Behavior Based Record Linkage *

Mohamed Yakout[1] Ahmed K. Elmagarmid[12] Hazem Elmeleegy[1] Mourad Ouzzani[1]
Alan Qi[1]
[1]Purdue University – West Lafayette, IN, USA
[2]Qatar Computing Research Institute, Qatar Foundation – Doha, Qatar
{myakout, ake, hazem, mourad, alanqi}@cs.purdue.edu

ABSTRACT

In this paper, we present a new record linkage approach that uses *entity behavior* to decide if potentially different entities are in fact the same. An entity's behavior is extracted from a transaction log that records the *actions* of this entity with respect to a given data source. The core of our approach is a technique that merges the behavior of two possible matched entities and computes the *gain* in recognizing behavior patterns as their matching score. The idea is that if we obtain a well recognized behavior after merge, then most likely, the original two behaviors belong to the same entity as the behavior becomes more complete after the merge. We present the necessary algorithms to model entities' behavior and compute a matching score for them. To improve the computational efficiency of our approach, we precede the actual matching phase with a fast candidate generation that uses a "quick and dirty" matching method. Extensive experiments on real data show that our approach can significantly enhance record linkage quality while being practical for large transaction logs.

1. INTRODUCTION

Record linkage is the process of identifying records that refer to the same real world entity. There has been a large body of research on this topic (refer to [10] for a recent survey). While most existing record linkage techniques focus on simple attribute similarities, more recent techniques are considering richer information extracted from the raw data for enhancing the matching process (e.g. [3, 15, 8, 6]).

In contrast to most existing techniques, we are considering *entity behavior* as a new source of information to enhance the record linkage quality. We observe that by interpreting massive transactional datasets, for example, transaction logs, we can discover behavior patterns and identify entities based on these patterns. Various applications such as retail stores, web sites, and surveillance systems, maintain transaction logs that track the actions performed by entities over time. Entities in these applications will usually perform actions, e.g., buying a specific quantity of milk at a specific point in time or browsing specific pages within a web site, which represent their behavior vis-à-vis the system.

To further motivate the importance of using the behavior for record linkage, consider the following real-life example. Yahoo has recently acquired a Jordanian Internet company called Maktoob, which, similar to Yahoo, provides a large number of Internet services to its customers in the region like e-mail, blogs, news, and online shopping. It was reported that with this acquisition, Yahoo will be able to add the 16 million Maktoob users to its 20 million users from the middle east region[1]. Clearly, Yahoo should expect that the overlap between these two groups of users can be quite significant, and hence the strong need for record linkage. However, user profile information stored by both companies may not be reliable enough because of different languages, unreal information, ...etc. In this scenario, analyzing the users behavior, in terms of how they use the different Internet services, will be an invaluable source of information to identify potentially common users. Record linkage analysis based on entity behavior has also many other applications. For example, identifying common customers for stores that are considering a merge, tracking users accessing web sites from different IP addresses, as well as helping in crime investigations.

A seemingly straightforward strategy to match two entities is to measure the *similarity* between their behaviors. However, a closer examination shows that this strategy may not be useful, for the following reasons. It is usually the case that the complete knowledge of an entity's behavior is not available to both sources, since each source is only aware of the entity's interaction with that same source. Hence, the comparison of entities' "behaviors" will in reality be a comparison of their "partial behaviors", which can easily be misleading. Moreover, even in the rare case when both sources have almost complete knowledge about the behavior of a given entity (e.g., a customer who did all his grocery shopping at Walmart for one year and then at Safeway for another year), the similarity strategy still will not help. The problem is that many entities do have very similar behaviors, and hence measuring the similarity can at best group the entities with similar behavior together (e.g., [21, 13, 1]), but not find their unique matches.

Fortunately, we developed an alternative strategy that works well even if complete behavior knowledge is not known to both sources. The key to our proposed strategy is that we *merge* the behavior information for each candidate pair of entities to be matched. If the two behaviors seem to *complete* one another, in the sense that stronger behavioral patterns become detectable after the merge, then this will be a strong indication that the two entities are, in fact, the same. The problem of distinct entities having similar overall behavior is also handled by the merge strategy, especially when their behaviors are split across the two sources with different splitting patterns (e.g., 20%-80% versus 60%-40%). In this case, two behaviors (from the first and second sources) will complete each other if they indeed correspond to the same real world entity, and not just two distinct entities who happen to share a similar behavior (which is one of the shortcomings of the similarity strategy).

In this paper, we develop principled computational algorithms to detect those behavior patterns which correspond to latent unique entities in *merged logs*. We compute the *gain* in recognizing a behavior before and after merging the entities transactions and use this gain as a matching score. In our empirical studies with real world data sets, the behavior merge strategy produced much better results

*This research was supported by NSF Grant Numbers IIS 0916614 and IIS 0811954, and by Purdue Cyber Center.

[1]http://www.techcrunch.com/2009/08/25/confirmed-yahoo-acquires-arab-internet-portal-maktoob/

than the behavior similarity strategy in different scenarios of splitting the entities' transactions among the data sources.

The contributions of this paper are:

- We present the first formulation of the record linkage problem using entity behavior and solve the problem by detecting consistent repeated patterns in merged transaction logs.
- To model entities' behavior, we develop an accurate, principled detection approach that models the statistical variations in the repeated behavior patterns and estimates them via expectation maximization [7].
- We present an alternative, more computationally efficient, detection technique that is based on information theory which detects recognized patterns through high compressibility.
- To speed up the linkage process, we propose a filtering procedure that produces candidate matches for the above detection algorithms through a fast but inaccurate matching. This filtering introduces a novel "winnowing" mechanism for zeroing in a small set of candidate pairs with few false positives and almost no false negatives.
- We conduct an extensive experimental study on real world datasets that demonstrates the effectiveness of our approach to enhance the linkage quality.

The rest of the paper is organized as follows; we discuss the problem and overview our approach in Section 2. Section 3 presents the candidates generation phase and Section 4 describes the accurate matching phase. The experiments are discussed in Section 5. Section 6 contains the related work and we conclude in section 7.

2. BEHAVIOR BASED APPROACH

2.1 Problem Statement

We are given two sets of entities $\{A_1, \ldots, A_{N_1}\}$ and $\{B_1, \ldots, B_{N_2}\}$, where for each entity A we have a transaction log $\{T_1, \ldots, T_{n_A}\}$ and each transaction T_i is a tuple in the form of $\langle t_i, a, F\_id \rangle$ where t_i represents the time of the transaction, a is the action (or event) that took place, and F_id refers to the set of features that describe how action a was performed.

Our goal is to return the most likely matches between entities from the two sets in the form of $\langle A_i, B_j, S_m(A_i, B_j) \rangle$, where $S_m(A_i, B_j)$ is the matching function. Given entities A, B (and their transactions), the matching function returns a score reflecting to what extent the transactions of both A and B correspond to the same entity.

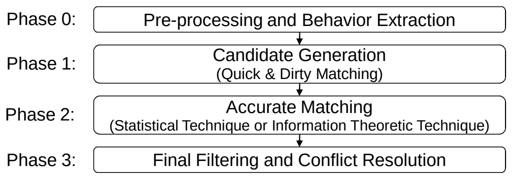
2.2 Approach Overview

We begin by giving an overview of our approach for record linkage, which can be summarized by the process depicted in Figure 1.

Phase 0: In the initial *pre-processing and behavior extraction* phase, we transform raw transaction logs from both sources into a standard format as shown on the right side of Figure 2. Next, we extract the behavior data for each single entity in each log. Behavior data is initially represented in a matrix format similar to those given in Figure 3, which we refer to as Behavior Matrix (BM).

Phases 1: Similar to most record linkage techniques, we start with a *candidate generation* phase that uses a "quick and dirty" matching function. When matching a pair of entities, we follow the *merge* strategy described in the introduction. Moreover, in this phase, we map each row in the BM to a 2-dimensional point resulting in a *very compact* representation for the behavior with some information loss. This mapping allows for very fast computations on the behavior data of both the original and merged entities. The mapping is discussed in Section 3 and we will show how we can use it to generate a relatively small set of candidate matches (with almost no false negatives).

It is worth mentioning that in some scenarios, and depending on the domain knowledge, more techniques can be applied to further discard candidate matches in this phase. For example, two customers in the shopping scenario should be deemed "un-mergeable" if they happened to shop in two different stores exactly at the same time. In our experiments, we did not rely on any such techniques to ensure the generality of the results.



Phase 0:
Pre-processing and Behavior Extraction
Phase 1:
Candidate Generation
(Quick & Dirty Matching)
Phase 2:
Accurate Matching
(Statistical Technique or Information Theoretic Technique)
Phase 3:
Final Filtering and Conflict Resolution

Figure 1: Process for behavior-based record linkage.

Phase 2: The core of our approach is to perform the *accurate* (yet more expensive) *matching* of entities. Accurate matching of the candidate pair of entities (A, B) is achieved by first modeling the behavior of entities A, B, and AB using a *statistical generative model*, where AB is the entity representing the merge of A, B. The estimated models' parameters are then used to compute the matching score.

In addition to the above statistical modeling technique, we also propose an alternative heuristic technique that is based on *information theoretic* principles for the accurate matching phase (See Appendix B). This alternative technique relies on measuring the increase in the level of *compressibility* as we merge the behavior data of pairs of entities. While, to some extent, it is less accurate than the statistical technique, it is computationally more efficient.

Phase 3: The *final filtering and conflict resolution* phase is where the final matches are selected. In our experiments, a simple filtering threshold, t_f, is applied to exclude low-scoring matches. To further resolve conflicting matches, more involved techniques such as *stable marriage* [11] can be used. However, the ultimate goal of this paper is to assign a matching score to each pair of entities and report the matches with the highest scores. Due to space limitations, we reported our experimental results without using any conflict resolution technique, although when used, we were able to gain at least 5% more accuracy over the reported results.

In the remainder of this section, we will first examine the details of phase 0, and then we will give an introduction to phases 1 and 2, whose detailed discussions will be presented in the following two sections.

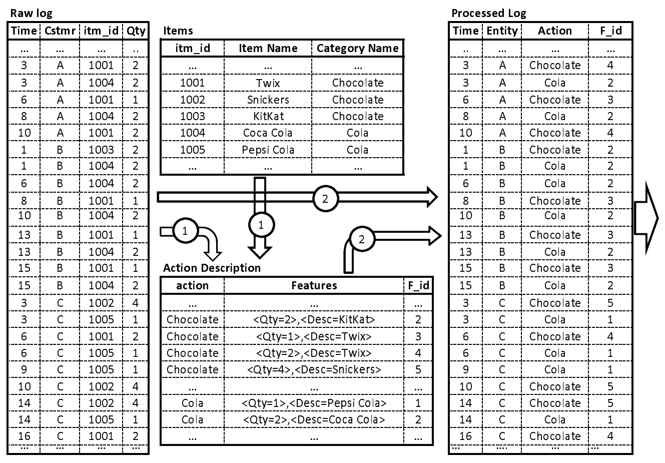
2.3 Pre-processing and Behavior Extraction

A transaction log, from any domain, would typically keep track of certain types of information for each action an entity performs. This information includes: (1) the time at which the action occurred, (2) the key object upon which the action was performed (e.g., buying a `Twix` bar), and (3) additional detailed information describing the object and how the action was performed (e.g., quantity, payment method, etc). For simplicity, we will be referring to each action just by its key object. For example, "`Twix`" can be used to refer to the action of buying a `Twix` bar.

The following example illustrates how we can transform a raw transaction log into a standard format with such information. Although the example is from retail stores, the same steps can be applied in other domains with the help of domain experts.

Example 1. An example of a raw log is shown in table "Raw log" in Figure 2 which has four columns representing the time, the customer (the entity to be matched), the ID of the item bought by the customer, and the quantity. Since the item name may be too specific to be the key identifier for the customer's buying behavior, an alternative is to use the item category name as the identifier for the different actions. This way, actions will correspond to buying `Chocolate` and `Cola` rather than `Twix` and `Coca Cola`. The main reason behind this generalization is that, for instance, buying a bar of `Twix` should not be considered as a completely different action from buying a bar of `Snickers`, and so on. In general, these decisions can be made by a domain expert to avoid over-fitting when modeling the behavior. In this case, the specific item name, along with the quantity, will be considered as additional detailed information, which we will refer to as the action *features*.

The next step is to assign an id, `F_id`, for each combination of



Raw log

Time	Cstmr	itm_id	Qty
...	...	...	..
3	A	1001	2
3	A	1004	2
6	A	1001	1
8	A	1004	2
10	A	1001	2
1	B	1003	2
1	B	1004	2
6	B	1004	2
8	B	1001	1
10	B	1004	2
13	B	1001	1
13	B	1004	2
15	B	1001	1
15	B	1004	2
3	C	1002	4
3	C	1005	1
6	C	1001	2
6	C	1005	1
9	C	1005	1
10	C	1002	4
14	C	1002	4
14	C	1005	1
16	C	1001	2
...	...	...	...

Items

itm_id	Item Name	Category Name
...	...	...
1001	Twix	Chocolate
1002	Snickers	Chocolate
1003	KitKat	Chocolate
1004	Coca Cola	Cola
1005	Pepsi Cola	Cola
...	...	...

Action Description

action	Features	F_id
...	...	...
Chocolate	<Qty=2>,<Desc=KitKat>	2
Chocolate	<Qty=1>,<Desc=Twix>	3
Chocolate	<Qty=2>,<Desc=Twix>	4
Chocolate	<Qty=4>,<Desc=Snickers>	5
...	...	...
Cola	<Qty=1>,<Desc=Pepsi Cola>	1
Cola	<Qty=2>,<Desc=Coca Cola>	2
...	...	...

Processed Log

Time	Entity	Action	F_id
..	...	...	...
3	A	Chocolate	4
3	A	Cola	2
6	A	Chocolate	3
8	A	Cola	2
10	A	Chocolate	4
1	B	Chocolate	2
1	B	Cola	2
6	B	Cola	2
8	B	Chocolate	3
10	B	Cola	2
13	B	Chocolate	3
13	B	Cola	2
15	B	Chocolate	3
15	B	Cola	2
3	C	Chocolate	5
3	C	Cola	1
6	C	Chocolate	4
6	C	Cola	1
9	C	Cola	1
10	C	Chocolate	5
14	C	Chocolate	5
14	C	Cola	1
16	C	Chocolate	4
...	....	...	...

Figure 2: Raw log pre-processing example: The first step is to decide on the action identifiers and the features describing each action to create the "Action Description" table. The second step is to use the identified actions to re-write the log.

features occurring with a specific action in "Raw Log", as shown in the "Action Description" table. This step ensures that even if we have multiple features, we can always reason about them as a single object using `F_id`. If there is only one feature, then it can be used directly with no need for `F_id`.

As a final step, we generate the "Processed Log" by scanning "Raw Log" and registering the time, entity, action, and `F_id` information for each line. □

Behavior Extraction and Representation: Given the standardized log, we extract the transactions of each entity and represent them in a matrix format, called *Behavior Matrix*.

DEFINITION 1. *Given a finite set of n actions performed over m time units by an entity A, the Behavior Matrix (BM) of entity A is an $n \times m$ matrix, such that:*

$$BM_{i,j} = \begin{cases} \mathcal{F}_{ij} & \textit{if action } a_i \textit{ is performed} \\ 0 & \textit{otherwise} \end{cases}$$

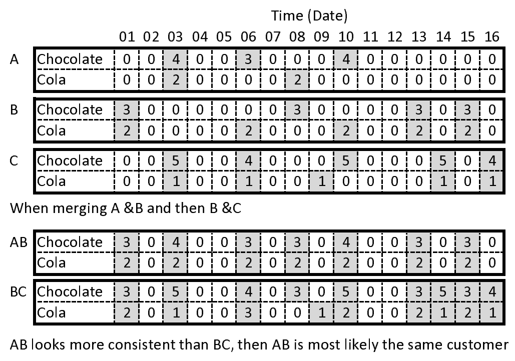
Where, $\mathcal{F}_{ij} \in \mathbf{F}_i$ is the `F_id` value for the combination of features describing action a_i when performed at time j, $\mathbf{F}_i$ is the domain of all possible `F_id` values for action a_i ,$i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Example 2. The BMs for customers A, B and C are shown in Figure 3. A non-zero value indicates that the action was performed and the value itself is the `F_id` that links to the description of the action at this time instant.□

A more compact representation for the entities' behavior is derived from the Behavior Matrix representation, and is constructed and used during the accurate matching phase. This second representation, which is based on the inter-arrival times, considers each row in the BM as a stream or sequence of pairs $\{v_{ij}, \mathcal{F}^{(v_{ij})}\}$, where v_{ij} is the inter-arrival time since the last time action a_i occurred, and $\mathcal{F}^{(v_{ij})} \in \mathbf{F}_i$ is a feature that describes a_i from L_{a_i} possible descriptions, $|\mathbf{F}_i| = L_{a_i}$. For example, in Figure 3, the row corresponding to action a_i = `chocolate` of entity C, $BM_i = \{0, 0, 5, 0, 0, 4, 0, 0, 0, 5, 0, 0, 0, 5, 0, 4\}$, will be represented as $\mathbf{X}_i = \{\{3, 5\}, \{3, 4\}, \{4, 5\}, \{4, 5\}, \{2, 4\}\}$.

The lossy behavior representation used in the candidate generation phase will be described in Section 3.

It is worth mentioning that the actions, along with their level of details (e.g., buying chocolate vs. buying `Twix`) and their associated features, are assumed to be *homogeneous* across the two sources. Otherwise, another pre-processing phase will be required to match the actions, and thereby ensure the homogeneity. Needless to say, the sources themselves must belong to the same domain (e.g., two grocery stores, two news web sites, etc) for the behavior-based approach to be meaningful.



Time (Date)

		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
A	Chocolate	0	0	4	0	0	3	0	0	0	4	0	0	0	0	0	0
	Cola	0	0	2	0	0	0	0	2	0	0	0	0	0	0	0	0
B	Chocolate	3	0	0	0	0	0	0	3	0	0	0	0	3	0	3	0
	Cola	2	0	0	0	0	2	0	0	0	2	0	0	2	0	2	0
C	Chocolate	0	0	5	0	0	4	0	0	0	5	0	0	0	5	0	4
	Cola	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	1

When merging A &B and then B &C

		01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
AB	Chocolate	3	0	4	0	0	3	0	3	0	4	0	0	3	0	3	0
	Cola	2	0	2	0	0	2	0	2	0	2	0	0	2	0	2	0
BC	Chocolate	3	0	5	0	0	4	0	3	0	5	0	0	3	5	3	4
	Cola	2	0	1	0	0	3	0	0	1	2	0	0	2	1	2	1

AB looks more consistent than BC, then AB is most likely the same customer

Figure 3: Resulting Behavior Matrices from the processed log.

2.4 Matching Strategy

As we explained in Section 2.2, matching entities based on their extracted behavior data is achieved in two consecutive phases: a candidate generation phase followed by an accurate matching phase. In this section, we describe our general matching strategy, which we apply in the two matching phases. Note that ultimately, we need to assign a matching score, S_m, for each pair of entities (A, B) deemed as a potential match, and then report the matches with the highest scores.

To compute $S_m(A, B)$, we first compute a *behavior recognition* score, $\mathcal{S}_r$, for each entity (i.e., $\mathcal{S}_r(A)$ and $\mathcal{S}_r(B)$). We then *merge* the behavior data of both A and B to construct the behavior of some hypothetical entity AB, whose score, $\mathcal{S}_r(AB)$, is also computed.

The next step is to check if this merge results in a more recognizable behavior compared to either of the two individual behaviors. Hence, the overall matching score should depend on the *gain* achieved for the recognition scores. More precisely, it can be stated as follows:

$$S_m(A, B) = \frac{n_A[\mathcal{S}_r(AB) - \mathcal{S}_r(A)] + n_B[\mathcal{S}_r(AB) - \mathcal{S}_r(B)]}{n_A + n_B} \tag{1}$$

where n_A and n_B are the total number of transactions in the BMs of A and B respectively. Note that the gains corresponding to the two entities are weighted based on the density of their respective BMs.

Example 3. To better understand the intuition behind the behavior merge strategy, we assume that entities A and C are from Source 1 and B is from Source 2 and their processed log is shown in table "Processed Log" in Figure 2. To find the best match for entity B, we first merge it with A, and then do the same with C. It is apparent from the resulting BMs in Figure 3 that A is potentially a good match for B; entity AB is likely to be an entity that buys chocolate every 2 or 3 days and prefers 2 liters of `Coca Cola` with either 2 bars `Twix` or 4 bars `Snikers` chocolates. However, it is hard to tell a behavior about entity BC. Of course, in a real scenario we will deal with much more actions. □

The key question now is: How to compute $\mathcal{S}_r(A)$? In fact, the goal of the recognition score, $\mathcal{S}_r$, is to capture the consistency of an entity's behavior along three main components: (1) consistency in repeating actions, (2) stability in the features describing the action, and (3) the association between actions. These three components, which will be explained shortly, correspond to three score components of $\mathcal{S}_r$; i.e., S_{r1}, S_{r2}, and S_{r3}. We compute $\mathcal{S}_r(A)$ as their geometric mean as given below.

$$\mathcal{S}_r(A) = \sqrt[3]{S_{r1}(A) \times S_{r2}(A) \times S_{r3}(A)} \tag{2}$$

1- Consistency in repeating actions: Entities tend to repeat specific actions on a regular basis following almost consistent inter-arrival times. For example, a user (entity) of a news web site may be checking the financial news (action) every morning (pattern).

2- Stability in the features describing actions: When an entity performs an action several times, almost the same features are expected to apply each time. For example, when a customer buys chocolate, s/he mostly buys either 2 `Twix` bars or 1 `Snickers` bar, as opposed to buying a different type of chocolate each time and in completely different quantities. The latter case is unlikely to occur in real scenarios.

3- Association between actions: Actions performed by entities are typically associated to each other, and the association patterns can be detected over time. For example, a customer may be used to buying `Twix` chocolate and `Pepsi` cola every Sunday afternoon, which implies an association between these two actions.

A major distinction between the matching techniques that we will describe next is in the method used to compute S_{r1}, S_{r2}, and S_{r3}. The candidate generation phase is a special case as it only considers the first behavior component; i.e. $\mathcal{S}_r(A) = S_{r1}(A)$.

The matching strategy we have described so far can be referred to as the **behavior merge** strategy, since it relies essentially on merging the entities' behaviors and then measuring the realized gain. This is to be contrasted to an alternative strategy, which can be referred to as the **behavior similarity** strategy, where the matching score can simply be a measure of the similarity between the two behaviors.

3. CANDIDATE GENERATION PHASE

To avoid examining all possible pairs of entities during the expensive phase of accurate matching, we introduce a candidate generation phase, which *quickly* determines pairs of entities that are likely to be matched. This phase results in almost no false negatives, at the expense of relatively low precision.

The high efficiency of this phase is primarily due to the use of a very compact (yet lossy) behavior representation, which allows for fast computations. In addition, only the first behavior component; i.e., consistency in repeating actions, which is captured by S_{r1}, is considered in this phase. Note that because the two other components are ignored, binary BMs are used with 1's replacing non-zero values.

Each row in the BM, which corresponds to an action, is considered as a binary time sequence. For each such sequence, we compute the first element of its Discrete Fourier Transform (DFT) [22], which is a 2-dimensional complex number. The complex number corresponding to an action a_i in the BM of an entity A is computed by:

$$C_A^{(a_i)} = \sum_{j=0}^{m-1} BM_{i,j} e^{\frac{2j\pi\sqrt{-1}}{m}} \tag{3}$$

An interesting property of this transformation is that the lower the magnitude of the complex number, the more consistent and regular the time sequence, and vice versa. This can be explained as follows. Consider each of the elements in the time series as a vector whose magnitude is either 0 or 1, and that their angles are uniformly distributed along the unit circle (i.e., the angle of the j^{th} vector is $\frac{2j\pi}{m}$). The complex number will then be the resultant of all these vectors. Now, if the time series was consistent in terms of the inter-arrival times between the non-zero values, then their corresponding vectors would be uniformly distributed along the unit circle, and hence they would cancel each other out. Thus, the resultant's magnitude will be close to zero.

Another interesting property is that merging the two rows corresponding to an action a in the BMs of two entities, A, B, is reduced to adding two complex numbers i.e., $C_{AB}^{(a)} = C_A^{(a)} + C_B^{(a)}$.

The following example shows how the candidate generation phase can distinguish between "”match” and “mismatch” candidates.

Example 4. Consider the example described in Figure 4. Let a_A, a_B and a_C be the rows of action a (chocolate) in the binary BMs of entities A, B and C from Figure 3. At the left of Figure 4, when merging a_A and a_B, the magnitude corresponding to the merged action, a_{AB} equals 0.19, which is smaller than the original magnitudes: 1.38 for a_A and 1.53 for a_B. The reduction in magnitude is because the sequence a_{AB} is more regular than either of a_A and a_B.

At the right of Figure 4, we apply the same process for a_B and a_C. The magnitudes we obtain are 2.03 for a_{BC}, 1.54 for a_B, and 0.09 for a_C. In this case, merging a_B and a_C resulted in an increase in magnitude because the sequence a_{BC} looks less regular than either of a_B and a_C. $\square$

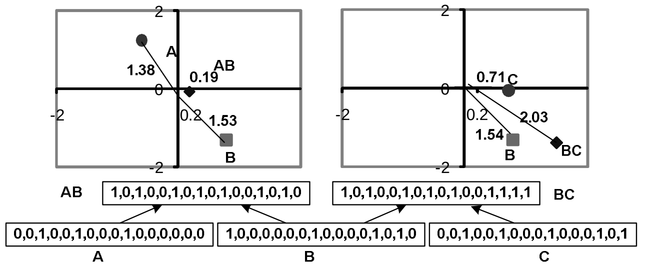


Figure 4: Actions patterns in the complex plane and the effect on the magnitude.

Based on the above discussion, we can compute a recognition score, $\mathcal{S}_r(a_A)$, for each individual action a that belongs to entity A such that it is inversely proportional to the magnitude of the complex number $C_A^{(a)}$. In particular, $\mathcal{S}_r(a_A) = M - mag(C_A^{(a)})$, where $mag(C_A^{(a)})$ is the magnitude of $C_A^{(a)}$ and M is the maximum computed magnitude.

To compute the overall $\mathcal{S}_r(A)$, we average the individual scores, $\mathcal{S}_r(a_A)$, each weighted by the number of times its respective action was repeated ($n_A^{(a)}$). The formula for $\mathcal{S}_r(A)$ is thus given as follows.

$$\mathcal{S}_r(A) = \frac{1}{n_A} \sum_{\forall\, a} n_A^{(a)} \cdot \mathcal{S}_r(a_A) \tag{4}$$

It is worth mentioning that we implemented the entire candidate generation phase as a single SQL query. The query is presented and discussed in Appendix C, where we also show the derivation of the final formula for the matching score starting from Eq. 1.

4. ACCURATE MATCHING USING A STATISTICAL MODELING TECHNIQUE

4.1 Building the Statistical Model

Our goal is to build a statistical model for the behavior of an entity given its observed actions. The two key variables defining an entity's behavior with respect to a specific action are (1) the inter-arrival time between the action occurrences, and (2) the feature id associated with each occurrence, which represents the features describing how the action was performed at that time, or in other words it reflects the entity's preferences when performing this action.

In general, we expect that a given entity will be *biased* to a narrow set of inter-arrival times and feature ids which is what will distinguish the entity's behavior. In merging two behavior matrices for the same entity, the bias should be enforced and made clearer. However, when the behavior matrices of two different entities are merged, the bias will instead be weakened and made harder to recognize. The statistical model that we build should enable us to measure these properties.

Our problem is similar to classifying a biological sequence as being a motif, i.e., a sequence that mostly contains a recognized repeated pattern, or not. A key objective in computational biology is to be able to discover motifs by separating them from some background sequence that is mostly random. In our case, a motif corresponds to a sequence of an action by the same entity. In view of this analogy, our statistical modeling will have the same spirit as the methods commonly used in computational biology. However our model has to fit the specifics of our problem which are as follows: (a) sequences are of two variables (inter-arrivals and feature id), rather than just one variable (DNA character) and (b) for ordinal variables (such as the inter-arrival time), neighboring values need to be treated similarly.

Modeling the Behavior for an Action: We model the behavior of an entity A with respect to a specific action a using a *finite mixture model* $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_K\}$, with *mixing coefficients* $\lambda^{(\mathbf{a_A})} = \{\lambda_1^{(a_A)}, \ldots, \lambda_K^{(a_A)}\}$, where $\mathcal{M}_k$ is its k^{th} component. Each component $\mathcal{M}_k$ is associated with two random variables: (i) The inter-arrival, which is generated from a uniform distribution over the range of inter-arrival times, $r_k = [start_k, end_k]$[2]. (ii) The feature id, is a discrete variable, which is modeled using a multinomial distribution with parameter $\theta_k^{(a_A)} = \{f_{k1}^{(a_A)}, \ldots, f_{kL}^{(a_A)}\}$, where L is the number of all possible feature ids, and $f_{kj}^{(a_A)}$ is the probability to describe the occurrence of action a using feature $\mathcal{F}_j$, $j = 1, \ldots, L$. In what follows, we omit the superscript a_A and assume that there is only one action in the system to simplify the notations.

What we described so far is essentially a generative model in the sense that once built, we can use it to generate new action occurrences for a given entity. For example, using λ, we can select the component $\mathcal{M}_k$ to generate the next action occurrence, which should occur after an inter-arrival time picked from the correspondng range $r_k = [start_k, end_k]$ and we can describe the action by selecting a feature id using θ_k. However, we do not use the model for this purpose. Instead, we use its estimated parameters (λ and the vectors θ_k) to determine the level of recognizing repeated patterns in the sequence corresponding to the action occurrences.

For the estimation of the model parameters, we use the *Expectation-Maximization* (EM) algorithm to fit the mixture model for each specific action a of an entity A to discover the optimal parameter values which maximize the likelihood function of the observed behavior data. Appendix A presents the algorithm and details all the derivations used to estimate the model parameters.

Below, we show an example of a behavior and the properties we desire for its corresponding model parameters. We demonstrate the challenge in finding those desired parameters, which we address by using the EM algorithm. We also show through the example how we choose the initial parameter values required for the EM algorithm.

Example 5. Consider that a customer's behavior with respect to the action of buying chocolate is represented by the sequence $\{\{6, s\}, \{15, l\}, \{6, s\}, \{8, s\}, \{15, l\}, \{14, l\}, \{13, l\}\}$, where s denotes a small quantity (e.g., 1-5 bars), and l denotes a large quantity (e.g., more than 5 bars). So s/he bought a small quantity of chocolate after 6 days, a large quantity after 15 days, and so on.

To characterize the inter-arrival times preferred by this customer, the best ranges of size 2 to use are $[6, 8]$ and $[13, 15]$. Their associated mixing coefficients (λ_k) should be $\frac{3}{7}$ and $\frac{4}{7}$, because the two ranges cover 3 and 4 respectively out of the 7 observed data points.

However, since in general, the best ranges in a behavior sequence will not be as clear as in this case, we need to systematically consider *all* the ranges of a given size (2 in this case), and assign mixing coefficients to each of them. The possible ranges for our example would be $\{[6, 8], [7, 9], [8, 10], \ldots, [13, 15]\}$.

A straightforward approach to compute λ_k for each range is to compute the normalized frequency of occurrence of the given range for all the observed data points. For instance, the normalized frequencies for the ranges $[6, 8]$, $[12, 14]$, and $[13, 15]$ are $\frac{3}{12}$, $\frac{2}{12}$, and $\frac{4}{12}$ (or $\frac{1}{4}$, $\frac{1}{6}$, and $\frac{1}{3}$) respectively, where 12 is the sum of frequencies for all possible ranges. Note that the same inter-arrival time may fall in multiple overlapping ranges. Clearly, these are not the desired values for λ_k. We would rather have zero values for all ranges other than $[6, 8]$ and $[13, 15]$. However, we still use these normalized frequencies as the initial values for λ_k to be fed into the EM algorithm.

Similarly, to compute the initial values for the θ_k probabilities, we first consider the data points covered by the range corresponding to component M_k only. Then, for each possible value of the feature id, we compute its normalized frequency across these data points. Clearly, in our example, the customer favors buying small quantities when s/he shops at short intervals (6-8 days apart), and large quantities when s/he shops at longer intervals (13-15 days apart). □

4.2 Computing Matching Scores

To match two entities A and B, we need to compute the gain $S_m(A, B)$ in recognizing a behavior after merging A and B using Eq. 1. This requires computing the scores $\mathcal{S}_r(A)$, $\mathcal{S}_r(B)$ and $\mathcal{S}_r(AB)$ using Eq. 2, which in turn requires computing the behavior recognition scores corresponding to the three behavior components, which, for entity A for example, are $S_{r1}(A)$, $S_{r2}(A)$, and $S_{r3}(A)$.

For the first behavior component, the consistency in repeating an action a is equivalent to classifying its sequence as a motif. We quantify the pattern strength to be inversely proportional to the uncertainty about selecting a model component using $\lambda^{(a_A)}$, i.e., action a's sequence is a motif if the uncertainty about $\lambda^{(a_A)}$ is low. Thus, we can use the entropy to compute $S_{r1}(a_A) = \log K - H(\lambda^{(a_A)})$, where $H(\lambda^{(a_A)}) = -\sum_{k=1}^{K} \lambda_k^{(a_A)} \log \lambda_k^{(a_A)}$, and the overall score $S_{r1}(A)$ is then computed by a weighted sum over all the actions according to their support, i.e., the number of times the action was repeated.

$$S_{r1}(A) = \frac{1}{n_A} \sum_{\forall\, a} n_A^{(a)} \cdot S_{r1}(a_A) \qquad (5)$$

For the second behavior component, the stability in describing the action (action features) is more recognizable when the uncertainty in picking the feature id values is low. The behavior score along this component can be evaluated by first computing $\theta'^{(a_A)} = \{f_1'^{(a_A)}, \ldots, f_L'^{(a_A)}\}$, which is the overall parameter to pick a feature id value for action a using the multinomial distribution such that the overall probability for entity A to describe its action a by feature $\mathcal{F}_j$ is $f_j'^{(a_A)}$. Here, $f_j'^{(a_A)} = \sum_{k=1}^{K} \lambda_k^{(a_A)} f_{kj}^{(a_A)}$ combined from the all K components for $j = 1, \ldots, L$, knowing that $\theta_k^{(a_A)} = \{f_{k1}^{(a_A)}, \ldots, f_{kL}^{(a_A)}\}$. Using the entropy of $\theta'^{(a_A)}$, we compute $S_{r2}(a_A) = \log L - H(\theta'^{(a_A)})$, where $H(\theta'^{(a_A)}) = -\sum_{j=1}^{L} f_j'^{(a_A)} \log f_j'^{(a_A)}$. Similar to Eq. 5, we can compute the overall score for $S_{r2}(A)$ as the weighted sum for $S_{r2}(a_A)$ according to the actions support.

For the third component, we look for evidence about the associations between actions. We estimate, for every pair of actions, its probability of being generated from components with the same inter-arrival ranges. The association between actions can be recognized when they occur close to each other. In other words, this can occur when both of them tend to prefer the same model components to generate their sequences. Consequently, the score for the third component can be computed over all possible pairs of actions for the same entity as follows:

$$S_{r3}(A) = \sum_{\forall\, a,b} \sum_{k=1}^{K} \lambda_k^{(a_A)} \lambda_k^{(b_A)}$$

Computing Behavior Similarity Score:

The similarity between two behaviors can be simply quantified by the closeness between the parameters of their corresponding behavior models according to the Euclidean distance. For two entities A and B, we compute the behavior similarity as follows:

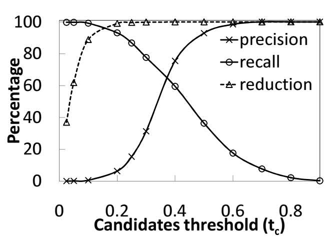
$$BSim(A, B) = 1 - \frac{1}{n_A + n_B} \sum_{\forall\, a} (n_A^a + n_B^a)$$
$$\sqrt{\sum_{k=1}^{K} [(\lambda_k^{(a_A)} - \lambda_k^{(a_B)})^2 + \sum_{j=1}^{L} (\lambda_k^{(a_A)} f_{kj}^{(a_A)} - \lambda_k^{(a_B)} f_{kj}^{(a_B)})^2]}.$$

Note that this method is preferred over directly comparing the BMs of the entities, since the latter method would require some sort of *alignment* for the time dimension of the BMs. In particular, deciding which cells to compare to which cells is not obvious.

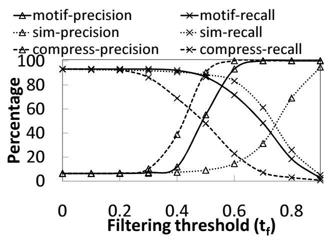
5. EXPERIMENTS

The goals of our experimental study are: (1) Evaluate the overall linkage quality of our behavior based record linkage for various situations of splitting the entities' transactions in the log. (2) Demonstrate the quality improvement when our technique is combined with

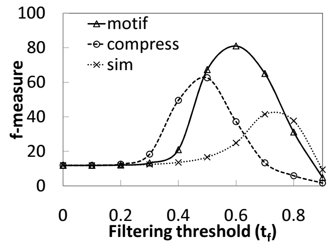
[2]The range size of r_k is user-configurable as it depends on the application and what values are considered close. In our experiments with retail store data from Walmart, we generated ranges by sliding, over the time period, a window of size 5 days with a step of 3 days. (i.e. {{1,6},{4,9},{7,12}, ...})



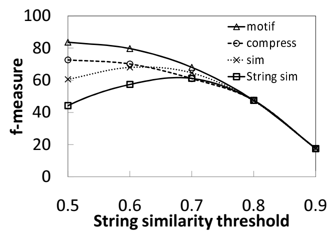
(a) Candidate generation quality.



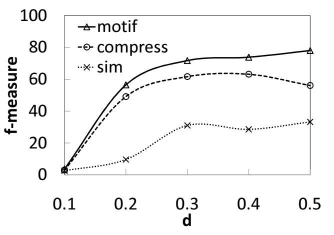
(b) Accurate matching comparisons: Precision & Recall



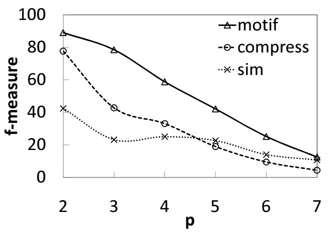
(c) Accurate matching comparisons: f-measure



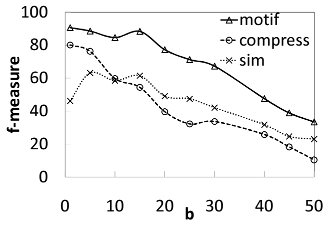
(d) Improving the textual matching quality.



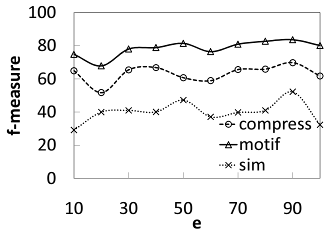
(e) Quality vs. different splitting probabilities



(f) Quality vs behavior exhaustiveness



(g) Quality vs behavior contiguousness



(h) Quality vs percentage of overlapping entities

Figure 5: Experimental Results.

a textual based record linkage technique.(3) Study the performance and scalability as well as the effectiveness of the candidate generation phase on the overall performance. We also include in our evaluation the compressibility technique, which is discussed in Appendix B. In the following, we will refer to the statistical model technique as motif.

To the best of our knowledge, this is the first approach to leverage entity's behavior for record linkage. Consequently, there is no other technique to directly compare to. Instead, we show how our technique can be combined with a textual record linkage technique. We should also mention that all existing techniques, including ours, depend on some domain information, e.g., relational [9] and aggregate constraints [6], which if present, the techniques will perform well. In our case, if the behavior information is rich enough and more reliable than other type of information, better record linkage is expected.

Dataset: We use a real world transaction log from Walmart which would cover many similar scenarios in the retail industry. These transactions cover a period of 16 months. An entry in the log represents an item that has been bought at a given time. Figure 2 shows a typical example for this log. We use the first level item grouping as the actions which are described by the quantity feature. This feature was grouped into {very low, low, medium, high, very high} for each individual item (high quantity for oranges is different from high quantity of milk gallons).

Setup and Parameters: To simulate the existence of two data sources whose customers (entities) need to be linked, a given entity's log is partitioned into contiguous blocks which are then randomly assigned to the sources. The log splitting operation is controlled by the following parameters with their assigned default values if not specified: (1) e: the percentage of overlapped entities between the two data sources (default 50%). (2) d: the probability of assigning a log block to the first data source (default 0.5). (3) b: the transactions block size as a percentage of the entity's log size. When b is very small, the log split is called a *random split* (default 1%), and for higher values we call the split a *block split* (default 30%). The block split represents the case where the customer alternates between stores in different places, e.g., because of moving during the summer to a different place. When b is 50%, the log is split into two equal contintigues halves. From the overlapping entities, 50% have their transactions random split and the rest is block split. These parameters allow us to test our techniques under various scenarios on how entities interact with the two systems.

All the matching scores within a phase are scalled to be between 0 and 1, by subtracting the minimum and dividing by the maximum scores. All of the experiments were conducted on a Linux box with a 3 GHz processor and 32 GB RAM. We implemented the proposed techniques in Java and we used MySQL DBMS to store and query the transactions and the intermediate results.

5.1 Quality

The matching quality of the proposed techniques is analyzed by reporting the classical precision and recall. We also report the *f-measure*$= \frac{2 \times precision \times recall}{precision + recall}$, which corresponds to the weighted harmonic mean of precision and recall. Since we control the number of overlapping entities, we know the actual unique entities to compute the precision and recall. In some cases and to provide more readable plots, we only report the f-measure as an overall quality measure.

Overall Quality: In this experiment, we use a log of a group of 1000 customers.

For the candidate generation phase, we report in Figure 5(a) the recall, precision and percentage of the reduction in the number of candidates against the candidate matching score threshold t_c. If the two data sources contain p and q entities and the number of generated candidates is c pairs, the reduction percentage corresponds to $r = 100(pq - c)/pq$.

We observe that high recall values close to 100% are achieved for $t_c \leq 0.3$. Moreover, the reduction in the number of candidates starts around 40% and quickly increases close to 100% for $t_c \geq 0.2$. The precision starts at very low values close to zero and increases with t_c.

The main purpose of this phase is to reduce the number of candidates while maintaining high recall using an approximate matching process. Therefore, low values for t_c should be used to relax the matching function and avoid false negatives. For low values around $t_c = 0.2$, the number of candidates are perfectly reduced with very few false negatives. This result was achieved on different datasets.

Figure 5(b) and 5(c) illustrate and compare the overall quality of the techniques of the behavior merge strategy; motif and compressibility, and the behavior similarity technique. In this experiment, we used the candidates produced at $t_c = 0.2$. The three techniques behave similarly with respect to t_f, Phase 2 filtering threshold. High recall is achieved for low values of t_f, while high precision is reached for high t_f. In Figure 5(c), f-measure values show that the motif technique can get an accuracy over 80% while the compressibility technique can hardly reach 65%. The behavior similarity technique was the worst as it can hardly reach 45%.

The difference between the motif and compressibility techniques is expected as the motif technique is based on an exhaustive statistical method that is more accurate, while the compressibility is based on efficient mathematical computations. The behavior similarity technique did not perform well because when a customer's shopping behavior is split randomly, it will be difficult to accurately model his/her behavior based on either source taken separately. Consequently, comparing the behavior model can hardly help in matching the customers. In the case where the transactions are block split, the behavior can be well modeled. However, since there are many distinct customers who have similar shopping behaviors, the matching quality will drop.

For subsequent experiments and to be fair to the three behavior matching techniques, we report the best achieved results when changing t_f. For the candidate generation phase, we used $t_c = 0.2$.

Improving Quality with Textual Linkage: In this experiment, we consider a situation that is similar to the Yahoo-Maktoob acquisition discussed in the introduction. We constructed a profile for each customer such that the textual information is not very reliable. Basically, we synthetically perturbed the string values. To match the customers textual information, we used the Levenshtein[3] distance function [17]. In the experiment, we first match the customers using different string similarity thresholds and we then pass the resulting matches to our behavior-based matching techniques.

In Figure 5(d), the overall accuracy improvement is illustrated by reporting the f-measure values. Generally, as we relax the matching using the textual information by reducing the string similarity threshold, the behavior linkage approach get more chance to improve the overall quality. Reducing the string similarity threshold resulted in very low precision and high recall with an overall low f-measure. Leveraging the behavior in such case improves the precision and consequently improves the f-measure. Although all the behavior matching techniques improved the matching quality, the motif technique is more accurate.

Split Transactions with Different Probabilities: In this experiment, we study the effect of changing the parameter d (i.e. we assess the quality when the entities' transactions are split between the data sources with different density). In Figure 5(e), we report and compare the f-measure when linking several datasets each splitted with different d value from 0.1 to 0.5.

We observe that for $d \leq 0.2$ (i.e, the first source contains below 20% of the entitiy's transactions), the matching quality drops sharply. The drop happens because one of the data sources will contain customers with fewer information about their behaviors. When matching such small behaviors, it is likely to fit and produce well recognized behavior with many other customers. The behavior merge techniques consistently produce better results in situations when each of the participating data sources contains at least 20% of an entity's behavior. This is a reasonable results especially when only the behavior information is used for linkage.

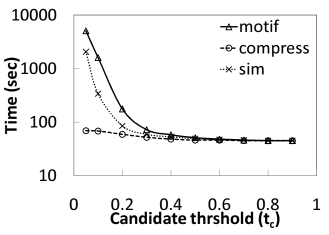
Split Incomplete Behaviors: This experiment evaluates the quality of the behavior linkage when matching incomplete behaviors (i.e., non-exhaustive[4]). More precisely, for a customer we split his/her transactions into p parts and proceed to link the behaviors using only two parts. We report in Figure 5(f) the resulting f-measure matching values when $p = 2, \ldots, 7$.

As expected, as we increase p the matching quality using all the behavior linkage approach drops. Although the customers we used in the experiments may not have their complete buying behavior in Walmart stores, the motif technique was able to get more than 50% quality for customers having about 25% of their behavior split between the two sources. The matching quality of the compressibility technique drops below 50% immediately if we split the transactions into 3 parts and link two of them. The behavior similarity technique was not helpful most of the time even when using an even transactions split.

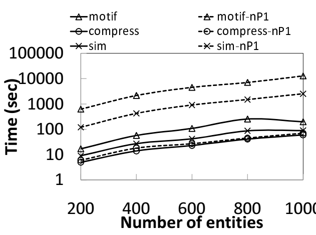
Split Contiguous Behavior Blocks: This experiment studies the effect of changing b, the transactions block size, to split an entity's transactions. In Figure 5(g), we report the matching quality using the f-measure as we change b. For $b = 50\%$, the transactions are split into two contiguous halves and for $b = 1\%$, it is almost as if we are randomly splitting the transactions. For low values of b, we get the best quality matching using the behavior merge, then as we increase the block size, b, the matching quality drops. For the behavior similarity, the quality starts low, about 45%, for very low values of b, then the quality improves to about 65% for b between 5% and 15%. After that the quality keeps decreasing with the increase of b.

[3]We used the implementation in the Second String library (http://secondstring.sourceforge.net/)

[4]An exhaustive behavior means that the customer does all of his/her purchases from the same store throughout the entire studied period



(a) Candidate generation effectiveness.



(b) Scalability.

Figure 6: Performance.

The behavior merge techniques benefit from having the original entities' behavior more random; when merging the transactions, the behavior patterns emerge. This is the main reason for having good accuracy for low values of b. Moreover, high values of b close to 50% mean that the behaviors can be well recognized and therefore the computed gain after merging the transaction will not be significant enough to distinguish between the entities. On the other hand, for the behavior similarity technique, there are two observations affecting its results: (i) there are many customers sharing the same buying behavior, and (ii) how much of the complete behavior can be recognized. For low values of b, the behavior can not be well recognized leading to poorly estimated behavior model parameters, and consequently low matching accuracy. For high values of b, the behavior is well recognized, however, because there are many customers sharing the same behavior the matching quality drops.

Changing the Number of Overlapping Entities : In this experiment, we study the effect of changing the overlapping percentage, e, from 10% to 100% (100% means that all the customers use both stores) and the results are reported in Figure 5(h). We see that the overlapping percentage parameter is not significantly affecting the matching quality for all techniques. This highlights the benefit of our approach to provide good results even when the expected number of overlapping entities is small.

5.2 Performance

Our next set of experiments study the execution time. We start by showing the positive effect of the candidate generation phase on the overall linking time, then we discuss the scalability of our approach.

Candidate Generation Phase Effectiveness: In this experiment, we used the same dataset as in Figure 5(a). In Figure 6(a), we report the total execution time of the motif, compressibility and similarity techniques against different values of Phase 1 threshold, t_c. Phase 1 took 45 sec; this execution time is not affected by t_c because all the pairs of entities should be compared anyway and then filtered based on t_c's selected value. For each value of t_c, the candidates are passed to the accurate matching phase to produce the final matching results.

The time spent in the whole matching process decreases as t_c increases because the number of produced candidates drops dramatically. This was illustrated in Figure 5(a) in terms of reduction in the percentage of the number of candidates. However, high values for t_c results in many false negatives. As mentioned earlier, values around $t_c = 0.2$ produce good quality candidates.

When comparing the performance of the accurate matching techniques at $t_c = 0.2$, the compressibility outperforms motif technique by a factor of about 3. This is because the compressibility uses a technique that does not require scanning the data many times while the motif technique uses an expensive iterative statistical method. The compressibility technique is thus more attractive for very large logs. The similarity technique requires less time than the motif, because the motif computes for each candidate pair 3 statistical models (two for the original two entities and one for the resulting merged entity), while the similarity computes models for only two entities.

Scalability : This experiment analyzes the scalability of the behavior linkage approach and compares the two cases of using or not the candidate generation phase. The evaluation was conducted using a sequential implementation of the techniques (i.e., no parallelization was introduced). In Figure 6(b), we report the overall linkage time for the three behavior matching techniques when using Phase 1 (motif, compress and sim) and without Phase 1 (motif-nP1, compress-nP1 and sim-nP1). When we used Phase 1, $t_c = 0.2$.

The behavior matching techniques require expensive computations and scale poorly without the help of the candidate generation phase, which resulted in around 2 orders of magnitude speedup for the case of motif technique. The processing time is governed by the generated number of candidates using the threshold t_c as discussed in the previous experiment.

For very large scale data processing, generating the candidates can benefit from standard database join performance. Moreover, the computations required for each candidate pair is independent from any other pair computation and hence can be easily parallelized. Therefore, in a parallel environment, all the behavior matching computations can be speeded up to the number of available processing units.

6. RELATED WORK

Most existing record linkage techniques based on textual attributes use various string approximate matching approaches (refer to [10] and [16] for recent surveys). Recent, more involved techniques use information extracted from the data to improve the linkage accuracy. For example, extracting information to capture similarities between entities has been recently explored in data mining and machine learning (e.g. [18, 9, 4, 15]). To the best of our knowledge, our work is the first to exploit the entities' behavior extracted from transactions log to perform entity matching. Moreover in contrast to the traditional concept of linkage based on similarity between entities pair, we used a new concept driven by computing the gain in recognizing behaviors upon merging the log of entities pair.

To improve the performance of the linkage run time, several techniques were introduced and used to avoid the entities cross product matching. For example, *blocking*, *Sorted Neighborhood* [12], traditional clustering [20], and *canopies* [19]. Our candidate generation phase is similar the canopies technique in the sense that we develope a *quick and dirty* matching function to quickly generate candidates for more accurate matching.

A closely related area to our work is users adaptive systems for web navigation and information retrieval (e.g., [21, 13, 1]). Most of these techniques focus on statistically modeling user interactions to extract domain specific features to understand users preferences. These models focus on the statistical significance of extracted features and may take into account the sequence of users actions. However, they do not take into account the time dimension to determine the repeated patterns of actions. Moreover, they are better suited to determine groups of common behaviors and may be used to evaluate the similarities between entities. However, they cannot be helpful for computing the strength in recognizing behaviors when merging entity behaviors along the time dimension.

7. CONCLUSIONS AND FUTURE WORK

We presented a new approach for record linkage that uses entity behavior extracted from transactions logs. When matching two entities, we measure the gain in recognizing a behavior in their merged logs. We proposed two different techniques for behavior recognition: a statistical modeling technique and a more computationally efficient technique that is based on information theory. To improve efficiency, we introduced a quick candidate generation phase. Our experiments demonstrate the high quality and performance of our approach.

One area that we left for future work is to automatically specify the actions and their features without the help of domain experts. Considering very general descriptions may lead to less information to distinguish between entities while very minute details may not lead to recognizable consistent patterns due to many sparse actions. This typically falls in the pathology of *attribute selection error* [14] for induction algorithms like our approach.

Other potential areas for future work include addressing multiple sources instead of two, the privacy issues for behavior data, and the integration of our approach with other relevant approaches proposed for web usage modeling and mining.

8. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *ACM SIGIR*, 2006.

[2] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transfom. In *IEEE Transactions on Computers*, 1974.

[3] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *9th SIGMOD Workshop on Research issues in data mining and knowledge discovery*, 2004.

[4] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 1(1), 2007.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[6] S. Chaudhuri, A. D. Sarma, V. Ganti, and R. Kaushik. Leveraging aggregate constraints for deduplication. In *ACM SIGMOD*, 2007.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. In *Journal of the Royal Statistical Society. Series B*, 1977.

[8] A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: A profiler-based approach. In *IIWeb*, 2003.

[9] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *ACM SIGMOD*, 2005.

[10] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, January 2007.

[11] D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.

[12] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*.

[13] S. Holland, M. Ester, and W. Kieling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, 2003.

[14] D. D. Jensen and P. R. Cohen. Multiple comparisons in induction algorithms. In *Machine Learning Journal*, 2000.

[15] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Int. Conf. on Data Mining*, 2005.

[16] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: Similarity measures and algorithms. In *ACM SIGMOD*, 2006.

[17] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[18] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, 2000.

[19] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *ACM SIGKDD*, 2000.

[20] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings Second ACM SIGMOD Workshop (DMKD 97)*, 1997.

[21] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *SIGKDD*, 2005.

[22] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Pub, 1997.

[23] G. K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

APPENDIX

A. BEHAVIOR MODEL PARAMETERS ESTIMATION

In the following, we use the expectation maximization (EM) algorithm to fit the finite mixture model of a given action sequence representing its occurrence and discover the parameters' values of the overall model which was discussed in Section 4.1. To simplify the notations, we assume there is only one action in the system, so we omit the superscript that link the entity and action names.

Recall that our model consists of K components $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_K\}$, where each component $\mathcal{M}_i$ describes the occurrence of an action using two variables: $r_k = [start, end]$ with a uniform distribution, which represents a range of inter-arrival time of the action, and θ_k, which represents an independent random variable describing a *multinomial trial* with parameters $\theta_k = \{f_{k1}, \ldots, f_{kL}\}$, where L is the number of possible features to describe the action when it occurs. f_{kj} is the probability to describe an action using feature $\mathcal{F}_j$ in component $\mathcal{M}_k$. The different parameters f_{kj}, with $k = \{1, \ldots, K\}$ and $j = \{1, \ldots, L\}$, are estimated from the entity's transaction log. The overall model of the pattern is achieved by estimating the components' *mixing coefficient* $\lambda = \{\lambda_1, \ldots, \lambda_K\}$. λ_k, with $\sum_{k=1}^{K} \lambda_k = 1$, is the probability of using component $\mathcal{M}_k$ to get the next entry $\{v, \mathcal{F}^{(v)}\}$ in the sequence $\mathbf{X}$; i.e. after how many time units, v, the action will occur and how it will be described, $\mathcal{F}^{(v)}$. In summary, the parameters for the overall model of an action are the mixing coefficient λ and the vector θ_k for each component $\mathcal{M}_k$, where $k = \{1, \ldots, M\}$.

As we mentioned earlier, we use the expectation maximization (EM) for finite mixture model to discover the parameters' values of the overall model which would maximize the likelihood of the data. The EM uses the concept of missing data and follows an iterative procedure to find values for λ and θ, which maximize the likelihood of the data given the model. In our case, the missing data is the knowledge of which components produced $\mathbf{X} = \{\{v_1, \mathcal{F}^{(v_1)}\}, \ldots, \{v_N, \mathcal{F}^{(v_N)}\}\}$. A finite mixture model assumes that the sequence $\mathbf{X}$ arises from two or more components with different, unknown parameters. Once we obtain these parameters, we use them to compute the behavior scores along each of the behavior three components.

Let us now introduce a K-dimensional binary random variable Z with a 1-of-K representation in which a particular z_k is equal to 1 and all other elements are equal to 0, i.e., $z_k \in \{0,1\}$ and $\sum_{k=1}^{K} z_k = 1$, such that the probability $p(z_k = 1) = \lambda_k$. Every entry in the sequence X_i will be assigned $Z_i = \{z_{i1}, z_{i2}, \ldots, z_{iK}\}$, We can easily show that the probability

$$\begin{aligned} p(X_i|\theta_1, \ldots, \theta_K) &= \sum_{k=1}^{K} p(z_{ik} = 1) p(X_i|Z_i, \theta_1, \ldots, \theta_K) \\ &= \sum_{k=1}^{K} \lambda_k p(X_i|\theta_k) \end{aligned}$$

Since we do not know z_{ik}, we consider the conditional probability $\gamma(z_{ik})$ of z_{ik} given X_i $p(z_{ik} = 1|X_i)$ which can be found using Bayes' theorem [5]:

$$\begin{aligned} \gamma(z_{ik}) &= \frac{p(z_{ik} = 1) p(X_i|z_{ik} = 1)}{\sum_{k=1}^{K} p(z_{ik} = 1) p(X_i|z_{ik} = 1)} \\ &= \frac{\lambda_k p(X_i|\theta_k)}{\sum_{k=1}^{K} \lambda_k p(X_i|\theta_k)} \end{aligned} \tag{6}$$

We shall view λ_k as the prior probability of $z_{ik} = 1$, and $\gamma(z_{ik})$ as the corresponding posterior probability once we got $\mathbf{X}$. $\gamma(z_{ik})$ can also be viewed as the *responsibility* that component $\mathcal{M}_k$ takes for explaining the observation X_i. Therefore, the likelihood or probability of the data given the parameters can be written in the log form as:

$$\ln p(X|\lambda, \theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{ik}) \ln [\lambda_k p(X_i|\theta_k)]$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{ik}) \ln p(X_i|\theta_k) + \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{ik}) \ln \lambda_k \tag{7}$$

The EM algorithm monotonically increases the log likelihood of the data until convergence by iteratively computing the expected log likelihood of the complete data $(\mathbf{X}, Z)$ in the E step and maximizing this expected log likelihood over the model parameters λ and θ. We first choose some initial values for the parameters $\lambda^{(0)}$ and $\theta^{(0)}$. Then, we alternate between the E-step and M-step of the algorithm until it converges.

In the **E-step**, to compute the expected log likelyhood of the complete data, we need to calculate the required conditional distribution $\gamma^{(0)}(z_{ik})$. We plug the $\lambda^{(0)}$ and $\theta^{(0)}$ in Eq. 6 to get $\gamma^{(0)}(z_{ik})$, where we can compute $p(X_i|\theta_k)$ as follows:

$$p(X_i|\theta_k) = \prod_{j=1}^{L} f_{kj}^{I(j,k,\mathcal{F}^{(v_i)})} \tag{8}$$

where $X_i = \{v_i, \mathcal{F}^{(v_i)}\}$ and $I(j, k, \mathcal{F}^{(v_i)})$ is an indicator function equal to 1 if $v_i \in r_k$ and $\mathcal{F}^{(v_i)} = \mathcal{F}_j$; otherwise it is 0.

Recall that $r_k = [start, end]$ is the period identifying the component $\mathcal{M}_k$.

The **M-step** of EM maximizes Eq. 7 over λ and θ in order to re-estimate new values for them $\lambda^{(1)}$ and $\theta^{(1)}$. The maximization over λ involves only the second term in Eq. 7,
$\text{argmax}_\lambda \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma(z_{ik}) \ln \lambda_k$, has the solution

$$\lambda_k^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \gamma^{(0)}(z_{ik}) \quad , k = 1, \ldots, K. \tag{9}$$

We can maximize over θ by maximizing the first term in Eq. 7 separately over each θ_k for $k = \{1, \ldots, K\}$. $\text{argmax}_\theta\, E(logp(X, Z|\theta_i, ..., \theta_K)]$ is equivalent to maximizing the right hand side of Eq. 10 over θ_k (only a piece of the parameter) for every k.

$$\theta_k = \underset{\theta_k}{\text{argmax}} \sum_{i=1}^{N} \gamma^{(0)}(z_{ik}) \ln p(X_i|\theta_k), \tag{10}$$

To do this, for $k = \{1, \ldots, K\}$ and $j = \{1, \ldots, L\}$ let

$$c_{kj} = \sum_{i=1}^{N} \gamma^{(0)}(z_{ik}) I(j, k, \mathcal{F}^{(v_i)}) \tag{11}$$

Then c_{kj} is in fact the expected number of times to describe the action by $\mathcal{F}_j$ when its inter-arrival falls in $\mathcal{M}_k$'s range r_k. We re-estimate θ_k by substituting Eq. 8 into Eq. 10 to get

$$\theta_k^{(1)} = \{\hat{f}_{k1}, \ldots, \hat{f}_{kL}\} = \underset{\theta_k}{\text{argmax}} \sum_{j=1}^{L} c_{kj} \ln f_{kj} \tag{12}$$

$$\text{Therefore, } \hat{f}_{kj} = \frac{c_{kj}}{\sum_{j=1}^{L} c_{kj}} \tag{13}$$

To find the initial parameters $\lambda^{(0)}$ and $\theta^{(0)}$, we scan the sequence $\mathbf{X}$ once and use Eq. 11 to get c_{kj} by setting all $\gamma^{(0)} = 1$. Afterward, we use Eq. 13 to get $\theta_k^{(0)}$ and compute

$$\lambda_k^{(0)} = \frac{\sum_{j=1}^{L} c_{kj}}{\sum_{k=1}^{K} \sum_{j=1}^{L} c_{kj}}$$

B. COMPRESSIBILITY: AN ACCURATE INFORMATION THEORETIC TECHNIQUE

We present an information theory-based technique for the computation of the matching scores. This technique is not as accurate as the motif-based technique (Section 4), but it is more computationally efficient. The underlying idea stems from observing that if we represent the BM as an image, we will see *horizontal repeated blocks* that would be more recognizable if the behavior is well recognized. The repeated blocks appear because of the repetition in the behavior patterns. Therefore, we expect more regularity along the rows than along the columns of the BM. In fact, the order of values in any of the columns depends on the order of the actions in the BM, which is not expected to follow any recognizable patterns. For these reasons, we compress the BM on a row by row basis, rather than compressing the entire matrix as a whole.

Most existing compression techniques exploit data repetition and encode it in a more compact representation. We thus introduce *compressibility* as a measure of confidence to recognize behaviors. In our experiments, we compress the BM with the DCT compression technique [2], being one of the most commonly used compression techniques in practice. We then use the compression ratios to compute the behavior recognition scores. Significantly higher compression ratios imply a more recognizable behavior.

Given the sequence representation of an action occurrence i.e. $\{\{v_j, \mathcal{F}^{(v_j)}\}\}$, if an entity follows stability in repeating an action, the values v_j's will follow a certain level of correlation showing the action rate. Moreover, the features values $\mathcal{F}^{(v_j)}$ will contain similar values to describe how the action was performed. To perform a compression of an action sequence, we follow the same approach used in JPEG [23] for a one dimentional sequence.

Our aim is to compute the three behavior recognition scores along the three behavior components (see Section 2.4). For the first behavior component, we compress the sequence $\{v_1, \dots, v_{n_A^{(a)}}\}$, which represents the inter-arrival times for each action a. The behavior score, $S_{r1}(a_A)$ for action a of entity A, will be the resulted compression ratio; the higher the compression ratio, the more we can recognize a consistent inter-interval time (motif). We then use Eq. 5 to compute the overall score $S_{r1}(A)$. Similarly, for the second behavior component, we compress the sequence $\{\mathcal{F}^{(v_1)}, \dots, \mathcal{F}^{(v_{n_A^{(a)}})}\}$, which represents the feature values that describe the action a. Again, the score $S_{r2}(a_A)$, is the produced compression ratio; the higher the compression ratio, the more we can recognize stability in action features. Similar to $S_{r2}(a_A)$, we can compute the overall score $S_{r2}(A)$.

Finally, for the third behavior component, which evaluates the relationship between the actions, we compress the concatenated sequences of inter-arrival times of every possible pair of actions.Given two actions a and b, we concatenate and then compress their inter-arrival times to get the compression ratio $cr_{a,b}$. If a and b are closely related, they will have similar inter-arrival times allowing for better compressibility of the concatenated sequence. On the contrary, if they are not related, the concatenated sequence will contain varying values. Thus, $cr_{a,b}$ quantifies the association between actions a and b. Hence, the overall pairwise association is an evidence for the strength in the relationship between the actions that can be computed by:

$$S_{r3}^{(A)} = \sum_{\forall\, a,b} cr_{a,b}$$

C. STANDARD SQL TO COMPUTE CANDIDATE MATCHES IN PHASE 1

In the following, we provide a derivation for a final formulation of the matching score in the candidate generation matching phase. At the end, we provide the corresponding SQL statement we used for this computation.

In Section 3, after computing the complex numbers representation for each action in an entity, we computed $\mathcal{S}_r(a_A) = M - mag(C_A^{(a)})$, where M is the maximum computed magnitude. Then, we obtain

$$\mathcal{S}(A) = \frac{1}{n_A} \sum_{\forall\, a} n_A^{(a)} (M - mag(C_A^{(a)})) \tag{14}$$

By substituting Eq. 14 into Eq. 1, we obtain the matching score $S_m(A, B)$:

$$\begin{aligned} S_m(A,B) \;=\; & \frac{n_A}{n_A + n_B} \\ & [\frac{1}{n_A+n_B} \sum_{\forall\, a} (n_A^{(a)} + n_B^{(a)})(M - mag(C_{AB}^{(a)})) \\ & \quad - \frac{1}{n_A} \sum_{\forall\, a} (n_A^{(a)})(M - mag(C_A^{(a)}))] \\ + \; & \frac{n_B}{n_A + n_B} \\ & [\frac{1}{n_A+n_B} \sum_{\forall\, a} (n_A^{(a)} + n_B^{(a)})(M - mag(C_{AB}^{(a)})) \\ & \quad - \frac{1}{n_B} \sum_{\forall\, a} (n_B^{(a)})(M - mag(C_B^{(a)}))] \end{aligned}$$

By Simple rearrangement to collect the terms related to $mag(C_{AB}^{(a)})$, we get

$$\begin{aligned} S_m(A,B) \;=\; & \frac{1}{n_A + n_B} \\ & \sum_{\forall\, a} [(n_A^{(a)} + n_B^{(a)})M - (n_A^{(a)} + n_B^{(a)})\, mag(C_{AB}^{(a)}) \\ & -n_A^{(a)}\, M + n_A^{(a)}\, mag(C_A^{(a)}) \\ & -n_B^{(a)}\, M + n_B^{(a)}\, mag(C_B^{(a)})] \end{aligned}$$

Note that the terms of M will cancel out and the final matching score will be

$$\begin{aligned} S_m(A,B) = \quad & \frac{1}{n_A + n_B} \\ & \sum_{\forall\, a} [\, n_A^{(a)}\, mag(C_A^{(a)}) + n_B^{(a)}\, mag(C_B^{(a)}) \\ & \quad - (n_A^{(a)} + n_B^{(a)})\, mag(C_{AB}^{(a)}) \,] \end{aligned} \tag{15}$$

We store the complex number information for each data source in a relation with the attributes (entity, action, Re, Im, mag, a_supp, e_supp), where there is a tuple for each entity and its actions. For each action of an entity, we store the real and imaginary components (Re and Im) of the complex number as well as the magnitude (mag). a_supp is the number of transaction for that action within the entities log and e_supp is total number of transactions for the entity repeated with each tuple corresponding an action. Thus, there are two tables representing each of the two data sources src1 and src2.

To generate the candidates, we need to compute Eq. 15 for each pair of entities and filter the result using the threshold t_c on the resulting matching score. The following SQL applies this computation and returns the candidate matches.

```
select
  c1.entity as e1 ,
  c2.entity as e2 ,
  ( c1.a_supp * c1.mag  // n^a_A * mag^a_A
  + c2.a_supp * c2.mag  // n^a_B * mag^a_B
  - (c1.a_supp + c2.a_supp ) *  // n^a_AB *
     SQRT(                      // mag^a_AB
          (c1.Re + c2.Re)*(c1.Re + c2.Re)
         +(c1.Im + c2.Im)*(c1.Im + c2.Im))
  )/ (c1.e_supp + c2.e_supp)   // n_AB
  as gain_score
from src1 c1 inner join src2 c2
          on c1.action = c2.action
where magscore > t_c
group by c1.entity, c2.entity
```