Expert Systems With Applications 147 (2020) 113115

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Swash: A collective personal name matching framework

Mohsen Raeesi[a], Masoud Asadpour[a,*], Azadeh Shakery[a,b]

[a] School of ECE, College of Engineering, University of Tehran, Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

A R T I C L E I N F O

Article history:
Received 14 April 2019
Revised 15 October 2019
Accepted 30 November 2019
Available online 2 December 2019

Keywords:
Personal name matching
Entity matching
Collective matching
Entity resolution
Heterogeneous information network
Unsupervised learning

A B S T R A C T

Having a unique personal identifier is a prerequisite to run person-centric analytical queries and data mining tasks, such as fraud detection, expert finding, and credit scoring. Personal names are the most commonly used identifier of individuals in datasets; however, the name of a person may not be unique across the dataset's records, especially where data are integrated from various sources. Intelligent systems utilize name matching methods to identify different name representations of persons. The performance of previous name matching methods is inadequate since they solely consider name similarities and ignore dissimilarities. Unavailability of Part of Name (PON, e.g., first name and last name) is an important limitation of dissimilarity consideration. To address this issue, this paper proposes an unsupervised personal name matching framework, namely Swash. This framework can model the information gatherable from a name dataset into a layered Heterogeneous Information Network, which facilitates control over the learning process. Swash predicts PON tags using a self-trainable algorithm and then collectively clusters the name vertices on the network. Evaluations on three public bibliographic datasets (i.e., CiteSeer, ArXiv, and DBLP) recognize the significance of the proposed framework. The results showed that Swash outperformed F1 of the state-of-the-art method up to 38%.

1. Introduction

Information retrieval and data mining are two key technologies of modern expert systems. These technologies require to know the underlying entities of datasets; however, the entities are often unavailable or noisy, especially when data is gathered from different sources. As an example, to select the best researcher in a country, all records of researchers should be aggregated, but there is no unique identifier on each researcher's records. Entity matching tries to find this identifier using available information of references. Entity matching resolves references in two levels, i.e., field matching and record matching. Field matching identifies different representations of available fields separately, and record matching aggregates the results of field matching to resolve entities. Names are often the most informative field of records for matching. Personal name matching identifies name representations of the same person only using the person name field. This paper suggests a comprehensive framework for personal name matching.

Personal name matching was initially studied as a string matching problem (Christen, 2006). After 2005, some research studies particularly focused on personal names. There are several different topics in this research, including utilizing name-specific patterns to increase matching performance (Gong, Wang, & Oard, 2009; Sukharev, Zhukov, & Popescul, 2014; Treeratpituk & Giles, 2012) and matching extension for other languages (Peng, Yu, & Dredze, 2015; Shaalan & Raza, 2007). Despite advances in name matching, the performance of previous research studies is still inadequate. According to our investigations, considering solely name similarities and ignoring dissimilarities is the main reason for this issue. For instance, two very similar names cannot be matched while their abbreviated middle names are dissimilar. To consider the dissimilarity, part of name (PON)[1] tags of names are required; however, these tags are absent in name datasets.

We model personal name matching problem using a layered heterogeneous information network (HIN)[2], which embeds infor-

* Corresponding author.
E-mail addresses: mohsen.raeesi@ut.ac.ir (M. Raeesi), asadpour@ut.ac.ir (M. Asadpour), shakery@ut.ac.ir (A. Shakery).

[1] PON: Part Of Name (analogous to Part Of Speech in NLP) refers to types of name tokens, including prefix, first name, middle name, last name, and suffix.

[2] HIN: Heterogeneous Information Network is a network that its vertices and edges have different types (Shi, Li, Zhang, Sun, & Yu, 2017).

https://doi.org/10.1016/j.eswa.2019.113115
0957-4174/© 2019 Elsevier Ltd. All rights reserved.

mation gatherable from a name dataset. In this network, each token of names is tagged with an initial PON. At first, a blocking scheme finds possible reference candidates using a message-passing algorithm. The blocking reduces the quadratic number of further comparisons to near-linear time. Afterward, an unsupervised scheme corrects the initial tagging mistakes using a self-trainable algorithm. This algorithm trains from confident tags and extends its knowledge to unconfident tags iteratively. The PON tags are used in a collective matching algorithm, which considers dissimilarity besides similarity. This matching algorithm collectively clusters reference vertices that belong to the same entities. We named the suggested solution 'Swash'[3] due to the resemblance between the matching process and waves breaking on beaches. According to our evaluations on three publicly available datasets, Swash significantly outperforms the state-of-the-art name matching methods.

The proposed approach, analogous to other name matching methods, is only responsible for the identification problem. Identification and disambiguation are two different problems that entity matching confronts (Bhattacharya & Getoor, 2007). The identification problem refers to identifying different representations of an entity, while the disambiguation problem tries to distinguish between different entities with the same or similar representations. Disambiguation is impossible in name matching due to the necessity of other fields' information. For instance, only based on the name field, we cannot recognize whether two 'John Doe' references refer to the same person or not?

The proposed solution outperforms state-of-the-art performance, particularly on more difficult datasets. Moreover, It makes several noteworthy contributions to previous methods. First, the approach is unsupervised and does not require gold data. Second, it introduces a self-trainable mechanism, which gradually predicts unconfident PON tags according to confident tags. The third contribution is the collectivity of matching decisions, which are dependent on each other. Finally, instead of reference similarity, the consistency concept is defined. The consistency considers the dissimilarity in addition to similarity.

The rest of this paper is structured as follows. In Section 2, the previous research on name matching is reviewed. Section 3 categorizes the typical variations in personal names and discusses the issues of the previous methods against matching these variations. The discussion leads to a comprehensive framework, which is proposed in section 4. The proposed framework is evaluated using four experiments. Section 5 and 6 describe the experiments' settings and discuss their results, respectively. Finally, Section 7 concludes the paper and reports several possible directions for future works.

2. Literature review

Personal name matching was initially studied as a string matching problem (Christen, 2006). The string matching measures can be divided into three types, i.e., *sequence-based, set-based*, and *phonetic hashing* (Doan, Halevy, & Ives, 2012). Sequence-based measures, such as edit-distance, compute the cost of transforming a string to another one (Levenshtein, 1966). Set-based measures transform strings to a multi-set of tokens and compute the similarity between them using set similarity indices such as Jaccard index (Cohen, Ravikumar, & Fienberg, 2003). Contrary to the mentioned measures, phonetic hashing is not pairwise. It hashes strings to several buckets such that strings in the same bucket are similar. Soundex is the most known phonetic hashing algorithm (Odell & Russell, 1918). These three types can be considered as the first generation of name matching methods. Extending this generation has continued so far. As an example, Christin et al. improved edit-distance measure utilizing contextual information such as string frequency and related strings (Christen & Gayler, 2015).

The next matching generation is *Hybrid matching* that combines sequence-based and set-based measures in order to consider structure and token variations simultaneously. Hybrid string matching greedily finds the most similar token pairs using a sequence-based measure and then computes the set-based similarity between aligned tokens. Monge and Elkan presented a 'recursive matching scheme', which is the most known hybrid matching algorithm (Monge & Elkan, 1996). Cohen et al. enhanced Monge-Elkan scheme using the TF-IDF weighting of tokens (Cohen et al., 2003).

The matching research that focused on personal names was mainly introduced after 2005. Most research has addressed side issues, including benchmarking previous methods (Gali, Mariescu-Istodor, & Fränti, 2016; Peng, Li, & Kennedy, 2012; Christen, 2006), matching of non-English names (Zhagorina, Braslavski, & Gusev, 2018; Peng, Yu, & Dredze, 2015; Shaalan & Raza, 2007), cross-language matching (Medhat, Hassan, & Salama, 2015), and ranking the alternate spellings of names (Sukharev et al., 2014; Varol & Bayrak, 2012). The core of name matching has targeted by little research, which will be reviewed in the next paragraph.

The third generation of name matching has been tackled the greediness of Hybrid matching. Hybrid matching aligns similar tokens using a greedy algorithm; hence, there is no guaranty to achieve optimal alignment. The third generation of name matching tries to find the optimal tokens alignment. We name it '*Maximum-similarity Alignment*' *matching*. To avoid greedy choice, considering some of thename structure variations was the first solution (Galvez & Moya-Anegón, 2007). This solution converts all valid variations of names to a canonical form using a finite-state graph. Having equal canonical form shows the equality of underlying entities. Gong et al. have suggested the state-of-the-art name matching method, which significantly outperforms hybrid matching (Gong et al., 2009). This method generates a graph of all valid transformations between two names and finds the optimal transformation using a shortest-path algorithm. There is a similar version of this method, which utilizes dynamic programming for optimization (Treeratpituk & Giles, 2012). The later research could not make a significant contribution in *Maximum-similarity Alignment* and only mashes up the previous methods, such as (Ash, 2017).

The proposed framework is the fourth generation of name matching. The previous methods only use the similarity between the forms of tokens; however, to improve performance, more in-depth knowledge of name structure is required. The proposed framework finds the structure of names using a self-trainable parser and matches them according to their structures. The framework will be detailed in the following sections.

3. Problem definition: personal name variations

A full personal name consists of several tokens that are arranged in a sequential structure. Each token has a specific PON (Part Of Name) tag, including prefix (pf), first name (fn), middle name (mn), last name (ln), and suffix (sf). There are two sources of variations in name matching: token variations and structure variations. Token variations refer to changes in the form of names, and structure variations are changes in the organization of name tokens. Several previous works have tried to categorize the types of person name errors (Arehart & Miller, 2008; Galvez & Moya-Anegón, 2007). Improving these works, a comprehensive and refined categorization of name variations is presented, as follows:

[3] Swash: the water that washes up onshore after an incoming wave has broken. Swash consists of two passes: uprush (onshore flow) and backwash (offshore flow).

Table 1
Robustness of previous name matching approaches to each type of name variations (only a single variation). Double ticks and single tick correspondingly represent total and partial robustness.

Name variations	Seq-based	Set-based	Phonetic hash	Hybrid	Max-Similarity Alignment
1.1. Data entry errors	√√	√√	√	√√	√√
1.2. Short forms					√
1.3. Spelling variations	√√	√√	√√	√√	√√
1.4. Nicknames and diminutives					
2.1. Deletion of optional parts					√
2.2. Order of tokens		√√		√√	√√
2.3. Insertion of extra delimiters	√√	√			
2.4. Deletion of valid delimiters	√√	√			

Table 2
Name pairs with multiple simultaneous variations from ArXiv bibliographic dataset.

Case 1	Case 2	Case 3	Case 4
c.linhares de jesus	jm figueroa-ofarrill	m.b.d.silva maia porto	christopher m.hull
c.d.linhares	jose figueroa-ofarrill	m.b.d.s.m.porto	chris hull

1 Token variations

1.1 Data entry errors such as typos and OCR errors (mostly non-word errors), e.g., *Ali* → *Alj*

1.2 Short forms

1.2.1 Abbreviations, e.g., *Muhammad* → *Mhd*

1.2.2 Initials, e.g., *John* → *J*

1.3 Spelling variations

1.3.1 Alternate spellings, e.g., *Jennifer* → *Jenifer*

1.3.2 Transliteration, e.g., *Saeed* → *Saeid*

1.4 Nicknames and diminutives, e.g., *Robert* → *Bob*

2 Structure variations

2.1 Deletions of optional parts (such as middle name), e.g., *John Charles Smith* → *John Smith*

2.2 Order of tokens, e.g., *Zhong Yushu* → *Yushu Zhong*

2.3 Insertion of extra delimiters, e.g., *A. Rodriguez* → *A. Rodr i guez*

2.4 Deletion of valid delimiters, e.g., *C. Korthals altes* → *C. Korthalsaltes*

A perfect name matching method should be robust to all types of the above variations. However, the previous methods cannot identify most of these variations. Table 1 compares the robustness of previous approaches to these name variations. According to Table 1, most of the approaches identify ‘data entry’ and ‘spelling variations’ of tokens when the variations do not dramatically change the token. There is no consensus among approaches on the detection of other variations. For example, while the sequence-based approach is sensitive to the ‘order of tokens’, it is relatively robust to delimiter errors. Almost none of the approaches can adequately cover variations of ‘short forms’, ‘nicknames’, and ‘optional part deletion’. Only the *Maximum-similarity Alignment* approach (3rd generation of name matching) can partially identify ‘short forms’ and ‘optional part deletion’ in some circumstances using contextual information.

Table 1 compares the robustness of different approaches, where names have a single type of variation. However, in real-world data, we encounter cases that names have multiple variations simultaneously, as illustrated in Table 2. In these cases, none of the previous approaches can effectively identify the variations. Consider Case 1 of Table 2, which includes two different variations, i.e., insertion of the middle name and deletions of suffixes. Co-occurring these two variations leads to a significant similarity decrease between the name pairs, such that even the *Maximum-similarity Alignment* approach cannot identify them.

In contrast to the previous matching approaches, people can identify that the name pairs of Table 2 refer to the same persons. However, why the human brain can identify and the previous approaches cannot? The main features of brain matching process can be categorized as follows:

1. **Collective approach**: When we decide on a name pair, other similar names in the corpus assist us in making a correct decision. For instance, the abundance of candidates, where ‘Christopher’ substitutes with ‘Chris’, increases the matching probability of Case 4 pairs. Moreover, if we have prior knowledge that ‘Chris’ is a nickname of ‘Christopher’, this can also result in matching this pair. Among the previous approaches, only *Maximum-similarity Alignment* approach tries to utilize contextual information; however, even this approach ignores a significant part of the information gatherable from a name dataset.
2. **Iterative self-training**: Our matching decisions are interrelated with each other. Every single high-confident match can provide pieces of evidence to match other names. In other words, every decision updates the current knowledge, and the updated knowledge affects future decisions. As an example, when ‘Carlo Linares’ is matched with ‘C. Linares De Jesus’, we find out that ‘Linares’ is the last name, and ‘De Jesus’ is the suffix. Based on this knowledge, ‘C. Linares De Jesus’ will be matched with ‘C. Jesus’ afterward.
3. **No need to label**: We can do name matching for the cultures that are unknown to us using limited facts on names and the information inferred from the whole dataset. Therefore, the initial labeled data or PON tags are unnecessary for self-training, and a limited number of simple facts are sufficient. As an example, in case 1, without knowing the meaning of ‘de’ in Spanish culture, we can guess it’s PON based on token alignment.
4. **More weight on less common tokens**: Co-occurrence of an uncommon token increases the similarity between two names more than a common token. It is IDF weighting heuristic, which is utilized in our daily matching decisions. For the first time, Cohen introduced TF-IDF heuristic in name matching literature (W. W. Cohen, 2000) and followed by other researchers later. However, TF-IDF weighting has two significant shortcomings. First, TF weighting is not meaningful in name matching, due to the fact that duplication of a co-occurred token does not increase the similarity of name pairs. As an example, the similarity of ‘J. Doe’ to itself is more than to ‘J. J. Doe’. The second shortcoming refers to the necessity for labeled data. Previous approaches require labeled data to estimate the weights, while people can estimate the commonness of tokens during the matching of unlabeled datasets.
5. **Considering dissimilarity in addition to similarity**: It is not enough to decide only based on the similarity of names. For instance, despite the high similarity of ‘Tomson, John M.’ and ‘Tomson, John A.’, they cannot be matched due to the dissimilarity of their middle names.

6. **Name parsing**: Parsing the name structure is a crucial prerequisite of name matching, especially to know optional parts that can be deleted (name variation of 2.1). Case 2 of Table 2 illustrates this point. If the PON tag of 'm' is middle name, its deletion in 'Jose Figueroa-Farrell' is valid. However, if 'm' is tagged to first name, the name pair cannot be matched due to the dissimilarity of their first names.

A proper matching should embed the above features in an integrated process. This paper proposes a comprehensive name matching framework, which includes the above features. This framework is quite robust to all name variations of Table 1, except the last two variations. The robustness enhancement of the framework to cover these two variations is possible; however, it is out of the aim of this paper, and we leave it for future research. A prominent advantage of the proposed framework over the previous methods is its robustness to multiple variations co-occurrence due to its intelligent perception of name structure.

4. Proposed approach

To address the discussed issues, we propose a personal name matching framework, namely Swash. This framework starts with confident matches and acquires knowledge based on this matching. Then, the acquired knowledge is utilized in further matches. It continues until the whole names are gradually matched. Swash framework splits the matching of token's variations and structure's variations into different modules that work together in an integrated process. Different kinds of similarity measures or prior knowledge are pluggable to these modules. By extending these two modules, we can generalize name matching to other types of data such as addresses, phone numbers, and even genome. Generally, the framework can potentially match any data, which includes a sequence of tokens.

As presented in Fig. 1, the proposed framework consists of two major phases: data modeling and matching. The names data is transformed into a rich HIN during the first phase. The matching phase exploits this HIN for measuring meta-path similarity and clustering. This phase alternates between the blocking stage, which finds possible candidate names, and clustering stage, which clusters the candidates. The remaining parts of this section details the data modeling phase, the blocking stage, and the clustering stage subsequently.

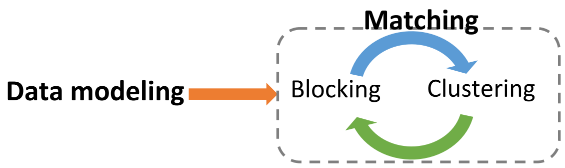


Fig. 1. The general process of the proposed matching framework.

4.1. Data modeling

A comprehensive name matching framework requires a comprehensive data model. This model would collect all available information in a name dataset. There are multiple items of information in the name dataset, including:

1. Tokens of a name, e.g., John Philip Doe → John, Philip, Doe
2. The sequence of token pairs, e.g., John →Philip, Philip →Doe
3. The frequency of names, e.g., John Philip Doe:5
4. The frequency of tokens, e.g., John: 153, Philip: 43, Doe: 5
5. The frequency of consecutive tokens' pairs, e.g., John →Philip: 12, Philip →Doe: 5
6. The similarity between tokens, e.g., John ~ Johnny

The data modeling phase collects the above items and embeds them into a rich HIN. Several steps should be taken to construct the HIN, as illustrated in Fig. 2. First, names are standardized to the same format. Afterward, standardized names are tokenized. If data includes 'deletion of valid delimiters' variation, applying a word breaking algorithm becomes necessary. This algorithm breaks words at proper boundaries in the absence of word delimiter (Wang, Thrasher, & Hsu, 2011). In the third step, names, tokens, and consecutive tokens' pairs are aggregated separately, and the frequency of each unique value is calculated. Finally, a layered HIN is constructed based on the results of previous steps.

The network schema of Fig. 3 sketches the proposed HIN. The types of vertices and edges are detailed in Table 3 and Table 4, respectively. This layered network consists of three main segments, i.e., references, clusters, and elements. The references segment (layer 0, REF vertices) contains unique name references that contain in the input data. These references should be clustered to identify their underlying entities. The clusters segment consists of the vertices of layer −1 (CLS and RID types), which points to the

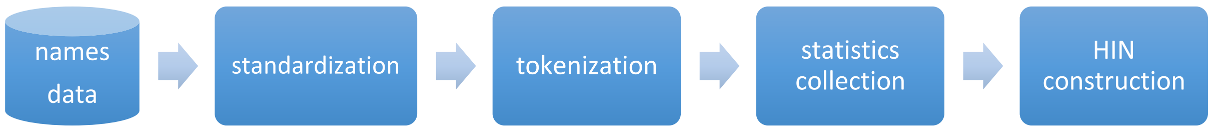


Fig. 2. The steps of the data modeling phase.

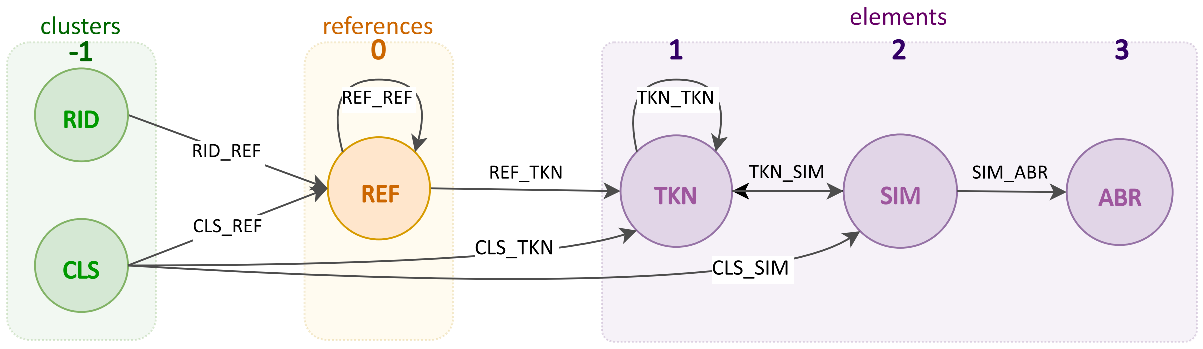


Fig. 3. Network schema of the proposed layered HIN.

Table 3
Types of vertices in the proposed layered HIN.

Label	Layer	Description	Segment	Properties
CLS	−1	Identified clusters	cluster	Frequency
RID	−1	Resolved IDs (if gold standard exists)	cluster	Frequency
REF	0	References of names	reference	Frequency, value
TKN	1	Tokens of names' references	element	Frequency, value, no. of clusters
SIM	2	The representative of similar tokens such as the same phonetic hash code or nickname	element	Frequency, value, no. of clusters
ABR	3	Abbreviated of tokens (the first character)	element	Frequency, value, no. of clusters

Table 4
Types of edges in the proposed layered HIN.

Label	Out-l	In-l	Description	Properties
RID_REF	−1	0	Reference names belonging to the same person	
CLS_REF	−1	0	Reference names in the same cluster	
CLS_TKN CLS_SIM	−1	1 2	Cluster profile that presents the names' elements of the cluster representative	Token's order, PON tag (prefix, first name, etc.)
REF_REF	0	0	The similarity between reference pair according to blocking the result	similarity
REF_TKN	0	1	Tokens of the reference name	Token's order, PON tag
TKN_TKN	1	1	Consecutive tokens' pairs	Frequency
TKN_SIM	1	2	Similarity representative of the token vertex	
SIM_ABR	2	3	Abbreviated form of the SIM vertex	

same entity references of layer 0. The positive layers of the proposed network (layer 1 to 3: TKN, SIM, and ABR) form elements segment, which is used to compute the similarity between references. These layers contain the tokens of references and the more abstract forms of these tokens. Applying a similarity hash-based function (like Soundex) on layer 1 forms layer 2, and abbreviating (first character) layer 2 forms layer 3. Every similarity hash-based function, which increases abstraction, can be utilized for layer 2, such as phonetic coding, nicknames prior knowledge, and token's n-grams.

The layered structure of the elements segment facilitates control over the learning process. Unsupervised learning algorithms direct the learning process to escape local optimum. Swash escapes local optimums by ordering candidates based on confidence. It starts learning from confident matches and extends its knowledge to unconfident matches. The upper layers are more abstract and, consequently, less confident. As a result, if two names have a common element in an upper layer, they are less probable to be matched. Moreover, this layered structure benefits the collectivity of reference similarity computation. To compute the similarity between names A and B via a common token t, other names containing t are accessible. Accessing these names is utilized to estimate token commonness (IDF heuristic mentioned in Section 3), which will be described in Section 4.2.

There are two types of vertices in layer -1, i.e., RID and CLS. RID refers to the gold resolved id of references. CLS is our cluster prediction from RID, which is generated dynamically during matching. The more equality between predicted clusters (CLS) and real clusters (RID) results in more clustering performance. Contrary to RID, a CLS vertex has a cluster profile (i.e., CLS_TKN and CLS_SIM edges). The cluster profile is a cluster belief on the name's parts. This profile is generated during the matching process and is updated gradually.

Fig. 4 illustrates a simple initial network instance of the proposed HIN. This network is constructed from a toy dataset of 32 names that has four unique references. These references belong to two different real entities. CLS_TKN and CLS_SIM edges are absent in this network to avoid more complexity of the figure. Moreover, it has no REF_REF edges because these edges are generated during matching, while the network is initial. The REF_TKN edges of the network are tagged with the initial PONs. Initial tagging considers the last non-abbreviated token as the last name and the first token of the remaining tokens as its first name. The remaining tokens before and after the last name are tagged as middle name and suffix, respectively. During the matching phase, the probable errors of initial tagging are corrected using the confident tags of similar references.

4.2. Blocking

Due to the quadratic complexity of matching each reference to the others, the blocking stage attempts to reduce the number of candidate pairs using a simple near-linear scheme. These candidate pairs will be accurately matched later in the clustering stage. The references' similarity, as a necessary condition for matching, is used for blocking. The proposed blocking scheme calculates the similarity between references via their common elements (TKN, SIM, and ABR) and chooses high-similar references as possible candidates. The blocking scheme includes four steps, as described in Fig. 5. First, a forward message-passing is performed to update the 'number of clusters' properties of element vertices. Second, a forward-backward message-passing algorithm finds the first common elements between references. In this step, each reference receives messages from other references that are common in an element vertex. In the third step, the messages from each sender are collected, and the aggregated similarity statistics are computed. Finally, similar references are chosen according to these statistics, and corresponding weighted REF_REF edges are added to the network. Consequently, the blocking stage constructs a homogenous similarity network between REF vertices, which represents similar references to each reference.

Token commonness is a crucial parameter in similarity computation. The co-occurrence of an uncommon token increases the similarity between two names more than a common token (IDF heuristic mentioned in Section 3). Thus, the number of persons having a token is a suitable index to measure token commonness. This index is embedded in the proposed HIN as a property of element vertices, i.e., 'number of clusters'. As the proposed framework is unsupervised, the current prediction of clusters, instead of real clusters, is used for commonness estimation. The 'number of clusters' is the sum of unique clusters in the preceding vertices, as follows:

$$number\ of\ clusters(e) = |cluster\ vertices\ in\ the\ preceding\ of\ e|$$

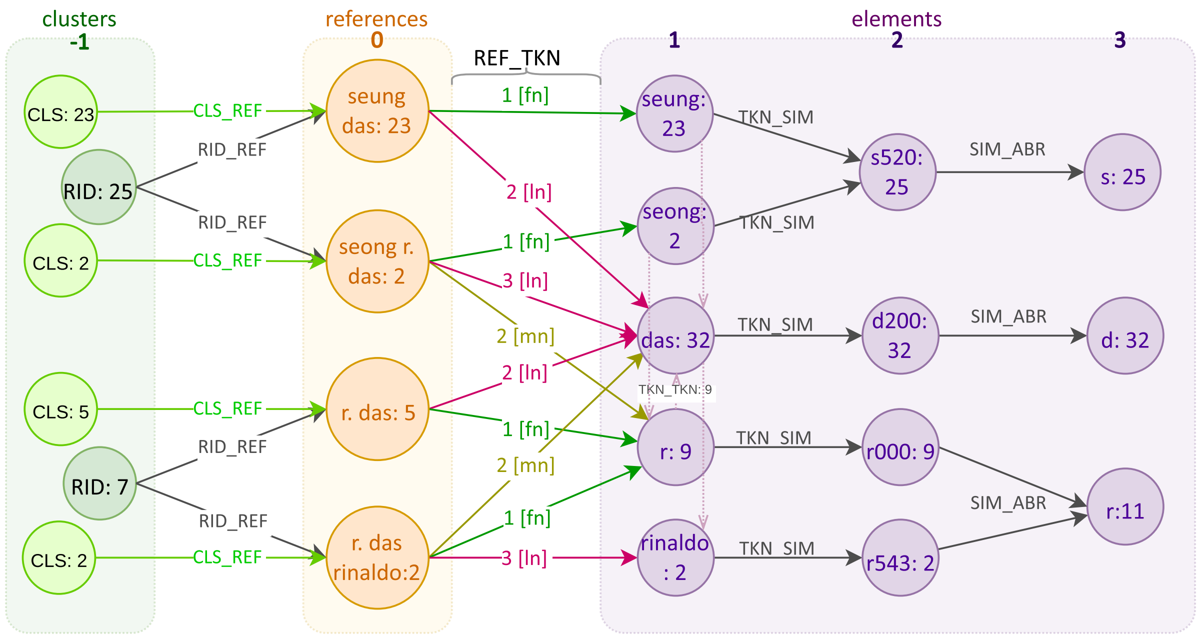


Fig. 4. An initial network instance of the proposed layered HIN. The numbers in vertices refer to their frequencies.

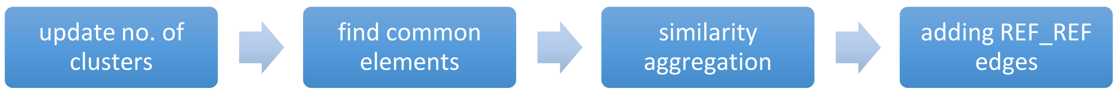


Fig. 5. The steps of the blocking stage.

(1)

where e is an element vertex. For example, in the network instance of Fig. 4, ‘seong’ vertex has one cluster, and ‘s520’ has two clusters. However, the clustering stage merges the cluster of ‘seung das’ and ‘seong r. das’ and accordingly, the ‘number of clusters’ property of ‘s520’ vertex becomes one.

The proposed blocking scheme computes the similarity between references via finding their common element vertices. Having a common element vertex in layer 1 (TKN) means equal token co-occurrence, in layer 2 (SIM) shows token similarity, and in layer 3 (ABR) represents token abbreviation. To find common element vertices, each reference sends out messages to its tokens. These messages are spread out to the succeeding layers up to a particular max layer and then return to layer 0. In other words, this message-passing has two different passes, i.e., a forward pass from REFs to a particular element layer and a backward pass from this layer to the REFs. These two passes are analogous to the uprush and backwash of a sea swash; therefore, we name the proposed framework to Swash. Max layer parameter can be changed from 1 to 3. For instance, three different meta-paths can be defined for the HIN of Fig. 4: REF_TKN_REF, REF_TKN_SIM_TKN_REF, and REF_TKN_SIM_ABR_SIM_TKN_REF. A message m has a similarity property, which is initially set to 1.0. When m reaches an element vertex e, the similarity value is updated as follows:

$$similarity^{new}(m) = \frac{similarity(m)}{number\ of\ clusters(e)} \tag{2}$$

As an example, consider the REF_TKN_REF message-passing from ‘seong das’ vertex in Fig. 4. The passed messages to ‘seong’ and ‘das’ obtain different similarity values, i.e., 1.0 and 0.25, respectively.

Received messages to a REF vertex indicate the similarity of senders to the receiver via common element vertices. Each REF vertex computes two aggregated measures per senders: the count of received messages and the sum of messages similarity. The aggregated similarity of messages received to r from a sender REF vertex s is computed as:

$$abs\_sim(r,s) = \sum_{m \epsilon m_{r,s}} similarity(m) \tag{3}$$

where $m_{r,s}$ is all messages received to r from s. According to aggregated similarity measure, adding REF_REF edges should be decided, but the similarity measure suffers from a severe inefficiency. The similarity values are not comparable to each other due to absoluteness. For instance, the similarity of an infrequent name, such as ‘Lee Janet Cash’, to itself is several times higher than a common name, such as ‘John Smith’. To tackle this issue, the relative similarity measure is defined:

$$rel\_sim(r,s) = \frac{abs\_sim(r,s)}{abs\_sim(r,r)} \tag{4}$$

where r and s are REF vertices, which respectively receive and send a message. Relative similarity normalizes the absolute similarity using self-similarity ($abs\_sim(r,r)$), which is the similarity of a sender to itself. Comparability of similarity values is the main advantage of relative similarity. At the final step, adding REF_REF weighted edges is decided by a simple conjunctive or disjunctive rule on message count and relative similarity. The threshold parameters of the rule should be set based on data quality. Fig. 6 illustrates REF_REF edges, which are added to the HIN after the blocking of the ArXiv dataset.

The proposed framework matches names in an iterative self-training process, as mentioned earlier. Thus, the proposed blocking starts from high-precision blocks to learn parameters correctly and

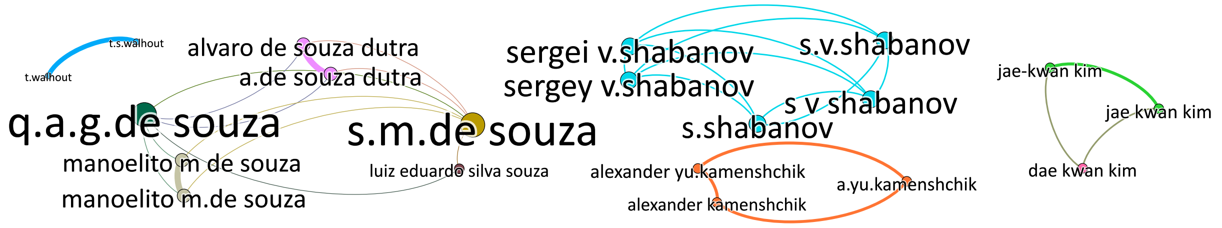


Fig. 6. A sample REF_REF similarity graph. The color and size of vertices represent their real clusters and their frequency, respectively. REF_REF graphs contain several disjoint components.

Table 5
Consistency conditions of an aligned name pair.

PON	Consistency Criteria	Representation in the proposed network
Last name	equal or similar	common in layers 1 or 2
First name	equal, similar, or abbreviated	common in layers 1, 2, or 3
Prefix, Middle name, and Suffix	equal, similar, abbreviated, or omitted from one side	common in layers 1, 2, or 3 + uncommon from one side

gradually relaxes blocking criterions every iteration to increase recall. Several blocking hyper-parameters can be changed during iterations, such as the max layer of the forward pass, passing messages to abbreviated element tokens or not, the structure of blocking decision rule, the message count threshold, and the relative similarity threshold. The values of hyper-parameters in each iteration should be planned according to a tradeoff between performance and running time. As blocking improvement is not the goal of this research, we set the hyper-parameters with naïve default values.

4.3. Clustering

While the blocking stage considers only name similarity, the clustering stage considers similarity and dissimilarity together using the consistency concept. Consistency is a necessary and sufficient condition for matching. An example can show the significance of consistency. Compare 'Christopher F. Williams' as a source with two different names: 'Christopher B. Williams' and 'C. Williams'. While the 1st name is more similar to the source, it is inconsistent. In contrast, the 2nd name is consistent with the source despite the lower similarity.

The clustering stage splits each connected component of the REF_REF similarity graph to several subgraphs such that their references are consistent with each other. Two names are consistent together if they satisfy three conditions (Table 5): first, their last name tokens are equal or similar (common in layers 1 or 2). Second, their first name tokens are equal, similar, or abbreviated (common in layers 1, 2, or 3). Finally, the rest of the matched tokens (prefix, middle name, and suffix) are equal, similar, abbreviated, or even omitted from one side of name pairs. As an example of omitting from one side, 'John M. Doe' is not consistent with 'John A. Doe' owing to the invalidity of omitting middle names from both names. By contrast, 'John M. Doe' is consistent with 'John M. A. Doe' because the middle name 'A.' is omitted only from the second name.

Fig. 7 presents the pseudo-code of the proposed name clustering algorithm. Initially, the REF vertices are inserted into a priority queue based on the confidence of their initial PON tagging. The algorithm aligns each dequeued vertex with its neighbors and checks the consistency of the alignment. The cluster of consistent neighbors is merged with the vertex cluster. If a neighbor is not consistent with the cluster, but a valid tag transformation makes it consistent, its PON tags are transformed to become consistent. The cluster profile is updated after adding each vertex to the cluster. The added vertices to clusters are removed from the queue, and the next iteration is started using the top vertex of the queue. The remaining parts of this subsection detail tagging confidence prioritization, tokens alignment, token's PON transformation, and cluster profile updating.

The proposed algorithm exploits names with high-confident PON tags in the tag correction of similar names. Three measures prioritize the initial tagging confidence of names: first, the lower number of tokens. The initial tags of names with the fewer number of tokens are more probable to become correct. For example, the tags of a name with two tokens are often first name and last name, but tagging a name with five tokens is not straightforward. Second, the lower number of abbreviated tokens. Abbreviated tokens have an intrinsic ambiguity, which makes their alignment difficult. As a result, the initial tags of names with the fewer abbreviated tokens are more confident. Third, higher name frequency. The frequent representations of names are often less noisy, and their tagging is more straightforward compared to the rare noisy forms.

Token alignment is a prerequisite of consistency checking. The proposed algorithm reduces the token alignment to finding the shortest path between the tokens of a reference and the element vertices of a cluster profile. The typical shortest path algorithms like Dijkstra are unidirectional, i.e. they start from the source vertex and traverse edges until reaching the target. The high-degree element vertices, such as abbreviated vertices, significantly increase traversal paths. The higher number of traversal paths, the longer running time. To remedy this deficiency, we propose a bidirectional shortest path algorithm. Fig. 8. illustrates the algorithm process on two sample references, 'seung das' and 'seong r. das'. Traversal starts from both references to their token vertices. In the first step, 'das' token is matched between them, and their first names are matches in the second step. After the second step, all tokens are matched except 'r' of the second name. Traversal goes on to the third layer to match this token, but it remains unmatched.

When a candidate pair is inconsistent, Swash tries to find a valid PON transformation, which makes it consistent. There are three valid transformations of PON tags, i.e., reverse order, shift left, and shift right. The reverse order inverts the tag of first name and last name. The shift left ignores the last middle name tag and slides backwards next tags. The last token is also tagged to the suffix. By contrast, shift right overwrites first middle name tag by sliding forward leading tags and considers the first token as the prefix. Fig. 9 exemplifies these transformations.

After matching each reference to a cluster profile, the cluster profile should be updated. If a token of the matched reference is

```
q ← the priority queue of REF vertices
WHILE q is not empty:
    v    ← dequeue from q
    clus ← the cluster of v
    prof ← the profile of clus
    bfs  ← the neighbors of v sorted by q order

    FOR each n in bfs:
       alg ← align the tokens of n to prof
       IF alg is consistent:
           add n to clus
       ELSE IF alg can become consistent by a valid PON transformation:
           transform the PON tags of n and add n to clus
       ENDIF

       IF n is added to clus:
           update prof
           add n's neighbors to bfs and remove n from q
       ENDIF
    ENDFOR
```

Fig. 7. The pseudo-code of the clustering stage that updates the cluster of each reference vertex.

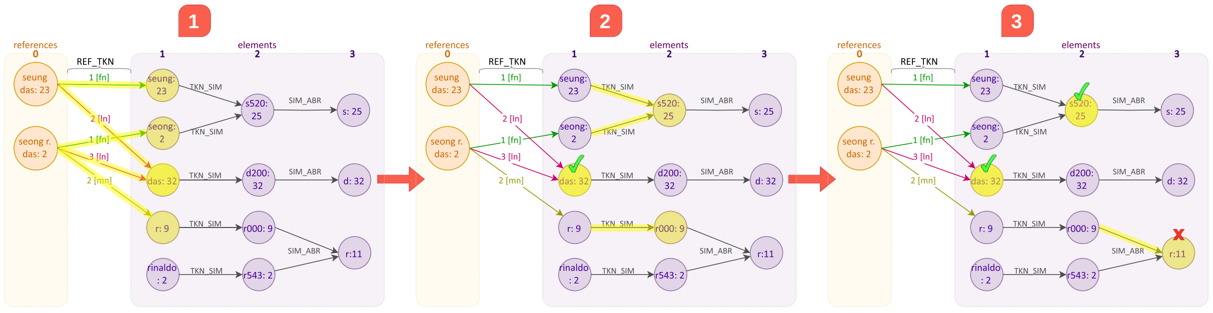


Fig. 8. An example of the bidirectional shortest path algorithm for tokens alignment.

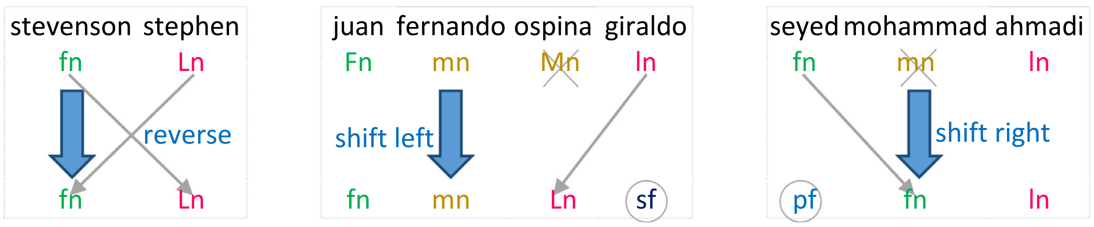


Fig. 9. Examples of three valid transformations of PONs.

absent in the cluster profile, the token should be added to the profile. Besides, matching tokens in the layer 2, such as 'seong' and 'seung' in Fig. 4, raises the first name vertex of the profile to layer 2 ('s520' vertex in Fig. 4).

5. Experimental setup

To evaluate[4] the proposed framework, three publicly available data sets are utilized, i.e., CiteSeer, ArXiv (HEP), and DBLP. First, the CiteSeer dataset was initially created by Giles, Bollacker, and Lawrence (1998). This dataset contains citations of four different areas of machine learning. Aron Culotta and Andrew McCallum later provided the ground truth of the dataset (Culotta & McCallum, 2005). Second, the ArXiv dataset is collected from high-energy physics publications. It was initially used in KDD Cup 2003. David Jensen, from the University of Massachusetts, provided the author entity labels for this dataset (McGovern et al., 2003). The third dataset is a subset of the Digital Bibliography & Library Project (DBLP), which is labeled by Reuther (2006). He has provided three different subsets., that the largest of them (DBLP-SUB-03) is used in this research.

These three datasets have different features such as paper name and author venue, which are ignored for name matching. Only person names and their corresponding entity labels are used in the proposed name matching framework. Table 6 reports several statistics on these datasets. The number of author entities on the DBLP dataset is unavailable due to the lack of entity labels for all references. The dataset has 2018 name pairs, which are different representations of the same entities. In contrast, ArXiv and CiteSeer contain the entity labels of all references in datasets. However, there are many errors in these labels, according to our investigation. Complex name variations, such as changing the order of tokens, were hidden from the sight of manual taggers. The resolved

[4] The source code of Swash has published at https://github.com/MohsenIT/swash-matching .

Table 6
Statistics of three datasets.

Statistic	CiteSeer	ArXiv	DBLP
Number of author names	2892	58,515	58,399
Number of unique author names	1636	12,732	21,688
Number of author entities	1159	8882	?
Average tokens per name	2.4	2.4	2.5

ids of 665 records on ArXiv and 32 records on CiteSeer are manually merged or fixed during our correction[5].

The performance of name matching is usually measured by comparing the correctness ratio of the matching decisions. To evaluate the proposed framework, we utilize pairwise F1 score, which is the harmonic mean of precision and recall measures. These measures are computed for all possible reference pairs due to the collectivity of our approach, as follows:

$$Precision = \frac{|algorithm\ true\ matches|}{|algorithm\ matched\ pairs|} = \frac{\sum_{c\,\in\,AC}\left(\sum_{u\,\in c}\sum_{v\,\in c,\ u\neq v}(1\ if\ u\ matches\ v\ else\ 0)\right)}{\sum_{c\,\in\,AC}(|c|\times(|c|-1))} \tag{5}$$

$$Recall = \frac{|algorithm\ true\ matches|}{|real\ matched\ pairs|} = \frac{\sum_{c\,\in\,AC}\left(\sum_{u\,\in c}\sum_{v\,\in c, u\neq v}(1\ if\ u\ matches\ v\ else\ 0)\right)}{\sum_{r\,\in\,RC}(|r|\times(|r|-1))} \tag{6}$$

$$F1\ score = 2\times\frac{Precision\ \times Recall}{Precision + Recall} \tag{7}$$

where *AC* is clusters identified by algorithms, and *RC* is the real entities' clusters. Pairwise F1 score has been widely used in entity matching literature. The selection of this pairwise evaluation enables the comparison of the proposed collective approach with previous methods.

6. Experimental results

In this section, the performance of the proposed framework is evaluated from two different points of view, i.e., overall performance and components performance. The first experiment presents the overall performance of the proposed framework in comparison with the state-of-the-art name matching methods. The next three experiments discuss the performance of clustering, blocking, and PON transformation components, respectively.

6.1. Overall performance

This experiment comprises the performance of the proposed framework with two recent generations of unsupervised name matching methods, i.e., Hybrid name matching and *Maximum-similarity Alignment matching*. Hybrid matching methods compute the sequence-based similarity between aligned tokens using a greedy algorithm. While this alignment is greedy, *Maximum-similarity Alignment* tries to find an optimal alignment. According to previous research, L2 Jaro-Winkler and graph-based name transformation achieve the best result of Hybrid matching (2nd generation) and *Maximum-similarity Alignment*, respectively (Gong et al., 2009). Fig. 10 illustrates the results of the name matching methods on three datasets, i.e., CiteSeer, ArXiv, and DBLP (ordered from easy to difficult). The proposed framework outperforms all previous methods in these three datasets. The higher difficulty of datasets results in higher F1 outperformance, such that on the DBLP dataset, which the F1 of the second-best method is half of the proposed framework. In addition to the unsupervised methods, the proposed framework even outperforms the previous supervised methods as well. The achieved F1 score is 10% and 20% more than the best-supervised alternatives (Gong et al., 2009) on ArXiv and DBLP, respectively. The rest of the experiments discuss the reasons behind this outperformance.

The significance of the result is an important concern of machine learning research, especially in supervised learning. The significance testing methods often study randomness and deviation of the achieved result. Lack of training data in unsupervised learning reduces deviation, which arises from train-test data selection. The proposed unsupervised framework has only one source of deviation in the results, i.e., the order of equal-rank references in the clustering priority queue. To test the impact of the order, we executed the proposed approach 30 times on the ArXiv data with random orders of equal-rank references. The standard deviation of resulted F1 values was 0.00027, which is much lower than the F1 improvement. This result confirms the significance of the proposed framework.

6.2. Consistency versus aligned tokens similarity

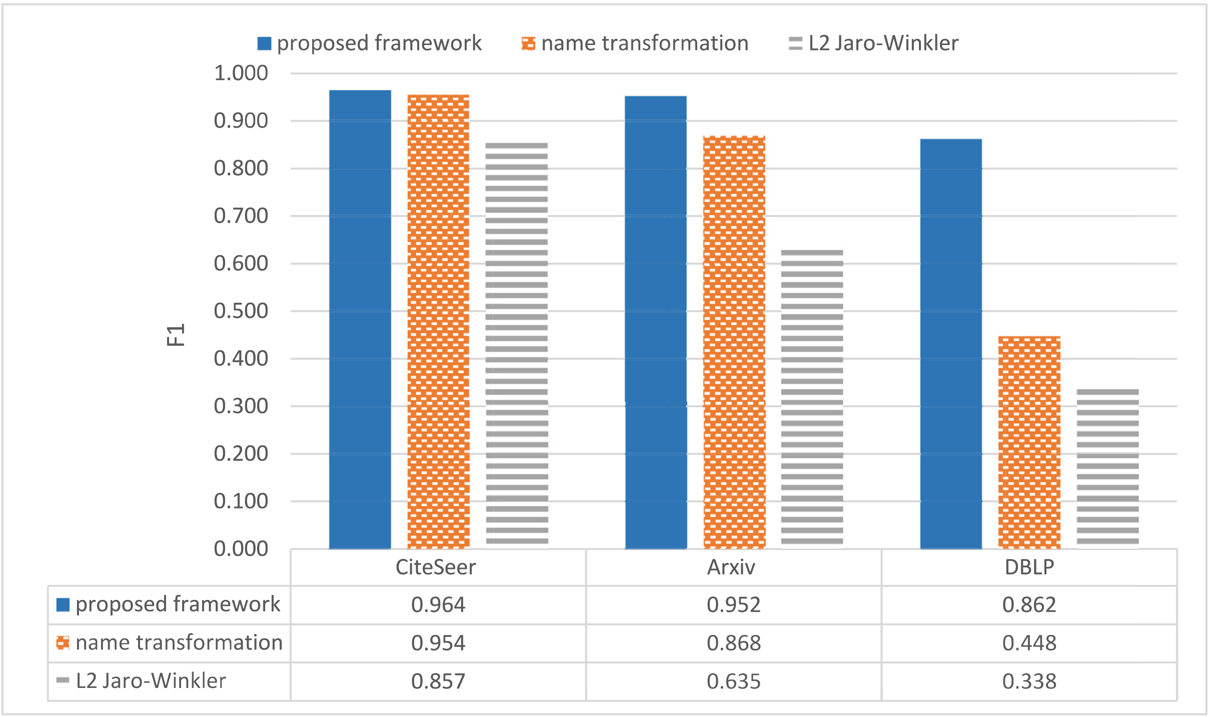
While the previous research relies on the similarity of the aligned tokens, this research raises the issue that token alignment cannot correctly identify structure variations of names. To address this issue, the consistency concept is defined, and a clustering algorithm is proposed based on this concept. This experiment studies the impact of consistency on matching. It compares the performance of community detection algorithms on the reference similarity graph with the proposed consistency clustering. The similarity graph is the output of the proposed blocking stage. The edge weights of this graph are computed using the similarity measure of Eq. (3). For instance, the similarity between 'seung das' and 'seong r. das' in Fig. 6 is 0.83, aggregating 0.33 from 'das' and 0.5 from 's520'. This graph represents the similarity of aligned tokens; therefore, the communities of similar references should represent real entities. Fig. 11 depicts the F1 score of the proposed consistency clustering in comparison with several popular community detection algorithms, including Label Propagation (Raghavan, Albert, & Kumara, 2007), Infomap (Rosvall & Bergstrom, 2008), Louvain modularity (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008), Fast greedy (Clauset, Newman, & Moore, 2004), Leading eigenvector (Newman, 2006), and connected components. This experiment is carried out on the ArXiv dataset due to cluster label availability for all references and its size.

According to Fig. 11, we can categorize methods to three levels according to their performance: Low-level with F1 lower than 0.1 (Louvain, Fast greedy, Leading eigenvector, and connected components), mid-level with 0.5-0.7 F1 (Infomap and Label Propagation), and high-level with F1 higher than 0.9, which only includes the proposed consistency clustering. There are two significant gaps between consequent levels. These gaps raise two questions. First,

Table 7
The performance and granularity of the community detection methods.

Method	F1	Precision	Recall	Cluster count	Cluster count of giant comp.
Proposed clustering	0.952	0.980	0.926	8895	2831
Label Propagation	0.675	0.532	0.923	3168	1100
Infomap	0.576	0.416	0.932	2722	802
Louvain	0.084	0.044	0.941	1930	70

[5] The datasets and their errata publicly are available at https://github.com/MohsenIT/swash-data-modeling/tree/master/data .



	CiteSeer	Arxiv	DBLP
■ proposed framework	0.964	0.952	0.862
▨ name transformation	0.954	0.868	0.448
▤ L2 Jaro-Winkler	0.857	0.635	0.338

Fig. 10. F1 comparison of the proposed matching framework with the best previous unsupervised methods.

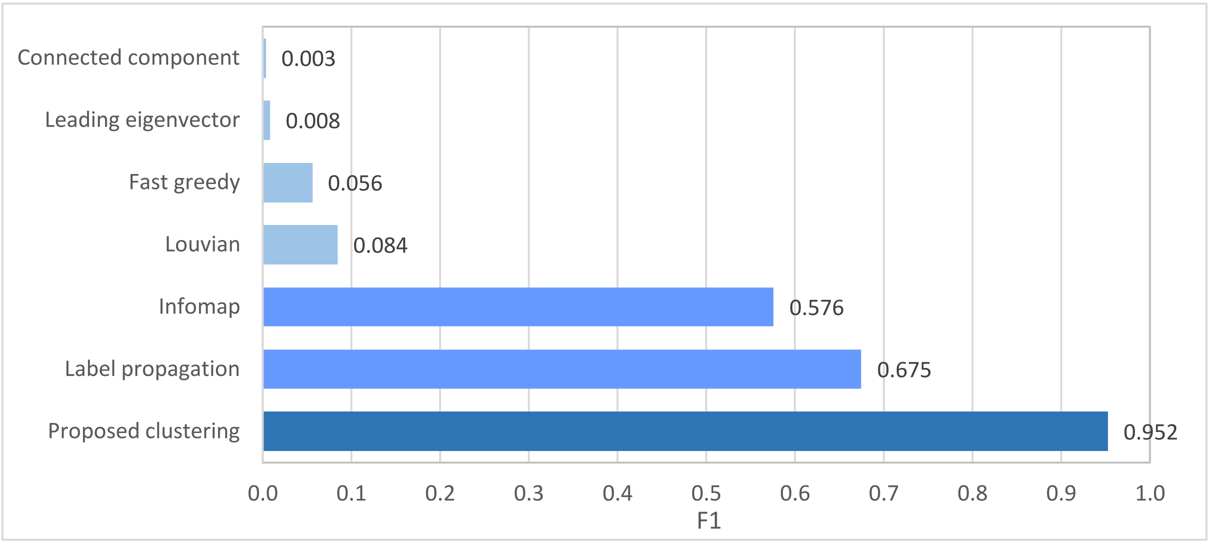


Fig. 11. The performance of the proposed consistency clustering in comparison with community detection algorithms.

what is the reason for the significant difference between F1 scores in low and middle-level community detection algorithms? Second, why could not the community detection algorithms achieve a high-performance level? Next two paragraphs answer these questions.

Granularity is the source of the large gap between low and mid-performance level methods. Table 7 presents the performance and cluster count of Louvain (as a representative of low-level methods) versus Infomap and Label Propagation (mid-level methods). The giant component of the similarity graph has 4372 vertices (34% of all vertices), which belongs to 2899 different real entities. Louvain clusters the giant component to only 70 clusters, while the mid-level algorithms find more than 800 clusters in this component. The coarse-grain clusters of low-level algorithms such as Louvain cause poor performance. These methods are initially developed for unweighted graphs; therefore, their extensions for the weighted graph are still weak. By contrast, Infomap and Label Propagation intrinsically utilize edge weights to cluster vertices. As a result, these methods divide the graph's giant component into more clusters, as displayed in Fig. 12.

None of the common community detection algorithms can achieve a high-performance level due to the inadequacy of aligned token similarity. In other words, the similarity between aligned tokens cannot correctly represent the required matching information. The subgraph of Fig. 13 supports this fact. There is no significant deviation in the edge weights of references having 'dolan' token, while these references belong to three different entities. On the other hand, the similarity of 'louise dolan' to 'louise c.martin' is much more than 'l. dolan', which belongs to the same entity of

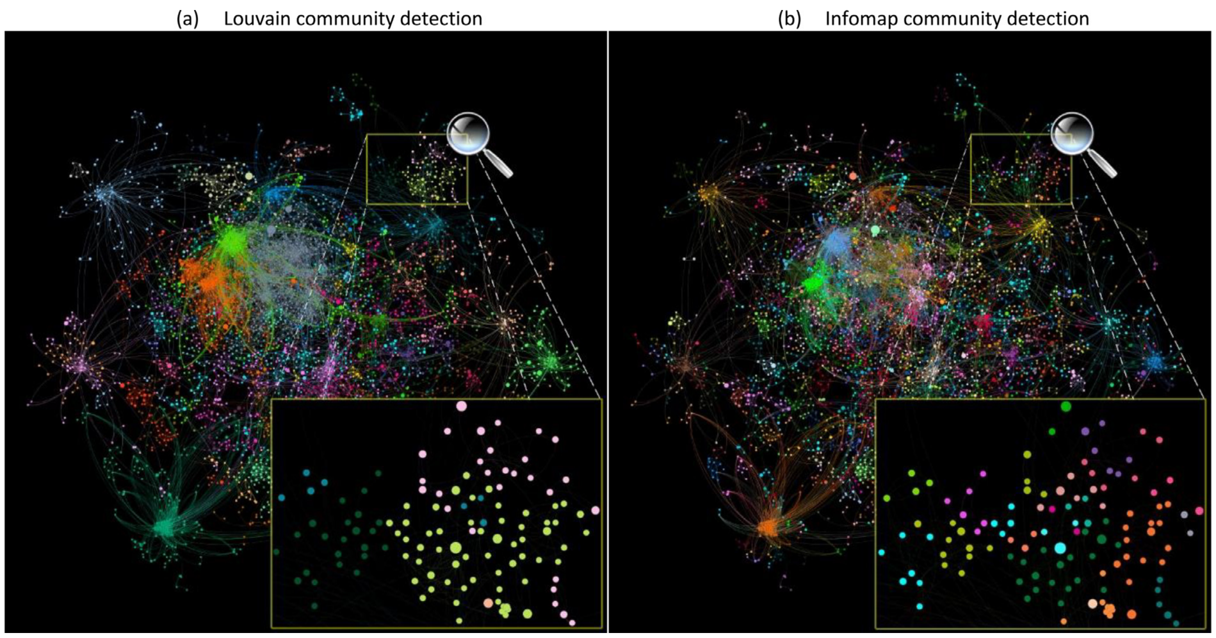

(a) Louvain community detection
(b) Infomap community detection

Fig. 12. Identified clusters in the giant component of the similarity graph. Louvain is more coarse grain than Infomap.

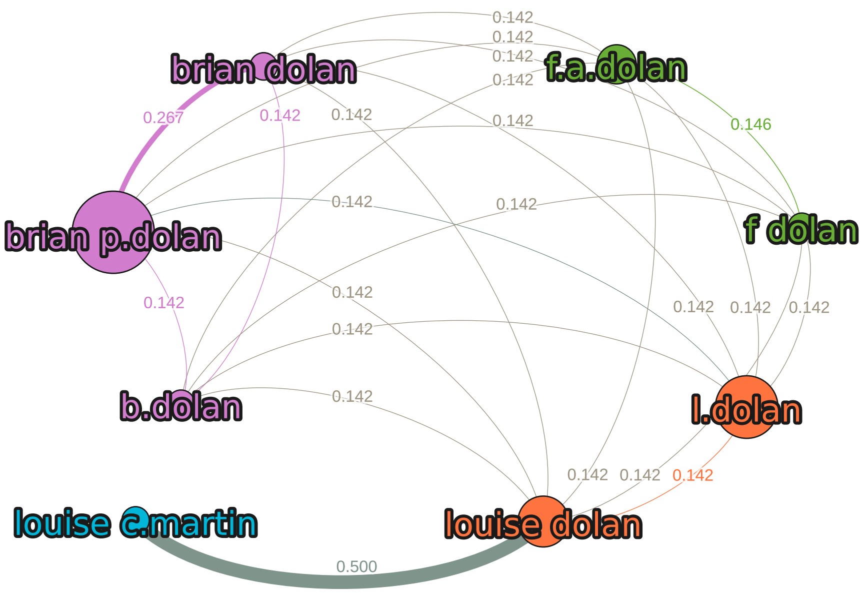


Fig. 13. A subgraph of aligned token similarity graph. Vertices color represents the real entity of references.

'louise dolan'. As a result, matching based on only the similarity graph is inadequate. The proposed solution settles this issue using the consistency clustering, which considers dissimilarity in addition to similarity. Accordingly, it can correctly resolve the entities of Fig. 13.

6.3. The blocking component performance

Scalability is the primary concern in the blocking stage; however, increasing scalability conventionally reduces efficiency. This experiment evaluates the scalability and efficiency of the proposed blocking using the number of comparisons and F1, respectively. The experiment and all other experiments in this research, utilize the same blocking settings, i.e., one iteration blocking that checks two conditions. First, reference candidate pairs should have at least one common token. Second, the relative similarity between them should be more than 0.5.

Table 8 reports the number of comparisons with and without blocking. If having one common token condition is considered as blocking, the number of comparisons is decreased significantly (more than 84%), but its absolute value is still high. Adding the second condition (relative similarity more than 0.5) also reduces the absolute value to less than 100 K, which is linear to the number of references.

Blocking increases scalability at the cost of missing several true matches. To study the quantity of missed true matches, we compute the maximum achievable F1 of the suggested blocking using a perfect matching assumption. This assumption assumes that the clustering stage correctly matches whole comparisons. Table 9 presents the performance result. According to the result, the maxi-

Table 8
The impact of the proposed blocking conditions in comparison reduction. The first blocking condition is one common token, and the second is that the relative similarity of candidates should be more than 0.5.

Dataset	References count	The number of comparisons (the proportion of all)		
		Without blocking	Only first condition	Both conditions
CiteSeer	1524	1160526	185106 (15.95%)	1588 (0.137%)
Arxiv	12,863	82721953	6404812 (07.74%)	32073 (0.039%)
DBLP	21,550	232190475	5767888 (02.48%)	79640 (0.034%)

Table 9
The maximum performance achievable based on the blocking vs. the achieved performance.

Dataset	Max achievable performance			Achieved F1	F1 gap
	Precision	Recall	F1		
CiteSeer	1.000	0.967	0.983	0.964	**1.9%**
ArXiv	1.000	0.941	0.969	0.952	**1.7%**
DBLP	1.000	0.899	0.947	0.862	**8.5%**

mum achievable F1 values are significantly lower than 1 (between 2% to 6%). It means that the utilized blocking scheme is responsible for a large proportion of the error. To decrease the error, the improvement of the blocking scheme is crucial. While the blocking improvement is not in the aim of this research, we leave this task for future research.

6.4. *The impact of PON transformation*

The proposed clustering tries to make inconsistent references consistent using the valid PON transformations. However, these transformations are not necessarily true. To be exact, PON transformations increase false-positive decisions besides true positives. Fig. 14 displays the overall F1 changes after using the proposed PON transformation. According to this figure, the transformation improves the F1 in all three datasets. Therefore, the increase in the true-positive rate is more than the increase in the false-positive rate.

The increase of F1 in ArXiv and CiteSeer datasets are minor. Contrary to the initial impression, the reason of this minority is not the weakness of the proposed solution. The reason is the minority of the references that can be made consistent using PON transformation, as depicted in Table 10. The proportion of these references is 1% or less; therefore, the maximum possible F1 increase is limited. The success rate of PON transformation in CiteSeer and ArXiv (100% and 64%, respectively) shows the proficiency of the proposed PON transformation.

Table 10
The impact of PON transformation (*F1 computation is impossible due to the unavailability of real clusters).

Dataset	F1 increase	Max possible F1 increase	Inconsistent count	Made consistent count	Success rate
CiteSeer	0.004	0.004	287	3	100%
ArXiv	0.001	0.002	12,525	66	64%
DBLP	0.016	?*	64,824	262	?*

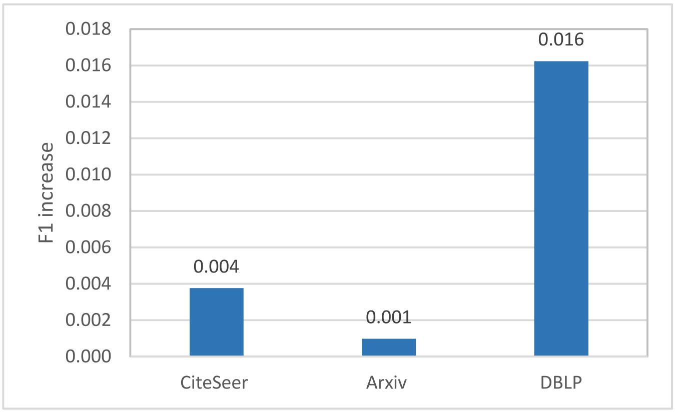


Fig. 14. Increase F1 after using the proposed PON transformations.

7. Conclusions and future works

This paper has extended our knowledge of personal name matching. It categorized possible variations of names and studied the limitations of previous name matching methods on these variations. To overcome these limitations, the Swash name matching framework was proposed. The framework consists of two phases: data modeling and matching. A comprehensive Heterogeneous Information Network was suggested to model names data. It embeds all collective information gatherable from names in a unified model. To match based on this model, a blocking scheme and a consistency clustering scheme were proposed. The blocking scheme finds candidates having similar tokens by a message-passing algorithm. The consistency clustering scheme acquires the PON (Part Of Name) tags of tokens using a self-trainable algorithm, and clusters candidates according to the consistency of their tags. Several experiments evaluated Swash from different aspects and presented three main findings, which will be reported in the next three paragraphs.

This research has found that the sole reliance of the previous methods on reference similarity is inadequate. The similarity is only the necessary condition for matching. Therefore, it should be utilized in blocking, which finds probable candidates. To match candidates, consistency should be checked. Consistency considers dissimilarity and similarity together and is a necessary and sufficient condition for matching. According to the experiment on the ArXiv dataset, the consistency clustering scheme improves the F1 score of the best similarity clustering method by more than 26%.

The second major finding is the importance of collective information gatherable from name datasets. A name dataset contains valuable information on frequency and sequence of name tokens. Attachment of tokens similarity also augments this information. Previous matching methods have neglected this global information due to their local pairwise approach. The proposed collective framework embeds this global information and matches based on it. Utilizing this information improves the results, especially on more noisy datasets. For instance, 38% F1 improvement is achieved in the DBLP dataset.

The evaluation of the Swash blocking scheme revealed the third finding. This scheme suggests a relative similarity measure, which specifies the candidate similarity threshold based on the similarity of a reference to itself. This measure has a significant advantage, i.e., comparability. The relative similarity solves the comparability issue using normalization of the absolute value with the self-similarity. According to the experiments, utilizing the relative similarity has successfully decreased the quadratic number of comparisons to near linear.

The proposed blocking scheme has an important limitation. While the blocking scheme significantly increases scalability, it misses a considerable number of true matches. The blocking scheme is responsible for about half of the performance shortage. As an example, 95.2% F1 score is achieved in the ArXiv dataset; however, even using a perfect clustering scheme, this score cannot exceed more than 96.9%. Accordingly, the blocking scheme improvement is crucial to achieving a perfect matching method.

The findings of current research provide insights for future research. Data modeling can be improved by developing a suitable name breaking algorithm and considering the case of token's letters. Moreover, extending the current modeling to identify 'Insertion of extra delimiter' variation is also desired. Consistency clustering also can be extended from different aspects. First, enhancing the output to a hierarchical structure, instead of flat clusters. This structure represents which name can be matched to which one. As an example, 'John Doe' and 'Jack Doe' are the children of 'J. Doe' in this hierarchy. The second aspect of the clustering component extension relates to the PON tagger. The distribution of confident tags can also be utilized to improve the tagger performance.

Declaration of Competing Interest

None.

Credit authorship contribution statement

Mohsen Raeesi: Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curation, Writing - original draft, Writing - review & editing, Visualization. **Masoud Asadpour:** Supervision. **Azadeh Shakery:** Writing - review & editing, Supervision.

Acknowledgment

This research was in part supported by a grant from the Institute for Research in Fundamental Sciences (No. CS1398-4-223).

References

Arehart, M., & Miller, K. J. (2008). A ground truth dataset for matching culturally diverse romanized person names. *International Conference on Language Resources and Evaluation (LREC).*

Ash, S. M. (2017). *Improving accuracy of patient demographic matching and identity resolution* PhD. Thesis. The University of Memphis.

Bhattacharya, I., & Getoor, L. (2007). Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD), 1*(1), 5. doi:10.1145/1217299.1217304.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment, 2008*(10), P10008. doi:10.1088/1742-5468/2008/10/P10008.

Christen, P. (2006). A Comparison of Personal Name Matching: Techniques and Practical Issues. In *Proceedings of the Sixth IEEE international conference on data mining workshops (ICDMW'06)* (pp. 290–294). doi:10.1109/ICDMW.2006.2.

Christen, P., & Gayler, R. W. (2015). Context-Aware Approximate String Matching for Large-Scale Real-Time Entity Resolution. In *Proceedings of the IEEE international conference on data mining workshop (ICDMW)* (pp. 211–217). doi:10.1109/ICDMW.2015.152.

Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E, 70*(6). doi:10.1103/PhysRevE.70.066111.

Cohen, W. W. (2000). Data integration using similarity joins and a word-based information representation language. *ACM Trans. Inf. Syst., 18*(3), 288–321. doi:10.1145/352595.352598.

Cohen, W., Ravikumar, P., & Fienberg, S. (2003). A comparison of string metrics for matching names and records. In *Proceedings of the KDD workshop on data cleaning and object consolidation: 3* (pp. 73–78).

Culotta, A., & McCallum, A. (2005). Joint deduplication of multiple record types in relational data. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 257–258). ACM. doi:10.1145/1099554.1099615.

Doan, A., Halevy, A., & Ives, Z. (2012). *Principles of data integration*. Elsevier Retrieved from. doi:10.1016/c2011-0-06130-6.

Gali, N., Mariescu-Istodor, R., & Fränti, P. (2016). Similarity measures for title matching. In *Proceedings of the 23rd international conference on pattern recognition (ICPR)* (pp. 1548–1553). doi:10.1109/ICPR.2016.7899857.

Galvez, C., & Moya-Anegón, F. (2007). Approximate personal name-matching through finite-state graphs. *Journal of the American Society for Information Science and Technology, 58*(13), 1960–1976. doi:10.1002/asi.20671.

Giles, C. L., Bollacker, K. D., & Lawrence, S. (1998). CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on digital libraries* (pp. 89–98). ACM. doi:10.1145/276675.276685.

Gong, J., Wang, L., & Oard, D. W. (2009). Matching person names through name transformation. In *Proceedings of the 18th ACM conference on information and knowledge management* (pp. 1875–1878). ACM. doi:10.1145/1645953.1646253.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In Soviet Physics Doklady (Vol. 10, p. 707). Retrieved from http://adsabs.harvard.edu/abs/1966SPhD10.707L%E5%AF%86

McGovern, A., Friedland, L., Hay, M., Gallagher, B., Fast, A., Neville, J., & Jensen, D. (2003). Exploiting relational structure to understand publication patterns in high-energy physics. *ACM SIGKDD Explorations Newsletter, 5*(2), 165–172. doi:10.1145/980972.980999.

Medhat, D., Hassan, A., & Salama, C. (2015). A hybrid cross-language name matching technique using novel modified Levenshtein Distance. In *Proceedings of the tenth international conference computer engineering & systems (ICCES)* (pp. 204–209). IEEE. *on*. doi:10.1109/icces.2015.7393046.

Monge, A. E., & Elkan, C. P. (1996). The field matching problem: algorithms and applications. In *Proceedings of the second international conference on knowledge discovery and data mining* (pp. 267–270). Portland: Oregon: AAAI Press. Retrieved from. http://dl.acm.org/citation.cfm?id=3001460.3001516 .

Newman, M. E. J. (2006). Finding community structure in networks using the eigenvectors of matrices. *Physical Review E, 74*(3). doi:10.1103/PhysRevE.74.036104.

Odell, M., & Russell, R. (1918). The Soundex coding system. US Patents, 1261167.

Peng, N., Yu, M., & Dredze, M. (2015). An empirical study of Chinese name matching and applications. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing: 2* (pp. 377–383). Association for Computational Linguistics. Volume 2: Short Papers.

Peng, T., Li, L., & Kennedy, J. (2012). A comparison of techniques for name matching. *GSTF Journal on Computing (JoC), 2*(1), 377–383.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E, 76*(3). doi:10.1103/PhysRevE.76.036106.

Reuther, P. (2006). Personal name matching: New test collections and a social network based approach. *Computer Science Technical Report, 1*, 1–22.

Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences, 105*(4), 1118–1123. doi:10.1073/pnas.0706851105.

Shaalan, K., & Raza, H. (2007). Person name entity recognition for Arabic. In *Proceedings of the workshop on computational approaches to semitic languages: common issues and resources* (pp. 17–24). Association for Computational Linguistics.

Shi, C., Li, Y., Zhang, J., Sun, Y., & Yu, P. S. (2017). A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering, 29*(1), 17–37. doi:10.1109/TKDE.2016.2598561.

Sukharev, J., Zhukov, L., & Popescul, A. (2014). Parallel corpus approach for name matching in record linkage. In *Proceedings of the IEEE international conference on data mining (ICDM)* (pp. 995–1000). IEEE.

Treeratpituk, P., & Giles, C. L. (2012). Name-ethnicity classification and ethnicity-sensitive name matching. In *Proceedings of the AAAI*.

Varol, C., & Bayrak, C. (2012). Hybrid matching algorithm for personal names. *Journal of Data and Information Quality (JDIQ), 3*(4), 8.

Wang, K., Thrasher, C., & Hsu, B.-J. P. (2011). Web Scale NLP: A Case Study on Url Word Breaking. In *Proceedings of the 20th international conference on world wide web* (pp. 357–366). USA: ACM. New York, NY. doi:10.1145/1963405.1963457.

Zhagorina, K., Braslavski, P., & Gusev, V. (2018). Personal names popularity estimation and its application to record linkage. In *Proceedings of the European conference on advances in databases and information systems* (pp. 71–79). Springer.