Robust Group Linkage

Pei Li
University of Zurich
peili@ifi.uzh.ch

Xin Luna Dong
Google Inc.
lunadong@google.com

Songtao Guo
LinkedIn
songtao.gg@gmail.com

Andrea Maurino
University of Milan-Bicocca
maurino@disco.unimib.it

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

ABSTRACT

We study the problem of *group linkage*: linking records that refer to multiple entities in the same group. Applications for group linkage include finding businesses in the same chain, finding social network users from the same organization, and so on. Group linkage faces new challenges compared to traditional entity resolution. First, although different members in the same group can share some similar *global* values of an attribute, they represent different entities so can also have distinct *local* values for the same or different attributes, requiring a high *tolerance* for value diversity. Second, we need to be able to distinguish local values from erroneous values.

We present a robust two-stage algorithm: the first stage identifies *pivots–maximal* sets of records that are very likely to belong to the same group, while being robust to possible erroneous values; the second stage collects strong evidence from the pivots and leverages it for merging more records into the same group, while being tolerant to differences in local values of an attribute. Experimental results show the high effectiveness and efficiency of our algorithm on various real-world data sets.

1. INTRODUCTION

Entity resolution aims at linking records that refer to the same real-world entity and has been extensively studied in the literature (surveyed in [7, 18]). In this paper we study a related but different problem that we call *group linkage*: linking records that refer to multiple entities in the same group.

One motivation for our work comes from the need to group the millions of social network users (*e.g.*, *LinkedIn*) by their organizations, which improves matching and recommendation activities in social networks. The organization information is often missing, incomplete, or simply too heterogeneous to be recognized as the same (*e.g.*, "International Business Machines Corporation", "IBM Corp.", "IBM", "IBM-Almaden", etc., all refer to the same organization). Contact phones, email addresses, and mailing addresses of people all provide extra evidence for group linkage, but they can also vary for different people even in the same organization.

We are also motivated by applications where we need to identify *business chains*, multiple business entities that share a brand name and provide similar products and services (*e.g.*, *Walmart, McDonald's*). With the advent of the Web and mobile devices, we are observing a boom in *local search*: that is, searching local businesses under geographical constraints. Local search engines include *Google Maps*, *Yahoo! Local*, *YellowPages, yelp, ezlocal*, etc. The knowledge of business chains can have a big economic value to local search engines. However, business listings are rarely associated with specific chains explicitly stated in real-world business-listing collections. Sharing the same name, phone number, or URL domain name can all serve as evidence of belonging to the same chain. But the same value is often presented in different ways and there are many erroneous values, as we soon show.

Table 1: Identified top-5 US business chains. For each chain, we show the number of stores, distinct business names, distinct phone numbers, distinct URL domain names, and distinct categories.

Name	#Store	#Name	#Phn	#URL	#Cat
SUBWAY	21,912	772	21,483	6	23
Bank of America	21,727	48	6,573	186	24
U-Haul	21,638	2,340	18,384	14	20
USPS - United State Post Office	19,225	12,345	5,761	282	22
McDonald's	17,289	2401	16,607	568	47

Group linkage differs from entity resolution in the following aspects. First, the type of heterogeneity in groups is different from that in entities, which is mainly caused by typographical errors and different representations of the same value. Instead, different members in the same group can share some similar *global* values as group identifier, and meanwhile can have distinct *local* values of the same attribute as entity identifier. For example, many branches in the same business chain provide a primary company-wide phone number, while a significant number of branches may provide different local phone numbers. Traditional methods learn different weights for different attributes so they can be tolerant on value variety for some less coherent attributes; they fall short in our context since global values and local values often occur in the same attribute. Second, it is non-trivial to distinguish such differences from erroneous values in the data. Finally, a group can contain tens of thousands of members. Computation within such huge groups can be very expensive; thus, *scalability* is a big challenge. We use the following example throughout the paper for illustration.

EXAMPLE 1.1. *We consider a set of 18M real-world business listings in the US extracted from a local search engine, each describing a business by its name, phone number, URL domain name, location, and category. Our algorithm automatically finds 600K business chains and 2.7M listings that belong to these chains. Table 1 lists the five largest chains we found. We observe that (1) each chain contains up to 22K different branch stores, (2) different branches from the same chain can have a large variety of names,*

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741118

Table 2: Real-world business listings. We show only state for location and simplify names of category. There is a wrong value in italic font.

RID	name	phone	URL (domain)	location	category
r_1	Home Depot, The	808		NJ	furniture
r_2	Home Depot, The	808		NY	furniture
r_3	Home Depot, The	808	homedepot	MD	furniture
r_4	Home Depot, The	808	homedepot	AK	furniture
r_5	Home Depot, The	808	homedepot	MI	furniture
r_6	Home Depot, The	101	homedepot	IN	furniture
r_7	Home Depot, The	102	homedepot	NY	furniture
r_8	Home Depot, USA	103	homedepot	WV	furniture
r_9	Home Depot USA	808		SD	furniture
r_{10}	Home Depot - Tools	808		FL	furniture
r_{11}	Taco Casa		tacocasa	AL	restaurant
r_{12}	Taco Casa	900	tacocasa	AL	restaurant
r_{13}	Taco Casa	900	tacocasa, *tacocasatexas*	AL	restaurant
r_{14}	Taco Casa	900		AL	food
r_{15}	Taco Casa	900		AL	food
r_{16}	Taco Casa	701	tacocasatexas	TX	restaurant
r_{17}	Taco Casa	702	tacocasatexas	TX	restaurant
r_{18}	Taco Casa	703	tacocasatexas	TX	restaurant
r_{19}	Taco Casa	704		NY	food store
r_{20}	Taco Casa		tacodelmar	AK	restaurant

phone numbers, and URL domain names, and (3) even chains of similar sizes can have very different numbers of distinct URL domains (same for other attributes). Thus, rule-based linkage can hardly succeed and scalability is essential.

Table 2 shows 20 business listings (with some abstraction) in this data set. After investigating their webpages manually, we find that $r_1 - r_{18}$ *belong to three business chains:* $Ch_1 = \{r_1 - r_{10}\}$, $Ch_2 = \{r_{11} - r_{15}\}$, *and* $Ch_3 = \{r_{16} - r_{18}\}$; r_{19} *and* r_{20} *do not belong to any chain. Note the slightly different names for businesses in chain* Ch_1*; also note that* r_{13} *is integrated from different sources and contains two URLs, one (*tacocasatexas*) being wrong.*

Simple linkage rules do not work well on this data set. For example, if we require only high similarity on name *for chain linkage, we may wrongly decide that* $r_{11} - r_{20}$ *all belong to the same chain as they share a popular restaurant name* Taco Casa*. Traditional linkage strategies do not work well either. If we apply Swoosh-style linkage [25] and iteratively merge records with high similarity on* name *and shared* phone *or* URL*, we can wrongly merge* Ch_2 *and* Ch_3 *because of the wrong URL from* r_{13}*. If we learn different weights for different attributes, a high weight for* phone *would split* $r_6 - r_8$ *out of chain* Ch_1 *because of their different local phone numbers, but a low weight for* phone *would split* $r_9 - r_{10}$ *out of chain* Ch_1 *since sharing the primary phone number, the main evidence, is downweighted.* □

The key idea in our solution is to find *sufficient strong evidence* that can glue group members together, while being tolerant to differences in values specific for individual group members. For example, we wish to reward sharing of primary values, such as *primary phone numbers* or *URL domain names* for chain linkage, but would not penalize differences in local values, such as *locations* and *local phone numbers*. For this purpose, our algorithm proceeds in two stages. First, we identify *pivots* containing *maximal sets of* records that are very likely to belong to the same group. Second, we collect *strong evidence* from the resulting pivots, such as primary phone numbers and URL domain names in business chains, based on which we cluster the pivots and remaining records into groups. Whereas our approach shares insights with other two-stage clustering techniques in the literature [1, 19, 22, 26, 28, 4], our pivot identification step guarantees both *robustness* to presence of erroneous values, which is critical for high precision, and generation of *maximal* pivots, which is critical for high recall. The advantages of our approach are verified in our experiments.

The group linkage problem we study in this paper is different from the group linkage problems in [17, 23], which compute *group-level* similarity between *pre-specified* groups of records from the same entity. Our goal is to find records of multiple entities that belong to the same group and we make three contributions.

1. We study pivot generation in the presence of erroneous data. Our pivot is *robust*: even if we remove a few possibly erroneous records from a pivot, we still have sufficient strong evidence that the other records belong to the same group.
2. We then reduce the group linkage problem to clustering pivots and the remaining records. We learn different weights *at the value level*, such that our clustering algorithm can leverage strong evidence collected from pivots and meanwhile be *tolerant* to value variety of records in the same group.
3. Experiments on two real-world data sets based on our motivating applications show high efficiency and effectiveness of our proposed approach.

Note that this paper focuses on finding records that belong to the same group. It does not require applying entity resolution to identify records that refer to the same individual entity. However, we show empirically that combining our algorithm and entity resolution can improve the results of both.

In the rest of the paper, Section 2 discusses related work. Section 3 defines the problem and provides an overview of our solution. Sections 4-5 describe the two stages in our solution. Section 6 describes experimental results. Section 7 concludes. For reasons of space, all proofs are omitted, and can be found in [21].

2. RELATED WORK

Entity resolution has been extensively studied in the past (surveyed in [7, 18]). Traditional entity resolution techniques aim at linking records that refer to the same real-world entity, so implicitly assume value consistency between records that should be linked. Group linkage is different in that it aims at linking records that refer to different entities in the same group. The variety of individual entities requires better use of strong evidence and tolerance on different values even within the same group. These two features differentiate our work from any previous linkage technique.

For record clustering in entity resolution, existing work may apply the transitive rule [16], do match-and-merge [25], or reduce it to an optimization problem [15]. Our work is different in that our pivot-linkage algorithm aims at being robust to a few erroneous records; and our clustering algorithm emphasizes leveraging the strong evidence collected from the pivots.

For record-similarity computation, existing work can be distance based [6], rule based [16], or classification based [10]. There has also been work on weight (or model) learning from labeled data [10, 27]. Our work is different in that in addition to learning a weight for each attribute, we also learn a *separate weight for each value* based on whether it serves as important evidence for the group. Note that some previous works are also tolerant to different values, but they leverage evidence that may not be available in our contexts: [9] is tolerant to schema heterogeneity from different relations by specifying matching rules; [14] is tolerant to possibly false values by considering agreement between different data providers; [20] is tolerant to out-of-date values by considering time stamps; we are tolerant to value diversity within the same group.

Two-stage clustering has been proposed in DB and IR communities [1, 19, 22, 26, 28, 4]; however, they identify pivots in different ways. Techniques in [19, 26] consider a pivot as a single record, either randomly selected or selected according to the weighted de-

grees of nodes in the graph. Techniques in [28] generate pivots using agglomerative clustering but can be too conservative and miss strong evidence. Techniques in [1] identify pivots as *bi-connected components*, where removing any node would not disconnect the graph. Although this corresponds to the *1-robustness* requirement in our solution (defined in Section 4), they generate overlapping clusters; it is not obvious how to derive non-overlapping clusters in applications such as business-chain linkage and how to extend their techniques to guarantee k-robustness. Techniques in [19, 22] require knowledge of the number of clusters for one of the stages, so are inapplicable for linkage applications, where the number of clusters is not known a priori. Techniques in [4] generate pivots to collect evolution evidence for resolving entities that evolve over time, but may not be robust against erroneous values. We experimentally compare with these methods whenever applicable (Section 6), showing that our algorithm is robust in presence of erroneous values and consistently generates high-accuracy results on data sets with different features.

Link prediction [12, 24] aims to predict generic links (e.g., referring to the same entity, being in the same group) between a pair of records. However, link prediction itself can make mistakes, and clustering entities (into groups) by using straightforward methods (such as the transitivity rule in [24]) is undesirable. Combining link prediction with our robust techniques for group linkage is an interesting direction of future work.

Finally, we distinguish our work from *group linkage* in [17, 23], which has a different goal of matching groups of records associated with the same entity from multiple databases. On et al. [23] compute similarity between pre-specified groups of records based on *record-level similarity*. With the same goal, Huang [17] decides group-level similarity using network evolution analysis. Our goal is to find records of *multiple entities* that belong to the same group.

3. OVERVIEW

3.1 Problem Definition

Let $\mathbf{R}$ be a set of records describing entities by a set of attributes $\mathbf{A}$. For a record $r \in \mathbf{R}$, we denote by $r.A$ its value of attribute $A \in \mathbf{A}$. Records may contain erroneous or missing values.

Group linkage aims to find records that represent entities belonging to the same real-world group. We focus on non-overlapping groups, which often hold in applications. As an example application, we wish to find *business chains*–a set of business entities with the same or highly similar names that provide similar products and services (e.g., *Walmart*, *Home Depot*, *Subway* and *McDonald's*).

DEFINITION 3.1 (GROUP LINKAGE). *Given a set of records, denoted by* $\mathbf{R}$, group linkage *identifies a set of clusters* $\mathbf{CH}$ *in* $\mathbf{R}$, *such that records representing real-world entities in the same group belong to one and the same cluster, and vice versa.* □

EXAMPLE 3.2. *Consider the records in Example 1.1, where each record describes a business listing by attributes* name, phone, URL, location, *and* category.

The ideal solution to the group linkage problem contains 5 clusters: $Ch_1 = \{r_1 - r_{10}\}$, $Ch_2 = \{r_{11} - r_{15}\}$, $Ch_3 = \{r_{16} - r_{18}\}$, $Ch_4 = \{r_{19}\}$, *and* $Ch_5 = \{r_{20}\}$. *Among them,* Ch_2 *and* Ch_3 *represent two different chains with the same name.* □

3.2 Overview of Our Solution

Group linkage is related to but different from traditional entity resolution because it essentially looks for records that represent entities in the same group, rather than records that represent exactly the same entity. Different members in the same group often share a certain amount of commonality (*e.g.*, common name, primary phone, and URL domain of chain stores), but meanwhile can also have a lot of differences (*e.g.*, different addresses, local phone numbers, and local URL domains); thus, we need to allow much higher variety in some attribute values to avoid false negatives. On the other hand, as we have shown in Example 1.1, simply lowering our requirement on similarity of records or similarity of a few attributes in clustering can lead to a lot of false positives.

The key intuition of our solution is to distinguish between *strong* and *weak* evidence. For example, branches in the same business chain often share URL domain names and those in North America often share a few 1-800 phone numbers. Thus, a URL domain or phone number shared among many business listings can serve as strong evidence for chain linkage. In contrast, a phone number shared by only a couple of entities is much weaker evidence, since one might be an erroneous or out-of-date value.

To facilitate leveraging strong evidence, our solution consists of two stages. The first stage collects records highly likely to belong to the same group; for example, a set of business listings with the same name and phone number are very likely to be in the same chain. We call the results *pivots*; from them we can collect strong evidence such as name, primary phone number, and primary URL domain of chains. The goal is to be robust against erroneous values and make as few false positives as possible, so we can avoid causing incorrect ripple effect later; however, we need to keep in mind that being too strict can miss important strong evidence.

The second stage puts pivots and remaining records into groups according to the discovered strong evidence. It decides whether several pivots belong to the same group, and whether a record not in any pivot actually belongs to some group. It treats weak evidence differently from strong evidence. The intuition is to leverage strong evidence and meanwhile be tolerant to value diversity in the same group, so we can reduce false negatives in the first stage. We illustrate our approach for business-chain linkage as follows.

EXAMPLE 3.3. *Continue with the motivating example. In the first stage we generate three pivots:* $Cr_1 = \{r_1 - r_7\}$, $Cr_2 = \{r_{14}, r_{15}\}$, $Cr_3 = \{r_{16} - r_{18}\}$. *Records* $r_1 - r_7$ *form a pivot because they have the same name, five of them* ($r_1 - r_5$) *share phone number* 808 *and five of them* ($r_3 - r_7$) *share URL* homedepot. *Similar for the other two pivots. Note that* r_{13} *does not belong to any pivot, because one of its URLs is the same as that of* $r_{11} - r_{12}$, *and one is the same as that of* $r_{16} - r_{18}$, *but except name, there are no other common values between these two groups of records. To avoid mistakes, we defer the decision on* r_{13}. *Indeed, recall that* tacocasatexas *is a wrong value for* r_{13}. *For a similar reason, we defer the decision on* r_{12}.

In the second stage, we generate groups – business chains. We merge $r_8 - r_{10}$ *with pivot* Cr_1, *because they have similar names and share either the primary phone number or the primary URL. We also merge* $r_{11} - r_{13}$ *with pivot* Cr_2, *because (1)* $r_{12} - r_{13}$ *share the primary phone* 900 *with* Cr_2, *and (2)* r_{11} *shares the primary URL* tacocasa *with* $r_{12} - r_{13}$. *We do not merge* Cr_2 *and* Cr_3 *though, because they share neither the primary phone nor the primary URL. We do not merge* r_{19} *or* r_{20} *to any pivot, because there is again not much strong evidence. We thus obtain the ideal result.* □

We describe pivot identification in Section 4 and group linkage in Section 5. While we present the algorithms for the setting of one machine, many components of our algorithms can be easily parallelized in the Hadoop infrastructure [3]; we omit the details as it is not the focus of this paper. Before we proceed, we first introduce an important concept used in our algorithms, which typically exists for groups in practice.

3.3 Attribute categorization

To facilitate distinguishing between strong and weak evidence, we classify attributes into three categories, based on different relationship-cardinalities between attributes and groups.

- *Common-value attribute:* We call an attribute A a common-value attribute if there is an $m : 1$ relation between groups and A-values; that is, all entities in the same group have the same or highly similar A-values (*e.g.*, business-name for chain linkage and organization for organization linkage).
- *Primary-value attribute:* We call an attribute A a primary-value attribute if there is a $1 : n$ relation between groups and A-values; that is, entities in the same group often share one or a few primary A-values (but there can also be other values), and these values are seldom used by entities outside the group (*e.g.*, phone and URL-domain for chain, and phone-prefix and email-server for organization).
- *Multi-value attribute:* We call an attribute A multi-value attribute if there is a $m : n$ relation between groups and A-values (*e.g.*, category for chain linkage).

According to the definition, for each attribute A we define three measures to capture its features on labeled datasets. First, for each group Ch and its primary A-value v^A_{Ch} (*i.e.*, the A-value that occurs most often in Ch), we define m^A_{Ch} as the percentage of records in Ch that contain v^A_{Ch}, define s^A_{Ch} as the average similarity between v^A_{Ch} and other A-values in Ch, and define n^A_{Ch} as the percentage of other groups that contain v^A_{Ch}. Then, we compute m^A, s^A, n^A as the average of the top-$k\%$ values; to avoid being biased by small groups, we may filter groups of small sizes. According to the categorization, A is a common-value attribute if s^A is high; A is a primary-value attribute if m^A is high and n^A is low; and A is a multi-value attribute if n^A is high. We decide whether a measure is *high* or *low* using the largest gap between continuous measures.

Finally, we point out that despite the importance of attribute categorization, simply applying traditional entity resolution methods while treating the attributes of different categories differently does not work well (Figure 3).

4. PIVOT LINKAGE

The first stage creates pivots consisting of records that are very likely to be in the same group. To this end, we only consider common-value and primary-value attributes. This section starts with pivot definition (Section 4.1), then describes how we construct similarity graphs to facilitate pivot finding (Section 4.2), and finally gives the algorithm for pivot linkage (Section 4.3).

4.1 Criteria for a Pivot

At the first stage, we wish to make only decisions that are highly likely to be correct. First, we require that each pivot contains as many highly similar records as possible so as not to miss important strong evidence of a group, and different pivots are easily distinguishable from each other. Second, we wish that our results are robust even in the presence of a few erroneous values in the data. In Table 2, $r_1 - r_7$ form a good pivot, because *808* and *homedepot* are popular values among these records. In contrast, $r_{13} - r_{18}$ do not form a good pivot, because records $r_{14} - r_{15}$ and $r_{16} - r_{18}$ do not share any phone number or URL domain; the only "connector" between them is r_{13}, so they can be wrongly merged if r_{13} contains erroneous values. Also, considering $r_{13} - r_{15}$ and $r_{16} - r_{18}$ as two different pivots is risky, because (1) it is not very clear whether r_{13} is in the same chain as $r_{14} - r_{15}$ or as $r_{16} - r_{18}$, and (2) these two pivots share one URL domain name so are not fully distinguishable.

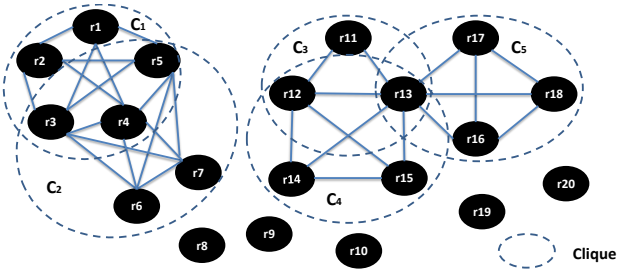


Figure 1: Similarity graph for records in Table 2.

We capture this intuition with *connectivity of a similarity graph*. We define the *similarity graph* of a set $\mathbf{R}$ of records as an undirected graph, where each node represents a record in $\mathbf{R}$, and an edge connects two nodes if they may contain strong evidence indicating a group. We consider two records sharing strong evidence if they agree on common-value attributes and (at least one) primary-value attribute. Note that our techniques are independent of the similarity criteria we apply. Figure 1 shows the similarity graph for Table 2.

Each pivot would correspond to a connected subgraph of the similarity graph. We would like such a subgraph to be *robust* such that even if we remove a few nodes the sub-graph remains connected; intuitively, even if there are some records with erroneous values, without them we still have enough evidence showing that the rest of the records should belong to the same group.

DEFINITION 4.1 (k-ROBUSTNESS). *A graph G is* k-robust *if after removing arbitrary k nodes and edges to these nodes, G is still connected. A clique or a single node is defined to be k-robust for any k.* □

In Figure 1, the subgraph with nodes $r_1 - r_7$ is 2-robust. It is not 3-robust as removing $r_3 - r_5$ can disconnect it.

According to the definition, we can partition the similarity graph into a set of k-robust subgraphs. As we do not wish to split any pivot unnecessarily, we require *maximal k-robust partitioning.*

DEFINITION 4.2 (MAXIMAL k-ROBUST PARTITIONING). *Let G be a similarity graph. A partitioning of G is a* maximal k-robust partitioning *if it satisfies the following properties.*

1. *Each node belongs to one and only one partition.*
2. *Each partition is k-robust.*
3. *The result of merging any partitions is not k-robust.* □

Note that a data set can have more than one maximal k-robust partitioning. Consider $r_{11} - r_{18}$ in Figure 1. There are three maximal 1-robust partitionings: $\{\{r_{11}\}, \{r_{12}, r_{14} - r_{15}\}, \{r_{13}, r_{16} - r_{18}\}\}$; $\{\{r_{11} - r_{12}\}, \{r_{14} - r_{15}\}, \{r_{13}, r_{16} - r_{18}\}\}$; and $\{\{r_{11} - r_{15}\}, \{r_{16} - r_{18}\}\}$. If we treat each partitioning as a possible world, records that belong to the same partition in all possible worlds have high probability to belong to the same group and so form a pivot. So, we define a pivot as follows.

DEFINITION 4.3 (k-PIVOT). *Let $\mathbf{R}$ be a set of records and G be the similarity graph of $\mathbf{R}$. The records that belong to the same subgraph in every maximal k-robust partitioning of G form a* k-pivot *of $\mathbf{R}$. A pivot contains at least 2 records.* □

PROPERTY 4.4. *A k-pivot is k-robust.* □

EXAMPLE 4.5. *Consider Figure 1 and assume $k = 1$. There are two connected sub-graphs. For records $r_1 - r_7$, the subgraph is 1-robust, so they form a 1-pivot. For records $r_{11} - r_{18}$, there are three maximal 1-robust partitionings, as we have shown. Two subsets of records belong to the same subgraph in each partitioning: $\{r_{14} - r_{15}\}$ and $\{r_{16} - r_{18}\}$; they form two 1-pivots.* □

Table 3: Simplified inverted index for the graph in Figure 1.

Record	V-Cliques	Represent
$r_{1/2}$	C_1	$r_1 - r_2$
r_3	C_1, C_2	r_3
r_4	C_1, C_2	r_4
r_5	C_1, C_2	r_5
$r_{6/7}$	C_2	$r_6 - r_7$
r_{11}	C_3	r_{11}
r_{12}	C_3, C_4	r_{12}
r_{13}	C_3, C_4, C_5	r_{13}
$r_{14/15}$	C_4	$r_{14} - r_{15}$
$r_{16/17/18}$	C_5	$r_{16} - r_{18}$

The higher the k, the stronger is the requirement for robustness; when $k = 0$, each connected subgraph is a pivot and the results would be vulnerable to erroneous values. In our motivating example, when $k = 0$, records $r_{11} - r_{18}$ would be wrongly considered as a pivot and thus belonging to the same chain. Our experiments show that a k in the range $[1, 4]$ is the best, improving the F-measure of the results by 10% over $k = 0$. Section 6.2.1 also shows that higher-precision, smaller-sized pivots with significantly lower recall can lead to low F-measure for the chains.

4.2 Constructing Similarity Graphs

Recall that we compare records by common-value and primary-value attribute values. All records agreeing on the common-value attributes and at least one value on a primary-value attribute form a clique, which we call a *v-clique*. We thus represent the similarity graph with a set of v-cliques, denoted by $\mathbf{C}$; for example, the graph in Figure 1 can be represented by five v-cliques ($C_1 - C_5$). In addition, we maintain an *inverted index* $\bar{L}$, where an entry corresponds to a record r and contains the v-cliques that r belongs to.

Given the sheer number of records in $\mathbf{R}$, it is not efficient to compare every pair of records, so we assume a *blocking* method [13] is applied to $\mathbf{R}$ to obtain a set of blocks. Still, the inverted index can be huge. In fact, according to Theorem 4.6, records in only one and the same v-clique belong to the same pivot, so we do not need to distinguish them. Thus, we simplify the inverted index such that for each v-clique we keep only a *representative* for nodes belonging only to this v-clique. Table 3 shows the simplified index for the similarity graph in Figure 1.

THEOREM 4.6. *Let G be a similarity graph and G' be a graph derived from G by merging nodes that belong to one and the same v-clique. Two nodes belong to the same pivot of G' if and only if they belong to the same pivot of G.* □

We can easily find v-cliques and construct the inverted index by scanning values of primary-value attributes. The running time is linear in the number of values from primary-value attributes.

Case study 1: On a data set with 18M records (Section 6), graph construction finished in 1.9 hours. The *original graph* contains 18M nodes and 4.2B edges. The *inverted index* is of size 89MB, containing 3.8M entries, each associated with at most 8 v-cliques; in total there are 1.2M v-cliques. The *simplified inverted index* is of size 34MB, containing 1.5M entries, where an entry can represent up to 11K records. The simplified inverted index reduces the size of the original graph by 3 orders of magnitude.

4.3 Identifying Pivots

We solve the pivot-linkage problem by reducing it to a Max-flow/Min-cut Problem. However, computing the max flow for a given graph G and a source-destination pair takes time $O(|G|^{2.5})$, where $|G|$ denotes the number of nodes in G; even the simplified inverted index can still contain millions of entries, so it can be very expensive. We thus first pre-process graph G in two steps (SCREEN in Section 4.3.1): (1) merging certain v-cliques according to a sufficient (but not necessary) condition for k-robustness and consider them as a whole in pivot linkage; then (2) splitting G into subgraphs according to a necessary (but not sufficient) condition for k-robustness. We find the max flow only on the resulting subgraphs, which are substantially smaller (SPLIT in Section 4.3.2). Section 4.3.3 gives the full algorithm, which iteratively applies SCREEN and SPLIT.

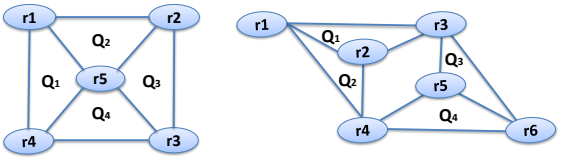


Figure 2: Two example graphs.

4.3.1 Screening

We explain how to reduce the search space for Max-flow by screening. Experiments show that we reduce the input size by 4 orders of magnitude on a real-world data set of 18M records.

A graph can be considered as a union of v-cliques, so essentially we need to decide if a union of v-cliques is k-robust. First, we can prove the following sufficient condition for k-robustness.

THEOREM 4.7 ($(k+1)$-CONNECTED CONDITION). *Let G be a graph consisting of a union Q of v-cliques. If for every pair of v-cliques $C, C' \in Q$, there is a path of v-cliques between C and C' and every pair of adjacent v-cliques on the path share at least $k+1$ nodes, graph G is k-robust.* □

Given Theorem 4.7, we define a $(k+1)$-*connected v-union* as a maximal union of v-cliques that satisfies $(k+1)$-connected condition. A $(k+1)$-connected v-union (simplified as v-union) must be k-robust but not vice versa. In Figure 1, subgraph $\{r_1 - r_7\}$ is a 3-connected v-union, because the only two v-cliques, C_1 and C_2, share 3 nodes. Indeed, it is 2-robust. Graph G_1 in Figure 2 is 2-robust but not 3-connected (there are 4 v-cliques, where each pair of adjacent v-cliques share only 1 or 2 nodes). From Theorem 4.7, we have that a $(k+1)$-connected v-union is k-robust and accordingly a k-pivot. Therefore for pivot linkage we consider graph G as a set of v-unions instead of v-cliques.

Next, we present a necessary condition for k-robustness.

THEOREM 4.8 ($(k+1)$-OVERLAP CONDITION). *Graph G is k-robust only if for every $(k+1)$-connected v-union $Q \in G$, Q shares at least $k+1$ common nodes with the subgraph consisting of the rest of the v-unions.* □

Accordingly, we define a $(k+1)$-*overlap graph* as a graph that satisfies $(k+1)$-overlap condition. A k-robust graph must be a $(k+1)$-overlap graph but not vice versa. In Figure 1, subgraph $\{r_{11} - r_{18}\}$ is not a 2-overlap graph, because there are two 2-connected v-unions, $\{r_{11} - r_{15}\}$ and $\{r_{13}, r_{16} - r_{18}\}$, but they share only one node; indeed, the subgraph is not 1-robust. On the other hand, graph G_2 in Figure 2 satisfies the 3-overlap condition, as it contains four v-unions, $Q_1 - Q_4$, and each v-union shares 3 nodes in total with the others; however, it is not 2-robust (removing r_3 and r_4 disconnects it). According to Theorem 4.8, if graph G is not a $(k+1)$-overlap graph, it cannot be k-robust. Therefore we split G into a set of maximal $(k+1)$-overlap subgraphs, and check k-robustness by Max-flow on each subgraph.

Now the problem is to split G into $(k+1)$-overlap subgraphs. Let G be a graph where a $(k+1)$-connected v-union overlaps with

the rest of the v-unions on no more than k nodes. We split G by removing these overlapping nodes. For subgraph $\{r_{11} - r_{18}\}$ in Figure 1, we remove r_{13} and obtain subgraphs $\{r_{11} - r_{12}, r_{14} - r_{15}\}$ and $\{r_{16} - r_{18}\}$ (recall from Example 4.5 that r_{13} cannot belong to any pivot). Note that the result subgraphs may not be $(k+1)$-overlap graphs (*e.g.*, $\{r_{11} - r_{12}, r_{14} - r_{15}\}$ contains two v-unions sharing one node), so we need to further screen them.

We now describe our screening algorithm, SCREEN, which takes a graph G, represented by a set $\mathbf{C}$ of v-cliques and inverted list $\bar{L}$, as input, finds $(k+1)$-connected v-unions in G and meanwhile decides if G is a $(k+1)$-overlap graph. If not, it splits G into subgraphs for further examination.

1. If G contains a single node, output it as a pivot if it represents records belonging only to one v-clique.
2. For each v-clique $C \in \mathbf{C}$, initialize a v-union $Q(C)$. We denote the set of v-unions by $\bar{Q}$, and the common nodes of C and C' by $\bar{B}(C, C')$.
3. For each v-clique $C \in \mathbf{C}$, we merge v-unions as follows.

   (a) For each unprocessed record $r \in C$, for every pair of v-cliques C_1 and C_2 in r's index entry, if they belong to different v-unions, add r to $\bar{B}(C_1, C_2)$.

   (b) For each v-union $Q \neq Q(C)$ where there exist $C_1 \in Q$ and $C_2 \in Q(C)$ such that $|\bar{B}(C_1, C_2)| \geq k+1$, merge Q and $Q(C)$.

   At the end, $\bar{Q}$ contains all $(k+1)$-connected v-unions.
4. For each v-union $Q \in \bar{Q}$, find its border nodes as $\bar{B}(Q) = \cup_{C \in Q, C' \notin Q} \bar{B}(C, C')$. If $|\bar{B}(Q)| \leq k$, split the subgraph it belongs to, denoted by $G(Q)$, into two subgraphs $Q \setminus \bar{B}(Q)$ and $G(Q) \setminus Q$.
5. Return the remaining subgraphs.

PROPOSITION 4.9. *Denote by $|\bar{L}|$ the number of entries in input $\bar{L}$. Let m be the maximum number of blocks a record belongs to, and a be the maximum number of adjacent v-unions a v-union has. Algorithm* SCREEN *finds $(k+1)$-overlap subgraphs in time $O((m^2 + a) \cdot |\bar{L}|)$ and the result is independent of the order in which we examine the v-cliques.* □

Note that m and a are typically very small, so SCREEN is basically linear in the size of the inverted index. For a similar reason as in Theorem 4.6, we further simplify G by keeping for each v-union a representative for all nodes that only belong to it.

EXAMPLE 4.10. *Consider Table 3 as input and $k = 1$. Step 2 creates five v-unions $Q_1 - Q_5$ for the input v-cliques.*

Step 3 starts with v-clique C_1. It has 4 nodes (in the simplified inverted index), among which 3 are shared with C_2. Thus, $\bar{B}(C_1, C_2) = \{r_3 - r_5\}$ and $|\bar{B}(C_1, C_2)| \geq 2$, so we merge Q_1 and Q_2 into $Q_{1/2}$. Examining C_2 reveals no other shared node.

Step 3 then considers v-clique C_3. It has three nodes, among which $r_{12} - r_{13}$ are shared with C_4 and r_{13} is also shared with C_5. Thus, $\bar{B}(C_3, C_4) = \{r_{12} - r_{13}\}$ and $\bar{B}(C_3, C_5) = \{r_{13}\}$. We merge Q_3 and Q_4 into $Q_{3/4}$. Examining C_4 and C_5 reveals no other shared node. We thus obtain three 2-connected v-unions: $\bar{Q} = \{Q_{1/2}, Q_{3/4}, Q_5\}$.

Step 4 then considers each v-union. For $Q_{1/2}$, $\bar{B}(Q_{1/2}) = \emptyset$ and we thus split subgraph $Q_{1/2}$ out and merge all of its nodes to one $r_{1/\ldots/7}$. For $Q_{3/4}$, $\bar{B}(Q_{3/4}) = \{r_{13}\}$ so $|\bar{B}(Q_{3/4})| < 2$. We split $Q_{3/4}$ out and obtain $\{r_{11} - r_{12}, r_{14/15}\}$ (r_{13} is excluded). Similar for Q_5 and we obtain $\{r_{16/17/18}\}$. Therefore, we return three subgraphs for further screening. □

4.3.2 Reduction

Each result $(k+1)$-overlap subgraph is typically very small, and as we soon show, in practice the majority of them are already k-robust. Thus, in many cases SCREEN is adequate in finding pivots. For completeness, we next briefly describe SPLIT that *guarantees* k-robustness of the pivots.

Between two nodes a, b in G, consider the paths that do not share any node except a and b. We denote the maximal number of such paths between a and b by $\kappa(a, b)$. According to Menger's Theorem [2], $\kappa(a, b)$ is the minimum number of nodes removing which disconnects a and b. Obviously, G is k-robust if $\kappa(a, b) > k$ for any nodes a, b in G. Note that for two nodes a, b in a $(k+1)$-connected v-union, we have $\kappa(a, b) \geq k+1$. Theorem 4.11 further gives k-robustness condition of G on adjacent v-unions in G.

THEOREM 4.11 (k-ROBUSTNESS CONDITION). *Let G be a similarity graph. Graph G is k-robust if and only if for every pair of adjacent $(k+1)$-connected v-unions Q and Q', there exist two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that $\kappa(a, b) > k$.* □

Computing $\kappa(a, b)$ is reduced to a Max-flow Problem in [8], for which we apply the well-known Ford & Fulkerson Algorithm [11]. If a graph G is not k-robust, we shall get the set $\bar{S}$ of nodes that separate G, which we call *separator nodes*. Suppose the separator nodes separate G into $\bar{X}$ and $\bar{Y}$ (there can be more subgraphs). We split G into $\bar{X} \cup \bar{S}$ and $\bar{Y} \cup \bar{S}$ for further processing. Note that we need to include $\bar{S}$ in both sub-graphs to maintain the integrity of each v-union. As proved in [21], the separator nodes do not belong to any pivot, so we mark them as "separators" and eventually exclude them from the returned pivots.

Algorithm SPLIT takes a $(k+1)$-overlap subgraph G as input and decides if G is k-robust. If not, it splits G into subgraphs on which we will then re-apply screening.

1. For each pair of adjacent $(k+1)$-connected v-unions $Q, Q' \in G$, find $a \in Q \setminus Q', b \in Q' \setminus Q$. Apply Ford & Fulkerson Algorithm [11] to compute $\kappa(a, b)$.
2. Once we find nodes a, b where $\kappa(a, b) \leq k$, find separator nodes $\bar{S}$ that separate G. Remove $\bar{S}$ and obtain several subgraphs. Add $\bar{S}$ back to each subgraph and mark $\bar{S}$ as "separators". Return the subgraphs for screening.
3. Otherwise, G is k-robust and output it as a k-pivot.

EXAMPLE 4.12. *Consider graph G_1 in Figure 2 and $k = 2$. There are four 3-connected v-unions (actually four v-cliques) and six pairs of adjacent v-unions. For Q_1 and Q_2, we check nodes r_2 and r_4 and find $\kappa(r_2, r_4) = 3$. Similarly we check for every other pair of adjacent v-unions and decide that the graph is 2-robust.*

Now consider graph G_2 and $k = 2$. There are four 3-connected v-unions. When we check $r_1 \in Q_1$ and $r_6 \in Q_3$, we find $\bar{S} = \{r_3, r_4\}$. We then split G_2 into subgraphs $\{r_1 - r_4\}$ and $\{r_3 - r_6\}$, marking r_3 and r_4 as "separators". □

PROPOSITION 4.13. *Let p be the total number of pairs of adjacent v-unions, and g be the number of nodes in the input graph. Algorithm* SPLIT *runs in time $O(pg^{2.5})$.* □

Despite its high complexity, SPLIT is not expensive in practice as g is very small (**Case Study 2**).

4.3.3 Full Algorithm

We now present the full algorithm, PIVOTLINKAGE (Algorithm 1). It first initializes the working queue $\mathbf{Q}$ with only input G (Line 1). Each time it pops a subgraph G' from $\mathbf{Q}$ and invokes SCREEN

Algorithm 1 PIVOTLINKAGE(G, k)

Input: G: Simplified similarity graph, represented by $\mathbf{C}$ and $\bar{L}$.
k: Robustness requirement.
Output: $\bar{C}$ Set of pivots in G.

```
1: Let Q = {G}, C̄ = ∅;
2: while Q ≠ ∅ do
3:    Pop G' from Q;
4:    Let P̄ = SCREEN(G', k, C̄);
5:    if P̄ = {G'} then
6:       Let S̄ = SPLIT(G', k, C̄);
7:       add graphs in S̄ to Q;
8:    else
9:       add graphs in P̄ to Q;
10:   end if
11: end while
```

Table 4: Step-by-step pivot linkage in Example 4.15.

Input	Method	Output
G	SCREEN	$G^1 = \{r_{1/\ldots/7}\}, G^2 = \{r_{11}, r_{12}, r_{14/15}\}$, $G^3 = \{r_{16/17/18}\}$
G^1	SCREEN	Pivot $\{r_1 - r_7\}$
G^2	SCREEN	$G^4 = \{r_{11}\}, G^5 = \{r_{14/15}\}$
G^3	SCREEN	Pivot $\{r_{16} - r_{18}\}$
G^4	SCREEN	-
G^5	SCREEN	Pivot $\{r_{14} - r_{15}\}$

(Lines 3-4). If the output of SCREEN is still G' (so G' is a $(k+1)$-overlap subgraph) (Line 5), it invokes SPLIT on G' (Line 6). Subgraphs output by SCREEN and SPLIT are added to the queue for further examination (Lines 7, 9) and identified pivots are added to pivot set $\bar{C}$. It terminates when $\mathbf{Q} = \emptyset$.

THEOREM 4.14. *Let G be the input graph and q be the number of v-unions in G. Define a, p, g, m, and $|\bar{L}|$ as in Propositions 4.9 and 4.13. Algorithm* PIVOTLINKAGE *finds correct pivots of G in time $O(q((m^2 + a)|\bar{L}| + pg^{2.5}))$ and is order independent.* □

EXAMPLE 4.15. *Consider the motivating example, with the input shown in Table 3 and $k = 1$. Table 4 shows the step-by-step pivot linkage process. Originally, $\mathbf{Q} = \{G\}$. After invoking* SCREEN*, we obtain three subgraphs G^1, G^2, and G^3.* SCREEN *outputs G^1 and G^3 as 1-pivots since each contains a single node that represents multiple records. It further splits G^2 into two single-node graphs G^4 and G^5, and outputs the latter as a 1-pivot.* □

Case study 2: On the data set with 18M records, we found 3-pivots in 5.6 minutes. SCREEN was invoked 114K times and took 51 seconds (15%) in total. Except the original graph, an input to SCREEN contains at most 39.3K nodes; for 97% inputs there are fewer than 10 nodes and running SCREEN was very fast. SPLIT was invoked only 41 times; an input contains at most 58 nodes (8 v-unions) and on average 10 nodes (2.7 v-unions). Recall that the simplified inverted index contains 1.5M entries, so SCREEN reduced the size of the input to SPLIT by 4 orders of magnitude. Only 1 input graph to SPLIT did not passed the 3-robustness check.

5. GROUP LINKAGE

The second stage clusters the pivots and the remaining records, which together we call *elements*, into groups. We apply an efficient state-of-the-art hill-climbing algorithm for clustering (details in [21]). Note that in this stage we also consider multi-value attributes that may contain *weak* evidence. Comparing with clustering in traditional entity resolution, we make two changes in element-cluster similarity computation. First, in addition to weighting attributes, we weight *attribute values according to their popularities within a group* such that similarity on primary values (strong evidence) is rewarded more. Second, instead of learning weights for each attribute, we treat all primary-value attributes as a whole so that diverse local values in the same group are penalized less. Experiments in Section 6.2.2 show benefits of the two changes.

Collecting strong evidence: We identify popular values within a cluster as strong evidence. When we maintain the signature for a pivot or a cluster, we keep all values of an attribute and assign a high *weight* to a popular value. Specifically, let $\bar{R}$ be a set of records. Consider value v and let $\bar{R}(v) \subseteq \bar{R}$ denote the records in $\bar{R}$ that contain v. The weight of v is computed by $w(v) = \frac{|\bar{R}(v)|}{|\bar{R}|}$.

EXAMPLE 5.1. *Consider* phone *for pivot $Cr_1 = \{r_1 - r_7\}$ in Table 2. There are* 7 *business listings in Cr_1,* 5 *providing* 808 *($r_1 - r_5$), one providing* 101 *(r_6), and one providing* 102 *(r_7). Thus, the weight of* 808 *is $\frac{5}{7} = .71$ and the weight for* 101 *and* 102 *is $\frac{1}{7} = .14$, showing that* 808 *is the primary phone.* □

Allowing diverse values: When we compute the similarity between an element e and a cluster Ch, we consider all primary-value attributes together. To compute primary-value attribute similarity, denoted by $sim_{prm}(e, Ch)$, we reward sharing primary values (values with a high weight) but not penalizing different values, unless there is no shared value. Specifically, if the primary value of e is the same as that of Ch, we consider them having probability p to be in the same group. Since we use weights to measure whether the value is primary and allow slight difference on values, with a value v from e and v' from Ch, the probability becomes $p \cdot w_e(v) \cdot w_{Ch}(v') \cdot s(v, v')$, where $w_e(v)$ measures the weight of v in e, $w_{Ch}(v')$ measures the weight of v' in Ch, and $s(v, v')$ measures the similarity between v and v'.

We compute $sim_{prm}(e, Ch)$ as the probability that they belong to the same group given several shared values.

$$sim_{prm}(e, Ch) = 1 - \prod_{v \in e, v' \in Ch} (1 - p \cdot w_e(v) \cdot w_{Ch}(v') \cdot s(v, v')). \quad (1)$$

When there is no shared primary value, sim_{prm} can be close to 0; once there is one such value, sim_{prm} can be significantly increased, since we typically set a large p.

EXAMPLE 5.2. *Consider element $e = r_8$ and cluster $Ch_1 = \{r_1 - r_7\}$ in Example 1.1, where* phone *and* domain *are primary-value attributes. Assume $p = .9$. Element e and Ch_1 share the primary domain, with weight* 1 *and $\frac{5}{7} = .71$ respectively, but have different phone numbers (assuming similarity of 0). We compute $sim_{prm}(e, Ch_1) = 1 - (1 - .9 \cdot 1 \cdot .71 \cdot 1) \cdot (1 - 0) \cdot (1 - 0) \cdot (1 - 0) = .639$; essentially, we do not penalize the difference in phone numbers. However if domain* homedepot *appeared only once, so was not a primary value, its weight would be* .14 *and accordingly $sim_{prm}(e, Ch_1) = .126$, indicating a much lower similarity.*

If we treat phone *and* domain *separately and compute sim_{prm} as average of sim_{ph} and sim_{dm}, we have $sim_{prm}(e, Ch_1) = \frac{0 + .71}{2} = .355$, also indicating a lower similarity.* □

We then learn weights for each attribute (treating primary-value attributes as a whole), and compute element-cluster similarity as a weighted sum of attribute similarities (details in [21]).

6. EXPERIMENTAL EVALUATION

We experiment on two real-world data sets, showing advantages of our algorithm over rule-based or traditional machine-learning methods on accuracy, and high scalability of our techniques.

Table 5: Statistics of the experimental data sets.

	#Records	#Groups (size > 1)	Group size	#Singletons (size $= 1$)	Level of distinction
BizLow	2446	1	2446	0	low
BizAvg	2062	30	[2, 308]	503	average
BizHigh	1149	14	[33, 269]	0	high
SIGMOD	590	71	[2, 41]	162	average

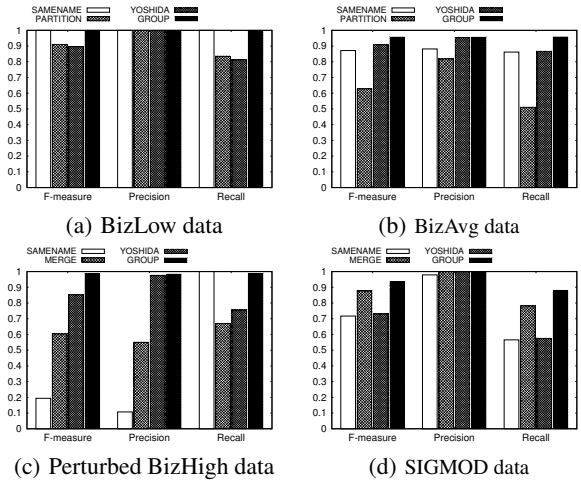
6.1 Experiment Settings

Data and gold standard: We experimented on two real-world data sets. *Biz* contains 18M US business listings and each listing has attributes name, phone, URL, location and category; we decide which listings belong to the same business chain. In lieu of using a social network data set with user privacy issues, we use *SIGMOD*, which contains records of about 590 attendees of SIGMOD'98 and each record has attributes name, affiliation, address, phone, fax and email; we decide which attendees belong to the same institute.

We experimented on the whole *Biz* data set to study scalability of our techniques. We evaluated accuracy of our techniques on four sets of data with different properties (seen in Table 5). The first three are from *Biz*. (1) *BizLow* contains 2446 listings for the same business chain *Allstate Insurance*. These listings have the same name, but 1499 provide URL *"allstate.com"*, 854 provide another URL *"allstateagencies.com"*, while 130 provide both, and 227 listings do not provide any value for phone or URL. (2) *BizAvg* contains 2062 listings from *Biz*, where 1559 belong to 30 randomly selected business chains, and 503 do not belong to any chain; among the 503 listings, 86 are highly similar in name to listings in the business chains and the rest are randomly selected. (3) *BizHigh data set* contains 1149 listings with similar names and highly similar category values; they belong to 14 different chains. Among the listings, 708 provide the same wrong name *Texas Farm Bureau Insurance* and meanwhile provide a wrong *URL farmbureauinsurance-mi.com*. Among these three subsets, the latter two are hard cases; for each data set, we manually verified all the chains by checking store locations provided by the business-chain websites and used it as the gold standard. The last is the whole *SIGMOD* data set. It has very few wrong values, but the same affiliation can be represented in various ways and some affiliation names can be very similar (e.g., *UCSC* vs. *UCSD*). We manually identified 71 institutes with multiple attendees and 162 attendees who do not belong to any of these institutes.

Measure: We considered each group as a cluster and compared pairwise linking decisions with the gold standard. We measured the quality of the results by *precision* (P), *recall* (R), and *F-measure* (F). If we denote the set of true-positive pairs by TP, the set of false-positive pairs by FP, and the set of false-negative pairs by FN, then, $P = \frac{|TP|}{|TP|+|FP|}$, $R = \frac{|TP|}{|TP|+|FN|}$, $F = \frac{2PR}{P+R}$. We also reported execution time.

Implementation: We implemented the technique we proposed in this paper, and call it GROUP. Before applying GROUP, we merge records that share highly similar values on pre-selected attributes (i.e., name, city and address for *biz*). In pivot generation, by default we considered two records are similar for *Biz* if (1) their name similarity is above .95; and (2) they share at least one phone or URL domain name. For *SIGMOD* we require (1) affiliation similarity is above .95; and (2) they share at least one of phone prefix (3-digit), fax prefix (3-digit), or email server. We required 2-robustness for pivots. In clustering, (1) for blocking, we put records whose name similarity is above .8 in the same block; (2) for similarity computation, we computed string similarity by Jaro-Winkler distance [5], we set $p = .8$, and we learned attribute weights from 1000 records randomly selected from *BizAvg* data for *Biz*, and 300 records randomly selected from *SIGMOD*.



(a) BizLow data (b) BizAvg data

(c) Perturbed BizHigh data (d) SIGMOD data

Figure 3: Overall results on *Biz* and *SIGMOD* data sets.

For comparison, we implemented the following baselines:

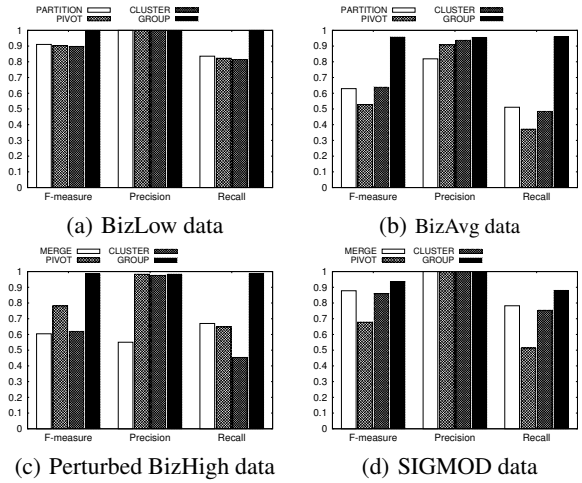
- SAMENAME groups *Biz* records with highly similar names and groups *SIGMOD* records with highly similar affiliations (similarity above .95);
- Traditional machine-learning methods include PARTITION, CENTER and MERGE [15]; each computes record similarity as wighted sum of attribute similarities with learnt attribute weights, and applies a state-of-the-art clustering algorithm.
- Two-stage method YOSHIDA [28] generates pivots by agglomerative clustering with threshold .9 in the first stage, uses TF/IDF weights for features and applies linear algebra to cluster records in the second stage.

We implemented the algorithms in Java. We used a Linux machine with Intel Xeon X5550 processor (2.66GHz, cache 8MB, 6.4GT/s QPI) and 8GB main memory. We used MySQL to store the data and stored the index as a database table. Note that after blocking, we can fit each block of nodes or elements in memory, which is typically the case with a good blocking strategy.

6.2 Evaluating Effectiveness

We first evaluate effectiveness of our algorithms. Figure 3 compares GROUP with the baseline methods, where for the three traditional linkage methods we plot only the best results. On *BizHigh*, all methods put all records in the same chain because a large number (708) of listings have both a wrong name and a wrong URL. We manually perturbed the data as follows: (1) among the 708 listings with wrong URLs, 408 provide a single (wrong) URL and we fixed it; (2) for all records we set name to *"Farm Bureau Insurance"*, so removed hints from business names. Even after perturbing, this data set remains the hardest and we use it hereafter instead of the original one for other experiments.

We observe that (1) GROUP obtains the highest F-measure (above .9) on each data set. It has the highest precision most of the time as it applies pivot identification and leverages the strong evidence collected from resulting pivots. It also has a very high recall (mostly above .95) on each subset because the clustering phase is tolerant to diversity of values within chains. (2) The F-measure of SAMENAME is up to 80% lower than GROUP. It can have false positives when listings of highly similar names belong to different chains and can also have false negatives when some listings in a chain have fairly different names from other listings. It only performs well in the *easiest dataset BizLow*, where it happens that all listings have the same name and belong to the same chain. (3) The



(a) BizLow data (b) BizAvg data

(c) Perturbed BizHigh data (d) SIGMOD data

Figure 4: Contribution of components on *Biz* and *SIGMOD*.

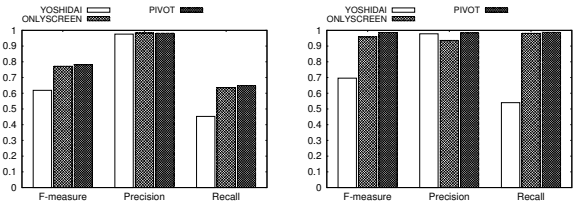
highest F-measure of traditional linkage methods can be 16-41% lower than SAMENAME. It requires higher similarity than sharing name values. As a result, it has a lower recall than SAMENAME. (4) YOSHIDA has comparable precision to GROUP since its first stage is conservative too, which makes it often improve over the best of traditional linkage methods on *Biz* dataset where reducing false positives is a big challenge; on the other hand, its first stage is often too conservative (requiring high record similarity) so the recall is 10-35% lower than GROUP, which also makes it perform worse than traditional linkage methods on *SIGMOD* dataset where reducing false negatives is challenging.

Contribution of different components: We compared GROUP with (1) PIVOT, which conducts only pivot identification, and (2) CLUSTER, which considers each individual record as a pivot (in the spirit of [19, 26]) and conducts only clustering. Figure 4 shows the results. First, we observe that PIVOT improves over traditional linkage methods on precision by up to 79% but has a lower recall (up to 34% lower) most of the time, because it sets a high requirement for merging records into groups. Note however that its goal is indeed to obtain a high precision such that the strong evidence collected from the pivots are trustworthy for the clustering phase. Second, CLUSTER often has higher precision (by up to 77%) but lower recall (by up to 32%) than the best traditional linkage methods; their F-measures are comparable on each data set. On *BizAvg* it can obtain an even higher precision than PIVOT, because PIVOT can make mistakes when too many records have erroneous values, but CLUSTER may avoid some of these mistakes by considering also weak evidence for similarity. However, applying clustering on the results of CLUSTER would not change the results, but applying clustering on the results of PIVOT can obtain a much higher F-measure, especially a higher recall (98% higher than CLUSTER on *BizAvg*). This is because the result of CLUSTER lacks the strong evidence collected from high-quality pivots so the final results would be less tolerant to diversity of values, showing the importance of pivot identification. Finally, we observe that GROUP obtains the best results in most of the data sets.

We next evaluate various choices in the two stages. Unless specified otherwise, we observed similar patterns on all data sets, and report the results on *BizAvg* or perturbed *BizHigh* data, whichever has more distinguishable results.

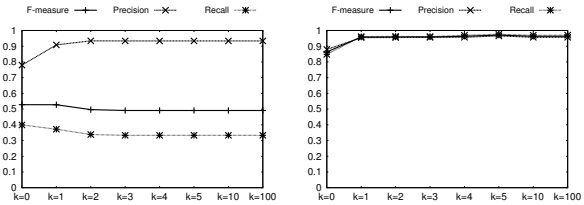
6.2.1 Pivot Identification

Pivot generation: We first compared three pivot-generation strategies: PIVOT iteratively invokes SCREEN and SPLIT, ONLYSCREEN only iteratively invokes SCREEN, and YOSHIDAI generates pivots by agglomerative clustering [28]. Recall that by default we apply PIVOT. Figure 5 compares them on the perturbed *BizHigh* data. First, we observe similar results of ONLYSCREEN and PIVOT on all data sets since most inputs to SPLIT pass the k-robustness test. Thus, although SCREEN in itself cannot guarantee soundness of the resulting pivots, it already does well in practice. Second, YOSHIDAI has lower recall in both pivot and clustering results, since it has stricter criteria in pivot generation.



(a) Pivot quality. (b) Chain quality.

Figure 5: Pivot identification on perturbed *BizHigh* data.



(a) Pivot quality. (b) Chain quality.

Figure 6: Effect of robustness requirement on *BizAvg* data.

Robustness requirement: We next studied how the robustness requirement can affect the results (Figure 6). We have three observations. (1) When $k = 0$, we essentially take every connected subgraph as a pivot, so the generated pivots can have a much lower precision; those false positives cause both a low precision and a low recall for the resulting chains because we do not collect high-quality strong evidence. (2) When we vary k from 1 to 4, the number of false positives decreases while that of false negatives increases for the pivots (Figure 6(a)), and the F-measure of the chains in Figure 6(b) increases but only very slightly. (3) When we continue increasing k, the results of pivots and clusters remain stable. This is because setting k=4 already splits the graph into single v-cliques, so further increasing k would not change the pivots. This shows that considering k-robustness is important, but k does not need to be too high.

Graph generation: We compared three edge-adding strategies for similarity graphs: SIM takes weighted similarity on each attribute except location and requires a similarity of over .8; TWODOM requires sharing name and at least two values on primary-value attributes; ONEDOM requires sharing name and one value on primary-value attributes. Recall that by default we applied ONEDOM. We observe that (1) SIM requires similar records so has a high precision, with a big sacrifice on recall for the pivots (0.00025); as a result, the F-measure of the chains is very low (.59); (2) TWODOM has the strongest requirements and so has even lower recall than SIM for the pivots (.00002), and in turn it has the lowest F-measure for the chains (.52). This shows that only requiring high precision for pivots with big sacrifice on recall can also lead to low F-measure for the chains.

6.2.2 Clustering

Clustering strategy: We first compared our clustering algorithm with two algorithms proposed for the second stage of two-stage clustering: LIUII [22] iteratively applies majority voting to assign each record to a cluster and collects a set of representative features

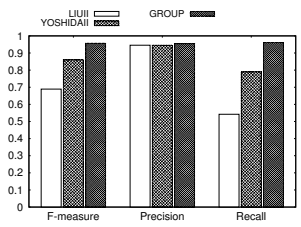


Figure 7: Clustering strategies on *BizAvg* data.

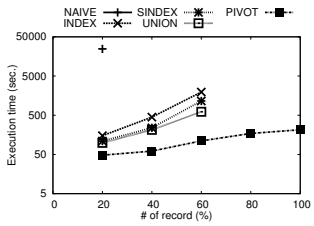


Figure 8: Execution time of pivot generation(we plot only those below 10 hours).

for each cluster using a threshold (we set it to 5, which leads to the best results); YOSHIDAII [28] is the second stage of YOSHIDA. Figure 7 compares their results. We observe that our clustering method improves the recall by 39% over LIUII and by 11% over YOSHIDAII. LIUII may filter strong evidence by the threshold; YOSHIDAII cannot handle records with null values well.

Value weight: We then compared the results with and without setting popularity weights for values. We observe that setting the popularity weight helps distinguish primary values from unpopular values, thus can improve the precision. Indeed, on perturbed *BizHigh* data it improves the precision from .11 to .98, and improves the F-measure by 403%.

Attribute weight: We next considered our attribute weight learning strategy. We first compared SEPARATEDPRIMARY, which learns separated weights for different primary-value attributes, and UNITEDPRIMARY (our default), which considers all such attributes as a whole and learns one single weight for them. On *BizAvg* the latter improves over the former by 95% on recall and obtains slightly higher precision, because it penalizes only if neither phone nor URL is shared and so is more tolerant to different values for primary-value attributes.

Robustness w.r.t. parameter p: We also ran experiments to test robustness against parameter setting. We observed very similar results when we ranged p from .8 to 1.

6.2.3 Preprocessing

Attribute categorization: We studied attribute categorization on both data sets. For *Biz* we used a labeled data set of 2062 records and identified name as a common-value attribute ($m = .94, s = .85, n = .001$); for *SIGMOD* we used a labeled data set of 369 records and identified affiliation as a common-value attribute ($m = .77, s = .85, n = 0$).

Then for *Biz* we used a set of randomly selected 166, 236 records and for *SIGMOD* we used all records. We considered only groups of size above 10; we averaged m, s, n among top 1% values for *Biz* and among all values for *SIGMOD*. For *Biz* data, we identify URL domain and phone as primary-value attributes, and state and category as multi-value attributes. For *SIGMOD* data, we identify phone-prefix, fax-prefix and email-domain as primary-value attributes, and state as a multi-value attribute.

Entity resolution vs group linkage: We studied the interaction between entity resolution and group linkage on *BizAvg*. We compared three strategies: NO-ER does not conduct entity resolution; PRE-ER (the default) conducts entity resolution before group linkage; FULL-ER first conducts entity resolution (merging records with highly similar name, address, city and the same phone or URL domain), then conducts group linkage, while removing duplicates within pivots or groups if they have highly similar address and city. We have three observations. First, the three approaches obtain similar results (F-measure all above .95). Second, applying entity resolution slightly improves group linkage (F-measure from .9639 to .9643). Third, among the 2062 records in *BizAvg*, PRE-ER identifies 21 pairs of duplicates while FULL-ER identifies 59 pairs, showing that group linkage can improve entity resolution too.

6.3 Evaluating Efficiency

Our algorithm finished in 8.3 hours on *Biz* data set with 18M listings on a single machine. Note that simpler methods (which we describe shortly) took over 10 hours for Stage I on 20% of the same data set. Also note that using Hadoop can reduce execution time for graph construction from 1.9 hours to 37 minutes.

Stage I: It spent 1.9 hours for graph construction and 2.2 minutes for pivot generation. To test scalability, we randomly divided the data set into five subsets of the same size; we started with one subset and gradually added the others. We compared five pivot generation methods: NAIVE applies SPLIT on the original graph; INDEX optimizes NAIVE by using an inverted index; SINDEX simplifies the inverted list by Theorem 4.6; UNION in addition merges v-cliques into v-unions by Theorem 4.7; PIVOT in addition splits the input graph by Theorem 4.8. Figure 8 shows the results and we have five observations. (1) NAIVE was very slow. Even though it applies SPLIT rather than finding the max flow for every pair of nodes, so already optimizes by Theorem 4.11, it took 6.8 hours on only 20% data and took more than 10 hours on 40% data. (2) INDEX improved NAIVE by two orders of magnitude just because the index simplifies finding neighborhood v-cliques; however, it still took more than 10 hours on 80% data. (3) SINDEX improved INDEX by 41% on 60% data as it reduces the size of the inverted index by 64%. (4) UNION improved SINDEX by 47% on 60% data; however, it also took more than 10 hours on 80% data. (5) PIVOT improved UNION significantly; it finished in 2.2 minutes on the whole data set so further reduced execution time by at least three orders of magnitude, showing importance of splitting.

Stage II: After Stage I we have .7M pivots and 17.3M remaining records. It spent 6.4 hours for Stage II: 1.7 hours for blocking and 4.7 hours for clustering. The long time for clustering is because of the huge number of blocks. There are 1.4M blocks with multiple elements (a pivot is counted as one element), with a maximum size of 22.5K and an average of 4.2. On only 35 blocks clustering took more than 1 minute and the maximum is 2.5 minutes, but for 99.6% blocks the size is less than 100 and CLUSTER took less than 60 ms. The average time spent on each block is only 9.6 ms.

6.4 Summary and Recommendations

We summarize our observations as follows.

1. Identifying pivots and leveraging strong evidence learned from the pivots is crucial in group linkage.
2. There are often erroneous values in real-world data and it is important to be robust against them; requiring k-robustness with $k \in [1, 4]$ already performs well on most data sets that have reasonable number of errors.
3. Setting weights of values according to their popularity is critical for obtaining good clustering results.
4. Our algorithm is efficient and scalable.

7. CONCLUSIONS

In this paper we studied how to link records to identify groups. We proposed a robust algorithm that is shown to be empirically scalable and accurate over two real-world data sets. Future work includes extending our techniques to find overlapping groups, and applying our framework in other contexts where tolerance to value diversity is critical, and erroneous data is prevalent.

8. REFERENCES

[1] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking stable clusters in the blogosphere. In *VLDB*, pages 806–817, 2007.

[2] H. Bruhn, R. Diestel, and M. Stein. Menger's theorem for infinite graphs with ends. *J. Graph Theory*, 50:199–211, November 2005.

[3] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. Flumejava: easy, efficient data-parallel pipelines. In *PLDI*, pages 363–375, 2010.

[4] Y.-H. Chiang, A. Doan, and J. F. Naughton. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB*, 7(6):469–480, 2014.

[5] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB*, pages 73–78, 2003.

[6] D. Dey. Entity matching in heterogeneous databases: A logistic regression approach. *Decis. Support Syst.*, 44:740–747, 2008.

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

[8] S. Even and E. R. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, 1975.

[9] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.

[10] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the Americal Statistical Association*, 64(328):1183–1210, 1969.

[11] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[12] L. Getoor and C. P. Diehl. Link mining: A survey. *SIGKDD Explor. Newsl.*, 7(2):3–12, 2005.

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

[14] S. Guo, X. Dong, D. Srivastava, and R. Zajac. Record linkage with uniqueness constraints and erroneous values. *PVLDB*, 3(1):417–428, 2010.

[15] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, pages 1282–1293, 2009.

[16] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998.

[17] S. Huang. Mixed group discovery: Incorporating group linkage with alternatively consistent social network analysis. *International Conference on Semantic Computing*, 0:369–376, 2010.

[18] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.

[19] B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, pages 16–22, 1999.

[20] P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.

[21] P. Li, X. Luna Dong, S. Guo, A. Maurino, and D. Srivastava. Robust Group Linkage. http://arxiv.org/abs/1503.00604, Mar. 2015.

[22] X. Liu, Y. Gong, W. Xu, and S. Zhu. Document clustering with cluster refinement and model selection capabilities. In *SIGIR*, pages 191–198, 2002.

[23] B. W. On, N. Koudas, D. Lee, and D. Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007.

[24] B. Taskar, M. fai Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Advances in Neural Information Processing Systems*, 2003.

[25] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[26] D. T. Wijaya and S. Bressan. Ricochet: A family of unconstrained algorithms for graph clustering. In *DASFAA*, pages 153–167, 2009.

[27] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, U.S. Bureau of the Census, 2002.

[28] M. Yoshida, M. Ikeda, S. Ono, I. Sato, and H. Nakagawa. Person name disambiguation by bootstrapping. In *SIGIR*, pages 10–17, 2010.