World Wide Web (2011) 14:157–186
DOI 10.1007/s11280-011-0109-5

Approximate entity extraction in temporal databases

Wei Lu · Gabriel Pui Cheong Fung · Xiaoyong Du · Xiaofang Zhou · Lijiang Chen · Ke Deng

Received: 25 December 2009 / Revised: 25 November 2010 /
Accepted: 4 January 2011 / Published online: 21 January 2011

Abstract We study the problem of efficiently extracting K entities, in a temporal database, which are most similar to a given search query. This problem is well studied in relational databases, where each entity is represented as a single record and there exist a variety of methods to define the similarity between a record and the search query. However, in temporal databases, each entity is represented as a sequence of historical records. How to properly define the similarity of each entity in the temporal

W. Lu · X. Du (✉) · X. Zhou
School of Information, Renmin University of China, 100872, Beijing, China
e-mail: duyong@ruc.edu.cn

W. Lu
e-mail: uqwlu@ruc.edu.cn

X. Zhou
e-mail: zxf@itee.uq.edu.au

W. Lu · X. Du · X. Zhou
Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education,
Beijing, China

G. P. C. Fung
Data Mining and Machine Learning Group, Arizona State University, 699 S. Mill Ave, Tempe, Arizona, USA
e-mail: g.fung@itee.uq.edu.au

X. Zhou · K. Deng
School of ITEE, The University of Queensland, GP78 South, St. Lucia, Brisbane, Australia

K. Deng
e-mail: dengke@itee.uq.edu.au

L. Chen
Department of Computer Science, Peking University, 100872, Beijing, China
e-mail: clj@pku.edu.cn

database still remains an open problem. The main challenging is that, when a user issues a search query for an entity, he or she is prone to mix up information of the same entity at different time points. As a result, methods, which are used in relational databases based on record granularity, cannot work any further. Instead, we regard each entity as a set of "virtual records", where attribute values of a "virtual record" can be from different records of the same entity. In this paper, we propose a *novel evaluation model*, based on which the similarity between each "virtual record" and the query can be effectively quantified, and the maximum similarity of its "virtual records" is taken as the similarity of an entity. For each entity, as the number of its "virtual records" is exponentially large, calculating the similarity of the entity is challenging. As a result, we further propose a **D**ominating **T**ree **A**lgorithm (DTA), which is based on the bounding-pruning-refining strategy, to efficiently extract K entities with greatest similarities. We conduct extensive experiments on both real and synthetic datasets. The encouraging results show that our model for defining the similarity between each entity and the search query is effective, and the proposed DTA can perform at least two orders of magnitude improvement on the performance comparing with the naive approach.

Keywords temporal databases · approximate entity extraction · bounding-pruning-refining · n-partite graph

1 Introduction

The problem of identifying the entities of a collection that are similar to a given search query has a variety of applications in many branches of computer science, including record linkage [4, 23], data cleaning [14], data integration [6, 21], approximate joins [38], error checking [27], etc. We refer to this problem as approximate entity extraction, which is also known as approximate match [1], object identification [30]. The problem of approximate entity extraction is induced by inconsistencies between user queries and the data, as misspellings, typographical and transcription errors, multiple conversions could exist in any of them. This problem is more serious when the data are crawled from unstructured or semi-structured documents like web pages [7, 23]. As a consequence, we have to relax query conditions since the equivalent or containment operations cannot work any further. In this paper, we are interested on extracting top K entities that are most similar to a given search query.

In the manner of relational databases,[1] an entity in a relation is represented as a single record and the entity score between an entity and the search query is calculated as follows: (1) a specific similarity function, such as Jaccard [18], edit distance [28], is offered to compute the attribute score between its record and the search query over each individual attribute; (2) the record score[2] between a record and the search query is computed by averaging these attributes scores; (3) the record score is offered as the entity score of the entity. After entity scores of all entities have been calculated, the top K entities can further be identified. A thorough study on evaluating the entity

[1]For the ease of differentiation, relational databases refer to non-temporal & relational databases.

[2]For the sake of clarity, we use "attribute score", "record score", "entity score" to measure the similarity between a query and an entity in terms of attribute, record and entity levels respectively.

score between an entity and the search query in the relational database could be found in [22].

Unlike relational databases, in the manner of temporal databases such as Postgresql, records are seldom deleted for ease of concurrency control and event tracking [13, 29]. Thus, an entity usually contains a sequence of records, one of which is the latest version, and the others are the historical versions. For better understanding, Table 1 shows an example of a relation in temporal databases logically. In temporal databases, records of the same entity share the same ID, and each record is attached with a timestamp, e.g. dates are shown in the last column of Table 1. A typical application scenario is that a user can issue a query to obtain historical records of entities [29]. For example, in Table 1, when a user issues a query with conditions: Name = Jennifer Smith, Title = Senior Officer, and Workplace = General Support, the first record of entity "0001" is most similar to the query and entity "0001" will be returned.

However, when a user issues a search query for an entity, he or she is prone to **mix up** demographic information from different periods. For example, when a user tries to search entity "0001" in Table 1, he or she may issue a query with conditions: Name = Jennifer, Title = Manager, and Workplace = General Support, where the user misremembers that entity "0001" still works in General Support, while actually in Infrastructure. This situation often happens in our daily life. Under such an application scenario, applying the method, which defines the entity score of each entity based on *record granularity* as described in relational databases, to temporal databases, poses some challenges.

1.1 Challenges

Challenge 1 *(Best virtual record match) Alternatively, the entity score between an entity and the search query can be quantified as:* (1) *for each record of the entity, we calculate the record score between the record and the search query based on record granularity;* (2) *the maximum record score of its records is taken as the entity score of the entity. For the ease of illustration, we call this evaluation method as* **per-record match**. *However, as we describe in the following example, incorrect result might be returned if per-record match is employed.*

Example 1 (Best virtual record match) Suppose a user wants to search the latest phone number of entity "0001" and input the following conditions in the search query: Name = Jennifer, Title = Manager, Workplace = General Support. In this

Table 1 Staff records in a temporal database.

ID	Name	Title	Workplace	Phone	Last update
0001	Jennifer Smith	Senior officer	General support	33650001	2008-01-01
0001	Jennifer Smith	Manager	Infrastructure	33650011	2009-01-01
...	...	...	...	...	...
0002	Jennifer Keith	Manager	General office	33650002	2008-05-01
...	...	...	...	...	...
0003	George Smith	Trainee	General support	33650003	1998-01-01
0003	George Smith	Manager	Security	33650023	2008-01-01
...	...	...	...	...	...

Table 2 An example of similarities between entities and a given search query.

ID	Name	Title	Workplace	Avg.	Last update
0001	0.5	0	1	0.5	2008-01-01
0001	0.5	1	0	0.5	2009-01-01
...	...	...	...	...	...
0002	0.5	1	0.26	0.59	2008-05-01
...	...	...	...	...	...
0003	0	0	1	0.33	1998-01-01
0003	0	1	0	0.33	2008-01-01
...	...	...	...	...	...

query, the user mixes up demographic information of entity "0001" from different periods into a single query. Using the Jaccard similarity function based on tokenizing each attribute value as a set of 3-grams,[3] attribute scores between the search query and the records over individual attributes are calculated, and shown in columns 2–4 of Table 2. We calculate the record score of each record by averaging attribute scores, and show it in the fifth column of Table 2. The entity score of each entity is quantified by the maximum record score of its records. From Table 2, we can observe that the entity "0002" has the greatest entity score (*not* entity "0001"). Hence, *incorrect result is returned based on the per-record match.*

As described above that a search query could be mixed up, it is natural to think whether we can use mixed up "virtual records" (see Definition 1*) to match this query.*

Definition 1 (Virtual record) Given an entity, (1) over each individual attribute that a search query is involved in, we select one and only one attribute value of its records; (2) such attribute values constitute a virtual record.

For example, (Jennifer Smith, Manager, General Support) is a virtual record of entity "0001" shown in Table 1 *with respect to the query shown in Example* 1*. Intuitively, for any mixed up query, if an entity is similar to this query, there should exist a virtual record of this entity similar to the query. Naturally, this virtual record is regarded as the best virtual record (see Definition* 2*) and its record score is offered as the entity score of the entity.*

Definition 2 (Best virtual record) Given an entity, (1) over each individual attribute that a search query is involved in, we select the attribute value of its records, which is most similar to the search query; (2) such attribute values constitute the best virtual record.

In Example 1*, the best virtual record of entity "*0001*" with respect to query (Jennifer, Manager, General Support) is (Jennifer Smith, Manager, General Support), and the entity score of entity "*0001*" is* $(0.5 + 1 + 1)/3 = 0.83$ *(see Table* 2*). Similarly, the entity score of entity "*0003*" is* $(0 + 1 + 1)/3 = 0.67$*. Thus, correct result can now be returned.*

[3]Although we only focus on Jaccard similarity function in this paper, our proposed work can also applied to handle all the other similarity functions.

For the ease of illustration, we refer to this evaluation method as **best virtual record match**.

To conclude, in the scenario where each entity contains a sequence of records, approximate entity extraction using **per-record match** *cannot properly identify entities that a user looks for. Instead, when we employ* **best virtual record match**, *in the above example, the correct result can be extracted.*

Challenge 2 (Heterogeneity penalty) *Unfortunately, simply employing best virtual record match is not always enough.*

Example 2 (Heterogeneity penalty) Suppose a user issues a search query with the following conditions: Name = Smith, Title = Manager, Workplace = General Support. Attribute scores between records and this search query over individual attributes can be calculated, and are shown in the columns 2–4 of Table 3. For each entity, the record score of its best virtual record is calculated and shown in the fifth column of Table 3, where attribute scores of the best virtual record are labeled with underlines. From the table, we can find that entity "0003" has the maximum entity score and is returned to the user. However, if we *further consider* the attribute "Last Update", we can observe that the time discrepancy between these two records of entity "0001" is *one year*, whereas it is *ten years* between these of entity "0003". Practically speaking, it is very unlikely to mix up the current information with the information that was ten years ago by users. As a result, we need to offer a larger **heterogeneity penalty** for such virtual record with respect to its record score such that entity "0001" is returned instead of entity "0003".

To conclude, if we compute the entity score of each entity only dependent on the record score of its best virtual record, we may easily get incorrect result. Therefore, we should take into consideration the time discrepancy among all involved attribute values of a best virtual record. In general, for a best virtual record, if the time discrepancy among its attribute values is large, then we will offer a larger **heterogeneity penalty** *with respect to its record score. It is worth mentioning that, as a extreme case, no* **heterogeneity penalty** *will be assigned if all attribute values are from a single record.*

Challenge 3 (Modeling) *As discussed above, on one hand, using per-record match to define the entity score of an entity, although no heterogeneity penalty is assigned, the record score might be low; on the other hand, using best virtual record match, although the record score is maximized, the heterogeneity penalty might be large.*

Table 3 Another example of similarities between entities and a given search query.

ID	Name	Title	Workplace	Avg.	Last update
0001	$\underline{0.25}$	0	$\underline{1}$	0.75	2008-01-01
0001	0.25	$\underline{1}$	0		2009-01-01
...	...	...	...	...	...
0002	$\underline{0}$	$\underline{1}$	$\underline{0.26}$	0.42	2008-05-01
...	...	...	...	...	...
0003	$\underline{0.3}$	0	$\underline{1}$	0.77	1998-01-01
0003	0.3	$\underline{1}$	0		2008-01-01
...	...	...	...	...	...

Actually, for other virtual records of an entity, although their record scores are not more than the best virtual record, their heterogeneity penalties might be less than that of the best virtual record such that the entity score might be larger if we use one of them to evaluate. Clearly, the challenge of effectively modeling the entity score of each entity is how to properly leverage record score and heterogeneity penalty. If we can effectively quantify the entity score of each entity, then after entity scores of all entities being computed, the top K entities will be extracted and returned.

1.2 Contributions

To sum up, we make the following contributions:

Contribution 1 (Problem definition) *We address a problem that is not yet being reported elsewhere to the best of our knowledge—approximate entity extraction in temporal databases. Our objective is to extract K entities, in a temporal database, which are most similar to a given search query. Existing work on approximate entity extraction problem can only be applied to the relational database.*

Contribution 2 (Modeling) *As described above, using neither per-record match nor best virtual record match can effectively quantify the entity scores of entities in temporal databases. To be more important, another virtual record, neither any record nor the best virtual record, might have larger record score after properly leveraging its record score and heterogeneity penalty. As such, we propose an effective model to quantify entity score of each entity:*

- *for each of its virtual records, we calculate its record score;*
- *its record score is adjusted by assigning a heterogeneity penalty, where the value of heterogeneity penalty is dependent on the time discrepancy among its attribute values. We refer to record score on this phase as* **adjusted record score***;*
- *the maximum adjusted similarity score will be offered as the entity score.*

Contribution 3 (Efficient computation) *For a given entity with n records and m attributes, there will be n^m virtual records. Identifying the maximum adjusted record score might need to evaluate all virtual records one by one. This is a very time consuming process. Worse more, if there exist l entities, we need to evaluate adjusted record scores of all virtual records of these l entities in order to extract K entities with greatest entity scores. For instance, consider a relation with* 6 *attributes. Suppose we want to extract* 100 *entities in this relation and each entity has* 10 *records, then we have* $10^6 \times 100 = 100{,}000{,}000$ *virtual records. Evaluating all virtual records is very expensive. This may not be always computationally feasible. Although we can apply some approximate algorithms to speed up the computational process, the solution obtained is not an exact one.*

In this paper, we develop an algorithm called **D***ominating* **T***ree* **A***lgorithm (DTA) to extract K entities with greatest entity scores. DTA computes the adjusted record scores of some selected virtual records of entities, and uses those virtual records to prune other virtual records and entities effectively. Although the time complexity of DTA is dependent on the model and the worst case is exponential, it performs surprisingly efficiently in practice from our experimental evaluations. We will describe in Section* 4 *that, although there might exist other evaluation models that can effectively measure the entity score of each entity, our DTA approach is a generic one that is independent on these models.*

The rest of the paper is organized as follows—Section 2 formally defines our problem; Section 3 describes our model on how to quantify the entity score between an entity and the search query; An effective approach, which is independent on our model, is proposed to identify the top K entities and presented in Section 4. The experimental results are presented in Section 5. Section 6 briefly discusses the related work; Section 7 concludes this paper.

2 Problem definition

Let R be a set of records, whose schema contains a set of attributes A. Let E be a set of entities, each of which contains a sequence of records of R. Each record of R is attached with a timestamp and belongs to one and only one entity in E. For reference, a set of symbols, shown in Table 4 will be used throughout this paper.

For a given search query q, in this paper, we study the problem of efficiently extracting K entities, in a temporal database, which are most similar to q, where each entity contains a sequence of records and K is pre-specified by the user. We refer to this problem as approximate entity extraction problem and these K entities as top K entities (denoted as top_K). The model on how to quantify the entity score between an entity and the search query is presented in Section 3, and here we assume that we have known how to compute the entity score. Formally, our objective is to extract a set $top_K \in E$ such that $|top_K| = K$, and $\forall e \in top_K$, $\forall e' \in E - top_K$, $sim(q, e) \geq sim(q, e')$. Without loss of generality, we assume the number of attributes in R equals that of q. If not, only the attributes of q will be considered for each record, while the rest of attributes are ignored.

Table 4 Symbols and their definitions.

Symbols	Definitions
A	A set of attributes
a	An attribute
E	A set of entities
e	An entity in E, $e \in E$
r_i	The i^{th} record of e
$r_i.a_j$	Attribute value of r_i over a_j
γ	A virtual record of e
$\gamma.a$	Attribute value of γ over a
$T(r_i)$	The valid time of r_i
$G_e(V, D)$	A complete n-partite graph for e
v_i	A vertex of $G_e(V, D)$
$w(v_i)$	Weight of vertex v_i
$w(v_i, v_j)$	Weight of edge that connects v_i and v_j
g_e	A candidate graph of $G_e(V, D)$
q	A search query
$sim(\cdot, \cdot)$	A similarity function
$sim_u(\cdot, \cdot)/sim_l(\cdot, \cdot)$	The upper/lower bound of $sim(\cdot, \cdot)$

2.1 Tokenization

For a given string, $r.a$ (or $q.a$), we can tokenize it as a set of n-grams. This tokenization is performed by sliding a window of length n over characters of a string. For example, consider the title of the latest record of entity "0001" shown in Table 1. "Manager" is tokenzied as "Man,ana,nag,age,ger" with respect to 3-grams. For ease of illustration, we use terms $T_{r.a}$ and $T_{q.a}$ to represent the set of n-grams of $r.a$ and $q.a$, respectively. $|T_{r.a}|$ denotes the number of grams in $T_{r.a}$.

2.2 Similarity function

Given a search query q, and an entity e, the attribute score between q and record $r \in e$ over attribute a can be quantified by one of the following commonly used similarity functions:

- **Intersect** similarity function: $sim_{Intersect}(q.a, r.a) = |T_{q.a} \cap T_{r.a}|$;
- **Dice** similarity function: $sim_{Dice}(q.a, r.a) = \dfrac{2 \cdot |T_{q.a} \cap T_{r.a}|}{|T_{q.a}| + |T_{r.a}|}$;
- **Jaccard** similarity function: $sim_{Jaccard}(q.a, r.a) = \dfrac{|T_{q.a} \cap T_{r.a}|}{|T_{q.a} \cup T_{r.a}|}$;
- **Cosine** similarity function: $sim_{Cosine}(q.a, r.a) = \dfrac{\mathbf{T_{q.a}} \cdot \mathbf{T_{r.a}}}{\|\mathbf{T_{q.a}}\| \|\mathbf{T_{r.a}}\|} = \dfrac{|T_{q.a} \cap T_{r.a}|}{\sqrt{|T_{q.a}|} \cdot \sqrt{|T_{r.a}|}}$.

In the rest of this paper, we use $sim(.,.)$ to denote any of the above similarity functions. Unless otherwise specified, the attribute score is quantified using Jaccard similarity function and each string is tokenized as a set of 3-grams.

According to Definition 1, $\forall r \in e$, clearly, r is a special virtual record of e. Unless otherwise specified, a virtual record mentioned in this paper can also refer to a record. Formally, a virtual record γ is represented as $(\gamma.a_1, \gamma.a_2, \ldots, \gamma.a_{|A|})$, where $\exists r_j \in e, \gamma.a_i = r_j.a_i$. In general, for each virtual record, we compute its record score based on Definition 3.

Definition 3 (Record score) For a virtual record γ, and a search query q, the record score of γ is defined as:

$$sim(q, \gamma) = \frac{1}{|A|} \dot{\sum}_{a \in A} sim(q.a, \gamma.a) \tag{1}$$

Due to the heterogeneity of attribute values of a virtual record, we are required to assign a heterogeneity penalty to their record scores that are computed based on (1). In the next part, we will present how to adjust the record score of each virtual record, which is denoted as **adjusted record score**, and define the entity score of each entity.

3 Modeling

In this section, we first model each entity as a complete $|A|$-partite graph [8], followed by a similarity measure exploiting the vertexes and edges of the graph to define the adjusted record score of each virtual record. Finally, we define the entity score of each entity based on adjusted record scores of its virtual records.

3.1 Overview of modeling the adjusted record score

The main challenge of modeling the adjusted record score of each virtual record is how to properly leverage its record score and heterogeneity penalty, where the record score is quantified based on (1) and the value of heterogeneity penalty is dependent on the time discrepancy of its attribute values.

However, how to model the adjusted record score of each virtual record is not intuitionistic if we simply take its attribute values into consideration. As such, we first model each entity as a complete $|A|$-partite graph, from which for each virtual record, we can find one and only one subgraph that corresponds to it. We then model the adjusted record score of each virtual record by the vertexes and edges of its corresponding subgraph. Finally, the maximum adjusted record score is offered as the entity score.

Definition 4 (Complete $|A|$-partite graph) A graph $G(V, D)$ (V, D represent the vertexes and edges in the graph) is said to be a complete $|A|$-partite graph if it can satisfy the following conditions:

1. vertexes in G are partitioned into $|A|$ groups;
2. every vertex in each group is connected to all the vertices of the graph which are not contained in that group;
3. no two vertexes in the same group are connected.

Formally, $G.V = \{V_1, \ldots, V_{|A|}\}$, and $G.D = \{(v, v') | \forall V_i \in V, \forall V_j \in V, i \neq j, \forall v \in V_i, \forall v' \in V_j\}$.

Lemma 1 *Each entity, e, can be modeled as a complete $|A|$-partite graph, denoted as $G_e(V, D)$, where attribute values involved in the same attribute are mapped to the vertexes of the same group. Formally, $\forall V_i \in G.V$, $V_i = \pi_{a_i}(e)$, where $\pi_{a_i}(e)$ is the projection of e over attribute a_i. A vertex in group V_i is represented as v_{ji} $(1 \leq j \leq |e|)$ and corresponds to attribute value $r_j.a_i$.*

Example 3 (A 4–partite graph) Consider entity “0001” with two records shown in Table 1. Let e denote entity “0001” and r_1, r_2 denote the second and first records of e respectively. The complete $|A|$-partite graph for entity e is shown in Figure 1a. In this figure, $v_{11} = r_1.a_1$, $v_{21} = r_2.a_1$, $v_{12} = r_1.a_2$, $v_{22} = r_2.a_2$, $v_{13} = r_1.a_3$, $v_{23} = r_2.a_3$, $v_{14} = r_1.a_4$, $v_{24} = r_2.a_4$.

Definition 5 (Candidate graph) A subgraph $g_e(V, D)$ in $G_e(V, D)$ is said to be a candidate graph if and only if: (1) each vertex in $g_e(V, D)$ comes from a different group; (2)The number of vertexes in $g_e(V, D$ equals to $|A|$; (3) Every two vertexes in $g_e(V, D)$ are connected. For ease of illustration, $g_e(V, D)$ is denoted as g_e for short.

Example 4 (Candidate graph) Let us refer to the graph in Example 3. Figure 1b shows one of the candidate graphs which is formulated by v_{11}, v_{12}, v_{23}, v_{14} and edges

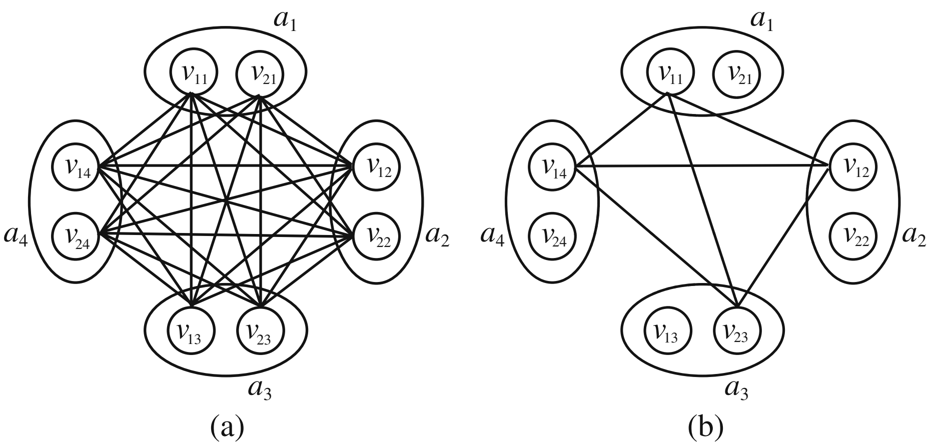


Figure 1 A complete $|A|$-partite graph and a candidate graph. **a** A complete 4-partite graph. **b** A candidate graph.

between every two of them. For some counter examples, a subgraph formulated by v_{11}, v_{12}, v_{23} is *not* a candidate graph because $|g_e| = 3$ which is less than $|A| = 4$. A graph formulated by $v_{11}, v_{12}, v_{23}, v_{13}$ is *not* a candidate graph because v_{23} and v_{13} are in the same group.

Lemma 2 *For a given entity e and $G_e(V, D)$, there exists a one-to-one correspondence between virtual records of e and candidate graphs of $G_e(V, D)$. For sake of illustration, a virtual record termed as γ^s is used to correspond to the candidate graph g_e^s using the same superscript.*

For example, a virtual record (Jennifer Smith, Manager, General Support, 33650011) of entity "0001" in Table 1 corresponds to the candidate graph g_e, vertexes and edges of which are $v_{11}, v_{12}, v_{23}, v_{14}$, and (v_{11}, v_{12}), (v_{11}, v_{23}), (v_{11}, v_{14}), (v_{12}, v_{23}), (v_{12}, v_{14}), (v_{23}, v_{14}), respectively.

So far, we model each entity as a complete $|A|$-partite graph, from which there exists a one-to-one correspondence between virtual records of the entity and candidate graphs of the graph. In the next part, we define the graph score of each candidate graph and offer it as the adjusted reord score of the corresponding virtual record.

3.2 Definition of graph score

We define the graph score of a candidate graph based on weights of its vertexes and edges, where weights of a vertex and an edge are defined as follows:

Definition 6 (Weight of a vertex) The weight of v_{ji}, denoted as $w(v_{ji})$, is defined as:

$$w(v_{ji}) = sim(q.a_i, r_j.a_i) \tag{2}$$

Definition 7 (Weight of an edge) The weight of edge $(v_{ji}, v_{j'i'})$, denoted as $w(v_{ji}, v_{j'i'})$ is defined as:

$$w(v_{ji}, v_{j'i'}) = 1 - \frac{|T(r_j.a_i) - T(r_{j'}.a_{i'})|}{T_{\max} - T_{\min}} \tag{3}$$

where $T(r_j.a_i)$ and $T(r_{j'}.a_{i'})$ are the timestamps of r_j and $r_{j'}$, $T_{\max}$ and $T_{\min}$ are the timestamps of the oldest and the latest records in the database. $T_{\max}$ and $T_{\min}$ are used for the purpose of normalization.

Equation 3 captures the idea that if the time discrepancy of two vertexes are very large, then they should not be mixed up easily, and $w(v_{ji}, v_{j'i'})$ will be small. Clearly, $0 \leq w(v_{ji}, v_{j'i'}) \leq 1$. As an extreme case, if $r_j.a_i$ and $r_{j'}.a_{i'}$ are from a single record, then $T(r_j.a_i) = T(r_{j'}.a_{i'})$ and $w(v_{ji}, v_{j'i'}) = 1$. Informally, we compute the graph score of a candidate graph, g_e, by the following steps:

- For each vertex v, we assign a penalty to its weight $w(v)$ by multiplying $w(v, v')$ with respect to its edge (v, v');
- The averaged weight of v is computed with respect to all its edges;
- Finally, the averaged weight of all its vertexes is computed.

Formally, the graph score is defined as follows:

Definition 8 (Graph score) For a given candidate graph g_e, the graph score of g_e, denoted as $sim(q, g_e)$, is defined as:

$$sim(q, g_e) = \frac{1}{|A|(|A|-1)} \sum_{v \in g_e} \left(\sum_{v' \in g_e \wedge v' \neq v} w(v) w\left(v, v'\right) \right) \tag{4}$$

Example 5 (Graph score) Consider a virtual record (Jennifer Smith, Manager, General Support, 33650011) of entity "0001" that is shown in Table 1. It corresponds to the candidate graph $(v_{11}, v_{12}, v_{23}, v_{14})$ that is shown in Figure 1b. Assume a user issues a search query (Smith, Manager, General Support, 33650017), and $|T_{\max} - T_{\min}| =$ 10 years. Weights of vertexes and edges are shown in Figure 2. Based on (4), we can compute the graph score of the candidate graph which is 0.69.

For each virtual record, we compute the graph score of its corresponding candidate graph, and the graph score is offered as the adjusted record score of the virtual record. It is worth mentioning that there might be a variety of similarity measures to quantify the adjusted record score of each virtual record. In the next section, however, we will demonstrate that our proposed algorithm is a generic one and can be extended to apply in other similarity measures.

3.3 Definition of entity score

For each candidate graph of an entity, suppose we have already computed its graph score, then the maximum graph score will be severed as the entity score.

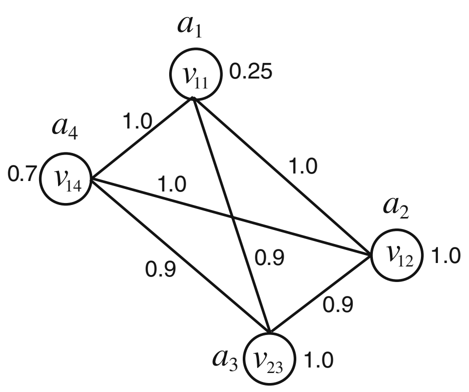


Figure 2 Graph score of the candidate graph shown in Figure 1b.

Definition 9 (Optimal candidate graph) Given a $|A|$-partite graph $G_e(V, D)$ for entity e, the candidate graph in $G_e(V, D)$ that is the most similar to q is called optimal candidate graph, and denoted as g_e^*. Mathematically,

$$sim(q, g_e^*) = \max \{sim(q, g_e) | \forall g_e \in G_e(V, D)\}. \tag{5}$$

Trivially, the entity score of e is modeled as:

Definition 10 (Entity score) Let q be a search query and e be an entity. The entity score of e, $sim(q, e)$, is:

$$sim(q, e) = sim\big(q, g_e^*\big). \tag{6}$$

For each of entity, we can compute its entity score based on (5) and (6). After entity scores of all entities have been computed, the top K entities can be identified.

4 Proposed work—dominating tree algorithm

In the previous section, for each entity, we propose a model to define the graph score of each candidate graph (corresponding to a virtual record) and the maximum graph score is offered as its entity score. However, as we described in Section 1, the number of candidate graphs is $|e|^{|A|}$ for each entity e and the total number of candidate graphs of entity set E is $\sum_{e \in E} |e|^{|A|}$. Clearly, extracting the top K entities by computing the graph score of each candidate graph is unpractical.

In this paper, we develop a novel algorithm called Dominating Tree Algorithm (DTA) to extract the top K entities. DTA is implemented based on the idea of bounding-pruning-refining.

– **Bounding:** For each entity, instead of computing its entity score directly, we first compute the range (lower bound and upper bound) of its entity score. As we will show later, the computational cost of obtaining ranges of each entity score is much cheaper than that of obtaining its exact entity score. Formally, we

use $sim_l(q, e)$ and $sim_u(q, e)$ to represent the lower bound and upper bound of $sim(q, e)$, respectively;
- **Pruning:** For an entity e, if there exist at least K entities, each of which has the lower bound of its entity score not less than the upper bound of the entity score of e, then e can be pruned; Also, if there exist less than K entities, each of which has the upper bound of its entity score greater than the lower bound of entity score of e, then e can be verified as a member of top K entities;
- **Refining:** If e cannot be pruned, then we need to get tighter bounds of $sim(q, e)$. We refine the bounds of $sim(q, e)$ according to a novel tree called Dominating Tree (DTree). With DTree, we can compute and refine the bounds of each entity by selectively computing graph scores of a small subset of candidate graphs only.

The iteration of bounding, pruning and refining goes on until we can extract the top K entities. As we will show later, DTA can extract the top K entities under other similarity measure criterion to define the adjusted record score besides using the complete $|A|$-partite graph model.

4.1 Initializing the upper and lower bounds

In order to initialize the upper and lower bounds of the entity score of e, we introduce a definition called base graph:

Definition 11 (Base graph) A candidate graph $g_e(V, D)$ in $G_e(V, D)$ is said to be the base graph if and only if: $\forall v_{ji} \in g_e.V, \forall v_{j'i} \in V_i, w(v_{ji}) \geq w(v_{j'i})$.[4] For the sake of brevity, the base graph of $G_e(V, D)$ is denoted as g_e^{base}.

Clearly, base graph g_e^{base} corresponds to the best virtual record γ^{base} that is defined in Definition 2. For entity e, as g_e^{base} is one of its candidate graphs, according to (5) and (6), we can initialize $sim_l(q, e)$ as follows:

Lemma 3 (Lower bound) *For a given entity e, let g_e^{base} be the base graph of e that corresponds to its best virtual record. Lower bound of entity score of e, $sim_l(q, e)$:*

$$sim_l(q, e) = sim\left(q, g_e^{\text{base}}\right) \tag{7}$$

Lemma 4 (Upper bound) *The upper bound of entity score of an entity e, $sim_u(q, e)$ is:*

$$sim_u(q, e) = sim\left(q, \gamma^{\text{base}}\right) = \frac{1}{|A|} \sum_{v \in g_e^{\text{base}}} w(v) \tag{8}$$

[4]Notice that v_{ji} is from group V_i (see Lemma 1). This sentence meas that the weight of v_{ji} is not less than that of any vertex in V_i.

Proof $\forall v, v' \in g_e.V, w(v, v') \leq 1$. As such, for each candidate graph, g_e, we can have:

$$\begin{aligned} sim(q, g_e) &= \frac{1}{|A|(|A|-1)} \sum_{v \in g_e} \left(\sum_{v' \in g_e \wedge v' \neq v} w(v) w\left(v, v'\right) \right) \\ &\leq \frac{1}{|A|(|A|-1)} \sum_{v \in g_e} \left(\sum_{v' \in g_e \wedge v' \neq v} w(v) \right) \\ &= \frac{1}{|A|} \sum_{v \in g_e} w(v) \leq \frac{1}{|A|} \sum_{v \in g_e^{\text{base}}} w(v) = sim\left(q, \gamma^{\text{base}}\right) \end{aligned}$$

As graph score of each candidate is not greater than $sim(q, \gamma^{\text{base}})$, according to (5) and (6), $sim_u(q, e) = sim(q, \gamma^{\text{base}}) = \frac{1}{|A|} \sum_{v \in g_e^{\text{base}}} w(v)$. □

Actually, (7) and (8) are in accord with the explanation that was described in Section 1. On one hand, if we only consider the record score of the best virtual record, and ignore the heterogeneity penalty, then the entity score will be maximized which is $sim_u(q, e)$. On the other hand, if we consider not only the record score, but also the heterogeneity penalty, then there might exist one virtual record, whose adjusted record score is greater than that of the best virtual record, which is $sim_l(q, e)$. Clearly, this initialization of bounding the entity score of each entity is independent on the similarity measures.

4.2 Pruning based on the initial upper and lower bounds

For each entity e, if we have obtained $sim_l(q, e)$ and $sim_u(q, e)$, then the following two rules can be applied.

Rule 1 (Entity pruning) *Given an entity e, if there exist K or more entities with their lower bound of entity scores not less than $sim_u(q, e)$, then e cannot be in the top K entities and can be pruned.*

Rule 2 (Entity selection) *Given an entity, e, if there exist less than K entities with their upper bounds of entity scores greater than $sim_l(q, e)$, then e must be a member of the top K entities.*

By Rules 1 and 2, for entities which can be determined whether they are in the top K entities or not, no further refinement of their bounds is required; otherwise, for the rest of entities, we need to refine $sim_l(q, e)$ and $sim_u(q, e)$, for each entity e of them.

4.3 Refining the upper and lower bounds

For an entity e, we expect to determine whether it is in the top K entities by computing graph scores of as few candidate graphs as possible. Finding an optimal subset of candidate graphs for each entity to compute is a very difficult online problem.

Here, we propose a heuristic approach based on a tree structure called Dominating Tree (DTree). The objective of proposing this tree is that candidate graphs in each entity will be accessed according to a certain order such that the number of candidate graphs can be pruned as many as possible. The formal definition of DTree is as follows:

Definition 12 (Dominating tree (DTree)) A DTree is constructed based on a given entity e, which must satisfy the following properties:

1. There exists one-to-one correspondence between nodes in a DTree and candidate graphs of $G_e(V, D)$;
2. The root node corresponds to the base graph of e;
3. Every node is labeled with a sequence of digits with length $|A|$ called graph ID. The meanings of a graph ID are: (1) each digit in position i of it corresponds to the vertex that is from group V_i, and weight of the vertex is ranked in the position of the value of digit; (2) this node corresponds to the candidate graph that consists of these vertexes. Especially, the graph ID of the root is all 1's;
4. The graph ID of a child node will always have one and only one digit one unit smaller than that of its parent node (if any), and keep the rest of digits the same.

DTree organizes the candidate graphs using its nodes. Formally, a node in the DTree corresponds to candidate graph g_e^s is termed as N^s, which refers to virtual record γ^s as well (see Lemma 2).

According to the property 1 of Definition 12, for an entity e, the total number of nodes in its DTree is $|e|^{|A|}$. The maximum number of children of a node is $|A|$ based on the property 1 of Definition 12.

Example 6 (DTree) Consider entity "0001", denoted as e, which is shown in Table 1. Two records of e are shown in Table 5. Suppose a user issues a search query (Smith, Manager, General Support, 33650017). The matrix of attribute scores of e is computed and shown as follows:

	a_1	a_2	a_3	a_4
r_2	0.25	0	1	0.5
r_1	0.25	1	0	0.7

Figure 3a shows the corresponding n-partite graph of e. Vertexes of each group are sorted according to the descending order (arrow direction) of their weights. We can see that vertexes v_{11}, v_{12}, v_{23}, v_{14}, are ranked in the first position in individual groups. As a result, according to Definition 11, vertexes v_{11}, v_{12}, v_{23},v_{14}, and edges connecting every two vertexes of them constitute the base graph. According to Definition 12, we can construct a complete DTree for e, which is shown in Figure 4. For example, the node with graph ID 1211 in the DTree corresponds to the candidate graph

Table 5 Records of entity "0001" shown in Table 1.

ID	Name	Title	Workplace	Phone	Last update
r_2	Jennifer Smith	Senior officer	General support	33650001	2008-01-01
r_1	Jennifer Smith	Manager	Infrastructure	33650011	2009-01-01

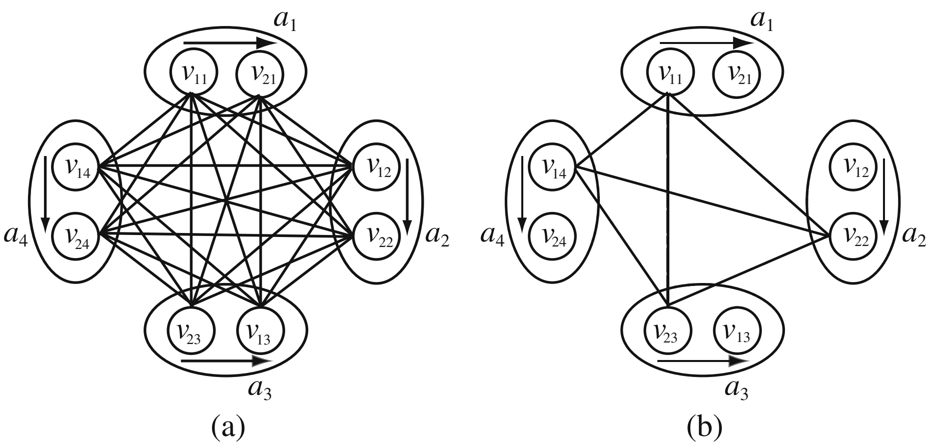


Figure 3 Another $|A|$-partite graph and a candidate graph. **a**. Vertexes of each group are sorted by weights. **b**. Candidate graph corresponds to graph ID 1211.

formulated by $\{v_{11}, v_{22}, v_{23}, v_{14}\}$, which is shown in Figure 3b. The reason why the node with graph ID 121 corresponds to candidate graph $\{v_{11}, v_{22}, v_{23}, v_{14}\}$ is that v_{11} is ranked in the 1st position in group over attribute a_1, v_{22} is ranked in the 2nd position in group over attribute a_2, v_{23} is ranked in the 1st position in group over attribute a_3, and v_{14} is ranked in the 1st position in group over attribute a_4, respectively.

It is worth mentioning that when we are trying to extract the top K entities, we *do not* need to formulate the whole DTree for each entity at the very beginning. Instead, we only identify the root node of the DTree for each entity first, by which the lower bound and the upper bound entity score of each entity can be calculated according to (7) and (8). Then, for those entities, which can be verified whether they are in the top K entities or not, we destroy their DTrees; for the rest of entities, we gradually extend the nodes of their DTrees to refine the lower and upper bounds of $sim(q, e)$.

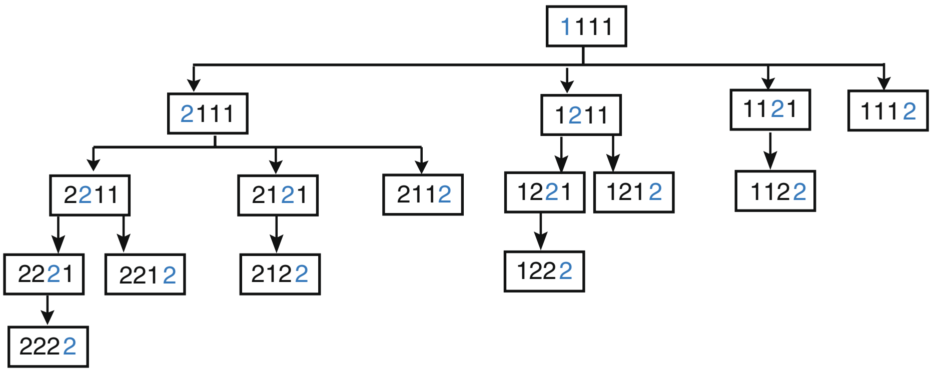


Figure 4 An example of dominating tree.

Before describing how to perform this refinement for the bounds of each entity, we define one relation, called dominance relation, in the following:

Definition 13 (Dominate, $\prec$) Given two nodes N^s, N^t in the DTree of an entity e, N^s is said to dominate N^t (denoted as $N^s \prec N^t$) if and only if: each digit of the graph ID of node N^s is less than or equal to the corresponding digit of the graph ID of N^t.

For example, a node with graph ID "1211" dominates another node with graph ID "1221" as each digit of the former is not greater than that of the latter over each individual position.

Lemma 5 *Given two nodes N^s, N^t in the DTree of an entity e, If $N^s \prec N^t$, then $sim(q, \gamma^s) \geq sim(q, \gamma^t)$.*

Proof Notice that candidate graphs g_e^s and g_e^t correspond to N^s and N^t, respectively, and virtual records γ^s and γ^t corresponds to g_e^s and g_e^t, respectively. As $N^s \prec N^t$, the weight of vertex of g_e^s is not less than that of g_e^s over individual attribute, that is, the attribute score of γ^s is not less than that of γ^t over each individual attribute. As a result, we can have:

$$\begin{aligned} sim(q, \gamma^s) - sim(q, \gamma^t) &= \sum_{a_i \in A} sim(q.a_i, \gamma^s.a_i) - \sum_{a_i \in A} sim(q.a_i, \gamma^t.a_i) \\ &= \sum_{a_i \in A} \left(sim(q.a_i, \gamma^s.a_i) - sim(q.a_i, \gamma^t.a_i) \right) \\ &\geq 0 \end{aligned}$$

□

Corollary 1 *Given a node N^s of DTree, for any of its offspring nodes (if any), denoted as N^t, $sim(q, \gamma^s) \geq sim(q, \gamma^t)$.*

Based on Corollary 1, for each node N^s and any of its offspring nodes, N^t, the record score of virtual record γ^s corresponding to N^s is not less than that of γ^t corresponding to N^t. As defined in Definition 8, for each virtual record γ^s and its corresponding candidate graph g_e^s, we can have $sim(q, \gamma^s) \geq sim(q, g_e^s)$. Thus, we can have another corollary.

Corollary 2 *Given a node N^s of DTree, for any of its offspring nodes (if any), denoted as N^t, $sim(q, g_e^t) \leq sim(q, \gamma^s)$.*

For a virtual record γ^s, if we can verify that $sim(q, \gamma^s)$ is not greater than $sim_l(q, e)$, then we are sure that γ^s and virtual records corresponding to offspring nodes of N^s cannot help refine the entity score of e. As a result, we can obtain the following pruning rule:

Rule 3 (Virtual record pruning) *Given a virtual record γ^s of entity e, if $sim_l(q, e) \geq sim(q, \gamma^s)$, then γ^s and virtual records corresponding to offspring nodes of N^s can be pruned.*

Example 7 (Virtual record pruning) Consider a node N^{s_1} with graph ID 2111 of the DTree which is shown in Figure 4. Suppose we know $sim_l(q, e) \geq sim(q, \gamma^{s_1})$, then all virtual records, which correspond to the offspring nodes of N^{s_1}, can be pruned. Specifically, from Figure 4 we can see that there are totally 7 nodes can be skipped, that is, 7 virtual records can be pruned. Similarly, for another node N^{s_2} with graph ID 1211, if we know $sim_l(q, e) \geq sim(q, \gamma^{s_2})$, then 3 more virtual records can be pruned. Finally, the optimal candidate graph, g_e^*, must be in the candidate graphs which correspond to the nodes with graph ID's either 1111, 1121,1122 or 1112. In summary, we only need to evaluate at most 6 out of 16 virtual records under this case.

For each entity e, we start from the root node of its DTree and $sim_l(q, e)$, $sim_u(q, e)$ are initialized using the best virtual record (corresponding to the root node) based on the (7) and (8). A priority queue, H, is exploited to maintain the virtual records of e which are sorted in the descending order of their record scores. At the first beginning, H contains γ^{base} only. $\forall e$, if e cannot be pruned by Rules 1 and 2, the **Refining Strategy** of e is conducted as follows:

1. At each iteration, we first pop the top virtual record (denoted as γ^s) from H. Then, for each child node N^t of N^s in the DTree, we compute the record score of γ^t (corresponding to N^t). If $sim(q, \gamma^t) \leq sim_l(q, e)$, then, based on Pruning Rule 3, all offspring nodes of N^t will be ignored; otherwise, we compute $sim(q, g_e^t)$, use it to refine $sim_l(q, e)$, and push the virtual record γ^t to H. Notice that, for any of the rest of virtual records that we have not seen, its record score is not more than that of the top virtual record of H. Let γ^v be the current top virtual record of H. Clearly, based on Corollary 1, $sim_u(q, e)$ can be refined to $sim(q, \gamma^v)$.
2. The above iteration will continue until H is empty or the entity score of e has been identified based on Pruning Rule 4.

Rule 4 (Similarity pruning) *If $sim_u(q, e) = sim_l(q, e)$, then we can immediately identify $sim(q, e)$ as $sim(q, e) = sim_l(q, e)$.*

According to Rule 4, for each entity e, if we know $sim_u(q, e) = sim_l(q, e)$, then $sim(q, e)$ can be derived and no further computation of e is needed.

4.4 Implementation details

The algorithm of DTA is described in Algorithm 1. As we have conducted a thorough discussion on the process of identifying top K entities, in this part, we explain some critical steps of implementation details.

Given a search query q, for each entity, attribute scores of its records over individual attributes are first calculated (line 3). Attribute scores over each individual attribute are sorted by the descending order of their values (line 3). Then we can identify the best virtual record γ^{base}, and push it into H_e (line 5). Based on (7) and (8), $sim_l(q, e)$ and $sim_u(q, e)$ can be initialized using γ^{base} (line 6).

After all entities are bounded, we sort these entities based on the descending order of their lower bound of entity scores such that we can obtain the K^{th} greatest lower bound and upper bound of entity scores (line 9). Starting from the $(K + 1)^{\text{th}}$ entity, e, we first check whether it can be pruned based on Rule 1 (line 11). If it can be pruned,

Algorithm 1: DTA(q, K)

```
input  : A search query ($q$) and a target ($K$)
output: Top $K$ entities ($Top_K$)
1  $Top_K = \emptyset$;
2  foreach $e \in E$ do
3      compute attribute scores of $e$ and sort them over individual attributes by the descending order ;
4      $H_e \leftarrow \emptyset$;
5      identify $g_e^{base}$ and push $\gamma^{base}$ into $H_e$;
6      initialize $sim_l(q,e)$ and $sim_u(q,e)$ using $\gamma^{base}$; // According to Eq. 7 and Eq. 8
7  end
8  while true do
9      sort all $e$'s based on the descending order of $sim_l(q,e)$, and obtain $l_k$, $u_k$ which are the $K^{\text{th}}$
       greatest lower bound and upper bound of entity scores, respectively;
10     foreach $e_i \in E$ do
11         if $i > K$ && $e_i$ can be pruned based on Rule 1 then
12             $H_{e_i}.clear()$,   $H_{e_i} \leftarrow \emptyset$;
13             remove $e_i$ from $E$;
14         else if $e_i$ can be verified based on Rule 2 then
15             add $e_i$ into $Top_K$;
16             if $|Top_K| = K$ then return $Top_K$;
17             $H_{e_i}.clear()$,   $H_{e_i} \leftarrow \emptyset$;
18             remove $e$ from $E$;
19         else
20             if $H_{e_i} = \emptyset$ or $sim_l(q,e_i) = sim_u(q,e_i)$ then $sim(q,e_i) = sim_l(q,e_i)$ ;
21             else refine$(e_i, l_k)$; // Algorithm 2 ;
22         end
23     end
24 end
```

then we clear its priority queue, and remove it from E (line 12–13). Starting from the first entity, e, of E, we check whether e can be verified as an element of Top_K based on Rule 2 (line 14). If it is, we add it into Top_K, release the resources that H_e occupies, and remove e (line 17–18). In addition, if we have extracted Top_K entities, then we return the results (line 16). If e cannot be determined whether it is in the top K entities or not (line 19), we will refine the lower bound and upper bound of its overall score (lines 20–22). Specifically, according to Rule 4, we only need to refine the entities whose overall scores have not been calculated yet (line 21). Algorithm 2 outlines the process of refinement.

In the following definition, for a given non-leaf node of DTree, we describe how to generate children nodes of it.

Definition 14 (Child) If N is a root node: N_a is a child of N iff (1) one digit of N_a's graph ID is 1 larger than that of N's graph ID; (2) other digits of N_a's graph ID are the same as N's. If N is neither root node nor leaf node: let k be the position of the digit in N's graph ID that is different from its parent's graph ID. N_b is a child of N iff (1) the first $k-1$ digits of N_b's graph ID are the same as those of N's; (2) one of the rest digits is 1 larger than that of N_b's and the other digits are the same.

In Algorithm 2, we first pop the virtual record γ^s from the top of H_e (line 1). Based on Definition 14, we can obtain all children nodes of node N^s that corresponds to γ_e^s (line 2). For each child node, N^t, we compute record score $sim(q, \gamma^t)$ and

Algorithm 2: `refine`(e, l_K)

```
   input : An entity $e$, the $K^{\text{th}}$ greatest lower bound of $E$, $l_K$
   output: The refined entity $e$
1  pop virtual record $\gamma^s$ from the top of $H_e$;
2  let $\tilde{N}^s$ be the children node of $N^s$;
3  foreach $N^t \in \tilde{N}^s$ do
4      compute $sim(q,\gamma^t)$ and $sim(q,g_e^t)$;
5      if $sim(q,\gamma^t) > \max\{sim_l(q,e), l_K\}$ then
6          $sim_l(q,e) \leftarrow \max\{sim_l(q,e), sim(q,g_e^t)\}$;
7          push $\gamma^t$ into $H_e$;
8      end
9  end
10 if $H_e$ is empty then
11     $sim_u(q,e) \leftarrow sim_l(q,e),\ sim(q,e) \leftarrow sim_l(q,e)$;
12 else
13     let $\gamma^v$ be the top virtual record in $H_e$;
14     $sim_u(q,e) \leftarrow sim(q,\gamma^v)$;
15     if $sim_u(q,e) \leq sim_l(q,e)$ then
16         $sim_u(q,e) \leftarrow sim_l(q,e)$;
17     end
18 end
```

adjusted record score $sim(q, g_e^a)$ (line 4). If record score $sim(q, \gamma^t)$ is not greater than $sim_l(q, e)$, we can be sure than all offspring nodes of N^s can be pruned safely. Furthermore, if $sim(q, \gamma^t)$ is not greater than l_K, we are sure as well that using virtual records corresponding to offspring nodes of N^t cannot make e as a member of top K entities (line 5), and all offspring nodes of N^s can be pruned safely as well. Otherwise, we use adjusted record score $sim(q, g_e^a)$ to refine $sim_l(q, e)$, and push γ^t into H_e as some offspring nodes can be exploited to refine $sim_l(q, e)$ (line 5–8). From line 10–18, we use the current top virtual record to refine the upper bound of entity score of e, $sim_u(q, e)$, and if all virtual records (including pruned virtual records) have been checked, $sim(q, e)$ can be obtained.

In general, the main idea of DTA approach is independent on our proposed evaluation model that is used to define the entity score of each entity.

– First, the initialization of $sim_l(q, e)$ and $sim_u(q, e)$ is based on the best virtual record of each entity, which is independent on the evaluation model;
– Second, the pruning strategy is generic and is independent on the evaluation model as well;
– Third, we refine $sim_l(q, e)$ and $sim_u(q, e)$ by computing the adjusted record scores of some selected virtual records of entities which are organized based on DTree. As described in the previous part, DTree is also independent on the evaluation model.

The time complexity of DTA approach is dependent on the evaluation model. For example, if we ignore the time discrepancy among attribute values of each virtual record, then the problem becomes the comparison with the best virtual record of each entity. The time complexity of DTA is $O(|A| * |e| * |E| * logK)$, where for each entity, the time complexity of identifying best virtual record is $O(|A| * |e|)$. For using $|A|$-complete partite Graph model, the time complexity is dependent on the data distribution of weights of vertexes and edges. Imaging that, distribution of weights of vertexes and edges is quite similar over each entity, then none entity will be easily pruned and at the worst case, each virtual record might be considered. Thus, the time complexity of DTA is exponential. However, this worse case rare happens in practice as, most entities are not similar to the search query according to [40], and based on our experiment results, DTA performs very efficient because of its powerful pruning rules.

5 Evaluation

5.1 Experiment setup

All experiments are conducted on a PC with Intel 1.66GHz dual core CPU and 1GB memory under Windows XP operating system and PostgreSQL. 8.2[5] DMBS. All programs are written using Java 1.6 as in-memory algorithms, with all records loaded into memory before they are run, and compiled using JDK 1.6.

As there lacks of public benchmark, we use three synthetic datasets, each of which has different distributions of the weight of vertexes, to study the scalability of our proposed algorithm.

- Uniform distribution: over each attribute, weights of vertexes follow uniform distribution between (0, 1). This should be the worst case of the data distribution as, in practice, most of entities are often not similar to the query, and their attribute scores are quite small. For each entity, the weight of edge between v_{ji} and $v_{j'i'}$ of e is set to $\frac{|j-j'|}{|e|}$, where each entity in the dataset has the same number of records. The definition of weight keeps the same under the other distributions.
- Normal distribution: over each attribute, weights of vertexes follow normal distribution between (0, 1) by its mean = 0, and variance = 0.5. Since each attribute score cannot be less than 0, we obtain its absolute value when it is less than 0. The confidence intervals between 0–0.5, i.e., the possibility of attribute scores locate between 0 and 0.5 is 68.2%, and the confidence intervals between 0.5–1 is 27.2%. When a value is greater than 1, we reset it to 1.
- Log-Normal distribution: over each attribute, weights of vertexes follow log-normal distribution, where we set the variance to 4, and the mean to 0 such that most values will locate between 0–0.5. Notice that the larger the variance is, the more the number of values near to zero will be.

Examples of the above three data distributions are shown in Figure 5a–c. We use NBA dataset to evaluate the effectiveness of our proposed model.[6]

- We craw this dataset from the website http://www.databasebasketball.com. It contains 3572 basketball players, where each player is regarded as an entity. Attribute names of this dataset are (fname, lname, team, opp, pts, reb, ast, ctime), and each record of this dataset describes the points, rebounds, and assists of a player at the team with the opponent on ctime. On average, each player contains 50 records. From this dataset, we randomly select 50 entities as the query entities.

[5] http://www.postgresql.org/

[6] As the size of NBA dataset is too small, we cannot effectively evaluate the scalability of DTA.

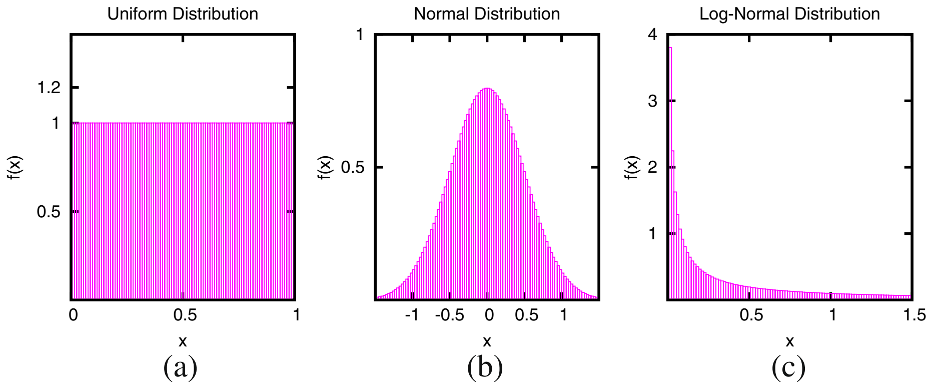


Figure 5 Examples of data distribution. **a** Uniform distribution. **b** Normal distribution. **c** Log-normal distribution.

For each entity, we randomly extract a virtual record, and use UIS database generator[7] to relax the values of the virtual record so as to mimic the scenario where some errors are introduced when the users type search queries (e.g. Kobe Bryant is changed to Koby Brient).

5.2 Performance on top K entity extraction

We evaluate the scalability of DTA on the above three synthetic datasets by varying the following four factors: (1) the number of entities, $|E|$; (2) the number of records, $|e|$; (3) the number of attributes, $|A|$; (4) the value of K. As the naive approach performs at least two order of magnitude worse than DTA over each dataset, for the sake of clarity, we only show the performance of DTA here. We conduct four experiments to evaluate the sensitivities of DTA. In each experiment, we vary one of the factor and keep the other factors constant.

Figure 6a–d show the computational time of DTA over the above synthetic datasets with different experimental settings.

- In Figure 6a, $|E|$ varies from 50,000 to 350,000, whereas $|A| = 4$, $|e| = 10$, and $K = 10$. The total number of virtual records varies from $50{,}000 \times 10^4$ to $350{,}000 \times 10^4$;
- In Figure 6b, $|e|$ varies from 5 to 35, whereas $|E| = 100{,}000$, $|A| = 4$, and $K = 10$. The number of virtual records varies from $100{,}000 \times 5^6$ to $100{,}000 \times 35^6$;
- In Figure 6c, $|A|$ varies from 2 to 8 whereas $|E| = 100{,}000$, $|e| = 10$, and $K = 10$. The total number of the virtual records varies from $100{,}000 \times 10^2$ to $100{,}000 \times 10^8$;
- In Figure 6d, K varies from 5 to 35, whereas $|E| = 100{,}000$, $|A| = 4$, and $|e| = 10$. The number of virtual records is $100{,}000 \times 10^4$.

[7] http://www.cs.utexas.edu/users/ml/riddle/data.html

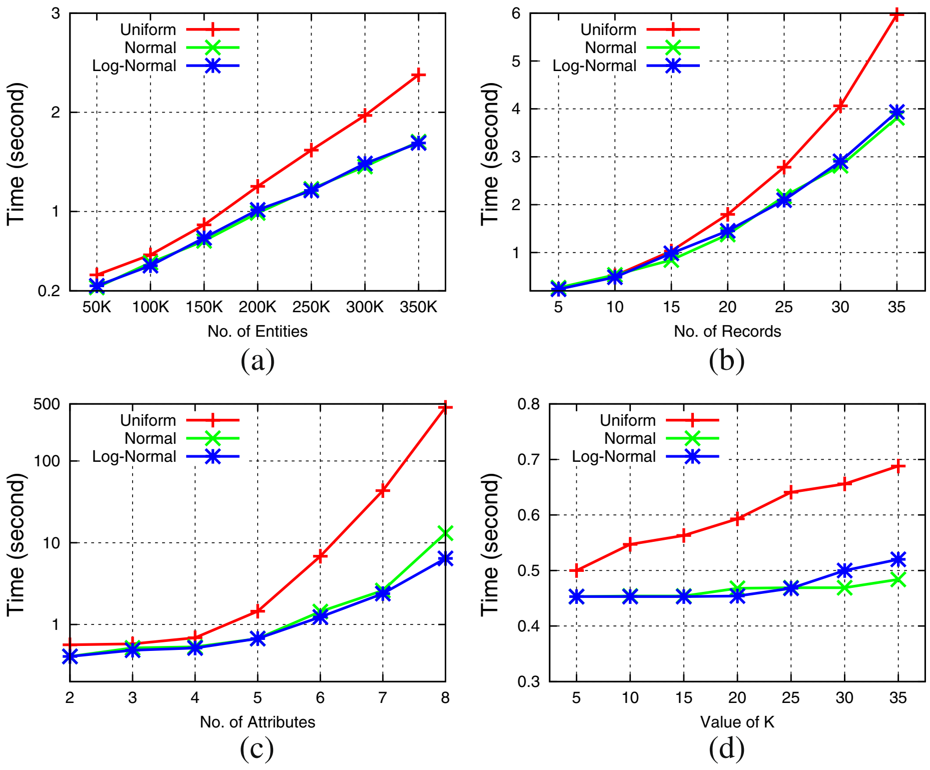


Figure 6 Efficiency of top K entity extraction. **a** $|e| = 10$, $|A| = 4$, $K = 10$. **b** $|E| = 100K$, $|A| = 4$, $K = 10$. **c** $|E| = 100K$, $|e| = 10$, $K = 10$. **d** $|E| = 100K$, $|e| = 10$, $|A| = 4$.

From all these four figures, we can find that DTA performs better on datasets with Normal Distribution and Log-Normal Distribution than the dataset with Uniform Distribution. This statement is easy to follow since entity score of each entity in Uniform Distribution dataset is quite similar such that it will take much time to differentiate them with each other. For Normal Distribution dataset and Log-Normal Distribution dataset, more than a half of values are less than 0.5 (see the confidence intervals), a large number of entities can be pruned. In addition, we can find that DTA performs nearly the same on Normal Distribution dataset and Log-Normal Distribution dataset.

Independent on the datasets, we can derive the following observations:

- from Figure 6c, the computational time of DTA increases exponentially when the number of attributes increases;
- from Figure 6b, the computational time of DTA is proportional to $|A|$ power of $|e|$. The above two observations are in accord with the intuition that the number of candidate graph is $|E| * |e|^{|A|}$;
- from Figure 6a, d, the computational time of DTA increases linearly with respect to the number of entities and the value of K.

In summary, the order of factors that most affect the performance are: the number of attributes $>$ the number of records $>$ the number of entities $>$ the value of K. Actually, attribute scores between most of records and the search query are quite near to zero. In [40], they performed an experimental analysis on the attribute scores of authors in the DBLP data. The result shows that majority of similarity entries have very small values which lie within a small range (0.005–0.015), and attribute scores of 1.4% of authors (about 123K of 8,692,365) are greater than 0.1. Clearly, the synthetic datasets we are using here actually are worse than the real datasets, and the performance of DTA will be much better on the real dataset. As such, to some extent, our synthetic datasets are of great generality.

Another point we need to clarify is that, in the synthetic datasets, we generate attribute scores according to different data distributions, however, in practice, we are required to compute attribute score of each record online. This is challenging especially when the cardinality of records in the dataset is large. To handle this problem where the token based similarity measures are exploited, a common solution is that, over each individual attribute, we create an inverted index using q-grams of each attribute value. The main idea is that, using the inverted index over each attribute, we only access such attribute values that have at least one common q-gram with the search query over this attribute and compute its attribute score. We will illustrate and solve this problem in our future work.

Figure 7a–d show the maximum number of candidate graphs of all entities when we use DTA approach. We are desired to use these four figures to illustrate the pruning power of our proposed algorithm. Notice that the total number of candidate graphs is $|e|^{|A|}$.

– In Figure 7a, the maximum number of candidate graphs varies from 7 to 17, 7 to 11, 7 to 12 for Uniform Distribution, Normal Distribution, and Log-Normal Distribution datasets, respectively, while the total number of candidate graph is 10,000.
– In Figure 7a, the maximum number of candidate graphs varies from 5 to 88, 5 to 31, 5 to 25 for Uniform Distribution, Normal Distribution, and Log-Normal Distribution datasets, respectively, while the total number of candidate graph varies from 625 to $1,500,625$.
– In Figure 7a, the maximum number of candidate graphs varies from 1 to 18,602, 1 to 3,887, 1 to 996 for Uniform Distribution, Normal Distribution, and Log-Normal Distribution datasets, respectively, while the total number of candidate graph varies from 100 to $100,000,000$.
– In Figure 7a, the maximum number of candidate graphs varies from 8 to 22, 7 to 17, 7 to 20 for Uniform Distribution, Normal Distribution, and Log-Normal Distribution datasets, respectively, while the total number of candidate graph is 10,000.

In general, only a small portion of candidate graphs of each entity is accessed based on the above analysis. Clearly, our proposed approach, DTA, is effective.

5.3 Effectiveness of entity resolution

We study the effectiveness of our proposed model on the NBA real dataset. We first randomly select 50 players from this dataset. For each selected player (entity), we

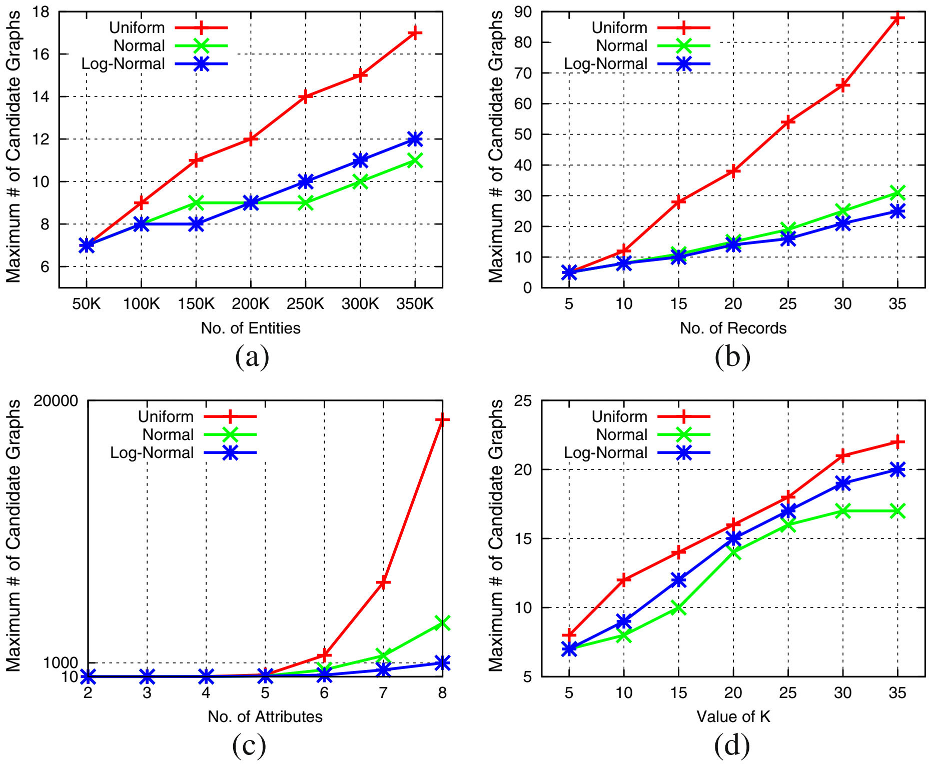
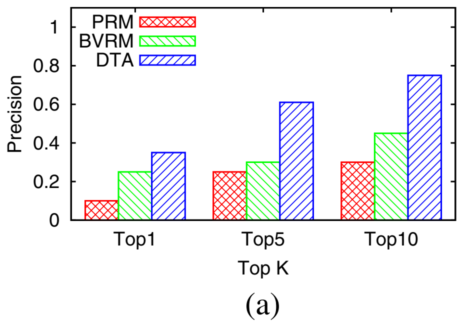


Figure 7 Maximum heap size of DTA in top K entity extraction. **a** $|e| = 10, |A| = 4, K = 10$. **b** $|E| = 100K, |A| = 4, K = 10$. **c** $|E| = 100K, |e| = 10, K = 10$. **d** $|E| = 100K, |e| = 10, A = 4$.

randomly combine his attribute values from his records to formulate a search query, where no two attribute values are involved in the same attribute and the number of attribute values is $|A|$. For each search query, we use UIS database generator to relax the values in the query so as to mimic the scenario where some errors are introduced when the users type search queries. UIS database generator allows us to control error types and error rates. We generate two collections of query records, one with low error rate (20%) and the other one with high error rate (40%). For the purpose of comparison, we implement the following two approaches:

– *Per-record match (PRM)* The entity score of each entity is defined based on the per-record match, which is described in Challenge 1 of Section 1.
– *"Best virtual record" Match (BVRM)* The entity score of each entity is defined based on the "best virtual record" match, which is described in Challenge 1 of Section 1.

We do not implement other existing work for comparison because none of existing work attempts to solve this problem *directly*. Modifications of the existing algorithms are necessary. This can potential be another research topic. In addition, the objective of this experiment is to demonstrate the feasibility of modeling the entity exaction problem in the way proposed in this paper, and to see whether DTA is possible to solve this problem effectively.



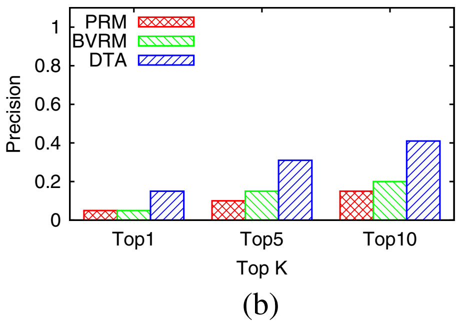


Figure 8 Results of entity disambiguation. **a** NBA$_{\text{large}}$ (low error). **b** NBA$_{\text{large}}$ (high error).

Figure 8 shows the results. The x-axes represent the value of K, where we set $K \in \{1, 5, 10\}$. Suppose a search query is generated from an entity e. When issuing this search query, if e can be identified in the top K entities, then the precision of this search is 100%, otherwise, the precision is 0. We conduct 50 different search queries to evaluate the precision for different values of K under approaches PRM, BVRM and DTA, respectively. The y-axes represent the averaged precision of the corresponding top K entity extraction over these 50 search queries. From these two figures, we can find that the results are very encouraging—DTA always performs much better than PRM and BVRM. In particular, the precision of DTA are nearly double of PRM and BVRM for most values of K in datasets. This indicates our proposed graph framework for modeling entity extraction is both theoretically and practically sound. In addition, BVRM is always better than PRM since the generated queries are mixed up and involved in different records of the same entity. PRM/BVRM are both very difficult to identify the correct entity for a search query. Besides, the averaged precision in low error rate situation is always higher than that of higher error rate situation. This is expected because the similarity match in the lower error situation must be more accurate than in the higher error situation.

It is worth mentioning that, there might be other effective evaluation methods to identify these top K entities, however, our proposed algorithm, DTA, is a generic approach and can be applied to these evaluation methods.

6 Related work

To the best of our knowledge, this is the first study on approximate entity extraction in temporal databases. Our study is related to the previous work on the approximate entity extraction problem in relational databases, and the graph optimization problem.

Approximate entity extraction in relational databases Given a search query q, the objective of approximate entity extraction is to identify all records in a relational database which refer to the same entity with q. One of the keys to solve this problem is how to define that a record r and q refer to the same entity since traditional containment or equivalence operation cannot work any further. Generally speaking, there are two ways to model this problem. (1) Threshold-based Similarity Model: if the record score between r and q is greater than a pre-specified threshold, then r is regarded as the same entity with q; (2) Top K based Similarity Model: if r is one of K records which have greatest record score in the database, where K is pre-defined by the user, then r is regarded as the same entity with q. The first method has been studied extensively in the literature [2, 10, 24, 25]. In recent years, the second method has attracted a lot of attention [33, 38]. It is worth mentioning that no matter which method is adopted, we still need to quantify how similar a record and the search query q is.

So far as we know, there exist four methods, which are explored to quantify the record score between a record r and the search query q. Without lose of generality, we assume that all attributes of r are involved in q.

- **Method 1:** Concatenate each record into an individual string such that the record score between a record and a search query can be quantified by calculating the attribute score of two corresponding strings. As we know, how to quantify the attribute score between two strings has been well-studied in the literature. According to [9], given two strings X and Y, the similarity functions, which are utilized to quantify the attribute score between X and Y can be classified into 5 classes. (1) Edit-based similarity functions calculate the attribute score based on the transformation cost of X to Y, including Edit Distance [40], Jaro-Winkler [35]; (2) Overlap-based similarity functions calculate the attribute score based on the common tokens of X and Y, including Intersect [28], Jaccard [28], Dice Coefficient [32]; (3) Weight-based similarity functions calculate the attribute score based on the common tokens of X and Y, where each token is attached with a weight, including Weighted Jaccard [28], TF-IDF Cosine Coefficient [12], BM25 [9]; (4) Language Modeling functions calculate the attribute score based on the probabilistic model imposed on elements of X and Y, including Language Modeling [9], and Hidden Markov Models [9]; (5) Hybrid functions calculate the attribute score based on at least two similarity functions above.
  In this method, each record and a search query are regarded as two individual strings, respectively, such that the record score between the record and the search query can be quantified by calculating the attribute score between the corresponding strings. Based on this method, Vernicaand etc. study the problem on how to efficiently extract the top K records, which are most similar to the search query [33], and Xiao etc. study the problem on how to efficiently extract K pairs of records, which are most similar to each other. In [3, 28, 34, 36, 37, 39], they focus on how to efficiently extract all records with record scores greater than a pre-specified threshold.
- **Method 2:** Pre-specify a threshold for each individual attribute such that each record, whose attribute score over the corresponding attribute is not less than the pre-specified threshold, is regarded to refer to the same entity with the search query. Given a search query q, for each record r in the database, we can quantify the attribute score between q and r over each attribute. For each attribute, we pre-specify a threshold and for any record r, if the attribute score between q and r over each attribute is not less than the corresponding threshold, then r and q refer to the same entity. This method is widely used in approximate search. In [5, 11, 20, 22], they utilize this method to verify whether a record refers to the same entity with the search query.

- **Method 3:** Explore voting theory. In [17], for a given record r, r is determined whether it refers to the same entity with a search query based on the voting theory. Specifically, for a given search query q, we can construct a ranking list of records based on their attribute scores over each individual attribute. Based on the voting theory, we can derive the final ranking list of relational records from these ranking lists over individual attributes. Finally, the top K records in the final ranking lists are said to refer to the same entity with the search query.
- **Method 4:** Utilize weight vector model. In [22], they pre-specify a weight for each attribute of the relation. For each record r and a given search query q, the record score between r and q ($sim(q, r)$) is define as:

  $$sim(q, r) = sim(q.a_i, r.a_i) \times w_i$$

  where $sim(q.a_i, r.a_i)$ is the attribute score between q and r over attribute a_i, and w_i is the weight of attribute a_i. As a result, a record with its record score greater than a pre-specified threshold refers to the same entity with q.

In order to improve the efficiency of approximate entity extraction in relational database, q-gram [15] and v-gram [24] are proposed. In [9, 16, 17], they present how to declaratively integrate similarity functions to the DBMS using an SQL interface and perform entity extraction tasks. Unfortunately, no single similarity function can always outperform the others. In [9], they evaluate the performances of some of the most widely used similarity functions upon a benchmark which contains some errors. It is worth mentioning that group linkage problem [26] proposed in 2007, is another problem that is similar to ours. Yet, the main focus between their problem and our's is quite different. In short, group linkage problem focuses on group matching in relational databases based on record granularity while ours is to extract top K entities based on record scores and attribute homogeneity.

To summarize, the existing research on approximate entity extraction problem aims to improve the accuracy and/or efficiency based on record level. They do not consider the integration of record score and attribute granularity.

Graph optimization For our graph model, we consider each entity as a complete n-partite graph [8], which is also known as Turan graph [31]. In a complete n-partite graph, there are totally $(1 - 1/n)(m^2/2)$ edges and $(m/n)^n$ candidate graphs, where m is the number of vertexes and n is the number of groups. Finding an optimal complete subgraph is a very difficult task which may need to explore all subgraphs [19]. Fortunately, our task is to extract the top K entities but not to identify an optimal complete subgraph for each entity, such that we can derive many efficient pruning strategies to have some early pruning based on the upper bound and lower bound of each optimal subgraph.

7 Conclusion and future work

In this paper, we extend the approximate entity extraction problem to the temporal database and propose a novel evaluation method, by which the entity score of each entity can be effectively quantified. The paper presents a first of breed effort to tackle this problem. We develop an efficient algorithm, DTA, to handle the top K entities extraction problem in a temporal database. Using one real dataset and three synthetic datasets, we demonstrate the efficiency of our proposed algorithm and the effectiveness of our proposed evaluation method.

Acknowledgements This work is partially supported by National Natural Science Foundation of China under Grant 60873017 and The 863 High-Tech Program 2009AA01Z149. Lijiang Chen's work is partially supported by the State Scholarship Fund of China Scholarship Council, File No. 2009601191.

References

1. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB, pp. 586–597 (2002)
2. Arasu, A., Ganti, V., Kaushik, R.: Efficient exact set-similarity joins. In: VLDB, pp. 918–929 (2006)
3. Behm, A., Ji, S., Li, C., Lu, J.: Space-constrained gram-based indexing for efficient approximate string search. In: ICDE, pp. 604–615 (2009)
4. Benjelloun, O., Garcia-Molina, H., Su, Q., Widom, J.: Swoosh: a generic approach to entity resolution. Stanford University (2005)
5. Benjelloun, O., Garcia-Molina, H., Kawai, H., Larson, T.E., Menestrina, D., Su, Q., Thavisomboon, S., Widom, J.: Generic entity resolution in the SERF project. J. IEEE Data Eng. Bull. **29**(2), 13–20 (2006)
6. Bergamaschi, S., Gelati, G., Guerra, F., Vincini, M.: An intelligent data integration approach for collaborative project management in virtual enterprises. World Wide Web **9**(1), 35–61 (2006)
7. Bilenko, M., Mooney, R.J.: Learning to combine trained distance metrics for duplicate detection in databases. Technical report, University of Texas, Austin (2002)
8. Brouwer, A.E., Cohen, A.M., Neumaier, A.: Distance-Regular Graphs. Springer, Berlin Heidelberg New York (1989)
9. Chandel, A., Hassanzadeh, O., Koudas, N., Sadoghi, M., Srivastava, D.: Benchmarking declarative approximate selection predicates. In: SIGMOD, pp. 353–364 (2007)
10. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: ICDE, pp. 5 (2006)
11. Chaudhuri, S., Chen, B.-C., Ganti, V., Kaushik, R.: Example-driven design of efficient record matching queries. In: VLDB, pp. 327–338 (2007)
12. Cohen, W.W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. In: SIGMOD, pp. 201–212 (1998)
13. Date, C.J., Darwen, H., Lorentzos, N.: Temporal Data & the Relational Model. Elsevier's Science & Technology (2002)
14. Do, H.-H., Rahm, E.: COMA–a system for flexible combination of schema matching approaches. In: VLDB, pp. 610–621 (2002)
15. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB, pp. 491–500 (2001)
16. Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D.: Text joins in an RDBMS for web data integration. In: WWW, pp. 90–101 (2003)
17. Guha, S., Koudas, N., Marathe, A., Srivastava, D.: Merging the results of approximate match operations. In: VLDB, pp. 636–647 (2004)
18. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)
19. Harary, F.: Graph Theory. Addison-Wesley, Reading (1994)
20. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: data cleansing and the merge/purge problem. J. Data Min. Knowl. Discov. **2**(1), 9–37 (1998)
21. Kappel, G., Kapsammeri, E., Retschitzegger, W.: Integrating XML and relational database systems. World Wide Web **7**(4), 343–384 (2004)
22. Koudas, N., Marathe, A., Srivastava, D.: Flexible string matching against large databases in practice. In: VLDB, pp. 1078–1086 (2004)
23. Li, C., Jin, L., Mehrotra, S.: Supporting efficient record linkage for large data sets using mapping techniques. World Wide Web **9**(4), 557–584 (2006)

24. Li, C., Wang, B., Yang, X.: VGRAM: improving performance of approximate queries on string collections using variable-length grams. In: VLDB, pp. 303–314 (2007)
25. Li, C., Lu, J., Lu, Y.: Efficient merging and filtering algorithms for approximate string searches. In: ICDE, pp. 257–266 (2008)
26. On, B.-W., Koudas, N., Lee, D., Srivastava, D.: Group linkage. In: ICDE, pp. 496–505 (2007)
27. Pak, A.N., Chung, C.-W.: A wikipedia matching approach to contextual advertising. World Wide Web **13**(3), 251–274 (2010)
28. Sarawagi, S., Kirpal, A.: Efficient set joins on similarity predicates. In: SIGMOD, pp. 743–754 (2004)
29. Stonebraker, M.: The design of the postgres storage system. In: VLDB, pp. 289–300 (1987)
30. Tejada, S., Knoblock, C., Minton, S.: Learning domain-independent string transformation weights for high accuracy object identification. In: SIGKDD, pp. 350–359 (2002)
31. Turn, P.: Onan extremal problem in graph theory. Journal of Matematiko Fizicki Lapok (in Hungarian) (1941)
32. Van Rijsbergen, C.J.: Information Retrieval, 2nd edn. Butterworth-Heinemann (1979)
33. Vernicaand, R., Li, C.: Efficient top-k algorithms for fuzzy search in string collections. In: KEYS, pp. 9 (2009)
34. Wang, W., Xiao, C., Lin, X., Zhang, C.: Efficient approximate entity extraction with edit distance constraints. In: SIGMOD, pp. 759–770 (2009)
35. Winkler, W.E.: The state of record linkage and current research problems. US Bureau of the Census (1999)
36. Xiao, C., Wang, W., Lin, X.: Ed-join: an efficient algorithm for similarity joins with edit distance constraints. In: VLDB, pp. 933–944 (2008)
37. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient similarity joins for near duplicate detection. In: WWW, pp. 131–140 (2008)
38. Xiao, C., Wang, W., Lin, X., Shang, H.: Top-k set similarity joins. In: ICDE, pp. 916–927 (2009)
39. Yang, X., Wang, B., Li, C.: Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In: SIGMOD, pp. 353–364 (2008)
40. Yin, X., Han, J., Yu, P.S.: LinkClus: efficient clustering via heterogeneous semantic links. In: VLDB, pp. 427–438 (2006)