This document is downloaded from DR-NTU (https://dr.ntu.edu.sg)
Nanyang Technological University, Singapore.

A simple and efficient approach to unsupervised instance matching and its application to linked data of power plants

Eibeck, Andreas; Zhang, Shaocong; Lim, Mei Qi; Kraft, Markus

2024

Eibeck, A., Zhang, S., Lim, M. Q. & Kraft, M. (2024). A simple and efficient approach to unsupervised instance matching and its application to linked data of power plants. Journal of Web Semantics, 80, 100815-. https://dx.doi.org/10.1016/j.websem.2024.100815

https://hdl.handle.net/10356/178350

https://doi.org/10.1016/j.websem.2024.100815

© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Downloaded on 25 Mar 2025 04:20:28 SGT

Web Semantics: Science, Services and Agents on the World Wide Web 80 (2024) 100815

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

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

A simple and efficient approach to unsupervised instance matching and its application to linked data of power plants

Andreas Eibeck [a], Shaocong Zhang [a], Mei Qi Lim [a], Markus Kraft [a,b,c,d,*]

[a] CARES Cambridge Centre for Advanced Research and Education in Singapore, 1 Create Way, CREATE Tower, #05-05, Singapore, 138602, Singapore
[b] Department of Chemical Engineering and Biotechnology, University of Cambridge, Philippa Fawcett Drive, Cambridge, CB3 0AS, United Kingdom
[c] School of Chemical and Biomedical Engineering, Nanyang Technological University, 62 Nanyang Drive, Singapore, 637459, Singapore
[d] The Alan Turing Institute, 2QR, John Dodson House, 96 Euston Road, London, NW1 2DB, United Kingdom

ARTICLE INFO

Keywords:
Semantic web
Linked data
Knowledge graph
Instance matching
Record linkage

ABSTRACT

Knowledge graphs store and link semantically annotated data about real-world entities from a variety of domains and on a large scale. The World Avatar is based on a dynamic decentralised knowledge graph and on semantic technologies to realise complex cross-domain scenarios. Accurate computational results for such scenarios require the availability of complete, high-quality data. This work focuses on instance matching — one of the subtasks of automatically populating the knowledge graph with data from a wide spectrum of external sources. Instance matching compares two data sets and seeks to identify instances (data, records) referring to the same real-world entity. We introduce AutoCal, a new instance matcher which does not require labelled data and runs out of the box for a wide range of domains without tuning method-specific parameters. AutoCal achieves results competitive to recently proposed unsupervised matchers from the field of Machine Learning. We also select an unsupervised state-of-the-art matcher from the field of Deep Learning for a thorough comparison. Our results show that neither AutoCal nor the state-of-the-art matcher is superior regarding matching quality while AutoCal has only moderate hardware requirements and runs 2.7 to 60 times faster. In summary, AutoCal is specifically well-suited to be used in an automated environment. We present its prototypical integration into the World Avatar and apply AutoCal to the domain of power plants which is relevant for practical environmental scenarios of the World Avatar.

1. Introduction

The Semantic Web [1] and Linked Data [2] enable data integration and semantic interoperability over the World Wide Web. Ontologies [3] define semantic terms and relationships of data in a formal way and provide the vocabularies necessary to describe entities and their properties. Recently, knowledge graphs [4] became popular as large-scale data structures storing and linking entity data.

The World Avatar builds on these concepts and related semantic technologies. While it originally focused on simulation and optimisation of eco-industrial parks to reduce waste and pollution [5], today it encompasses domains such as combustion chemistry [6], power systems [7], and urban planning [8]. Its architecture consists of a dynamic, decentralised knowledge graph. Computational agents operate on the knowledge graph to query, process and update its data. The same agents can be composed to solve complex cross-domain tasks with varying scales and initial conditions.

The following three air pollution scenarios exemplify the idea of agent composition: In the scenario presented by [9], a *power plant* agent computes emissions of a power plant located in Berlin, and a *dispersion* agent estimates the dispersion profile in the plant's neighbourhood with respect to surrounding buildings and real-time data on weather conditions. In a similar scenario [6], *ship* agents use real-time data of ship positions and types in the vicinity of Singapore to simulate their emissions while the same *dispersion* agent estimates the impact on a district of Singapore. In a future scenario, greenhouse gas emissions from various emitters could be computed and aggregated nationwide by creating more complex ensembles of agents. This scenario refers to a knowledge-graph approach for national digital twins proposed by [10].

Such scenarios assume the availability of high-quality data on entities such as power plants, ships and buildings to achieve accurate results. Additional agents which populate and update the knowledge graph with data from a multitude of providers are desirable but their

* Corresponding author at: Department of Chemical Engineering and Biotechnology, University of Cambridge, Philippa Fawcett Drive, Cambridge, CB3 0AS, United Kingdom.
E-mail address: mk306@cam.ac.uk (M. Kraft).

https://doi.org/10.1016/j.websem.2024.100815
Received 12 March 2023; Received in revised form 14 June 2023; Accepted 25 January 2024
Available online 20 February 2024
1570-8268/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Nomenclature

AG	IM scenario for products — Amazon vs. Google Products
DA	IM scenario for bibliography — DBLP vs. ACM
DG	IM scenario for power plants in the United Kingdom
DL	Deep Learning
DS	IM scenario for bibliography — DBLP vs. Google Scholar
FZ	IM scenario for restaurants — Fodor vs. Zagat
GPPD	Global Power Plant Database
HTTP	Hypertext Transfer Protocol
IM	Instance Matching
KG	IM scenario for power plants in Germany
ML	Machine Learning
MTF	Maximum Token Frequency
OWL	Web Ontology Language
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
TWA	The World Avatar
URI	Uniform Resource Identifier

automation is challenging because it involves several steps such as pre-processing, schema matching, instance matching and data fusion. If these steps are not performed correctly, the quality of the knowledge graph will degrade. In this paper, we focus on the subtask of instance matching on tabular data sets as the first step towards automation.

Instance matching considers two data sets and aims to find all instances denoting the same real-world entity. It can be regarded as a binary classification of pairs of instances which are either a match or nonmatch. Providers frequently publish data in tabular form, e.g. tables of power plants with properties (columns) for a plant's name, owner, locality, fuel type, design capacity, commissioning year, etc. In the case of tabular data, instance matching is also called record linkage. For many years, both fields are under active research, and a broad range of approaches was published, in particular from Machine Learning (ML) and Deep Learning (DL). For an overview of recent approaches and frameworks we refer to [11–16].

DL methods for instance matching often concatenate property labels and values for each instance to generate high-dimensional embeddings from pre-trained neural models for training. By contrast, many ML methods rely on features which measure the similarity between property values of two instances. The latter requires the alignment of related or equivalent properties of both data sets. This is commonly provided manually or by a preceding schema-matching step.

Supervised methods of instance matching require manual effort to label instance pairs. The models trained can be validated automatically. This allows for the optimisation of hyperparameters which are critical for matching performance. Unsupervised methods do not need labelled pairs but frequently rely critically on the adjustment of specific parameters for the data sets at hand [12,13]. While specifying domain-related parameters, such as property alignments and similarity measures, is commonly not an issue, choosing the proper values for technical, rather method-specific parameters can be difficult. Challenges are thus to be expected when such methods are performed in an automated environment. This was our starting point for designing a new instance matcher.

The **purpose of this paper** is to present our results and experiences with matching real-world data such as data of power plants, with respect to matching quality, runtime behaviour and degree of automation. We introduce AutoCal, a new algorithm which can be applied stand-alone to match tabular data or be wrapped as an agent populating the knowledge graph of the World Avatar.

The **novelty of AutoCal** is marked by its readiness of deployment in a plethora of domains, which derives from its comparatively simple algorithm without requirement of neither labelled data nor parameter tuning nor specialised hardware. Nonetheless, AutoCal achieves results competitive to recently proposed unsupervised matchers from the field of Machine Learning and still runs 2.7 to 60 times faster than the state-of-the-art matcher. All of the above makes AutoCal specifically well-suited to be used in an automated environment.

AutoCal does not require any labelled instance pairs. Like many ML approaches, AutoCal expects the similarity features of properties as input. Since data sets frequently consist of a mixture of expressive properties such as plant name or locality and more or less discriminative properties, AutoCal "calibrates" automatically the similarity feature values such that they become directly comparable and summable, and uses the resulting total scores for predicting matches.

We evaluate AutoCal on two test scenarios for the domain of power plants and on four test scenarios for other domains. The latter is used widely in the literature but gives more weight to textual than numerical property values. AutoCal achieves matching results comparable to results published for semi-supervised and unsupervised ML methods. At the same time, AutoCal exhibits only two technical parameters and shows stable behaviour for the parameters' default values. Hence, it can be used out of the box for new matching scenarios and is particularly suitable for integration into an automated environment such as the World Avatar.

Unsupervised DL methods usually perform better for the latter four test scenarios. For a deeper analysis, we choose the unsupervised state-of-the-art DL matcher CollaborEM [17] and compare it to AutoCal regarding the applicability, matching quality, and runtime behaviour. It turned out that AutoCal performs significantly better on one of the four widely used test scenarios and – after adding a missing "stop" criterion to CollaborEM – slightly better on both power plant scenarios. AutoCal runs at least 2.6 times faster than CollaborEM on a dedicated ML machine in the cloud and between 15 and 60 times faster on a local machine with a powerful CPU but less GPU memory. AutoCal's runtime behaviour and moderate GPU requirements make it a good choice to run on the same decentralised infrastructure as the World Avatar.

The paper is structured as follows: Section 2 outlines AutoCal and its matching pipeline. Section 3 gives an overview of related work. Section 4 summarises token-blocking and similarity functions, and presents AutoCal in detail. Section 5 evaluates AutoCal for the six test scenarios, presents published results of existing instance matchers, and compares the performance of AutoCal and CollaborEM. Section 6 summarises the World Avatar's architecture and AutoCal's prototypical integration and demonstrates how matching is applied in the case of the power plant domain. Conclusions are outlined in Section 7.

2. Outline of AutoCal

Instance matching can be regarded as a binary classification task where two data sets A and B are given and each pair of instances (a, b), $a \in A$, $b \in B$, is either a matching pair (match) or a nonmatching pair (nonmatch). A pair is a match if a and b refer to the same entity in the real world. The number of all pairs is $|A| \cdot |B|$ while the number of matches only grows linearly with the size of A and B. For instance, 905 out of around 2 million pairs are matches in one of the power plant scenarios considered in this work. This indicates that instance matching suffers from extreme imbalance and scalability issues.

Fig. 1 presents our complete pipeline for instance matching which consists of the preparatory steps such as blocking and the computation of similarity vectors and the characteristic steps of the AutoCal instance matcher. After pre-processing the data sets, blocking discards pairs that

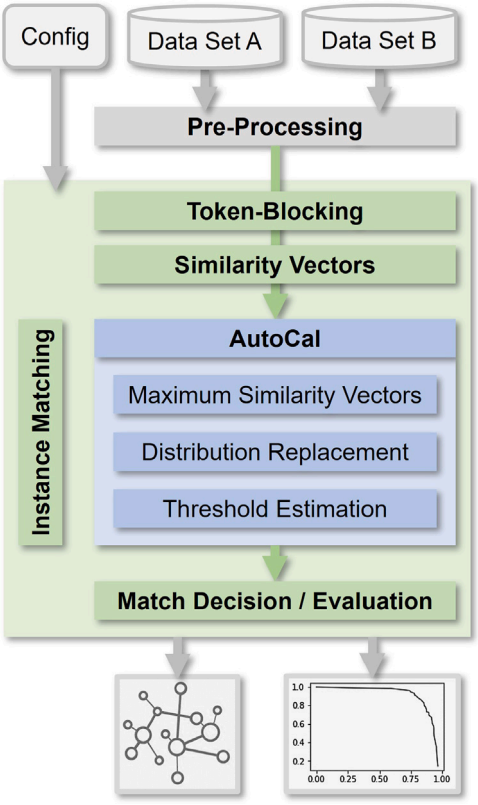

Fig. 1. Pipeline for instance matching: The green boxes denote preparatory steps commonly taken by methods of instance matching. The blue area refers to steps of our proposed instance matcher AutoCal. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

are very likely to be nonmatches and keeps a much smaller set of candidate pairs for classification. In practical applications, the overall performance of instance matching is crucial, and the chosen blocking technique and set of similarity features strongly impact it.

Before detailing token-blocking, similarity functions and AutoCal in Section 4, we want to summarise the core ideas behind AutoCal. The corresponding Python code was published on GitHub [18].

AutoCal is based on a heuristic approach motivated by our own experience when matching data manually: Firstly, a human editor defines a suitable criterion to pre-select a feasible set of potential matching candidates $b \in B$ for a given $a \in A$. This refers to the blocking step already mentioned. By comparing the property values between a and all its potential matching candidates, the editor is able to identify the matching candidate b which is "most similar" to a. This refers to the step *Maximum Similarity Vectors* in Fig. 1 in which AutoCal selects the largest similarity values for each property.

Finally, the human editor has to decide whether the pair (a, b) is actually a match or nonmatch. This step involves the editor's background knowledge. Instead, AutoCal considers the set of all largest similarity values and applies a statistical approach to measure how discriminative properties and similarity values are for matches and nonmatches. This is realised in the step *Distribution Replacement* which aims towards the conditional probability that the pair (a, b) is a match given $s_k(a, b) \in I$. Here, $I \subset [0, 1]$ is a small interval and $s_k(a, b)$ denotes the similarity value between instances a and b regarding the kth property. The conditional probability can be approximated by the ratio between all true matches having their similarity value in I and all pairs (both matches and nonmatches) having their similarity value in I as well. This ratio measures much better than the similarity value itself how relevant a similarity value is for matching. We illustrate this observation for instances representing power plants: If two instances have the same value for property *fuel*, the corresponding similarity value is 1, but it is unlikely that both instances match because there are many more nonmatching than matching pairs with the same fuel type. In this case, the ratio would be quite small. On the other hand, if two instances have only a similarity value of 0.6 for property *name* but corresponding values for nonmatches usually fall below 0.5, the ratio may be pushed towards 1.

Of course, the counts for true matches are not known. For this reason, AutoCal computes approximated ratios by replacing the unknown counts with the known counts for the "most similar" matching candidates from step 1. In summary, AutoCal converts the original similarity values property-wise into matching-relevant ratios. We call these ratios *calibrated scores* because – in contrast to the original scores – the ratios for different properties are directly comparable to each other in the context of instance matching. The last step, *Threshold Estimation* in Fig. 1, uses the averaged calibrated scores as total score and predicts pairs with a total score above a certain threshold as matches. Since the calibration works in an unsupervised way without any labelled pairs, we call our algorithm "auto calibration" or "AutoCal" for short.

We regard AutoCal as a simple algorithm which is based on statistics and heuristics: AutoCal employs only basic statistics throughout all steps, such as histograms with equally-sized bins to represent distributions of similarity values on the range $[0, 1]$. In addition, the replacement of the unknown distribution of true matches by the empirical distribution of "most similar" pairs is motivated heuristically. However, AutoCal does also make use of a pre-trained language model to derive string similarity features.

3. Related work

In this section, we outline some existing methods for instance matching which are unsupervised or require only a small amount of labelled pairs, and which are thus candidates for an automated environment. The selected methods cover a variety of approaches and achieve results close to or among state-of-the-art in ML and DL. Many of them use similarity functions which score the similarity between two property values, where the score ranges from 0 (no similarity) to 1 (equality). We refer to [13] for details about semi- and unsupervised methods in instance matching. For the sake of simplicity, we use the terms *instance*, *entity* and *record* synonymously, and likewise the terms *property*, *column* and *attribute*.

Auto-FuzzyJoin [19] considers instance matching as an optimisation problem that maximises recall for a user-defined minimum precision value. The algorithm is based on a geometric argument and searches iteratively for combinations of distance functions and upper distance bounds to define proper matching rules.

Wombat [20] is a state-of-the-art approach for link discovery. Link discovery can predict relations of any type between instances in knowledge graphs and is not restricted to instance matching or tabular data. Wombat relies exclusively on positive examples, i.e., on an initial subset of labelled matches in case of instance matching. It learns rules which specify properties, similarity functions and similarity thresholds for linking (matching) two instances. Wombat starts with simple rules and applies set-valued operators to combine and refine them such that the resulting more complex rules provide a higher performance on the initial subset of labelled matches.

Kejriwal and Miranker [21] propose an iterative semi-supervised method requiring a small seed set of labelled training pairs initially. Their method trains an ensemble of base classifiers (either Random Forest or Multilayer Perceptron) using boosting. The most confident predictions for matches and nonmatches are added to the training set for training a new boosted classifier. The training loop terminates after a configured number of iterations.

Jurek et al. [22] also train an ensemble of base classifiers (Support Vector Machine) iteratively but their approach is unsupervised and

includes several initial steps to select a set of seed pairs for each base classifier such that the sets exhibit a high diversity. These steps include the selection of similarity functions and the computation of property weights that express the distinguishing power of the property values.

ZeroER [23] models the unknown match and nonmatch distributions as a Gaussian mixture. It is based on the expectation-maximisation algorithm to estimate the unknown parameters for the Gaussian distributions and the mixture ratio between matches and nonmatches. The method exploits the fact that matches tend to have higher similarity values than nonmatches and provides an adaptive regularisation strategy to prevent overfitting.

EmbDI [24] converts records from both data sets into a common graph and learns neural embeddings for the graph nodes. The graph defines three types of nodes to represent record IDs (entities), column IDs (properties), and tokens. Nodes representing two records are connected indirectly via a common token node if the records share a token. Next, random walks are performed on the graph nodes, and the visited nodes of each random walk are concatenated to a sentence containing record IDs, column IDs and tokens. The total of all sentences generated allows it to learn embeddings in an unsupervised fashion. Finally, the cosine distance between embeddings representing a pair of records serves as a matching criterion.

ErGAN [25] is a semi-supervised generative adversarial network which is specifically designed for instance matching. Initially, it computes the median of all similarity values for each property and divides the set of unlabelled pairs into subsets according to whether their similarity values are smaller or greater than the median values. Then, for each training iteration, its diversity module gives unlabelled pairs a higher priority if they belong to a smaller subset. In combination with the propagation module, the diversity module tackles the problem of imbalance between matches and nonmatches and avoids overfitting.

CollaborEM [17] is a self-supervised framework running two phases: The first phase computes embedding vectors for each record by means of Sentence-BERT [26]. By using the cosine distance and additional constraints, record pairs are identified that are very likely to be true matches or true nonmatches. The second phase uses this auto-generated seed set of labelled pairs for supervised training: each record is converted into a multi-relational graph, and a graph neural network is trained to learn four types of graph features (embeddings) for labelled pairs. Afterwards, both the trained graph features and the sentence representations of record pairs are used in a collaborative way to fine-tune a pre-trained transformer model and to make a final matching decision.

Regarding its paradigm, AutoCal is most comparable to Auto-FuzzyJoin. By contrast, many previously outlined methods train standard ML classifiers or use neural embeddings. Of course, all ML methods take advantage of the fact that the similarity of property values tends to be greater for matches than for nonmatches. But AutoCal strictly exploits this fact in its first step when identifying "most similar" instance pairs. On closer inspection, some of them also reveal implicit assumptions or steps similar to those of AutoCal. For example, EmbDi's graph conversion relies on common tokens, and ErGAN's propagation module uses some statistics.

While the previously outlined methods are unsupervised and partly achieve state-of-the-art results, some of them have requirements which may reduce their applicability in an automated environment. For instance, Auto-FuzzyJoin assumes a reference data set not containing any duplicates, while Wombat requires a specific coverage threshold for properties. Jurek-Loughrey and Deepak [13] point out that it can be problematic to adapt method-specific parameters for new data sets without labelled pairs. CollaborEM lacks a criterion to stop the fine-tuning. In contrast to all of those methods, AutoCal can be used out of the box.

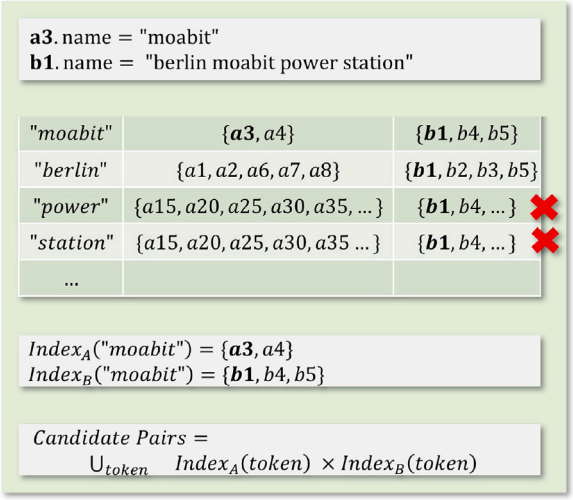

Fig. 2. Token-blocking identifies those entity pairs that share at least one token. This technique reduces the number of candidate pairs that have to be classified as match or nonmatch. It thus avoids issues of scalability and extreme imbalance.

4. Instance matching

4.1. Token-blocking

Many data sets contain columns with highly expressive string values, like instance names, descriptions, or addresses. Because of their expressiveness, such values are valuable for instance matching. Each of the string values can be split into a set of tokens. While the respective strings related to matching entities may differ in detail, they usually contain some equal tokens. We thus choose token-blocking, which considers two instances as a candidate pair for matching if they share at least one token. While AutoCal may be combined with other blocking techniques, token-blocking suits AutoCal in a natural way since the subsets for which AutoCal's first step identifies the "most similar" instance pairs can be easily created and managed in parallel. Token-blocking was introduced in [27]; we refer the reader to [28] for an overview of further unsupervised blocking approaches.

Fig. 2 demonstrates token-blocking by means of instances from two data sets A and B. Here, A and B are two data sets of power plants in Germany which we will also use for evaluation later on in this paper. Among the instances of the data sets, we find an instance $a3 \in A$ with name "moabit" and an instance $b1 \in B$ with name "berlin moabit power station". Both instances share the token "moabit" in their name, which denotes a district of the city of Berlin. We thus add them to the set of all instances from A and B, resp., which also contain the token. Since the name of instance $b1$ contains three more tokens "berlin", "power", and "station", the instance is also added to the corresponding sets of these three tokens.

We create a mapping $Index_A$ from the set of all tokens to their related instances in data set A by iterating over all instances in A, tokenising their names and further string values of other properties, and finally adding the instances to the corresponding sets. For $Index_B$, we proceed likewise. The block for a given token t is defined as the Cartesian product $Index_A(t) \times Index_B(t)$. The set of candidate pairs is the union of all blocks for all found tokens. Blocks for two different tokens overlap if some instances $a \in A$ and $b \in B$ share both tokens. However, the candidate pair (a, b) only appears once in the union set. A token found only in instances from one of the data sets would lead to an empty block and is ignored.

If the index considered only tokens derived from equivalent properties in A and B, then there may be instances of A that do not share any tokens with instances of B. For example, 23% of the instances in one of

Table 1
Similarity functions.

Data type	Name	Description
any	equality	$1_{\{x=y\}}$
number	absolute error	$1 - g(\lvert x - y \rvert)$
number	relative error	$1 - \frac{\lvert x-y \rvert}{max\{\lvert x \rvert, \lvert y \rvert\}}$
string	cosine similarity - weighted	tf-idf weights, [29]
string	cosine similarity - embeddings	[26]

the data sets of power plants in Germany miss values for the property *name*. On the other hand, most of these instances provide a non-empty string for the property *address locality*. For this reason, token-blocking takes tokens from multiple properties in order to create the index. It subsequently considers instances as potential matching partners that would be filtered out otherwise.

Frequent tokens are less distinctive and increase the number of candidate pairs disproportionately. Our implementation of token-blocking provides the "maximum token frequency" (MTF) as a configuration parameter and ignores those tokens for which the number of corresponding instances either in A or in B exceeds the configured MTF value. In Fig. 2, this is indicated by two red crosses for tokens "power" and "station" for the case MFT = 5. MFT strongly impacts the number of candidate pairs and plays an important role in the overall performance of instance matching.

4.2. Similarity vectors

Two instances a and b can be regarded as similar if their property values are similar. The similarity of two property values $v_1, v_2 \in V$ is measured by a similarity function $f : V \times V \to [0, 1]$. The similarity value $f(v_1, v_2)$ is 1 if v_1 and v_2 are equal, close to 1 if they are very similar, and 0 if they are completely different. A wide range of similarity functions was devised, each being applicable for particular domains, e.g. [29] evaluates various similarity functions for matching names.

Table 1 lists the similarity functions considered for matching in this paper. Note that a distance function such as the absolute error can easily be converted into a similarity function by composing with a suitable increasing function $g : [0, \infty) \to [0, 1]$ and subtracting from 1. The last two rows refer to cosine similarity which is defined for two (non-zero) vectors v and w by the normalised dot product $v \cdot w / \|v\| \|w\|$. Here, v and w are vector representations of two given strings. "Weighted" means that each vector component corresponds to a separate token and is set to its tf-idf weight, see e.g. [29], while the last row refers to sentence embeddings [26].

AutoCal requires a mapping between related properties of data sets A and B as well as respective suitable similarity functions. This mapping may be derived manually or automatically by schema matching, for example. The selection of similarity function for a pair of mapped properties may also be automated, e.g., by preliminary analysis of type and range of property values in data set A and B. The similarity functions from Table 1 cover all data types relevant to this work and may be modified or extended, for instance for a new data type such as *date*. For the evaluation of AutoCal in Section 5, we applied both cosine similarity functions to each mapped pair of string-valued properties since they complement each other: The cosine similarity between weighted vectors works well for short strings with a few expressive tokens such as names or locations while the cosine similarity between sentence embeddings expresses the semantic similarity for longer strings such as product descriptions.

We describe the mapping by K triples of the form (p_k, q_k, f_k), $k = 1, \ldots, K$. Here, p_k and q_k denote properties of instances $a \in A$ and $b \in B$ resp., and f_k a suitable similarity function that can be applied to the pair $a.p_k$ and $b.q_k$ of property values. The definition of a mapping allows the application of several distinct similarity functions for the same related properties. Each f_k may also map to the special case *null* to indicate missing property values. Finally, a given mapping of K triples defines a mapping from the set of candidate pairs into the space of similarity vectors by

$$s(a, b) = (f_1(a.p_1, b.q_1), \ldots, f_K(a.p_K, b.q_K)) \in ([0, 1] \cup \{null\})^K.$$

Each similarity vector consists of K similarity values. In the following, we will use the term kth component, kth triple or kth property interchangeably to refer to its individual similarity values.

4.3. AutoCal

AutoCal consists of three steps as shown in Fig. 1. For sake of clarity, we present the details of the steps as pseudo code in Algorithm 1. In addition to the original datasets A and B, AutoCal requires the set C of candidate pairs and corresponding similarity vectors $s(\cdot, \cdot)$ from the two previous subsections as input; moreover, a configurable parameter named *bin width* Δ and the mapped properties $(p_k, q_k)_{k=1}^K$ while the similarity functions are not needed any more.

Lines 1 to 5 of Algorithm 1 reflect the main structure of AutoCal and are completed by line 6 which returns a subset of C as predicted matches. For a given candidate pair (a, b), AutoCal's three steps often perform calculations and comparisons on the subsets

$$C(a) = \{(a, \tilde{b}) \in C\} \quad \text{and} \quad C(b) = \{(\tilde{a}, b) \in C\}$$

and hence consider two directions: The first keeps a fixed and varies associated $\tilde{b} \in B$, while the second keeps b fixed and varies associated $\tilde{a} \in A$. Due to token-blocking, associated instances $\tilde{b}$ are those that share at least one token with a. However, two instances $\tilde{b}_1$ and $\tilde{b}_2$ associated both with a might not have any tokens in common by themselves. The same holds for the other direction.

The **first step** of AutoCal, the computation of **maximum similarity vectors**, can be expressed compactly by the first two lines of Algorithm 1. For a given $a \in A$, the maximum similarity vector $s^{max,A}(a)$ is larger than or equal to all associated similarity vectors in any of the K components. Likewise, for a given $b \in B$. The left diagram in Fig. 3 illustrates this concept for a power plant instance $a3$ with only $K = 2$ properties *fuel* and *name*. Of the three pairs shown, the maximum similarity vector is the one coloured turquoise. It may happen that none of the similarity vectors can be identified as best. This situation is illustrated in the right diagram of Fig. 3. In this case, $s^{max,A}(a)$ is not associated to any pair $(a, \tilde{b}) \in C$ and we consider it as "virtual" vector. Moreover, if the value for the kth property is missing for a or $\tilde{b}$, then $s_k(a, \tilde{b})$ is also missing, and is ignored when computing the maximum in lines 1 and 2. If the kth value is missing for all $(a, \tilde{b})$, $s_k^{max,A}(a)$ is set to *null*. This propagation of missing values through operations and comparisons is straightforward and will not be mentioned further in the following.

The heuristic behind the definition of maximum similarity vectors is that if we choose any $a \in A$ and look at the subset $C(a)$ of candidate pairs, we would suppose that a pair with similarity vector in the vicinity of $s^{max,A}(a)$ has a higher chance to be a match than pairs with similarity vectors further away — as long as the similarity functions employed are reasonable and do not contradict the concept of similarity. Likewise for $b \in B$ and $C(b)$. Nevertheless, such a pair does not have to be a match.

The **second step** of AutoCal uses the maximum similarity vectors and a statistical approach to decide for each of the K components which similarity values are both typical and relevant for matches. It is based (a) on the **replacement of the unknown match distribution** by the known empirical distribution for the maximum similarity vectors, and (b) on a rough estimation for the mixture of the marginal match and nonmatch distributions. Fig. 4 illustrates this concept for the kth triple referring to property *name*. The upper left diagram depicts the counts of matches and nonmatches for the kth component. In the lower left diagram, the match counts are replaced by counts derived from the kth component of all maximum similarity vectors. Usually, the counts

Algorithm 1 AUTOCAL

input:
data sets A, B
candidate pairs $C \subset A \times B$
similarity vectors s for C
dimension K of similarity vectors
property mapping $P = (p_k, q_k)_{k=1}^{K}$
bin width Δ

1: $s^{max,A}(a) \leftarrow (\max\{s_k(a,\tilde{b}) : (a,\tilde{b}) \in C\})_{k=1}^{K}, \quad a \in A$
2: $s^{max,B}(b) \leftarrow (\max\{s_k(\tilde{a},b) : (\tilde{a},b) \in C\})_{k=1}^{K}, \quad b \in B$
3: $score^A \leftarrow$ REPL-DISTRIBUTION$(A, C, s, K, P, \Delta, s^{max,A})$
4: $score^B \leftarrow$ REPL-DISTRIBUTION$(B, C, s, K, P, \Delta, s^{max,B})$
5: $matches \leftarrow$ ESTIM-THRESHOLD$(C, score^A, score^B, \Delta)$
6: **return** $matches$

7: **function** REPL-DISTRIBUTION$(E, C, s, K, P, \Delta, s^{max})$
8: $r \leftarrow []$
9: **for** $k \in \{1, \dots, K\}$ **do**
10: **for** $j \in \{0, \dots, \lfloor 1/\Delta \rfloor\}$ **do**
11: $n^{max} \leftarrow \#\{e \in E : j\Delta \le s_k^{max}(e) < (j+1)\Delta\}$
12: $n \leftarrow \#\{(a,b) \in C : j\Delta \le s_k(a,b) < (j+1)\Delta\}$
13: $r(k,j) \leftarrow \max\{1, n^{max}/(n+1)\}$
14: $w \leftarrow []$
15: **for** $k \in \{1, \dots, K\}$ **do**
16: $w(k) \leftarrow (\#\{(p_l, q_l)\})^{-1} \cdot (\#\{l : (p_l, q_l) = (p_k, q_k)\})^{-1}$
17: $score \leftarrow []$
18: **for** $(a,b) \in C$ **do**
19: $score(a,b) \leftarrow 0$
20: **for** $k \in \{1, \dots, K\}$ **do**
21: $pscore \leftarrow r(k, \lfloor s_k(a,b)/\Delta \rfloor)$
22: $score(a,b) \leftarrow score(a,b) + w(k) \cdot pscore$
23: **return** $score$

24: **function** ESTIM-THRESHOLD$(C, score^A, score^B, \Delta)$
25: $C_M^A \leftarrow \{(a,b) \in C : score^A(a,b) = \max_{(a,\tilde{b}) \in C} score^A(a,\tilde{b})\}$
26: $C_M^B \leftarrow \{(a,b) \in C : score^B(a,b) = \max_{(\tilde{a},b) \in C} score^B(\tilde{a},b)\}$
27: $C_M \leftarrow C_M^A \cup C_M^B$
28: $score \leftarrow []$
29: **for** $(a,b) \in C$ **do**
30: $score(a,b) \leftarrow \max\{score^A(a,b), score^B(a,b)\}$
31: $r \leftarrow []$
32: **for** $j \in \{0, \dots, \lfloor 1/\Delta \rfloor\}$ **do**
33: $n_M \leftarrow \#\{(a,b) \in C_M : j\Delta \le score(a,b) < (j+1)\Delta\}$
34: $n \leftarrow \#\{(a,b) \in C : j\Delta \le score(a,b) < (j+1)\Delta\}$
35: $r(j) \leftarrow n_M/(n+1)$
36: $j_{min} \leftarrow \min\{j : r(j) \ge \frac{1}{2}, r(j+1) \ge \frac{1}{2}, r(j+2) \ge \frac{1}{2}\}$
37: $t_{est} \leftarrow j_{min}\Delta$
38: **return** $\{(a,b) \in C_M : score(a,b) \ge t_{est}\}$

from the maximum similarity vectors differ from the match counts, in particular in the vicinity of 0 and 1. They provide a very rough approximation of the actual match distribution; however, if the bias is not too strong, the counts help to rate the relevance of similarity values for instance matching in form of scores.

The function declared in lines 7 to 23 of Algorithm 1 formalises the second step. The function is called twice — once for dataset A in line 3 and once for dataset B in line 4 in combination with the corresponding maximum similarity vectors and further input parameters.

The first part of the function (lines 8 to 13) discretises the range $[0, 1]$ of similarity values by subdivision into bins $[0, \Delta), [\Delta, 2\Delta), \dots$. For each component k, lines 11 and 12 count how many of the maximum

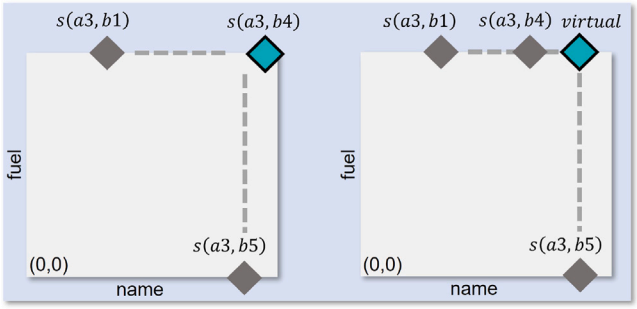

Fig. 3. Candidate pair $(a3, b4)$ is more likely to be a match compared to $(a3, b1)$ and $(a3, b5)$ since its similarity vector (turquoise diamond in the left diagram) is larger in all its components. If no such "best" pair exists, a "virtual" maximum similarity vector (turquoise diamond in the right diagram) is computed.

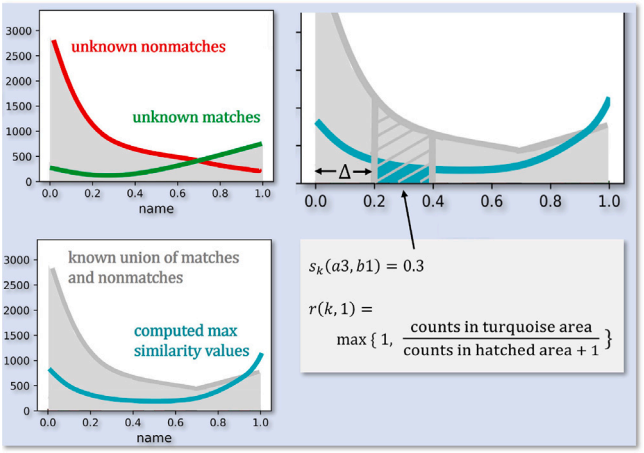

Fig. 4. The unknown marginal distributions of matches (upper left diagram) are replaced by the empirical marginal distributions of all calculated maximum similarity vectors (lower left diagram). For each property, an auto-calibrated score is calculated as the ratio between the replacement distribution and the overall distribution of matches and nonmatches (right diagrams).

similarity values and of the original similarity values, resp., are placed in the jth bin. Line 13 computes the ratio $r(k,j)$ between both counts — restricted to maximum 1. We use the +1 term in the denominator to avoid division by zero.

The second part of the function (lines 14 to 22) calculates the total score for each candidate pair (a,b) as weighted average of the computed ratios. The essential line is line 21 which maps the kth similarity value to the bin due to index $j = \lfloor s_k(a,b)/\Delta \rfloor$ and replaces it by the "property score" $r(k,j)$. The right side of Fig. 4 demonstrates how AutoCal "calibrates" the similarity value $s_k(a3, b1) = 0.3$ and transforms it into the property score in case of the specific candidate pair $(a3, b1)$. The property score measures how relevant the similarity value is for matching.

The total score is computed in line 22 by summing the property scores multiplied with weights $w(k)$. The weights are pre-computed in lines 16 which accounts for the possibility that the same property pair may occur several times. If this is not the case, it gives $w(k) = K^{-1}$, and the total score is just the mean of the property scores. If, for example, we configure the equality function for property *fuel* and a string similarity function for property *name*, then $K = 2$ and $w(k) = \frac{1}{2}$. But if we duplicate the second pair, then $K = 3$ and using the same weight $w(k) = \frac{1}{3}$ would undermine the idea of calibration. In contrast, line 16 correctly sets $w(1) = \frac{1}{2}$, $w(2) = w(3) = \frac{1}{4}$ which gives the same total scores as in the case $K = 2$.

The **third step** of AutoCal **estimates a threshold** to separate matches from nonmatches. It is formally described by the second

Table 2
Scenarios for evaluating instance matchers.

ID	Data Set A	$\#A$	Data Set B	$\#B$	#Prop	#Eval	#Matches
KG	KWL	1983	GPPD DE	982	5 (+1)	10,392	905
DG	DUKES	1184	GPPD GB	2536	4	13,509	950
FZ	Fodor	533	Zagat	331	5	946	110
DA	DBLP	2616	ACM	2294	4	12,363	2220
DS	DBLP	2616	Scholar	64,263	4	28,707	5347
AG	Amazon	1363	Google	3226	3	11,460	1167

function (lines 24 to 38) of Algorithm 1 which is called with the results $score^A$ and $score^B$ of the previous step (line 5). The function applies operations to these total scores similar to the property-wise operations in the previous two steps. Lines 25 to 27 shortlist all pairs with maximum total scores on the subsets $C(a)$ and $C(b)$ as potential matches C_M. This relates to the idea behind AutoCal's first step.

While line 30 aggregates $score^A$ and $score^B$ by taking their maximum, lines 32 to 35 resemble lines 10 to 13 and transfer the idea of discretisation to aggregated total scores. The ratio $r(j)$ is defined as the portion of potential matches to all pairs in the jth bin. Line 36 chooses the index j_{min} as the first bin for which the potential matches gain the majority. Actually, we require that this condition is satisfied three times in sequence to prevent possible unstable behaviour in case of low bin counts. The estimated threshold t_{est} is simply the lower bound for the chosen bin (line 37). The condition is a compromise between values far below t_{est} which may lead to a low precision, and values far above t_{est} which may lead to a low recall. Finally, AutoCal predicts those potential matches in C_M having an aggregated score above the threshold t_{est} as matches (line 38).

5. Evaluation

5.1. Data sets

Table 2 presents the scenarios for the evaluation of instance matching (IM) in this work. Each IM scenario consists of two data sets A and B and an evaluation set with labelled ground truth matches and nonmatches. The table contains the number of instances in A and B, the number of properties used for matching, the size of the evaluation set, the number of matches within the evaluation set, and an ID that we will use for reference in this work.

The first two rows present the scenarios of the power plant domain. Scenario KG refers to matching data of power plants in Germany published by the *Federal Network Agency*[1] with data from the Global Power Plant Database (GPPD),[2] while scenario DG refers to matching data of power plants in the United Kingdom published by the *Department for Business, Energy and Industrial Strategy*[3] with data from GPPD. All data sets involved provide the properties *name*, *primary fuel*, *capacity* (in MW) and *owner*, which are considered for matching. In scenario KG, we also take into account the property *commissioning year*.

For both KG and DG, we created the ground truth manually step by step: Candidate pairs were generated iteratively by token-blocking for increasing MTF and by means of the *Python Record Linkage Toolkit* [30], and were classified as match or nonmatch manually in each iteration. In addition, we considered the property *address locality* for token-blocking in scenario KG. This property is only available for data set A and helps to compensate for missing power plant names in this data set. Finally, for both KG and DG, the candidate set created by token-blocking for MTF = 50, combined with the manually identified labels, serves as an evaluation set. Candidate pairs not identified previously are considered nonmatches.

Scenarios in all but the first two rows of Table 2 were widely used for evaluation in other papers and refer to the domains *restaurant*, *bibliography*, and *product*, respectively. The labelled data for DA, DS and AG originate from [31]. For a fair comparison, we downloaded the data from the Magellan Data Repository[4] which is frequently used for recent research in matching. Since AutoCal is unsupervised and CollaborEM is self-supervised, we use the union of the downloaded train, validation and test sets as an evaluation set.

For future use, the new data for KG and DG is publicly available in the same format as provided by the Magellan Data Repository [32]. The Ontology Alignment Evaluation Initiative[5] (OAEI) publishes benchmarks for various matching tasks. In particular, the task of matching tabular data to a knowledge graph is related to our use case. Due to the significant preprocessing effort required for adaptation, we are deferring this task to future work.

5.2. Quality metrics and setup

Given an evaluation set, the quality metrics precision P, recall R, and F_1-score are defined by

$$P = \frac{TM}{TM + FM}, \quad R = \frac{TM}{TM + FN}, \quad F_1 = \frac{2PR}{P + R},$$

where TM, FM and FN denote the number of true predicted matches (true positives), false predicted matches (false positives) and false predicted nonmatches (false negatives) on the evaluation set. AutoCal estimates a threshold to make matching decisions for all pairs within the candidate set. Since the candidate set is no superset of the evaluation set in general, we set all pairs outside the candidate set as predicted nonmatches, i.e. true matches outside the candidate set are counted correctly as false predicted nonmatches.

AutoCal requires some parameter values to be provided: The bin width Δ is set to 0.025 for all six IM scenarios. First, we will vary the MTF value to analyse its effect on the matching quality. Then we fix it to the suitable default value of 50. We defined a fixed property (column) mapping for each scenario, comprising similarity functions from Table 1: For numerical properties, we either applied the absolute or the relative error, for properties with enumerated values (such as fuel type) we used the equality function, and for string-valued properties, we used the cosine similarity both for TF-IDF weights and sentence embeddings. In effect, we employed two similarity values per string-valued property and otherwise one.

Concerning CollaborEM, we applied the same setup including default parameter values as is described in [17]. For DS we use the same graph features as for DA, while for KG and DG we use all available four graph features.

Both AutoCal and CollaborEM ran on machines of a major cloud provider — equipped with an Intel Xeon CPU, 89 GB RAM and NVIDIA A100 GPU with 40 GB memory. In addition, we run both methods on a laptop with an Intel Core i7-11800H CPU, 32 GB RAM and NVIDIA GeForce RTX 3070 GPU with 8 GB memory. In the first case, we trained CollaborEM with batch size 64 as in [17]; in the second case, we had to reduce the batch size to 16 to avoid GPU memory problems.

5.3. Results regarding matching quality

Fig. 5 summarises the results of our ablation study for AutoCal and answers the following questions: How does the choice of MTF impact AutoCal's matching quality, and how much do AutoCal's steps contribute to it respectively? Each diagram refers to one of the IM

[1] https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/Versorgungssicherheit/Erzeugungskapazitaeten/Kraftwerksliste/start.html, version 01/04/2020

[2] https://datasets.wri.org/dataset/globalpowerplantdatabase, version 1.2.0

[3] DUKES 5.11 2019, https://www.gov.uk/government/statistics/electricity-chapter-5-digest-of-united-kingdom-energy-statistics-dukes

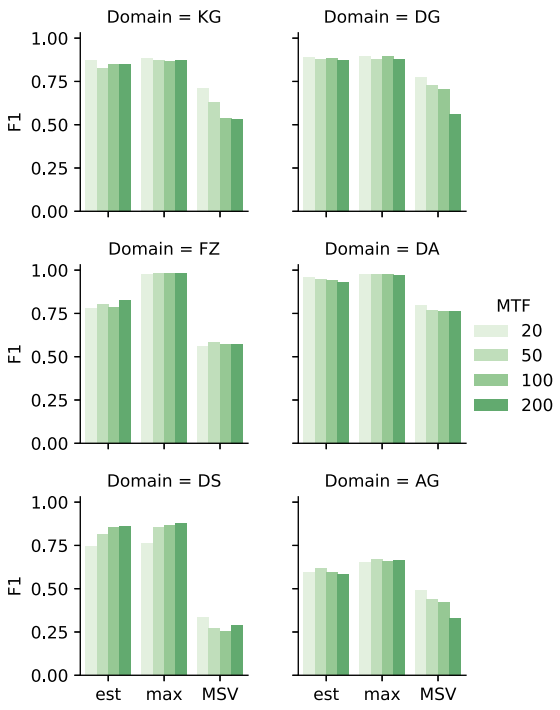

Fig. 5. Dependency of F_1-scores on maximum token frequency (MTF) and on AutoCal's steps for both threshold estimation ("est") and maximum similarity vectors ("MSV").

scenarios and contains three groups that we will discuss one after another.

The first group "est" shows F_1-scores on the evaluation sets for the estimated threshold t_{est} with MTF values 20, 50, 100 and 200. The second group "max" presents maximum F_1-scores. They are calculated by increasing the threshold t from 0 to 1 in small steps, computing the F_1-score for each threshold value and taking the maximum $\max_t F_1(t)$.

The maximum F_1-scores are slightly larger than their counterparts in the first group — except for small-sized data sets of FZ. The large difference for the "easiest" scenario FZ is caused by a relatively small number of matches compared to the bin width Δ which we fixed to 0.025 for all six IM scenarios. This leads to sparsely populated bins and hence, to an inaccurate estimation of the threshold. In contrast, the threshold estimation step achieves sufficiently accurate results for mid-sized data sets.

The F_1-scores in both groups are surprisingly stable for a wide range of MTF-values — except for DS in combination with MTF = 20. This indicates that MTF = 50 is a reasonable choice when performing instance matching with AutoCal on other mid-sized data sets. Therefore, we use MTF = 50 as the default value in the following. Table 3 shows the resulting number of tokens and candidate pairs and AutoCal's matching quality in terms of precision, recall, and F_1-score for the estimated threshold t_{est}.

The results in the last group "MSV" of Fig. 5 are obtained only when relying on AutoCal's first step. In this case, a candidate pair is predicted as a match if and only if its similarity vector is a maximum similarity vector. The resulting F_1-scores do not depend on any threshold. They indicate that maximum similarity vectors – taken in isolation – do not provide acceptable F_1-scores. Nevertheless, computing maximum similarity vectors is necessary to obtain an empirical distribution that produces useful auto-calibrated scores for all candidate pairs.

[4] https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md
[5] http://oaei.ontologymatching.org

Table 3
The number of tokens and candidate pairs due to token-blocking for MTF = 50 and resulting precision, recall and F_1-scores in % for AutoCal.

ID	# Tokens	# Candidates	P	R	F_1
KG	962	10,392	95.1	73.5	82.9
DG	1043	13,509	90.6	84.9	87.7
FZ	727	8979	67.1	100.0	80.3
DA	6647	202,565	97.8	92.2	94.9
DS	4238	124,545	92.5	72.7	81.4
AG	1781	54,831	61.7	61.5	61.6

Table 4
F_1-scores in % for unsupervised, semi-supervised and supervised approaches. The best F_1-score for each scenario in each category is underlined.

	Method	Setup	FZ	DA	DS	AG
zero and low resources	AutoCal	t_{est}	80.3	94.9	81.4	61.6
	Auto-FuzzyJoin [19]		88.9	97.7	–	–
	SBC [21]	2% seed matches	94.7	94.7	–	43.0
	Wombat [20]	2% seed matches	98	94	–	53
	Ensemble Learning [22]		94	92	55	–
	ZeroER [23]		100	96	86	48
	EmbDI [24]		99	95	92	59
	ErGAN [25]	1% seed matches	–	96.9	85.9	–
	CollaborEM [17]		100	98.6	–	68.6
mid to high	Wombat [20]	30% matches	97	95	91	54
	Magellan [33] [34]	strat. 60% + 20%	100	98.4	92.3	49.1
	DeepMatcher [34]	strat. 60% + 20%	100	98.4	94.7	69.3
	DITTO [35]	strat. 60% + 20%	100	99.0	95.6	75.6
	DeepER [36]	10% + 5-fold CV	100	98.6	97.7	96.0

5.4. Comparison

In this section, we compare AutoCal's matching quality with some unsupervised, semi-supervised and supervised methods of instance matching. Table 4 summarises published F_1-scores for FZ, DA, DS and AG. When a result was achieved by using at most 2% of labelled matches from the complete ground truth, we assigned it to the upper category, otherwise to the lower category. The highest score for each IM scenario in each category is underlined. The "Setup" column provides additional information that might be helpful for a fair comparison: "strat. 60% + 20%" means that stratified sampling was applied to create training and validation sets with 60% and 20%, resp., of the labelled candidate pairs while the remaining 20% were used for testing, "CV" stands for "cross validation", and if no percentage is specified, it means no manual labelling is required. There are further relevant differences that are not shown in the table such as the use of various blocking methods and sets of similarity functions.

AutoCal belongs to the upper category; the first row shows its F_1-scores copied over from Table 3 for convenient comparison. All other methods in the upper category were already introduced in Section 3.

We also added supervised ML and DL methods as the second category to Table 4 since they define an upper limit for the matching quality and give an idea of what we can presently expect at most in an automated environment. The second category starts with Wombat [20] as a state-of-the-art approach for link discovery; this time, the results shown were achieved using 30% of the labelled matches. Magellan [33] is an ML framework for instance matching which integrates various methods for blocking, feature generation and matching (such as Random Forests and Support Vector Machine). The last three rows refer to state-of-the-art DL models DeepMatcher [34], DITTO [35] and DeepER [36]. Supervised methods usually achieve higher F_1-scores than those from the upper category but require a large number of labelled matching pairs. It is out of scope here to describe these models in detail; instead, we refer to [16] for a survey of DL instance matchers.

The upper category of Table 4 shows that unsupervised methods employing embeddings and deep learning, i.e. EmbDI [24], ErGAN [25] and CollaborEM [17], frequently achieve higher F_1-scores. In particular, self-supervised CollaborEM obtains the best scores for FZ, DA and

Table 5
Maximum F_1-scores and F_1-scores in practice (in %) for AutoCal and CollaborEM.

Method	Setup	FZ	DA	DS	AG	KG	DG
AutoCal	max. $t \in [0,1]$	98.2	97.4	85.4	66.8	87.2	87.7
CollaborEM	max. 30 epochs	100.0	98.3	66.9	67.1	86.5	88.3
AutoCal	t_{est}	80.3	94.9	81.4	61.6	82.9	87.7
CollaborEM	val. 30 epochs	94.1	97.1	63.5	66.2	81.4	86.6
CollaborEM	val. 10 epochs	94.1	96.6	62.5	64.1	80.1	85.9

AG. DL approaches benefit fully from high-dimensional embedding vectors instead of only using the cosine similarity between them. AutoCal achieves F_1-scores which are competitive to results for ML methods, but at first sight, DL approaches seem to achieve superior matching results.

On the other hand, DL approaches often interpret numerical data such as product prices as strings and missing data as empty strings. Since the power plant scenarios KG and DG contain numerical data and short strings, it is not clear whether unsupervised DL approaches can use this information effectively and achieve better performance for KG and DG – out of the box, to be specific – than AutoCal. Moreover, we are also interested in further aspects that are relevant when applying instance matching in real-world projects. For this reason, we will have a deeper look at these aspects in the next subsection and choose the state-of-the-art DL matcher CollaborEM for intensive study.

5.5. Results regarding applicability and runtime behaviour

This section compares AutoCal and CollaborEM in more detail and focuses on applicability in real-world projects as well as runtime behaviour. For this purpose, the number of epochs is crucial for CollaborEM's training process, i.e., for fine-tuning a pre-trained language model. According to [17], the number of epochs depends on the size of the data sets, but it is not specified how to choose the number in case of new datasets such as KG and DG. We thus run CollaborEM in two ways.

Firstly, we run CollaborEM as described in [17] but always for 30 epochs. After each epoch, we apply the fined-tuned model to the evaluation set and calculate the F_1-score. Finally, we take the maximum F_1-score over all 30 epochs. Since the maximum slightly depends on the initial random seed, we run CollaborEM for initial values 1 to 5. The second row of Table 5 shows the resulting maximum F_1-scores averaged over these five runs. The average scores are close to the published results for CollaborEM as shown in Table 4. However, the evaluation set consists of ground truth matches and nonmatches which are not available in an automated environment or in the context of unsupervised methods. Thus, the evaluation set cannot be consulted for selecting the best epoch. Consequently, the values in the second row should rather be interpreted as upper bounds for F_1-scores achievable in real-world projects. The first line shows AutoCal's maximum F_1-scores for MTF = 50 which are also upper bounds and can only be obtained by an optimal procedure for threshold estimation. Please note that averaging is not necessary in the case of AutoCal because it is a completely deterministic algorithm which does not depend on any random seed.

Secondly, we consider the following approach: At the beginning, CollaborEM automatically generates a seed set of matches and nonmatches. Instead of using the entire seed set for fine-tuning, we use only 90% for fine-tuning and the rest as the validation set. Again, we run CollaborEM for 30 epochs, but this time we apply the fine-tuned model to the validation set instead of the evaluation set after each epoch. Then, we choose the epoch ≤ 30 with the maximum F_1-score on the validation set. If the maximum value results from more than one epoch, then we choose the smallest of those epochs. We finally apply the fine-tuned model corresponding to the selected epoch for instance matching and calculate the F_1-score on the evaluation set. The fourth row of Table 5 shows the resulting F_1-scores - again averaged over five runs. The last row shows the analogous results for running CollaborEM for only 10 epochs. Not surprisingly, the averaged scores tend to decrease.

Table 6
Runtime in seconds of AutoCal and CollaborEM in two different hardware environments.

	Method	Setup	FZ	DA	DS	AG	KG	DG
Cloud	AutoCal		32	464	548	113	31	46
	CollaborEM	val. 10 epochs	99	1231	2982	310	364	275
Laptop	AutoCal		18	260	369	71	19	24
	CollaborEM	val. 10 epochs	393	4588	5687	1103	1145	1143

For the sake of comparison, the third row contains AutoCal's F_1-scores for the estimated threshold t_{est}, copied over from Table 3. The best score for each IM scenario within the lower block is underlined. AutoCal's F_1-scores for KG and DG are slightly better than CollaborEM's F_1-scores. However, AutoCal's F_1-score for DS is even significantly larger than CollaborEM's maximum F_1-score. We did not vary the ratio for the train-validation split or apply any further criterion for selecting the "best" epoch, possibly improving CollaborEM's F_1-scores. Nevertheless, Table 3 illustrates AutoCal's matching quality in real-world projects and indicates that none of both methods is clearly superior when applied in practice.

All matching results presented so far were achieved on a machine in the cloud with 40 GB GPU memory. Another important aspect in practice is the runtime behaviour which we checked additionally on a complementary environment: a laptop with 8 GB GPU memory. Table 6 summarises the total times (in seconds).

AutoCal makes use of the GPU only when computing sentence embeddings [26] for string-valued properties (using the CPU for this task is possible but slower). Due to an efficient implementation of this task, AutoCal benefits from a powerful GPU with large memory only for a small fraction of its total time. On the other hand, AutoCal runs faster on the laptop due to a CPU being more powerful than the CPU on the machine in the cloud.

In contrast, CollaborEM's runtime behaviour relies heavily on a powerful GPU, allowing large batches for fine-tuning the language model. According to Table 5, fine-tuning over 10 epochs comes with an acceptable loss in matching quality. Therefore, we also consider only 10 epochs when computing CollaborEM's total time. The total times in Table 6 contain all of CollaborEM's processing steps, except the computation of F_1-scores on the evaluation set. Contrary to AutoCal, the total times increase when running CollaborEM on the laptop since fine-tuning contributes to the largest portion of the total time and slows down with a smaller batch size (16 instead of 64 samples) on a less powerful GPU.

A comparison of the total times in Table 6 shows that AutoCal runs at least 2.6 times faster than CollaborEM in the cloud. On the laptop, AutoCal performs even 15 to 20 times faster for FZ, DA, DS and AG, and 60 and 47 times faster for KG and DG, respectively.

6. Integration into the World Avatar

The World Avatar (TWA) project initially focused on decarbonisation of the chemical industry on Jurong Island in Singapore. One outcome is the representation of an eco-industrial park based on Semantic Web technology [5]. The scope of TWA project has since broadened considerably. Consequently, manual data curation became unsustainably expensive.

AutoCal was designed as a first step towards the automated merging and linking of data from various providers, and towards continuously populating TWA's knowledge graph. This section summarises TWA's core principles and presents how AutoCal is integrated into TWA. Finally, we demonstrate how the prototype is applied in the domain of power plants.

6.1. Core principles of the World Avatar

TWA is built on the Semantic Web [1] and the principles of Linked Data [2]. The project strives to connect data and reusable computational agents to create a digital "avatar" of the real world, or of parts thereof. Thus, TWA consists of two layers – the knowledge graph and the agents – which we outline in the following. We refer to [9,10,37] for details of TWA and its architecture.

TWA employs ontologies to annotate data semantically, to link instances and to provide interoperability between agents. An ontology is a collection of classes and properties in a domain of interest. It can be considered a vocabulary used to express statements about the instances of this domain and about the relations between instances of this domain or other domains. Each statement is expressed as a triple consisting of a subject, a predicate and an object. For instance, power plant $a3$ which was already introduced in Fig. 2 may be described by triples such as *(a3, type, FossilFuelPlant)* and *(a3, name, "moabit")*. Here, *type* and *name* denote properties, $a3$ is an instance of type (class) *FossilFuelPlant* and *"moabit"* is a plain string.

TWA uses the Resource Description Framework (RDF)[6] and the Web Ontology Language (OWL)[7] to define properties, classes and instances formally. While we rely on short terms in this work for better readability, TWA actually employs Uniform Resource Identifiers (URIs) to identify properties, classes and instances. We refer to the supplementary material for the formal and complete description of $a3$ as an example.

Since each triple can be regarded as two nodes (the subject and the object) connected by an edge (the predicate), all triples in total form a graph. The scientific literature offers a variety of criteria of what turns such a graph into a knowledge graph. A common description is presented in [38]: "A knowledge graph

1. mainly describes real world entities and their interrelations, organised in a graph.
2. defines possible classes and relations of entities in a schema.
3. allows for potentially interrelating arbitrary entities with each other.
4. covers various topical domains."

TWA's knowledge graph conforms to the previous criteria: TWA takes ontologies from a wide range of domains such as urban planning [8], eco-industrial parks [39], and combustion chemistry [6]. Its graph contains a huge number of instances such as buildings, chemical plants, and chemical species. Another example is OntoPowSys, an ontology for electrical power systems. Among other fields, it is used to model the electrical grid on Jurong Island in Singapore [7]. The resulting representation is composed of bus nodes, electrical lines, power converters etc. and is connected to instances of fossil fuel generators and chemical plants. This allows a holistic cross-domain approach to eco-industrial parks and illustrates the importance of interrelating instances in the context of TWA.

Agents form the second layer of TWA. Agents are pieces of software that can invoke each other by HTTP requests. They query and update data in the knowledge graph by means of SPARQL,[8] and perform a range of activities from simple tasks, such as presenting data from the knowledge graph, to more complex tasks, such as optimising the power flow of an electrical grid. Agents themselves can be described by an ontology named OntoAgent which allows to reuse and compose agents automatically to solve cross-domain tasks [37]. In this way, their descriptions become instances of type agent, stored in TWA's knowledge graph.

[6] https://www.w3.org/TR/rdf11-primer/
[7] https://www.w3.org/TR/owl2-overview/
[8] https://www.w3.org/TR/sparql11-query/

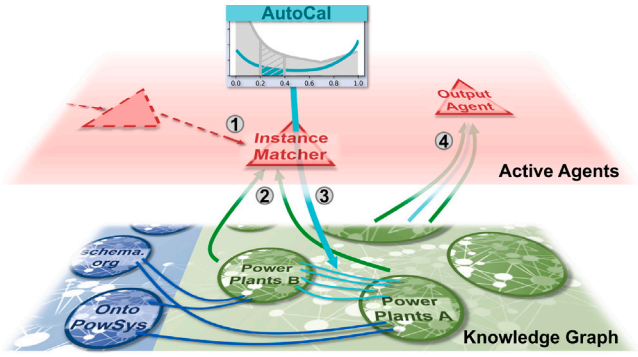

Fig. 6. Architecture of the World Avatar using the example of the instance matching agent and triples for the power plant domain.

It is worth noting some differences between TWA's knowledge graph and popular knowledge graphs such as DBpedia [40]. TWA's knowledge graph focuses on technical domains such as infrastructure and chemical engineering. Moreover, it is distributed over several Web nodes. The most striking difference shows in its dynamics: Agents simulate the behaviour of connected real-world entities and optimise their current state by incorporating data from time series or real-time data (such as weather conditions, energy demand, or ship positions). The optimised states are updated for the corresponding instances in the knowledge graph. This creates a "living digital twin" of the real world. Consequently, state-related triples may change on a much smaller time-scale compared to other knowledge graphs.

6.2. Integration of AutoCal and use case

Fig. 6 illustrates TWA's core architecture and the prototypical integration of instance matching. The lower layer shows some part of the knowledge graph which is relevant for our prototype. Power plants in TWA's knowledge graph are represented by means of OntoPowSys and partly by TWA's ontology for eco-industrial parks. Non-technical information such as address and country are expressed by properties *streetAddress*, *postalCode* and *addressLocality* from the *schema.org* vocabulary[9] and by referencing country instances from DBpedia knowledge graph, respectively. In the lower layer, the references of classes and properties are indicated by blue lines between sets of instances (green circles) and ontologies (blue circles).

For the prototype, we implemented a new *Instance Matcher* agent which wraps AutoCal and the matching pipeline from Fig. 1. The agent requires a configuration set that specifies data sets A and B, and parameters for token-blocking, property mapping, and optional evaluation. In Section 5, the data sets A and B were provided in Magellan's CSV format and described by the same column names which simplifies the evaluation of AutoCal and CollaborEM. Here, the matching context is different: Both data sets are triple sets. Data set A contains new data that we want to integrate into the knowledge graph while data set B is a suitable existing subset of the knowledge graph. As soon as the *Instance Matcher* agent is called (step 1 in Fig. 6), it reads the triple sets (step 2), maps the values for the configured properties into a tabular format, performs the pipeline and adds a triple of the form *(a, sameAs, b)* to the knowledge graph for each predicted match (a, b) (step 3). The property *sameAs* is a standard property defined as part of OWL. It declares instances a and b as the same. In Fig. 6, the turquoise lines between the triple sets for A and B indicate resulting *sameAs*-triples.

The upper layer also shows an *Output Agent* which in this case, allows a user to query or visualise data from the knowledge graph (step

[9] https://schema.org/

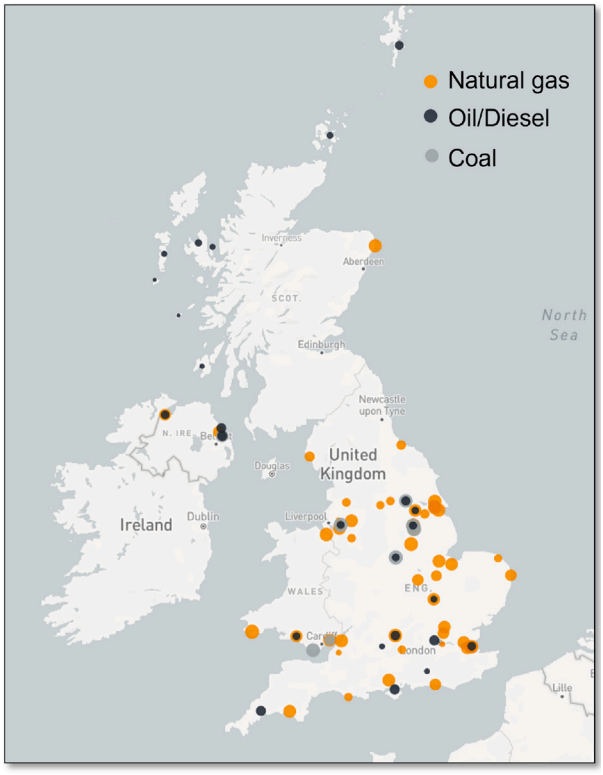

Fig. 7. Geographical distribution of fossil fuel power plants in the UK.

4). This agent could be a standard SPARQL query browser to query matched power plants, or a visualisation agent which queries certain types of power plants and their geographic coordinates and creates a map as shown in Fig. 7.

Regarding TWA, power plant instances are relevant for a variety of scenarios of different scale. As mentioned in the previous subsection, they may represent energy suppliers in the electrical grid as part of the industrial park on Jurong Island in Singapore. Another example of local scale is the air pollution scenario described in [9] in which the dispersion profile of the power plant's emissions is estimated under consideration of surrounding buildings and real-time data on weather conditions. In [10], a national digital twin based on TWA's knowledge graph is proposed which helps to optimise, plan and develop the infrastructure in the UK including power systems and the electrical and gas transmission systems.

While the real world undergoes changes, e.g., fossil-fuel power plants are decommissioned and renewable energy sources are being developed continuously, agents should work on current data to achieve accurate results in the aforementioned scenarios. Some data sets such as the British DUKES data set in Table 2 are updated and published yearly. However, manual curation of an increasing number of instances is laborious.

The datasets related to matching scenarios DG and KG in Table 2, i.e., DUKES, KWL, and GPPD, contain both fossil fuel, nuclear, and renewable energy power plants. In contrast, prior to applying the instance matching, TWA's knowledge graph contained 5225 instances of power plants worldwide, most of them running on fossil fuel. This set of original instances is based on data selected from GPPD, Global Energy Observatory,[10] and Enipedia.[11] Its size is much smaller than the size of the complete GPPD dataset and may affect matching quality negatively. For this reason, we run AutoCal to match instances of all types in DUKES and KWL, resp., and fossil-fuel-only instances in GPPD. AutoCal's F_1-score (in percent) improved from 87.7 to 98.8 and from 82.9 to 90.1, resp., in comparison to the results of Table 5 for the complete GPPD dataset.

The matching results illustrate how AutoCal supports the task of keeping information up-to-date: The additional renewable power plants in the original data sources KWL and DUKES have been integrated as new instances in TWA's knowledge graph. When data providers subsequently publish a new version of their data, AutoCal facilitates matching the new version to the expanded set of power plant instances. This helps to avoid instance duplication and to preserve a consistent knowledge graph with less curation effort.

Populating the knowledge graph enables modelling and analysis of advanced scenarios in the framework of TWA: The authors of [41] examine the curtailment of British wind energy and the causes of associated energy losses. They integrate numerous data sources for a deeper investigation into the energy landscape, based on TWA's knowledge graph to facilitate a multi-source analysis. Study [42] investigates energy storage systems as a potential solution to the rising issues of electricity price volatility and British wind curtailment. It applies OntoPowSys to describe the energy system (including wind power plants) and an extension of OntoPowSys to specify energy storage systems. Agents simulate the behaviour of energy storage systems and update the state of corresponding instances in the knowledge graph. The unique knowledge graph approach in the study allows to coordinate a variety of data sources and agents and to perform more sophisticated analyses.

The TWA project aims for a high degree of automation in many respects. What are the implications for instance matching? First of all, neither AutoCal nor CollaborEM requires manually labelled instance pairs. Moreover, we showed that AutoCal's default parameter values ensure an acceptable matching quality for a wide range of IM scenarios. This suggests that AutoCal can be applied to new incoming data (from other domains) out-of-the-box, i.e., without the need to fine-tune parameters on an extra set of labelled pairs. As pointed out in Section 5.5, CollaborEM must be equipped with a criterion for selecting the "best" epoch. Once this is done, its default parameter values can be used likewise for instance matching in real-world projects.

In general, the names of the respective columns or properties describing new data and data in the knowledge graph differ. This requires preliminary property matching. In future, an agent could automate this task by applying existing methods from the field of schema matching. Likewise, the choice of similarity functions in the case of AutoCal and the choice of graph features in the case of CollaborEM could be automated by analysing the type and range of data for each column or property.

In summary, both AutoCal and CollaborEM are suitable building blocks for automated instance matching within TWA. While none of them is superior with respect to matching quality they significantly differ regarding runtime behaviour and hardware requirements. Contrary to knowledge graphs such as DBpedia, TWA's knowledge graph is not centralised but its triples are distributed over several Web nodes. Moreover, its agents may run on the same nodes which often are server machines with strong CPUs but less powerful GPUs. Since AutoCal allows faster instance matching on the same hardware, it is particularly suited for deployment on TWA.

7. Conclusions

In the first part of this paper, we presented AutoCal, a new algorithm for unsupervised instance matching of tabular data sets. AutoCal consists of three steps based on heuristics and statistics: It computes maximum similarity vectors on subsets of paired instances sharing tokens. It uses their empirical marginal distributions to derive calibrated property scores. Finally, it estimates a matching threshold for the aggregated property scores. Token-blocking suits AutoCal's first step in

[10] http://globalenergyobservatory.org
[11] https://archive.ph/20140610231532/http://enipedia.tudelft.nl/

a natural way and is applied – in combination with the maximum token frequency – as a preparatory step to reduce the number of candidate pairs to a feasible size.

We evaluated AutoCal with six different scenarios for instance matching: Two new scenarios with power plant entities exhibiting a mixture of properties with short string, numerical and missing values, as well as four scenarios from other domains which are frequently used in the literature and which contain a large portion of text. We collected a wide variety of state-of-the-art methods for instance matching for a comprehensive comparison with AutoCal and summarised the results published for the four scenarios. Our results for AutoCal show that its matching quality is competitive to recent unsupervised and semi-supervised instance matchers from the field of Machine Learning.

Unsupervised Deep Learning methods frequently achieve better matching results on said four common scenarios. Within this class of methods, we selected CollaborEM as a state-of-the-art matcher for a detailed comparison with AutoCal. Both CollaborEM equipped with a stop criterion and AutoCal can be applied out-of-the-box for new matching tasks without manually adapting default values for method-specific parameters. Unexpectedly, it turned out that neither AutoCal nor CollaborEM are superior regarding matching quality. Moreover, AutoCal runs 2.7 up to 60 times faster than CollaborEM for the six matching scenarios.

While AutoCal can run stand-alone, we illustrated in the second part of the paper how the new method is integrated prototypically into the World Avatar (TWA) as an agent. TWA is a dynamic knowledge graph approach for digital twins and aims for a high degree of automation. Since AutoCal achieves competitive matching results and can run out of the box, it is an important step towards automatically and continuously populating the knowledge graph with new or updated data from external sources. TWA's knowledge graph and agents are distributed over several Web nodes. Due to its runtime behaviour and moderate hardware requirements, AutoCal is particularly suitable to run on the same decentralised infrastructure.

Finally, we demonstrated how AutoCal matches new data from the power plant domain with existing instances in the knowledge graph. In the presence of additional information and/or discrepancies in data, the current instance matching prototype is not yet able to determine whether they are explainable or pure data errors. Despite this, the prototype has increased the amount of power plant data that can be queried in TWA. It can highlight data that need more investigation. As the next step, we will extend existing ontologies and develop new ontologies to integrate publicly available data, which can act as supplementary information, into TWA to discern such situations. We will also examine state-of-the-art approaches to automatically discover data relevant to TWA on the World Wide Web.

CRediT authorship contribution statement

Andreas Eibeck: Conceptualisation, Methodology, Software, Data curation, Writing – original draft, Visualisation. **Shaocong Zhang:** Conceptualisation, Methodology, Software, Writing – review & editing. **Mei Qi Lim:** Conceptualisation, Writing – review & editing, Supervision, Project administration. **Markus Kraft:** Conceptualisation, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Research data supporting this publication is available in the University of Cambridge data repository (doi:10.17863/CAM.82548)

Acknowledgements

M.K. gratefully acknowledges the support of the Alexander von Humboldt Foundation. The authors thank Dr Feroz Farazi for discussions, Jörg Preisendörfer for commenting extensively on the paper, Andrew Breeson for proofreading, and Wanni Xie for providing semantically annotated data. All authors approved the version of the manuscript to be published.

Funding

This project is funded by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Part of this work was supported by Towards Turing 2.0 under the EPSRC Grant EP/W037211/1 & The Alan Turing Institute.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.websem.2024.100815.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Sci. Am. 284 (5) (2001) 34–43.
[2] C. Bizer, T. Heath, T. Berners-Lee, Linked data: The story so far, in: Semantic Services, Interoperability and Web Applications: Emerging Concepts, IGI global, 2011, pp. 205–227.
[3] T.R. Gruber, A translation approach to portable ontology specifications, Knowl. Acquis. 5 (2) (1993) 199–220.
[4] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G.d. Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, Synth. Lect. Data, Semant., Knowl. 12 (2) (2021) 1–257.
[5] L. Zhou, M. Pan, J.J. Sikorski, S. Garud, L.K. Aditya, M.J. Kleinelanghorst, I.A. Karimi, M. Kraft, Towards an ontological infrastructure for chemical process simulation and optimization in the context of eco-industrial parks, Appl. Energy 204 (2017) 1284–1298.
[6] F. Farazi, M. Salamanca, S. Mosbach, J. Akroyd, A. Eibeck, L.K. Aditya, A. Chadzynski, K. Pan, X. Zhou, S. Zhang, et al., Knowledge graph approach to combustion chemistry and interoperability, ACS Omega 5 (29) (2020) 18342–18348.
[7] A. Devanand, G. Karmakar, N. Krdzavac, R. Rigo-Mariani, Y.F. Eddy, I.A. Karimi, M. Kraft, OntoPowSys: A power system ontology for cross domain interactions in an eco industrial park, Energy AI 1 (2020) 100008.
[8] A. Chadzynski, S. Li, A. Grisiute, F. Farazi, C. Lindberg, S. Mosbach, P. Herthogs, M. Kraft, Semantic 3D city agents — An intelligent automation for dynamic geospatial knowledge graphs, Energy AI (2022) 100137.
[9] A. Eibeck, M.Q. Lim, M. Kraft, J-Park Simulator: An ontology-based platform for cross-domain scenarios in process industry, Comput. Chem. Eng. 131 (2019) 106586.
[10] J. Akroyd, S. Mosbach, A. Bhave, M. Kraft, Universal digital twin – A dynamic knowledge graph, Data-Centric Eng. 2 (2021).
[11] A. Ferrara, A. Nikolov, F. Scharffe, Data linking for the Semantic Web, Int. J. Semant. Web Inf. Syst. (IJSWIS) 7 (3) (2011) 46–76.
[12] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, E. Rahm, A survey of current link discovery frameworks, Semant. Web 8 (3) (2017) 419–436.
[13] A. Jurek-Loughrey, P. Deepak, Semi-supervised and unsupervised approaches to record pairs classification in multi-source data linkage, in: Linking and Mining Heterogeneous and Multi-View Data, Springer, 2019, pp. 55–78.
[14] P. Christen, Data Matching, Springer, 2012.
[15] H. Köpcke, E. Rahm, Frameworks for entity matching: A comparison, Data Knowl. Eng. 69 (2) (2010) 197–210.
[16] N. Barlaug, J.A. Gulla, Neural networks for entity matching: A survey, ACM Trans. Knowl. Discov. Data (TKDD) 15 (3) (2021) 1–37.
[17] C. Ge, P. Wang, L. Chen, X. Liu, B. Zheng, Y. Gao, CollaborEM: A self-supervised entity matching framework using multi-features collaboration, IEEE Trans. Knowl. Data Eng. (2021).
[18] A. Eibeck, S. Zhang, OntoMatch, 2022, https://github.com/cambridge-cares/TheWorldAvatar/tree/main/Agents/OntoMatchAgent.
[19] P. Li, X. Cheng, X. Chu, Y. He, S. Chaudhuri, Auto-FuzzyJoin: Auto-program fuzzy similarity joins without labeled examples, in: Proceedings of the 2021 International Conference on Management of Data, 2021, pp. 1064–1076.
[20] M.A. Sherif, A.-C. Ngonga Ngomo, J. Lehmann, WOMBAT – A generalization approach for automatic link discovery, in: European Semantic Web Conference, Springer, 2017, pp. 103–119.

[21] M. Kejriwal, D.P. Miranker, Semi-supervised instance matching using boosted classifiers, in: European Semantic Web Conference, Springer, 2015, pp. 388–402.
[22] A. Jurek, J. Hong, Y. Chi, W. Liu, A novel ensemble learning approach to unsupervised record linkage, Inf. Syst. 71 (2017) 40–54.
[23] R. Wu, S. Chaba, S. Sawlani, X. Chu, S. Thirumuruganathan, ZeroER: Entity resolution using zero labeled examples, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 1149–1164.
[24] R. Cappuzzo, P. Papotti, S. Thirumuruganathan, Creating embeddings of heterogeneous relational datasets for data integration tasks, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 1335–1349.
[25] J. Shao, Q. Wang, A. Wijesinghe, E. Rahm, ErGAN: Generative adversarial networks for entity resolution, in: 2020 IEEE International Conference on Data Mining, ICDM, IEEE, 2020, pp. 1250–1255.
[26] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using Siamese BERT-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019.
[27] G. Papadakis, E. Ioannou, C. Niederée, P. Fankhauser, Efficient entity resolution for large heterogeneous information spaces, in: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, 2011, pp. 535–544.
[28] K. O'Hare, A. Jurek-Loughrey, C. de Campos, A review of unsupervised and semi-supervised blocking methods for record linkage, Linking and Mining Heterogeneous and Multi-view Data (2019) 79–105.
[29] W. Cohen, P. Ravikumar, S. Fienberg, A comparison of string metrics for matching names and records, in: KDD Workshop on Data Cleaning and Object Consolidation, vol. 3, 2003, pp. 73–78.
[30] J. De Bruin, Python record linkage toolkit: A toolkit for record linkage and duplicate detection in Python, 2019, http://dx.doi.org/10.5281/zenodo.3559043.
[31] H. Köpcke, A. Thor, E. Rahm, Evaluation of entity resolution approaches on real-world match problems, Proc. VLDB Endow. 3 (1–2) (2010) 484–493.
[32] A. Eibeck, S. Zhang, M.Q. Lim, M. Kraft, Research data, 2022, http://dx.doi.org/10.17863/CAM.82548.
[33] P. Konda, S. Das, P.S. GC, A. Doan, A. Ardalan, J.R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al., Magellan: Toward building entity matching management systems, Proc. VLDB Endow. 9 (12) (2016).
[34] 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: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 19–34.
[35] Y. Li, J. Li, Y. Suhara, A. Doan, W.-C. Tan, Deep entity matching with pre-trained language models, 2020, arXiv preprint arXiv:2004.00584.
[36] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, N. Tang, Distributed representations of tuples for entity resolution, Proc. VLDB Endow. 11 (11) (2018) 1454–1467.
[37] X. Zhou, A. Eibeck, M.Q. Lim, N.B. Krdzavac, M. Kraft, An agent composition framework for the J-Park Simulator – A knowledge graph for the process industry, Comput. Chem. Eng. 130 (2019) 106577.
[38] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semant. Web 8 (3) (2017) 489–508.
[39] L. Zhou, C. Zhang, I.A. Karimi, M. Kraft, An ontology framework towards decentralized information management for eco-industrial parks, Comput. Chem. Eng. 118 (2018) 49–63.
[40] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al., DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, Semant. Web 6 (2) (2015) 167–195.
[41] J. Atherton, W. Xie, F. Farazi, S. Mosbach, J. Akroyd, M. Kraft, Breakdown of British wind curtailment using a multi-source knowledge graph approach, 2023, in press, https://como.ceb.cam.ac.uk/media/preprints/c4e-preprint-304.pdf. (Accessed 2 June 2023).
[42] J. Atherton, J. Akroyd, F. Farazi, S. Mosbach, M.Q. Lim, M. Kraft, British wind farm ESS attachments: curtailment reduction vs. price arbitrage, Energy & Environmental Science 16 (9) (2023) 4020–4040.