skyex: an R Package for Entity Linkage

Suela Isaj
Aalborg University
suela@cs.aau.dk

Torben Bach Pedersen
Aalborg University
tbp@cs.aau.dk

ABSTRACT

As the data is becoming bigger, more heterogeneous, and originating from different sources, the availability of the same information in different forms leads to various entity linkage problems. We demonstrate our `skyex` package, an R package that supports all three steps of entity linkage: blocking, pairwise comparison, and labeling. Thus, the user can solve the whole process using `skyex`, but not necessarily; the `skyex` modules are independent, meaning that the user can easily integrate them with other packages or even other environments. Additionally, we are the first to provide the implementation of two skyline-based algorithms (*SkyEx-F* and *SkyEx-D*) that can label the compared pairs without the need for weights, scoring functions, etc. `skyex` supports the typical workflow of entity linkage, using minimalist, user-friendly function calls.

1 INTRODUCTION

The *entity linkage* problem, sometimes called *data matching*, *entity resolution*, *duplicate detection*, *reconciliation*, etc., detects different records that belong to the same entity. Even though the process varies in different domains, the main steps are the same: blocking, pairwise comparison, and labeling the pairs (Fig.1). The entity linkage process starts with a set of entities that might contain duplicates. First, a blocking method is used to group entities that show a certain level of similarity and are of interest to compare further. Then, the pairwise comparison step compares the entities in the same blocks, e.g., using similarity metrics of the attributes of the entities or comparing the structure of their connections. Finally, the labeling step decides whether a pair of candidates belongs to the same entity or not. The entity linkage process results in a set of labeled pairs.

We present an R package, `skyex`, that supports all three steps of the entity linkage problem. In the labeling step, we provide the novel *SkyEx-F* and *SkyEx-D* algorithms in [6, 8]. The R language is in the top five languages of data science, and even more importantly, R is the second most used software in data science scientific papers, corresponding to 50,000 articles [1]. Moreover, R is used by different industries besides academia, such as healthcare, government, insurance, etc., where entity resolution is a common obstacle [2]. The current entity linkage tools [1–4, 9–11] offer rule-based solutions with blocking and comparison functions [3, 10], crowdsourcing solutions [4, 9], or machine learning solutions [1, 2, 11]. In contrast to all the current tools, we contribute with a Labeling module that implements two novel algorithms (*SkyEx-F* and *SkyEx-D*) [6, 8], which can label the pairs without the need of weights, scoring functions, or exhaustive experiments. In order to support the full entity linkage workflow, we provide functions to perform blocking based on text and spatial attributes, and we offer a module for textual, spatial, semantic pairwise comparison. Similarly to [2], we support analysis and visualization functions that assist in the interpretation of the results and assessing the quality of the labeling. Analogously to [11] that uses Python, `skyex` uses the R ecosystem and can be easily integrated with other packages, in contrast to the current standalone tools. Finally, we demonstrate the different scenarios that can be supported by our tool using three real-world datasets. Overall, `skyex` solves the entity linkage problem with minimal effort and background knowledge.

The remainder of the paper continues with the functionalities covered by our `skyex` package in Section 2, a description of our on-site demonstration in Section 3, and finally, concluding in Section 4.

[1] http://r4stats.com/articles/popularity/
[2] https://stackoverflow.blog/2017/10/10/impressive-growth-r/

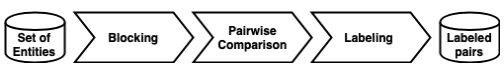

Figure 1: The entity linkage process

2 SKYEX PACKAGE FUNCTIONALITIES

The `skyex` package is composed of 17 functions corresponding to four main modules: Blocking, Pairwise Comparison, Labeling, and Analysis and Visualization. The workflow of using `skyex` is illustrated in Fig. 2. The user starts with a dataframe (a common data type for storing tables in R) of entities. In order to illustrate the workflow and our functions, we will use a real-world dataset of spatial entities extracted as in [5] and used in the experiments of [8]. The dataset contains spatial entities in the North Denmark region, originating from four sources, Google Places, Yelp, Krak (online yellow pages in Denmark, www.krak.dk), and Foursquare. We also introduce the running example of six records of entities (`entities`) from this dataset in Fig. 2, which are identified by an ID, by geographic coordinates latitude and longitude, categories that explain the type of spatial entity, and the address.

Blocking module. After loading the data, we can use a blocking technique (textual or spatial) from the Blocking module. The textual blocking is executed by the `textual.blocking` function, choosing a similarity metric among *levenshtein*, *cosine*, *jaccard*, *jaro-winker*, and *qgram*, and setting a maximal distance allowed. For example, `textual.blocking` on the attribute "name" with *levenshtein* and maximal distance 4 will group the entities with names "Bilhuset Biersted A/S" and "Bilhuset Biersted" (from `entities` in Fig. 2). Note that `textual.blocking` is accurate but time-consuming. Alternatively, `prefix.blocking` and `suffix.blocking` produce faster results. Besides, in some domains, e.g., for species names, these methods can be more relevant than textual blocking. For spatial entities, being spatially close is often a better indicator of block quality than the name. For example, two records with the same name, e.g., Fakta supermarkets in different cities, are two different entities. `spatial.blocking` creates blocks of entities that are at most *max_distance* meters apart. The code snippets for these blocking methods are as follows:

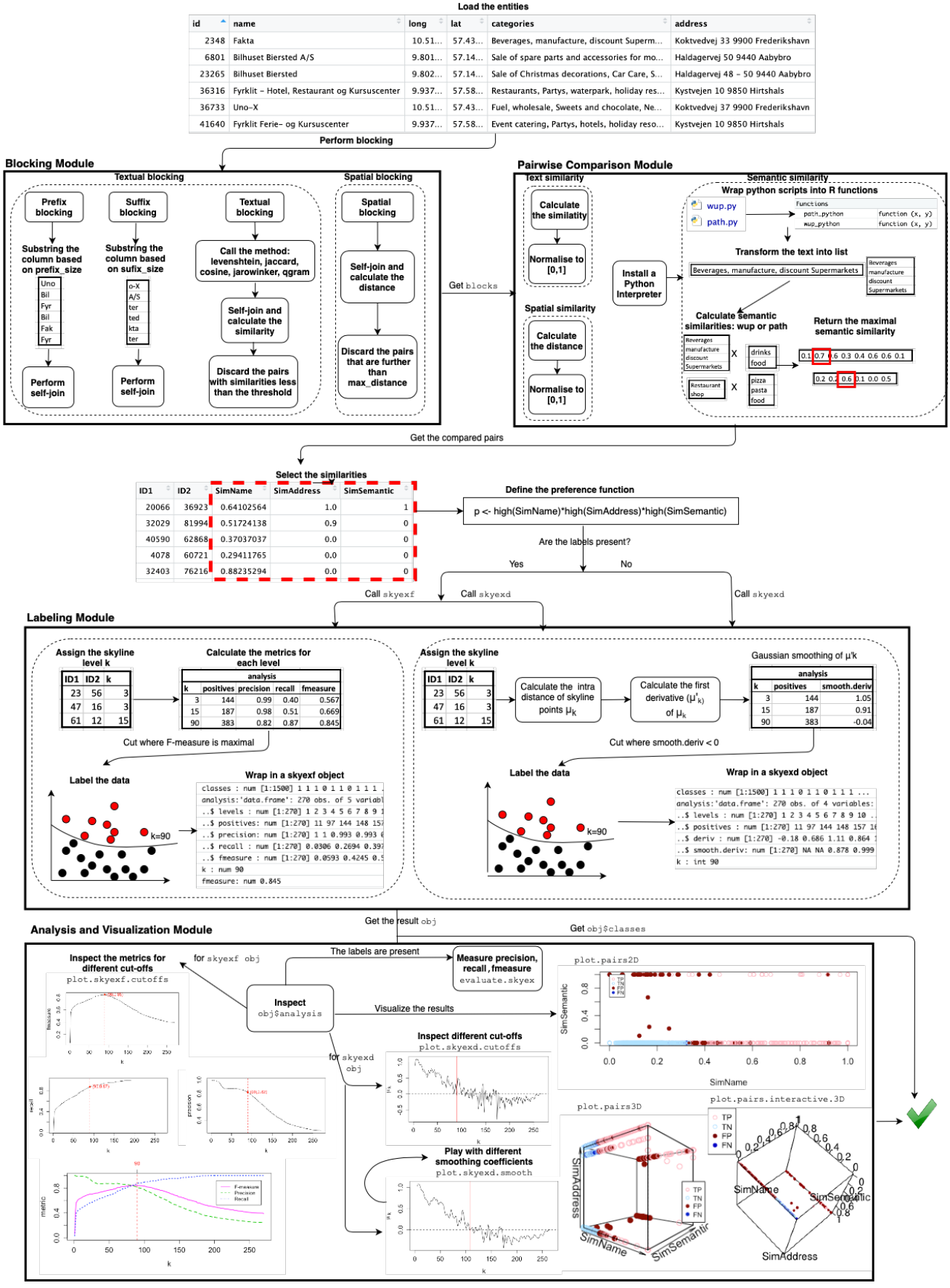

Figure 2: skyex workflow

```
#Textual blocking using levenshtein distance and max_distance=4
blocks<-textual.blocking(data=entities, column = "name",
        method = "levenshtein", max_distance = 4)
#Prefix blocking for the first 4 characters
blocks<-prefix.blocking(data=entities, column = "name", prefix_size = 4)
#Spatial blocking for entities at most 50 m apart
blocks<-spatial.blocking(data=entities, longitude = "long",
        latitude = "lat", max_distance = 50)
```

Pairwise Comparison module. The Blocking module outputs a dataframe of pairs, which saves the user from the task of having to create the pairs from each block. The Pairwise Comparison module offers three functions that compare text syntactically and semantically, as well as spatial attributes. Moreover, all three functions output normalized values, which can be directly used in the Labeling module. `text.similarity` calculates the similarity of the pairs based on a text attribute using similarity metrics such as *levenshtein*, *cosine*, *jaccard*, *jaro-winker*. *levenshtein* similarity is calculated using the formula in [7, 8] in order to return a normalized value. `spatial.similarity` also requires a maximal distance for the normalization. For example, for a *max_distance* = 70, "Uno-X" and "Fakta" will have a similarity of 0.0, because their distance of 83 meters is beyond the threshold. In the case of Bilhuset Biersted A/S and Bilhuset Biersted, this distance is 63 meters, which translates to a similarity of 0.09.

Regarding the semantic similarity, our work in [8] uses the Wu&Palmer metric from Wordnet. There exists a `wordnet` library in R, but it does not provide the metrics. Moreover, Wu&Palmer needs the whole path of both words that need to be compared, which in R, it could be resolved only through recursive calls. Through experimentation, this implementation turned out to be non-efficient. Thus, we include two Python scripts in the `skyex` package for two different metrics in Wordnet. These scripts are wrapped in R functions; thus, the user only needs to have a Python interpreter installed and give its path to the R function. The code for the pairwise comparisons is as follows:

```
#Text similarity using cosine
blocks$SimName<-text.similarity(data=blocks, method = "cosine",
               column1 = "name.x", column2 = "name.y")
#Spatial similarity with max_distance=70
blocks$SimSpatial<-spatial.similarity(data=blocks, lat1 = "lat.x",
                  long1 = "long.x", lat2 = "lat.y", long2 = "long.y",
                  max_distance = 70)
#Semantic similarity with Wu&Palmer
blocks$SimSemantic<-semantic.similarity(data=blocks,
                   column1 = "categories.x", column2 = "categories.y",
                   pythonpath = "/Users/..", method = "wup" )
```

Labeling module. After the pairs are compared, the user can decide which similarities should go into the labeling process. Usually, he would select those similarities that are likely to indicate a match, e.g., the similarity of the names of the entities. We will consider the similarity of the name "SimName", the similarity of the address "SimAddress", and the semantic similarity of the categories "SimSemantic" as in [8]. Then, the user decides on the preference function for the Pareto Optimality calculations. In our case, we prefer a high value for each similarity. Depending on the availability of the labels, the user can choose between running `skyexf` or `skyexd`, corresponding to the threshold-based *SkyEx-F*, or to the fully unsupervised *SkyEx-D*, respectively [6]. *SkyEx-F* finds that skyline level k that separates best the classes and maximizes the F-measure. It starts with assigning the skyline to all the points and then checking different cut-offs while measuring precision, recall, and f-measure. Finally, it labels the data, and the `skyexf obj` is returned, containing the classes, an analysis data frame, the proposed cut-off k, and the corresponding f-measure.

For unlabeled data, *SkyEx-D* finds the skyline level k where the mean distance of the points in the positive class starts to drop, meaning that we are entering the denser area of the negative class. It starts by assigning the corresponding skyline to each point; then, calculating the cumulative mean distance in the positive class and its first derivative; later, finding where the first derivative becomes negative for the first time. Finally, *SkyEx-D* labels the data and wraps the classes, the analysis data frame, and the proposed cut-off k in a `skyexd obj`. Detailed explanations about both algorithms can be found in [6]. Our `skyex` package hides all the details above from the user, meaning that the processes inside the dotted line boxes (Fig. 2) are performed simply by the `skyexf` and `skyexd` function calls. The script for running both algorithms, storing the results of each labeling algorithm in separate objects, and attaching the predicted classes to the dataset is as follows:

```
#Define the preference
p<-high(SimName)*high(SimSemantic)*high(SimAddress)
#Call SkyEx-F algorithm and store the result in f.obj
f.obj<-skyexf(data=blocks, p=p, label="Class",posclass=1, negclass=0)
#Call SkyEx-D algorithm and store the result in d.obj
d.obj<-skyexd(data=blocks, p=p, simlist=c("SimName", "SimSemantic",
        "SimAddress"), posclass=1, negclass=0, smooth.coefficient=5)
blocks$fpred<-f.obj$classes
blocks$dpred<-d.obj$classes
```

We thus provide a labeling procedure that can be used with only two lines of code: defining the preference and calling the labeling function. However, for a more knowledgeable user, we offer the possibility to do analysis and visualize the results through the Analysis and Visualization module.

Analysis and Visualization module. To illustrate the analysis of the labeling, we will use the 1500 manually-labeled pairs in [8]. Additionally, this dataset is also available in our package under the name `pairsManual` and can be loaded simply by `data(pairsManual)`. The Analysis and Visualization module needs the output of the Labeling module as input, which is a `skyexd` or `skyexf` object. The raw analysis can be accessed simply by calling the dataframe `analysis` from *obj* (inspect `obj$analysis` in Fig. 2). In the case of a `skyexf` object, `analysis` contains all the cut-offs, the size of the positive class, precision, recall, and f-measure. In order to facilitate the exploration of `analysis`, the user can call `plot.skyexf.cutoffs`, which produces graphs that monitor the evolution of the metrics when passing to the deeper skylines (see Fig. 2). `plot.skyexf.cutoffs` by default plots the f-measure. However, it is possible to plot the precision and the recall separately, and also all metrics together. The code snippets for plotting the f-measure (first two), the precision, the recall, and all the metrics are as follows:

```
plot.skyexf.cutoffs(f.obj)
plot.skyexf.cutoffs(f.obj, "fmeasure")
plot.skyexf.cutoffs(f.obj, "precision")
plot.skyexf.cutoffs(f.obj, "recall")
plot.skyexf.cutoffs(f.obj, "all")
```

The resulting plots from the above script on `pairsManual` are shown in Fig. 2 in the Analysis and Visualization module. Understandably, precision is high in the first skylines because it is very likely that the pairs in the first skylines that we label as positives are actual positives, but it degrades while moving in deeper cut-offs. On the contrary, recall is always increasing, the more we label as positive, the more likely it is to find an actual positive. The F-measure gives the trade-off between both metrics. All graphs show the suggested cut-off by `f.obj` in the red dotted line. However, the user can explore different trade-offs for his problem. In that case, plotting all metrics in a graph (the last script) gives a better overview.

In the case of a `skyexd` object, `analysis` keeps the cut-offs, the size of the positive class, the first derivative, and the smoothed

values. Similarly, `plot.skyexd.cutoffs` aids exploring the raw `analysis` by plotting the smoothed first derivative function for each cut-off. If the plot looks too "smoothed" or too "raw", it is possible to play with different smoothing coefficients without having to re-run `skyexd` again by calling `plot.skyexd.smooth`. (see the code below). Fig. 2 shows the analysis of `skyexd`, which was run with `smooth.coefficient=5`, and also the results of `plot.skyexd.smooth(d.obj, 10)`. The higher the smoothing coefficient, the higher the cut-off k, since smoother values push the cut-off towards deeper skylines.

```
#Plot the first derivative and the current cut-off k
plot.skyexd.cutoffs(d.obj)
#Smooth the first derivative with 10
plot.skyexd.smooth(d.obj, 10)
```

`evaluate.skyex` can also be called as in the code below, to measure precision, recall, and f-measure when the labels are available. The values of these metrics will be printed in the console.

```
evaluate.skyex(prediction=d.obj$classes, labels=data$Class, posclass = 1)
```

Additionally, we offer user-friendly functions to plot the data and the *obj* results. We offer 2D plots, 3D plots, and interactive 3D plots, where the user can play and move the dimensions while looking at the data. The color of the points reflects if the pair is a true positive TP (an actual positive labeled as positive), a true negative TN (an actual negative labeled as negative), a false positive FP (an actual negative labeled as positive), and a false negative FN (an actual positive labeled as negative). The user can decide to change the colors of the points based on his preference. The code for these plots is as follows:

```
#Plot 2D using SimName and SimSemantic
plot.pairs2D(data=data, sim1="SimName", sim2="SimSemantic",
            prediction=f.obj$classes, labels=data$Class, posclass = 1)
#Plot 3D using SimName, SimSemantic, and SimAddress
plot.pairs3D(data=data, sim1="SimName", sim2="SimSemantic",sim3="SimAddress",
            prediction=f.obj$classes, labels=data$Class, posclass = 1)
#Plot 3D interactive plot using SimName, SimSemantic, and SimAddress
plot.pairs.interactive.3D(data=data, sim1="SimName", sim2="SimSemantic",
            sim3="SimAddress", prediction=f.obj$classes,
            labels=data$Class, posclass = 1)
```

Fig. 2 shows the results of `pairsManual` with the three types of plots. These graphs can also be considered as an analysis since they show the problems with labeling and where to locate them. For example, it is noticeable that we have a bigger problem with the false positives then with the false negatives, thus if precision is fundamental to the domain, we could go back to the analysis and evaluation module and consider a smaller k for the cut-off. The interactive 3D plot offers a better view of the data points since it is possible to move and rotate the graph.

Summary. The workflow of `skyex` supports typical entity linkage tasks, from blocking to evaluating the quality of the labels. The Blocking, Pairwise Comparison, and Labeling modules are completely independent, which means that the user can decide to perform his own methods and still be able to connect to the workflow of `skyex`. The labeling task can be as simple as just calling two lines of code to get the classes and as detailed as performing analysis, playing with the parameters, visualizing the labels, and highlighting the errors, etc. Moreover, the user can always go back, choosing new similarities and new preferences until the results are satisfactory. The `skyex` package is dependent on `rPref`, `dplyr`, `fields`, `rgl`, `plot3D`, `smoother`, `fuzzyjoin`, `stringr`, `stringdist`, `geosphere`, `reticulate`, and `pracma` which support some basic functionalities in our functions. `skyexf` and `skyexd`) scale relatively well for an R environment; e.g. they run in less than a minute for 50,000 pairs, less than 15 minutes for 150,000 pairs, and around 1 hour for 300,000 pairs.

3 DEMONSTRATION OVERVIEW

In the on-site demonstration, the user can download `skyex`[3], which is publicly available in GitHub, by following the README instructions, or use our pre-installed R environment. We will provide three datasets: `entities` (2814 spatial entities in the North Denmark region with an ID, name, categories, and address) [8], `restaurants`[4] (a collection of 864 restaurant records with name, address, city, and type), and `pairsManual` (1500 labeled pairs with pre-compared similarities of the name, address, and categories) [8]. Additionally, we have published a full video[5] demonstrating our functionalities for all three datasets, and a short video[6] for the `restaurants` dataset. We will provide example scripts, which the user can adapt based on his preference. The user will start with different blocking techniques on `entities` and `restaurants`, discussing with us what would be a good blocking technique for this dataset. Afterwards, he can play with different similarity metrics and different thresholds for the pairwise comparison. Later, the user can decide either to continue with the dataset of pairs he created so far from `entities` and `restaurants`, or move to the pre-compared `pairsManual` and play with the labeling parameters. The user can try both algorithms and will be guided by us through the Analysis and Visualization module. He can try the visualizations (including the interactive plotting) in order to detect problems with the labeling. Finally, he can discuss with us the applicability of the method across domains and possibilities for improvement.

4 CONCLUSIONS AND FUTURE WORK

We introduced the `skyex` package, a user-friendly R package that supports all three steps of the entity linkage process. We demonstrated the functions of `skyex` with scripts and sample data, and supported the full workflow of the user. We showed that our Labeling module could solve the labeling problem with only two lines of code, but at the same time, offer the possibility for deeper analysis for the knowledgeable user. As future work, we intend to work on the scalability of our tool for big data, as well as on a similar package in Python.

REFERENCES

[1] E. C. Dragut et al. 2016. ORLF: A flexible framework for online record linkage and fusion. In *ICDE*. 1378–1381.
[2] A. Ebaid et al. 2019. EXPLAINER: Entity Resolution Explanations. In *ICDE*. 2000–2003.
[3] A. Elmagarmid et al. 2014. NADEEF/ER: Generic and interactive entity resolution. In *SIGMOD*. 1071–1074.
[4] Y. Govind et al. 2018. Cloudmatcher: a hands-off cloud/crowd service for entity matching. *PVLDB* 11, 12 (2018), 2042–2045.
[5] S. Isaj and T. B. Pedersen. 2019. Seed-Driven Geo-Social Data Extraction. In *SSTD*. 11–20.
[6] S. Isaj, T. B. Pedersen, and E. Zimányi. 2019. Multi-Source Spatial Entity Linkage. arXiv:1911.09016
[7] S. Isaj, N. B. Seghouani, and G. Quercini. 2019. Profile Reconciliation Through Dynamic Activities Across Social Networks. In *CAiSE*. 126–141.
[8] S. Isaj, E. Zimányi, and T. B Pedersen. 2019. Multi-Source Spatial Entity Linkage. In *SSTD*. 1–10.
[9] X. Ke et al. 2018. A demonstration of PERC: probabilistic entity resolution with crowd errors. *PVLDB* 11, 12 (2018), 1922–1925.
[10] L. Kolb, A. Thor, and E. Rahm. 2012. Dedoop: Efficient deduplication with hadoop. *PVLDB* 5, 12 (2012), 1878–1881.
[11] P. Konda et al. 2016. Magellan: toward building entity matching management systems over data science stacks. *PVLDB* 9, 13 (2016), 1581–1584.

[3] https://github.com/suelai/skyex
[4] source: https://www.cs.utexas.edu/users/ml/riddle/data.html
[5] https://youtu.be/TdxVsUtKRjw
[6] https://youtu.be/Zn8FOOh_xwA