hello-world  
Download the latest JedAI Docker image from Docker Hub.  
$ sudo docker pull gmandi/jedai-webapp:latest |
| 4    | 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](https://data.mendeley.com/public-files/datasets/4whpm32y47/files/79f5ccdd-e60a-4f9c-99cb-8f2d7ef0fc25/file_downloaded)  
$ sudo docker load  
Launch the JedAI Web application.  
Note that parameter -Xmx4g allows JedAI to use up to 4Gb 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 |

where Docker will store and unpack the dataset files as long as it has user-write permissions.

The following steps are reproduceable on any Windows system.

| 8    | Add Docker's official GPG key.  
$ sudo apt-key add -  
Set up the stable repository.  
$ sudo add-apt-repository “deb [arch=amd64] https://download.docker.com/linux/ubuntu/gpg” |
| 9    | $ sudo apt-get update  
Install the latest version of Docker Engine.  
$ sudo apt-get -y install docker-ce docker-ce-cli containerd.io  
Verify that Docker Engine is installed correctly.  
$ sudo docker run hello-world  
Download the latest JedAI Docker image from Docker Hub.  
$ sudo docker pull gmandi/jedai-webapp:latest |

203 it has user-write permissions.

204 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.

210 Finally, it is worth noting that it is also possible to use Docker on Windows 10. The installation is a straightforward process that merely needs some additional steps. After the successful installation, all experiments can be seamlessly run, without any performance issue. Indeed, one of our testing platforms forms runs on Windows 10 Pro (Windows – base1 in Table 5).

241 2.4. System requirements and performance evaluation

242 All single-core experiments in [1] can be reproduced on any Java 8 compliant platform, which practically includes all major 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.

250 Our original configuration corresponds to Ubuntu – base1 for the Performance and Scalability Tests and to Ubuntu – base1V 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 – base1V, all experiments were run through script files 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.

270 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. 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.

277 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 executions, one can simply run the command sudo apt-get update followed by sudo apt-get upgrade to recover the space occupied after multiple executions.

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

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

[12]The source code of all tests is available at: [https://github.com/scify/JedAItoolkit/tree/master/src/test/java/org/scify/](https://github.com/scify/JedAItoolkit/tree/master/src/test/java/org/scify/)  
[13]For more details, please refer to [https://stackoverflow.com/questions/30248794/run-docker-in-ubuntu-live-disk](https://stackoverflow.com/questions/30248794/run-docker-in-ubuntu-live-disk)
reported in Tables 8 and 9, while Ubuntu was only used for the experiments in Table 10. The testing platforms that were successfully used to reproduce our experiments. Note that Ubuntu was used in [1] for performing the experiments reported in Tables 8 and 9, while Ubuntu was only used for the experiments in Table 10.

Table 5: The testing platforms that were successfully used to reproduce our experiments. Note that Ubuntu was used in [1] for performing the experiments reported in Tables 8 and 9, while Ubuntu 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     | 1 Intel Xeon E5-4603 v2 @2.20GHz, 128 Gb DDR3 RAM, 1.6 Tb mechanical disk | Authors   |
| Ubuntu - base'   | Server | Ubuntu 14.04 LTS       | 1 Intel Xeon E5-2670 v2 @2.50GHz, 80GB DDR3 RAM, 1Tb mechanical disk | Authors   |
| Ubuntu - base2   | Server | Ubuntu 14.04.6 LTS     | 1 AMD Opteron 6320 @2.80GHz, 128 Gb DDR3 RAM, 1.6 Tb mechanical disk | Authors   |
| Ubuntu - base3   | Laptop | Ubuntu 18.04.5 LTS     | 1 Intel Core i7-4710MQ @2.50GHz, 16 Gb DDR3 RAM, 120 Gb SSD | Authors   |
| Ubuntu - base4   | Laptop | Ubuntu 20.04 LTS       | 1 Intel Core i5-1035G1 @1.00GHz, 4 Gb DDR4 RAM, 32 Gb flash drive | Authors   |
| Ubuntu - base5   | Laptop | Linux Mint 19.1 Tessa  | 16 Gb DDR3 RAM, 1 Tb mechanical disk | Authors   |
| Ubuntu - base6   | Laptop | Ubuntu 20.04.2 LTS     | Intel Core i7-9750H @2.60GHz, 32 GB RAM, 2.5Tb mechanical disk | Reviewer  |
| Ubuntu - base7   | Server | Ubuntu 20.04.2 LTS     | 1 Intel Xeon Bronze 3204 @1.9GHz, 512 Gb DDR4 RAM, 120Gb mechanical disk | Reviewer  |
| Ubuntu - base8   | Server | Ubuntu 16.04.7 LTS     | 1 Intel Core i7 8700k @3.7GHz, 64Gb swap, 64 Gb DDR4 RAM, 3Tb mechanical disk | Reviewer  |
| Ubuntu - base9   | Laptop | Ubuntu 20.04.1 LTS     | 1 Intel Core i5 8265u @1.6GHz, 16 DDR4 RAM, no swap, 34Gb virtual disk over SSD | Reviewer  |
| Windows - base1  | Laptop | Windows 10 Pro v. 20H2 | 1 Intel Core i5-1035G1 @1.00GHz, 6 Gb DDR4 RAM, 240 Gb SSD | Authors   |

Table 6: The aggregate time required to run all the experiments included in Tables 8, 9, and 10 (that could be completed in less than 40 hours) for each testing platform, while reproducing most experiments from [1]. The testing platforms Ubuntu - base3, Ubuntu - base4, Ubuntu - base5, Ubuntu - base6, Ubuntu - base 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   | 5,526 min ≈ 92.1 hrs | Authors   |
| 2   | Ubuntu - base2   | 6,832 min ≈ 113.9 hrs | Authors   |
| 3   | Ubuntu - base3   | 2,678 min ≈ 44.6 hrs | Authors   |
| 4   | Ubuntu - base4   | 187 min ≈ 3.1 hrs | Authors   |
| 5   | Ubuntu - base5   | 2,198 min ≈ 36.6 hrs | Authors   |
| 6   | Ubuntu - base6   | 1,428 min ≈ 23.8 hrs | Reviewer  |
| 7   | Ubuntu - base7   | 6,393 min ≈ 106.5 hrs | Reviewer  |
| 8   | Ubuntu - base8   | 3,212 min ≈ 53.5 hrs | Reviewer  |
| 9   | Ubuntu - base9   | 1,731 min ≈ 28.8 hrs | Reviewer  |
| 10  | Windows - base1  | 1,743 min ≈ 29.1 hrs | Authors   |

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, 9, and 10 but 4 Gb are suggested to ensure Docker's stability, as explained above. However, the experiments with the two largest synthetic datasets, D1M and D2M, require up to 25 Gb, whereas the largest real dataset, D8K, 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, recover disk space for unused volumes: 

```
sudo docker volume prune
```

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 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 [1] 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 Figure 2. This means that minimal human intervention is required. For example, all datasets in Tables 1, 2 and 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 tab ‘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 to 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 Figure 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.
Figure 2: The screens of JedAI’s Web application for reproducing all single core experiments in [1]: (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.
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 |
|----------|----------------|----------|
|          | Precision      | Recall   | F-Measure | Precision              | Recall              | F-Measure          |
|          | D_{14}         | D_{24}   | D_{34}    | D_{15}                | D_{25}              | D_{35}             |
| Ubuntu   | 0.82           | 0.81     | 0.82      | 0.81                  | 0.82                | 0.82               |
| Windows  | 0.90           | 0.93     | 0.90      | 0.93                  | 0.93                | 0.93               |
|          | (Gb)            | (Gb)     | (Gb)      | (Gb)                  | (Gb)                | (Gb)               |
|          | 1.1 sec         | 1.0 sec  | 1.0 sec   | 1.0 sec               | 1.0 sec             | 1.0 sec            |
| Ubuntu   | 1.1 sec         | 1.0 sec  | 1.0 sec   | 1.0 sec               | 1.0 sec             | 1.0 sec            |
| Windows  | 1.1 sec         | 1.0 sec  | 1.0 sec   | 1.0 sec               | 1.0 sec             | 1.0 sec            |

(a) Default configuration of the budget- and schema-agnostic pipeline

(b) Best configuration of the budget- and schema-agnostic pipeline

(c) Best configuration of the budget-agnostic, schema-based pipeline

The outcomes of the Performance, the Scalability and the Budget-awareness tests over all testing platforms are reported in Tables S5, S9 and S10, respectively. In all cases, the effectiveness measures per dataset are common among all platforms, with the only differences corresponding to the running times. Compared to the experiments reported in [1], the effectiveness results of the Budget-awareness tests are practically identical in most cases. The only significant exceptions pertain to the best schema-agnostic pipeline over D_{24}, D_{34} and D_{44}, 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_{34} has also increased from 0.586 to 0.594. The effectiveness results of the Scalability and the Budget-awareness tests are also identical.
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. \(1M\) 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.

|                | \(D_{10K}\) | \(D_{50K}\) | \(D_{100K}\) | \(D_{200K}\) | \(D_{300K}\) | \(D_{1M}\) | \(D_{2M}\) |
|----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 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 sec     | 12.8 sec    | 35.1 sec    | 120.2 sec   | 193.1 sec   | 32.3 min    | 147.1 min   |
| Ubuntu – base2 | 1.6 sec     | 11.4 sec    | 37.3 sec    | 130.8 sec   | 199.3 sec   | 33.4 min    | 145.4 min   |
| Ubuntu – base3 | 1.4 sec     | 5.2 sec     | 19.8 sec    | 51.9 sec    | 141.9 sec   | 22.1 min    | 145.4 min   |
| Ubuntu – base4 | 0.9 sec     | 4.3 sec     | 10.7 sec    | 54.3 sec    | IM          | IM          | IM          |
| Ubuntu – base5 | 1.0 sec     | 3.7 sec     | 11.5 sec    | 37.8 sec    | 77.0 sec    | 16.9 min    | IM          |
| Ubuntu – base6 | 0.7 sec     | 4.2 sec     | 10.8 sec    | 36.7 sec    | 114.6 sec   | –           | –           |
| Ubuntu – base7 | 0.8 sec     | 6.3 sec     | 19.9 sec    | 63.4 sec    | 148.0 sec   | 24.2 min    | 93.5 min    |
| Ubuntu – base8 | 0.5 sec     | 3.6 sec     | 9.0 sec     | 27.1 sec    | 71.3 sec    | 12.9 min    | 51.4 min    |
| Ubuntu – base9 | 1.5 sec     | 8.2 sec     | 15.7 sec    | 46.9 sec    | 118.4 sec   | 22.6 min    | IM          |
| Windows – base1| 1.4 sec     | 5.0 sec     | 10.8 sec    | 47.2 sec    | 232.5 sec   | IM          | IM          |

(a) Default configuration of the budget- and schema-agnostic pipeline

|                | 1.000       | 1.000       | 1.000       | 1.000       | 1.000       | 1.000       | 1.000       |
|----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| Precision      | 0.593       | 0.598       | 0.602       | 0.600       | 0.602       | 0.603       | 0.602       |
| Recall         | 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 sec     | 137.2 sec   | 695.3 sec   | 55.6 min    | 140.3 min   | 17.8 hrs    | >40 hrs     |
| Ubuntu – base2 | 5.3 sec     | 120.3 sec   | 534.8 sec   | 49.6 min    | 96.9 min    | 19.3 hrs    | >40 hrs     |
| Ubuntu – base3 | 4.0 sec     | 89.4 sec    | 367.8 sec   | 26.3 min    | 69.4 min    | 13.0 hrs    | >40 hrs     |
| Ubuntu – base4 | 3.9 sec     | 74.6 sec    | 316.5 sec   | 24.0 min    | 48.4 min    | IM          | IM          |
| Ubuntu – base5 | 3.6 sec     | 67.9 sec    | 298.2 sec   | 21.3 min    | 55.9 min    | 10.3 hrs    | >40 hrs     |
| Ubuntu – base6 | 3.8 sec     | 78.6 sec    | 341.5 sec   | 23.6 min    | 57.1 min    | –           | >40 hrs     |
| Ubuntu – base7 | 7.9 sec     | 172.2 sec   | 704.1 sec   | 49.5 min    | 111.9 min   | 19.8 hrs    | >40 hrs     |
| Ubuntu – base8 | 3.1 sec     | 140.1 sec   | 375.2 sec   | 22.3 min    | 49.7 min    | 10.2 hrs    | 39.8 hrs    |
| Ubuntu – base9 | 5.3 sec     | 96.8 sec    | 376.4 sec   | 28.7 min    | 64.5 min    | 10.8 hrs    | >40 hrs     |
| Windows – base1| 4.3 sec     | 87.3 sec    | 376.7 sec   | 26.7 min    | 56.8 min    | IM          | IM          |

(b) Best configuration of the budget-agnostic, schema-based pipeline

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 \(D_{5M}\). This delay is caused by a process that converts all entity profiles into the representation 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 few minutes for \(D_{1M}\), \(D_{2M}\) and \(D_{5M}\). This delay is caused by a process that converts all entity profiles into the representation 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 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 and 3.1.2 for each type of experiments.

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 Table 1. The 'Automatic' configuration applies grid or random search over numerous iterations.
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 $D_{sk}$ is omitted, as in [1], due to the excessively large running time and the very high memory requirements of the corresponding experiment.

|                  | Ubuntu ~ base1 | Ubuntu ~ base2 | Ubuntu ~ base3 | Ubuntu ~ base4 | Ubuntu ~ base5 | Ubuntu ~ base6 | Ubuntu ~ base7 | Ubuntu ~ base8 | Ubuntu ~ base9 | Windows ~ base1 | Abt | Buy | GP | ACM | Amazon | DBLP | Walmart | DBLP | IMDB | DBpedia |
|------------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|------|-----|----|-----|--------|------|---------|------|------|--------|
| 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          | 6.00          | 0.30          | 3.50          |                |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base1   | 0.3 sec       | 13.7 sec      | 1.8 min       | 32.9 sec      | 5.2 min       | 46.3 min      | 18.4 hrs      | 16.9 sec      | 79.3 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base2   | 0.6 sec       | 19.0 sec      | 3.4 min       | 49.8 sec      | 7.8 min       | 68.2 min      | 20.1 hrs      | 15.8 sec      | 96.5 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base3   | 0.4 sec       | 14.6 sec      | 2.2 min       | 48.5 sec      | 7.4 min       | 54.1 min      | 12.7 hrs      | 12.2 sec      | 73.5 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base4   | 0.3 sec       | 12.5 sec      | 1.5 min       | 35.7 sec      | 6.1 min       | 40.7 min      | IM           | 9.9 sec       | 55.6 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base5   | 0.6 sec       | 12.2 sec      | 2.0 min       | 31.5 sec      | 5.1 min       | 37.2 min      | 10.8 hrs      | 9.4 sec       | 49.9 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base6   | 0.3 sec       | 11.9 sec      | 1.4 min       | 33.5 sec      | 4.9 min       | 34.5 min      | 9.6 hrs       | 10.1 sec      | 53.8 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base7   | 0.4 sec       | 20.9 sec      | 2.7 min       | 63.1 sec      | 9.4 min       | 64.7 min      | 22.0 hrs      | 19.6 sec      | 107.8 sec     |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base8   | 0.3 sec       | 10.2 sec      | 1.2 min       | 30.0 sec      | 4.2 min       | 28.7 min      | 9.5 hrs       | 8.8 sec       | 45.8 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Ubuntu ~ base9   | 0.8 sec       | 16.6 sec      | 2.2 min       | 48.8 sec      | 6.9 min       | 38.8 min      | 13.9 hrs      | 14.3 sec      | 66.9 sec      |                |     |     |    |     |        |      |         |      |      |        |
| Windows ~ base1  | 0.5 sec       | 14.9 sec      | 1.9 min       | 20.4 sec      | 8.3 min       | 52.4 min      | 11.3 hrs      | 15.3 sec      | 80.0 sec      |                |     |     |    |     |        |      |         |      |      |        |

(a) Budget-aware, schema-agnostic pipeline

(b) Budget- and schema-agnostic pipeline
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 of a method.

Figure 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.

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 Figure 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 its documentation 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 Figure 3(b).

Below, we explain the restrictions that apply to each pipeline step with respect to the methods that can be selected.
explained in [1], 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 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 [1].

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 [1] with the parent paper 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.

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

References

[1] 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.

[2] G. Papadakis, E. Ioannou, E. Thanos, T. Palpanas, The four generations of entity resolution, Synthesis Lectures on Data Management.

[3] P. Christen, Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.

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

[5] V. Christophides, V. Efthymiou, K. Stefanidis, Entity Resolution in the Web of Data, Synthesis Lectures on the Semantic Web.

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

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

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

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

[10] K. Stefanidis, V. Efthymiou, M. Herschel, V. Christophides, Entity resolution in the web of data, in: WWW, 2014, pp. 203–204.

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

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

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

[14] J. Euzenat, A. Ferrara, C. Meilicke, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváˇb-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. http://oaei.ontologymatching.org/2010 (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. https://hpi.de/naumann/projects/repeatability/datasets.html

[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, P. S. G. C., C. Gokhale, P. Konda, Y. Govind, D. Paulsen, The magellan data repository, https://sites.google.com/site/anhaiddotgroup/projects/data

[20] G. Papadakis, E. Ioannou, C. Niederére, P. Funkhauser, 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: ACM SIGKDD, 2000, pp. 169–178.

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

[24] U. Draissbach, 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. Koenig, A. Gal, Mitblocks: An effective blocking algorithm for entity resolution, Inf. Syst. 38 (6) (2013) 908–926.

[26] 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, 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.