[Vision] Self-configured Entity Resolution with pyJedAI

Vasilis Efthymiou[1], Ekaterini Ioannou[2], Manos Karvounis[3], Manolis Koubarakis[4], Jakub Maciejewski[4], Konstantinos Nikoletos[4], George Papadakis[4], Dimitris Skoutas[5], Yannis Velegrakis[6], Alexandros Zeakis[5] (alphabetical order)

[1]*Foundation for Research and Technology - Hellas, Greece* vefthym@ics.forth.gr
[2]*Tilburg University, The Netherlands* Ekaterini.Ioannou@tilburguniversity.edu
[3]*Agroknow - Greece* manos.karvounis@agroknow.com
[4]*University of Athens, Greece* {koubarak,sdi1700080,k.nikoletos,gpapadis}@di.uoa.gr
[5]*Athena Research Center, Greece* {dskoutas,azeakis}@athenarc.gr
[6]*Utrecht University, The Netherlands* i.velegrakis@uu.nl

***Abstract*—Entity Resolution has been an active research topic for the last three decades, with numerous algorithms proposed in the literature. However, putting them into practice is often a complex task that requires implementing, combining and configuring complementary individual algorithms into comprehensive end-to-end workflows. To facilitate this process, we are developing pyJedAI, a novel system that provides a unifying framework for any type of main works in the field (i.e., both unsupervised and learning-based ones). Our vision is to facilitate both novice and expert users to use and combine these algorithms through a series of principled approaches for automatically configuring and benchmarking end-to-end pipelines.**

***Index Terms*—entity resolution, automatic configuration**

I. Introduction

Entity Resolution (ER) has been an active area of research for the last decades, as it constitutes a core data integration task [1]. Most works on ER, though, focus on the development of individual methods that pertain to a single step in end-to-end ER pipelines [2]. As a result, many open challenges remain towards building unified, holistic ER systems that are capable of addressing modern data integration needs, especially as organizations increasingly turn to data lakes to enable ad hoc, self-service analytics [3].

In such contexts, ER has to face massive collections of datasets from different sources, which may change autonomously over time, and which are characterized by high variety in terms of structure, format, content, metadata and quality [4]. Examples of related ER challenges are given in the following two use cases:

1) Agroknow (https://agroknow.com) maintains a data lake with reports about food safety incidents (e.g., salmonella in a particular poultry product) that are crawled from the web sites of various regional, national or international authorities from all over the world. Typically, every incident is reported by multiple food safety authorities, as the problematic lots are imported and distributed in various countries. The high complexity of the food supply chain results in the same underlying issues being reported at different dates, from different countries, or in different points in the overall supply chain (e.g., at the market or at a distribution or processing level), and might even involve multiple and distinct lot numbers, all of which might be affected by the same higher-level issue. A food safety expert needs to know as fast as possible if a particular food incident is the same as an already seen one, or is an entirely new incident that requires immediate action. Currently, this process requires time-consuming human involvement, archival search and cross-department or cross-organisational communication and coordination. ER can automate this process, assisting the food safety experts to handle urgent issues of public health more decisively and timely.

2) Marketplace platforms recently focused on gaining a competitive advantage by adopting digital transformation strategies, including the use of data lakes for various tasks. Consider, for example, a platform that creates a network of resources based on current market demands and information that is then decomposed to determine its product prices and/or discounts [5]. Constructing this network requires being able to detect identical products from other platforms. Another example is bidding on advertising slots for products through real-time auctions. For instance, approaches have been suggested to automatically determine the applications to bid as well as the bid prices [6]. Relevant approaches are not restricted to detecting identical products, but can actually benefit from a loose form of matching between the products.

In such data lakes, pay-as-you-go entity-centric approaches are necessary for modern data integration use cases in addition to the traditional batch-based integration. The latter produce results only after processing the entire input and thus are inappropriate for ER applications with restricted computational or temporal resources (e.g., cloud-based applications with a limited budget for AWS Lambda functions); these are served by the former, which maximize the throughput [7]. Besides tackling efficiency, scalability, accuracy and data variety, modern ER systems also need to offer increased automation, trustworthiness and ease of integration with other widely used data science tools.

There exist numerous ER tools, both commercial and open-source ones. However, the former encompass algorithms and features that are largely unclear, as explained in the extended version of [8]. The latter come from the database and the Semantic Web communities, mainly focusing on structured (e.g., Magellan [8]) and semi-structured (e.g., LIMES [9]) data, respectively. There exist also certain systems, namely

MinoanER [3] and Machop [10], that bridge the gap between the two groups by applying seamlessly to structured and semi-structured data. However, all these tools implement a few methods, which are typically crafted for batch ER, as explained by the detailed comparisons in [11]. Thus, they suffer from a narrow scope, while requiring expert users to fine-tune and deploy them to practical use cases. This means that they are not suitable for covering the modern ER challenges, such as those arising in the context of data lakes.

To address these issues, we are developing pyJedAI, a novel open-source system in Python that aims to become the ultimate library of ER. Its goal is to unify all state-of-the-art approaches into a common framework that facilitates their use through ample documentation and principled approaches for *automatic* configuration optimization. pyJedAI will also allow for benchmarking individual methods and entire pipelines. In what follows, we elaborate on our plan for pyJedAI.

II. Problem Definition

The main ER task can be formally defined as follows [3]:

Problem 1 (Clean-Clean ER): Given two duplicate-free entity collections, E_1 and E_2, detect all the duplicate entity profiles $\{(e_i, e_j)|\ e_i \in E_1 \wedge e_j \in E_2 \wedge e_i \equiv e_j\}$, where $e_i \equiv e_j$ indicates that e_i and e_j describe the same real-world object.

Clean-Clean ER is also called Record Linkage, in contrast to Deduplication or *Dirty ER*, where a single entity collection E is given as input with duplicates in itself [12], [1].

Typically, ER deals with entity profiles containing textual attributes (e.g., names, descriptions, addresses). However, many entities are also associated with a geolocation (e.g., entities corresponding to physical objects). In that case, the geometries (e.g., points, lines, polygons) of these entities need to be taken into account. This may also lead to additional types of relationships, besides equivalence, such as containment or adjacency – e.g., a store may be located within a shopping mall, or a parking lot may be adjacent to a building. Linking these entities is often crucial for many applications, like providing routing directions in a navigation application.

Geospatial Interlinking [13] is formally defined as [14]:

Problem 2 (Geospatial Interlinking): Given a source dataset S, a target dataset T along with a set of topological relations R, discover the set of links $L_R = \{(s, r, t)|s \in S \wedge t \in T \wedge r(s, t) \wedge r \in R\}$ as efficiently as possible.

Note that popular topological relations are those defined by the DE-9IM model [14]: intersects, contains, within, covers, covered-by, equals, touches, crosses, overlap and disjoint. Note also that both ER and Geospatial Interlinking suffer from a quadratic time complexity, as their brute-force solutions have to consider all valid pairs of input profiles. Hence, both tasks are solved through a **filtering-verification framework** in practice: *the filtering step, a.k.a., blocking*, applies a quick and approximate procedure that reduces the computational cost to the candidate pairs, i.e., profiles that are most likely to be true positives, while *the verification step, a.k.a., matching*, performs a time-consuming process that examines the resulting candidates in detail. Methods of these two steps are typically combined in (complex) end-to-end workflows or pipelines.

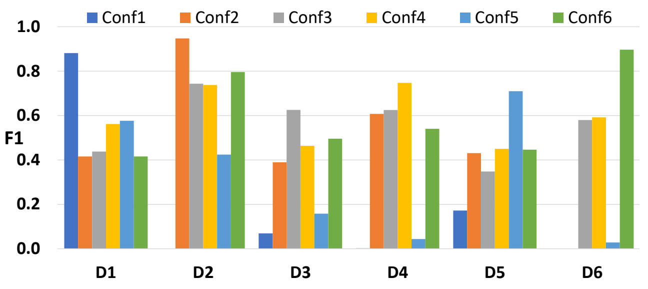

Fig. 1. Impact of parameter configuration on ER performance.

III. Challenges

ER researchers and practitioners face the following challenges when implementing ER solutions:

- The internal parameters typically affect the performance of individual methods and, thus, entire workflows to a large extent. As an example, consider Figure 1. We have fine-tuned pyJedAI on six different datasets from various domains that are widely used in the literature: D_1 (restaurants), D_2 (abt.com-buy.com), D_3 (Amazon-Google Products), D_4 (IMDB-TMDB), D_5 (Walmart-Amazon) and D_6 (Movies). $Conf_x$ denotes the fine-tuned pipeline for D_x.[1] Applying these pipelines on all datasets, we observe that the relative difference in F-measure between $Conf_x$ and the second best pipeline on D_x, for x an integer in $[1, 6]$, is rather high, fluctuating between 16% (D_2) and 36.5% (D_5).[2] In other words, none of the pipelines approximates the optimal performance in any dataset apart from the fine-tuned one. Note also that every pipeline consists of at least five consecutive methods and, thus, their configuration is non-trivial, especially for novice users. This clearly demonstrates the critical role played by the configuration parameters of end-to-end ER pipelines. Yet, fine-tuning an entire ER pipeline or even one of its components is a non-trivial task, as it requires knowledge of the domain of each configuration parameter as well as principled techniques for minimizing the search space.

- Similarly, benchmarking individual state-of-the-art algorithms on the same task is another challenge, because their relative performance depends heavily on their internal parameters, thus calling for configuration optimization.

- It is hard for users, especially the novice ones, to keep up with the latest advancements in ER. For example, at least six open-source matching algorithms based on deep learning have been recently published [15]. Using a tool that timely incorporates latest developments would alleviate this difficulty.

- Applying an existing ER technique to a task that is slightly different from the one that it was originally crafted for is not straightforward. Consider as an example a DL-based matching algorithm for batch ER that is applied to streaming data.

[1] For more details about the performance and configuration per dataset, please see: https://pyjedai.readthedocs.io/en/latest/pages/benchmarks.html.

[2] One could argue that a common, high-performing workflow for all datasets can be determined by performing grid search on their union. This is not practical, though, as ER practitioners want to reuse workflows fine-tuned on one domain to another.

• To the best of our knowledge, none of the existing open-source systems justifies its results, apart from providing a similarity score between candidate pairs of entities. In many applications, though, detailed explanations are required about (non-)matching decisions, especially when processing sensitive data, in order to ensure the absence of bias. Related to this issue is the lack sufficient documentation, which can be used for automatically generating explanations. The documentation is also crucial for users, even experienced ones, who try to build workflows, facilitating them to select the most suitable techniques for the task(s) at hand. This pertains both to the configuration of individual methods and to the resulting performance in terms of effectiveness and time efficiency (e.g., 'the similarity threshold of matching algorithm X was set high, to 0.8, in order to trade slightly lower F-measure for significantly lower run-time').

IV. Past and Present

To address these challenges, we are developing pyJedAI, which currently offers a comprehensive set of established, learning-free methods for Clean-Clean ER and Geospatial Interlinking. pyJedAI has been under intense development since 2017 [11], [16], [17], with all versions publicly available[3], offering the following unique characteristics:

• It is released under the Apache License, Version 2.0, thus supporting both academic and industrial applications.

• It is *format-agnostic* in the sense that it allows for resolving datasets in any data format: structured (relational databases, csv files), semi-structured (SPARQL endpoints, RDF dumps, XML and OWL files) and unstructured (free text). Any combination of these datasets is also possible. E.g., pyJedAI supports scenarios of Problem 1, where E_1 is a structured data source, E_2 an unstructured one, while their ground-truth is expressed as semi-structured data. This versatility is achieved through a simple, generic data model based on name-value pairs, and through an inherently schema-agnostic functionality in every step of every end-to-end workflow.

• It is *extensible* in the sense that it facilitates the integration of any algorithm that targets specific workflow steps in the available end-to-end pipelines. For example, every component in Figure 2 specifies a concise interface, based on appropriate data structures. Thus, any ER technique can be added to this pipeline through a wrapper that receives the input of the corresponding workflow step and transforms it into the required format. The wrapper should also transform the output of the technique into the data structure produced by the corresponding workflow step. This adds new techniques in a transparent way, without requiring heavy user involvement – just routine data transformations. We have already applied this approach to: (i) the state-of-the-art approximate nearest-neighbor search techniques FAISS [18] and FALCONN [19], (ii) DeepBlocker [20], which leverages deep learning without requiring a training set, and (iii) the main language models, like the (Sentence)BERT-based ones, which can be used in the schema matching, block building and/or entity matching steps of the pipeline of Figure 2 [21].

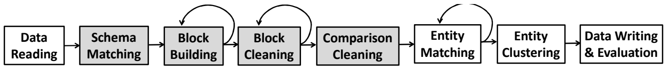

Fig. 2. One of pyJedAI's pipelines for batch ER. Gray rectangles indicate optional steps. Self-loops show that multiple methods can be selected per step.

• It is *holistic* in the sense that it supports end-to-end pipelines of any type with respect to schema- and budget-awareness for Problems 1 and 2. Schema-awareness distinguishes ER pipelines into schema-based (leveraging similarity joins) and schema-agnostic ones (leveraging blocking), while budget-awareness distinguishes them into budget-agnostic (batch) and budget-aware (progressive/pay-as-you-go) ones.

• It is *efficient* with respect to time and memory requirements, due to implementation optimizations (e.g., using native Python's data structures) and the largest set of relevant techniques than any other open-source ER tool [11].

• It is *versatile* in the sense that it supports all major execution modes, i.e., stand-alone execution on a single CPU-core, massively parallel execution on top of Apache Spark and multi-core execution. These execution models are uniformly provided to all methods addressing Problems 1 and 2.

• It is *usable* in the sense that it supports a wide variety of user interfaces. These include a command-line interface, a desktop application (based on JavaFX) with an intuitive wizard-like interface[4], a Web application based on ReactJS and Spring (boot+ MVC) that can be easily installed and run through Docker[5] as well as Python notebooks and ready-to-use interfaces that require no installation, as they are built on top of Colab.[6] Each type of interface demonstrates the relative performance of evaluated pipelines in terms of effectiveness and evaluation measures (i,e., recall, precision, F1 and run-times). Most of them also provide more advanced visualizations that allow for examining the input, the intermediate and the output data of an end-to-end ER workflow as well as for comparing the relative performance of two or more workflows in more detail. Equally important is the documentation that accompany pyJedAI's interfaces. They include how-to guides and examples for all available methods, explaining how to apply them through each user interface[7], but also how to use them in code – the latter consists of a set of scripts that demonstrate the application of highly performing pipelines on a series of benchmark datasets. Note also that every method added to pyJedAI implements the `Documentation` interface, which provides a short description of its functionality, the role of its configuration parameters and their domain. These interfaces are available for methods addressing both Problems 1 and 2.

[3] https://github.com/AI-team-UoA/pyJedAI

[4] See https://github.com/scify/jedai-ui for more details.

[5] See https://github.com/AI-team-UoA/JedAI-WebApp for more details.

[6] https://colab.research.google.com

[7] E.g., video tutorials like https://www.youtube.com/watch?v=OJY1DUrUAe8&t=3s.

V. Future Outlook

Our vision is to turn pyJedAI into the ultimate community-driven library for ER that addresses all challenges in Section III through configuration optimization. To this end, we are actively working towards integrating the state-of-the-art open-source matching techniques that leverage machine and deep-learning (see [15] for an overview). Special care is also taken to make the most of the open-source large language models that are publicly available through Hugging Face[8], building matching algorithms based on prompt engineering [22], [23].

With the existing methods and the ones that will be added soon, pyJedAI can be used to creating millions of end-to-end ER workflows, i.e., realizing any possible ER situation. Yet, fine-tuning a particular pipeline is a non-trivial task, especially for a novice user, who typically is not familiar with most of the available techniques. Therefore, our main goal is to equip pyJedAI with techniques that facilitate or even optimize the configuration of *any* pipeline. More specifically, pyJedAI should support the following five scenarios:

- *Scenario 1: Known ground-truth and pipeline.* In this case, the goal is to fine-tune a specific end-to-end workflow over a given dataset, which is accompanied by the golden standard of real duplicates (ground-truth). Several techniques can be leveraged, such as grid and random search, but the former is too time-consuming and even impractical for complex pipelines, while the latter is not guaranteed to identify the best configuration for all workflow steps in an end-to-end pipeline. Therefore, *more advanced algorithms for configuration optimization are required to effectively reduce the extremely large search space of configuration parameters. For higher efficiency, the new algorithms should provide inherent support to multi-core or MapReduce parallelization. They should also apply to* any *possible pipeline generated by pyJedAI.* To this end, pyJedAI will adapt the idea of model cards for machine learning models [24] to ER methods and pipelines: every implemented method will be associated with a reasonable set of values (e.g., q in q-grams blocking is usually defined in $[2, 6]$ with a step of 1), the appropriate metrics, its intended use, factors affecting its performance etc.

- *Scenario 2: Known ground-truth, but unknown pipeline.* This is a generalization of the previous task. Instead of trying to optimize a specific ER pipeline, its goal is to identify the best workflow for the data at hand among those supported by pyJedAI based on its golden standard of duplicates. The key in this process is to significantly curtail the search space of possible end-to-end workflows. No relevant methods have been proposed for ER in the literature. Inspiration will be drawn from the AutoML literature [25] as well as from recent works on meta-learning [26]. The final step in this process is to optimize the configuration of the selected pipeline using the methods of the previous scenario.

- *Scenario 3: Unknown ground-truth, but known pipeline.* In these settings, the user tries to apply a specific pipeline that is expected to yield high performance to a dataset that lacks a golden standard of duplicates. Apparently, this requires fine-tuning the configuration parameters of its constituent methods. However, there is no relevant method in the literature at the moment. *We thus plan to develop new techniques that extract generic features from the input data and are capable of predicting near-optimal parameter values for each workflow step based on learning models that have been trained on instances that capture the performance of state-of-the-art pipelines over a series of established ER datasets.*

- *Scenario 4: Unknown ground-truth and pipeline.* Given that pyJedAI aims at a broad audience, its user base will primarily comprise novice users who perform exploratory analyses. As a result, most of its practical applications will involve datasets with unknown ground-truth and unknown pipelines. Unfortunately, the literature lacks any approaches for fine-tuning end-to-end workflows under these settings. To address this issue, *we will develop and implement novel algorithms for configuration recommendation*. They receive as input a specific dataset along with the performance of numerous pipelines over established benchmark datasets. Based on generic dataset profiles, they produce a ranking list of the most suitable pipelines together with the corresponding parameters.

- *Scenario 5: Type-specific configurations.* Data lakes and knowledge graphs typically involve a large number of entity profiles that stem from a rich diversity of data sources and domains, thus differing significantly in their characteristics. For example, most product descriptions are usually associated with a single long textual description, while music and person profiles are associated with multiple, rather short attributes. In these settings, applying a common end-to-end workflow to all entity profiles is probably less effective and efficient than applying a different pipeline to the entities of each type. Even though several techniques have been proposed for automatically detecting entity types [27], *there is no relevant approach for a-priori specifying the best workflow and configuration per entity type in a data lake. We intend to equip pyJedAI with novel algorithms for parameter fine-tuning based on generic characteristics of the automatically detected entity types. The goal of these configurations is to minimize the overall run-time, while maximizing accuracy.*

Seemingly, these tasks can be solved by existing approaches for transfer learning or hyperparameters fine-tuning. In practice, though, the automatic configuration of end-to-end ER pipelines differs substantially from these two approaches, which focus on configuring individual methods. Instead, our vision is to develop data-driven approaches that automatically build effective pipelines by combining multiple methods, each of which needs its own fine-tuning. Given that there is no clear winner in any ER workflow step, this process is quite complex even for a simple batch pipeline, especially when considering the trade-offs between recall, precision and run-time.

Challenges: All methods addressing the above scenarios should offer the following characteristics:

1) They should be self-configurable, i.e., they should be

[8] https://huggingface.co/models?pipeline_tag=text-generation

carried out automatically, without any human intervention.

2) They should offer visualizations that demonstrate the effect of every configuration parameter in any method.
3) They should support a step-wise as well as an holistic functionality. The former operates in a local manner that fine-tunes the methods in a pipeline one-by-one, starting from the first, while the latter simultaneously fine-tunes all pipeline methods at once.
4) They should consider the user requirements about performance in terms of accuracy vs prediction time.
5) They should log all experimental results similar to MLflow (https://mlflow.org) for ML pipelines. The recorded information will allow for: (i) backtracking to fix the configuration of a previous step before fine-tuning the current one, and (ii) terminating the search for optimal configuration early on, based on a continuous cost-benefit analysis.
6) They should should inherently support fairness and explainability. Existing works are crafted for batch matching algorithms [28]. We plan to incorporate it into all methods and pipelines implemented by pyJedAI, including blocking methods and progressive pipelines. A critical issue in this effort is to impose the fairness constraints as early as possible so as to increase time efficiency.
7) New benchmark datasets should be developed, given that current research is "overfitting" the existing ones [15].

At the moment, no existing approach for configuration optimization covers any of the above scenarios or requirements.

VI. Conclusions

We plan to democratize Entity Resolution in two ways: (i) through a unified framework that implements the main existing algorithms, and (ii) through the hands-off construction and configuration of complex, end-to-end pipelines that can be built by this framework. The first goal is served by pyJedAI, which provides a solid foundation – starting from zero, or any other open-source ER system would require much work for bringing it to the maturity of pyJedAI. The second goal constitutes the vision proposed in this work, i.e., the development of algorithms that automatically synthesize and fine-tune existing solutions into end-to-end pipelines of high effectiveness and time efficiency.

This vision will be pursued in the next two years, within the context of the STELAR research project[9]. After that, we plan to turn pyJedAI into a public, community-driven project that keeps pace with the latest development in the field, perhaps through the Apache Incubator[10]. To this end, pyJedAI will rely on a modular architecture, such that a new technique can be seamlessly integrated into a workflow step as long as it implements the corresponding interface. The community can thus work on any of the scenarios described in Section V, providing novel solutions that with little effort and curation can be added to pyJedAI.

The success of our effort will be measured with respect to: (i) The number of successfully supported scenarios from those described in Section V. (ii) The number of implemented techniques per scenario. (iii) The performance of auto-configuration methods in terms of effectiveness (f-measure) and time efficiency (run-time) in the real-world use cases of Section I as well as in a series of benchmark datasets. (iv) The number of downloads from PyPI. (v) The number of endorsements/stars in pyJedAI's repository.

Ackowledgements. This research was partially funded by the EU Horizon Europe GA No 101070122 (STELAR) and the Hellenic Foundation for Research and Innovation (Project Number: HFRI-FM17-2351 GeoQA, and GA No 969).

References

[1] P. Christen, *Data Matching*. Springer, 2012.

[2] X. Dong and D. Srivastava, "Big data integration," *PVLDB*, vol. 6, no. 11, pp. 1188–1189, 2013.

[3] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis, "An overview of end-to-end entity resolution for big data," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 127:1–127:42, 2021.

[4] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: Challenges and opportunities," *PVLDB*, vol. 12, no. 12, pp. 1986–1989, 2019.

[5] D. Zhang and Z. Lu, "Assessing the value of dynamic pricing in network revenue management," *INFORMS J. Comput.*, vol. 25, no. 1, 2013.

[6] S. Adikari and K. Dutta, "Adaptive ad network selection for publisher-return optimization in mobile-app advertising," *Decis. Sci.*, vol. 52, no. 4, pp. 986–1017, 2021.

[7] S. E. Whang, D. Marmaros, and H. Garcia-Molina, "Pay-as-you-go entity resolution," *TKDE*, vol. 25, no. 5, pp. 1111–1124, 2013.

[8] P. Konda *et al.*, "Magellan: Toward building entity matching management systems," *PVLDB*, vol. 9, no. 12, pp. 1197–1208, 2016.

[9] A. N. Ngomo *et al.*, "LIMES: A framework for link discovery on the semantic web," *Künstliche Intell.*, vol. 35, no. 3, pp. 413–423, 2021.

[10] J. Wang, Y. Li, W. Hirota, and E. Kandogan, "Machop: an end-to-end generalized entity matching framework," in *aiDM*, 2022, pp. 2:1–2:10.

[11] G. Papadakis *et al.*, "Three-dimensional entity resolution with jedai," *Inf. Syst.*, vol. 93, p. 101565, 2020.

[12] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *TKDE*, vol. 24, no. 9, pp. 1537–1555, 2012.

[13] M. A. Sherif, K. Dreßler, P. Smeros, and A. N. Ngomo, "Radon - rapid discovery of topological relations," in *AAAI*, 2017, pp. 175–181.

[14] G. Papadakis, G. Mandilaras, N. Mamoulis, and M. Koubarakis, "Progressive, holistic geospatial interlinking," in *WWW*, 2021, pp. 833–844.

[15] G. Papadakis, N. Kirielle, P. Christen, and T. Palpanas, "A critical re-evaluation of benchmark datasets for (deep) learning-based matching algorithms," *CoRR*, vol. abs/2307.01231, 2023.

[16] M. Papamichalopoulos *et al.*, "Three-dimensional geospatial interlinking with jedai-spatial," *CoRR*, vol. abs/2205.01905, 2022.

[17] K. Nikoletos, G. Papadakis, and M. Koubarakis, "pyjedai: a lightsaber for link discovery," in *Demo at ISWC*, 2022.

[18] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[19] A. Andoni, P. Indyk, T. Laarhoven, I. P. Razenshteyn, and L. Schmidt, "Practical and optimal LSH for angular distance," in *NIPS*, 2015.

[20] S. Thirumuruganathan *et al.*, "Deep learning for blocking in entity matching: A design space exploration," *PVLDB*, vol. 14, no. 11, pp. 2459–2472, 2021.

[21] A. Zeakis, G. Papadakis, D. Skoutas, and M. Koubarakis, "Pre-trained embeddings for entity resolution: An experimental analysis," *PVLDB*, vol. 16, no. 9, pp. 2225–2238, 2023.

[22] R. Peeters and C. Bizer, "Using chatgpt for entity matching," in *ADBIS*, vol. 1850, 2023, pp. 221–230.

[23] A. Narayan, I. Chami, L. J. Orr, and C. Ré, "Can foundation models wrangle your data?" *PVLDB*, vol. 16, no. 4, pp. 738–746, 2022.

[24] M. Mitchell *et al.*, "Model cards for model reporting," in *FAT*, 2019, pp. 220–229.

[25] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowl. Based Syst.*, vol. 212, p. 106622, 2021.

[9] https://stelar-project.eu

[10] https://incubator.apache.org

[26] T. M. Hospedales *et al.*, “Meta-learning in neural networks: A survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5149–5169, 2022.

[27] Y. Ma, T. Tran, and V. Bicer, “Typifier: Inferring the type semantics of structured data,” in *ICDE*, 2013, pp. 206–217.

[28] V. Efthymiou, K. Stefanidis, E. Pitoura, and V. Christophides, “FairER: Entity resolution with fairness constraints,” in *CIKM*, 2021.