Similarity Joins on Item Set Collections Using Zero-Suppressed Binary Decision Diagrams

Yasuyuki Shirai[1][3], Hiroyuki Takashima[1], Koji Tsuruma[2], and Satoshi Oyama[3]

[1] JST-ERATO MINATO Discrete Structure Manipulation System Project,
Hokkaido University, Sapporo, Japan
{shirai,takashima}@erato.ist.hokudai.ac.jp
[2] NEC Corporation, Kanagawa, Japan
k-tsuruma@ak.jp.nec.com
[3] Graduate School of Information Science and Technology,
Hokkaido University, Sapporo, Japan
oyama@ist.hokudai.ac.jp

Abstract. Similarity joins between two collections of item sets have recently been investigated and have attracted significant attention, especially for linguistic applications such as those involving spelling error corrections and data cleaning. In this paper, we propose a new approach to similarity joins for general item set collections, such as purchase history data and research keyword data. The main objective of our research is to efficiently find similar records between two data collections under the constraints of the number of added and deleted items. Efficient matching algorithms are urgently needed in similarity joins because of the combinatorial explosion between two data collections. We developed a matching algorithm based on Zero-suppressed Binary Decision Diagrams (ZDDs) to overcome this difficulty and make matching process more efficient. ZDDs are special types of Binary Decision Diagrams (BDDs), and are suitable for implicitly handling large-scale combinatorial item set data. We present, in this paper, the algorithms for similarity joins between two data collections represented as ZDDs and pruning techniques. We also present the experimental results obtained by comparing their performance with other systems and the results obtained by using real huge data collections to demonstrate their efficiency in actual applications.

Keywords: similarity joins, error-tolerant matching, recommendation, zero-suppressed binary decision diagram

1 Introduction

Similarity joins or error-tolerant matching algorithms between two collections of item sets have recently been investigated and have attracted significant attention [1, 2, 4, 6, 12–14, 17–19] for linguistic applications such as those involving spelling error corrections, data integration, data cleaning, and detection of similar words. The aims in these researches are to find similar records between two huge data sets based on similarity definitions such as cosine measure, Jaccard similarity, and edit distance.

Here, we propose a new efficient method of implementing similarity joins for general item set collections, such as those of purchase history and research keyword data. The main objective of our research is to efficiently find similar records between two data collections under the constraints of the number of added and deleted items. These constraints can be considered to be natural constraints for general unordered item set collections, and they have a broad range of possible applications. For example, consider the following item sets: $D_0 = \{a, b\}, D_1 = \{a, c\}, D_2 = \{a\}$ and $D_3 = \{a, b, c\}$, where all of the edit distances from D_0 to D_1, D_2 and D_3 are one. It is natural, however, to consider that the distance from D_0 to D_1 would be longer than that from D_0 to D_2 and D_0 to D_3 in real world data such as those in purchase histories rather than those in linguistic applications.

In this research, the similarity between records is not defined approximately but "exactly". Efficient matching algorithms are urgently needed in exact similarity joins because of the combinatorial explosion between two data collections. To overcome this difficulty and make matching process more efficient, we developed a matching algorithm based on Zero-suppressed Binary Decision Diagrams (ZDDs) [7, 9, 11]. ZDDs are special types of Binary Decision Diagrams (BDDs) [3] and suitable for implicitly handling large-scale combinatorial item set data. In our previous work [16], we developed an efficient method of set recommendation using ZDD structures. In our set recommendation, a set of items to be added or deleted is recommended to the initial item set so that the modified set is classified to the target class. In this paper, we extend this approach to establish set similarity joins between two collections of item sets.

The applications of our approach can cover a wide area of real applications where we need to find similar records between two huge collections of data sets such as those in :

- Patent searches
  To find similar previous patents by other research institutes, organizations, or countries.
- Data cleaning and duplicate entry detection
  To fix errors in databases compared with other databases, or detect duplicate entries between two databases.
- Similar research
  To find similar research papers or research activities in previous decades, those published by other organizations or countries, or those in different areas.
- Fraud detection
  To find fraudulent data in the database by comparing with previous fraud cases. In fraud detection, rule based method is one of the most realistic approaches. However, in fact, we can hardly define the rules explicitly to detect individual case of fraud. Our approach in this paper enables us to directly find similar cases to those in previous fraud data.

– Customer classification
  To find loyal or potentially-dependable customers, or to find unacceptable customers by comparing with those in previous customer databases.

In this paper, we present the experimental results obtained from comparing the performance of our approach with ordinary matching algorithms and other similar methods, and also the results from other experiments using real huge data collections such as those in DBLP research titles and NSF (National Science Foundation) research abstracts, to demonstrate its efficiency and availability in real applications.

The rest of the paper is organized as follows: Section 2 discusses some works related to our research, especially previous researches on similarity joins. Section 3 provides some definitions and describes the implementation of our framework using ZDD data structures. We present and discuss the results from evaluating the performance of our approach in Section 4, and applications using real data such as those from DBLP and NSF in Section 5. We conclude this paper in Section 6 with a brief summary and some additional comments, and mention future works.

2 Related Work

There have been many works about similarity joins [1, 2, 4, 6, 17–19] or error tolerant matching [12–14], which can be considered to be essential operations in many applications. The objectives in these researches have been to find similar records based on “exact” matching under various constraints such as cosine measure, Jaccard similarity and edit distance.

Chaudhuri [4] and Arasu [1] introduced a general operator called SSJoin, which could be extended to various measures such as edit distance, Jaccard similarity, Hamming distance. The algorithms PartEnum and WtEnum implemented with SSJoin are based on a filtering method, i.e., filtering effective similarity joins based on two ideas for signature generation called partitioning and enumeration.

Bayardo [2] proposed All-Pairs algorithms, on which irrelevant records could be filtered out using inverted lists that were dynamically created on the process according to constraints. They showed that All-Pairs algorithm was highly efficient and scalable to huge size of records.

Xiao [18] introduced a filtering approach focused on the number of “mismatching” and proposed Ed-Join algorithms. The mismatch-based filtering methods could reduce the numbers of candidates.

These approaches described above, were based on filtering-based methods to avoid redundant matching as much as possible. First, by generating the signatures for each data (e.g., string sequence), the candidates of similar pairs could be generated (filtering phase). Then, to find exactly similar pairs with defined similarities, these candidates were verified in a test phase. However, if the database consisted of sets of relatively short strings, signature based filtering methods could be an overhead since these approaches generally generated huge numbers of candidate pairs.

Feng [6] and Wang [17] proposed another approach rather than filtering methods, which was a trie based framework called Trie-Join, in which the data set could be represented as trie structures. Trie structures can share common prefixes for a set of strings. They proposed search procedures on trie structures and pruning methods for sub-structures of the trie.

In our work, we do not consider filtering methods, but focus on efficient representational structures and search algorithms to find similar records in two data collections. Although our approaches and motivation are close to those in Trie-Join [6, 17], Trie-Join assumed sequential patterns as input data and edit distance as constraints, because they considered language processing as a promising area of applications. However, our method takes into considerations general item sets (unordered sets) and the constraints are slightly different from the edit distance. Our system searches similar pairs under the constraints of the number of added and deleted items. We consider that our constraints on item addition and deletion are rather natural and generic for applications of general item set collections or various applications in bioinformatics [15].

In this research, we adopt Zero-suppressed Binary Decision Diagrams (ZDDs) [7, 9, 11] to implement the algorithms efficiently rather than trie structures. ZDDs are not tree structures but directed acyclic graph (DAG) structures that can share the same sub-structure.

3 Set Similarity Joins Using ZDDs

This section provides some definitions and examples of our set similarity joins based on the addition/deletion constraints. We then briefly introduce ZDDs to handle huge numbers of data sets, and present the algorithm for similarity joins on ZDD structures.

3.1 Preliminaries

We will provide various definitions and notations as follows.

Definition 1 (item). *An item is an atomic entity that represents a characteristic or feature and is denoted by a lower-case character* $(a, b, c, \ldots)$*. A set of all items to be considered is denoted by* Σ*.*

Definition 2 (item set and collection of item sets). *An item set is a set of items that represents the characteristics or features of an object (we use* D *to represent an item set). A collection of item sets is a set of item sets, and is represented by* $\mathcal{S}$ *or* $\mathcal{T}$*.*

We sometimes call an item set a "record", and an item set collection a "database" for simplicity.

Definition 3 (add/delete-constraints). *The upper bounds on the numbers of items that can be added or deleted for an item set are denoted by N^+, N^-, respectively.*

If we have item set $D_1 = \{a, b, c, d\}$ and $N^+ = 2$, $N^- = 2$, then item set $D_2 = \{a, e, f, d\}$ is a modification to D_1 under the constraints of N^+ and N^-.

Note that we do not consider a data set as an ordered sequence. Although the edit distance between D_1 and D_2 in the above example is two (if we consider the distance of replacement as one), the edit distance between D_1 and $D_3 = \{a, d, e, f\}$ is four, while D_1 and D_3 also satisfy the condition $N^+ = 2$ and $N^- = 2$ as well as D_1 and D_2.

As an example, suppose we have two collections of item sets, $\mathcal{S}$ and $\mathcal{T}$:

$$\mathcal{S} = \{\{a, b\}, \{a, c\}, \{c\}\} \tag{1}$$

$$\mathcal{T} = \{\{a, d\}, \{a, b, c\}, \{b\}\} \tag{2}$$

The similarity joins of $\mathcal{T}$ for $\mathcal{S}$ under the condition of $N^+ = N^- = 1$ would consist of the following six pairs : $\{\{a, b\}, \{a, d\}\}$, $\{\{a, c\}, \{a, d\}\}$, $\{\{a, b\}, \{a, b, c\}\}$, $\{\{a, c\}, \{a, b, c\}\}$, $\{\{a, b\}, \{b\}\}$, $\{\{c\}, \{b\}\}$.

If the conditions of addition and deletion are $N^+ = 1$ and $N^- = 0$, only the pair of $\{\{a, b\}, \{b\}\}$ satisfies the condition. If $N^+ = 0$ and $N^- = 1$, two pairs $\{\{a, b\}, \{a, b, c\}\}$ and $\{\{a, c\}, \{a, b, c\}\}$ satisfy the condition.

3.2 Zero-Suppressed Binary Decision Diagrams

This subsection reviews ZDDs, which are a variant of Binary Decision Diagrams [3, 7] (BDDs). BDDs are well-known and widely used for efficiently manipulating large-scale Boolean function data. A BDD is a directed graph representation of the Boolean function. The reduction rules in a BDD consist of a "node deletion rule" (delete all redundant nodes with two edges that point to the same node) and a "node sharing rule" (share all equivalent sub-graphs).

Zero-suppressed BDDs (ZDDs) [7, 9, 11] are special types of BDDs that are suitable for implicitly handling large-scale combinatorial item set data. The reduction rules for ZDDs are slightly different from those for BDDs and are outlined in Fig. 1.

- Share equivalent nodes as well as ordinary BDDs ((1) in Fig. 1).
- Delete all nodes whose 1-edge directly points to the 0-terminal node, and jump through to the 0-edge's destination ((2) in Fig. 1).

ZDDs are especially much more effective than BDDs for representing "sparse" combinations such as purchase history data. For instance, sets of combinations selecting 10 out of 1000 items can be represented by ZDDs up to 100 times more compactly than those by using ordinary BDDs.

Fig. 1 has an example of reduced ZDDs for $\mathcal{S}$ in 3.1. Note that the non-existence of a node on each path means that the item is negated in the ZDD representation. For example, $\{c\}$, which is a model of $\mathcal{S}$, can be represented by the heavy line path in ZDD representation. $\mathcal{S}$ can be rewritten in a sum of product representations as $ab + ac + c$ within the ZDD context.

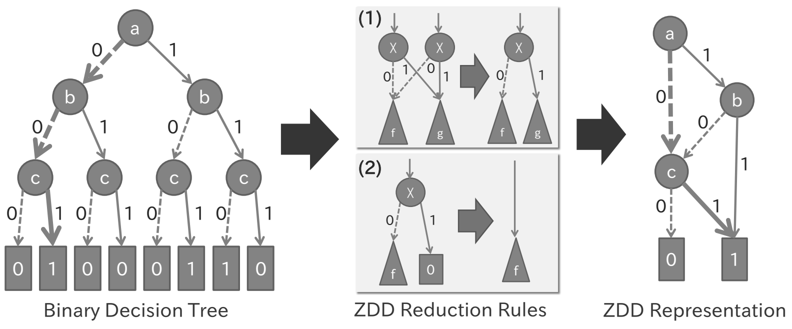


Fig. 1. Reduced Binary Decision Tree with ZDDs

3.3 ZDD based Similarity Joins

In our approach, two collections of item sets in text format are transformed into ZDD structures via the ZDD package [10]. The input data of our system are two ZDD structures and the constraints description. This subsection presents the method for calculating similarity joins for two collections of item sets, i.e., for two ZDD structures.

Algorithm 1 outlines the search algorithm on ZDD structures. Fig. 2 has a simple example of set similarity joins using ZDD based on Algorithm 1, where the objective is to find similar item sets between $\mathcal{S}$ (diagram at left) and $\mathcal{T}$ (diagram at right) under the constraints of $N^+ = N^- = 1$. The search step begins to work from top node a on $\mathcal{S}$ ($search\_zdd(n)$ in the algorithm).

Each square box on the edge indicates the search results, which is a list of a pair: the current count of addition and deletion, and the corresponding path on $\mathcal{T}$. For example, box (1) on $\mathcal{S}$ is attached to the edge which indicates the negation of a. `+0-1:2` in box (1) indicates that the current count is "no additional item (`+0`) and that one item has been deleted (`-1`)" for edge 2 on $\mathcal{T}$. In the same manner, `+0-0:1` in box (1) indicates that the current count is "no additional or deleted items" for edge 1 on $\mathcal{T}$. The *update_candidate* in the algorithm adds new candidates to the current candidates, $n_1.cand$ and $n_0.cand$. The *reduce* function reduces the candidate set and checks the constraints.

Similarly, box (2) is created based on the results for box (1). We need the result for box (3) as well as the result for box (2) to create the box (4). The search process in our algorithm only proceeds if all parents of the node have already finished their processes. After box (3) is calculated, the calculation of box (4) starts based on the results for box (2) and (3).

For box (6) which means the addition of d in $\mathcal{S}$, all the counts in boxes (4) and (5) must add 1 to the addition of items, because no item sets in $\mathcal{T}$ have d as their element. As a result, none of the counts in boxes (4) and (5) satisfy constraints N^+. If no elements in the box satisfy the condition N^+ or N^-, searching along that path is terminated. Hence, box (6) becomes ϕ.

Algorithm 1 Search Algorithm on ZDD structures

```
$n_0$ is a top node of the ZDD $\mathcal{S}$;
$n_0.cand = \{\{+0-0:0\}\}$;
$search\_zdd(n_0)$;

function SEARCH_ZDD($n$)
    if all of other ancestors of node $n$ have not been processed then return
    end if
    if $n$ is a terminal node then return $cand$; // output candidates
    else
        $n_1 = n.edge_1.dest$; // $n_1$ : destination of 1 edge of node $n$
        $n_0 = n.edge_0.dest$; // $n_0$ : destination of 0 edge of node $n$
        $n_1.cand = update\_candidate(n.edge_1, n.cand, n_1.cand)$;
        $n_0.cand = update\_candidate(n.edge_0, n.cand, n_0.cand)$;
        // update candidates for edge 1 and 0
        $n_1.cand = reduce(n_1.cand)$;
        $n_0.cand = reduce(n_0.cand)$;
        // reduction of the candidate set and check the constraints
        if $n_1.cand$ is not NULL then return $search\_zdd(n_1)$;
        end if
        if $n_0.cand$ is not NULL then return $search\_zdd(n_0)$;
        end if
    end if
end function
```

Although there are no solutions under the constraints for the example in Fig. 2, another example in Fig. 3 can obtain two sets of results, where the square box labeled `+0-1:2-4-5-6-7` indicates one item has been deleted from path `2-4-5-6-7` (i.e., $\{a, c, d\}$). We can hence, obtain the final result, $\{a, c, d\} - \{c\} = \{a, d\}$. We can similarly obtain $\{b, c, d, e\}$ (addition of e to $\{b, c, d\}$) from the label of `+1-0:1-3-5-6-7`.

4 Performance Evaluation

4.1 Comparison on Sorted Text Search

We first evaluate the efficiency of our approach based on ZDDs, using artificial data. The problem we provided to evaluate performance in this experiment consists of 170 items in total ($|\Sigma| = 170$), and each record randomly contains five items. We prepare two data sets as $\mathcal{S}$ which contain one million data and 10 million data respectively, and four data sets as $\mathcal{T}$ with sizes from $1000-1000000$. We compare three types of programs for set similarity joins for two given data sets, i.e., "(1) Text-Linear" : between two sorted text data sets, "(2) ZDD-Linear": between ZDD and a sorted text data set, "(3) ZDD-ZDD" : between two ZDDs presented in this paper.

All the systems were implemented in C++, and the experiments were run on a SUSE Linux Enterprise Server 11 with 32 Intel Xeon CPUs (2.66 GHz) and

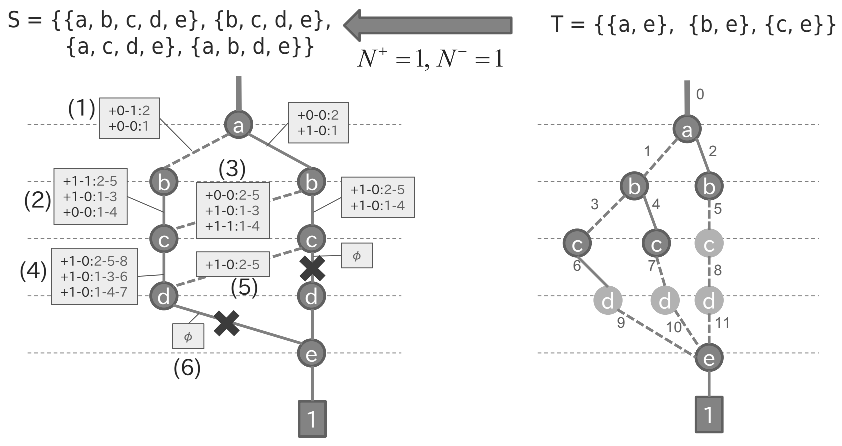


Fig. 2. Example of set-similarity joins using ZDD (1)

1.024 TB RAM. Table 4 compares the execution times. None of the execution times include the time for preparatory data processing, i.e., data sorting or ZDD construction.

As we can see from the table, the execution times for (1) Text-Linear and (2) ZDD-Linear increase linearly for the size of search data, while (3) ZDD-ZDD method can suppress the increase in execution time. We could conclude from these results that our algorithms based on two ZDD structures worked efficiently than linear searches (1) and (2).

4.2 Comparison with Trie-Join

As we previously described, Trie-Join [6, 17] is a similar approach to our system. It, however, only treats totally ordered sequences. Although it is difficult to make precise comparisons with our systems, the experimental results in this subsection provide some indications about their efficiency.

The data sets used in the experiments were as follows :

– Item sets are generated by the alphabet ("a" to "z" in Trie-Join, and "x01" to "x26" in the ZDD-based method) in alphabetical (or numerical) order.
– The length (number of items) of each record is 10. However, if duplicate items occur in the records, we suppress them since the ZDD approach does not distinguish the plural occurrence of items.

Table 1 lists the sample data we used in this experiment.

There are three variations of data sets, each of which consists of 100000, 500000, and 1000000 data records. The results from execution by Trie-Join[4] and

[4] We used the Trie-Join program on :
http://dbgroup.cs.tsinghua.edu.cn/wangjn/codes/triejoin.tar.gz

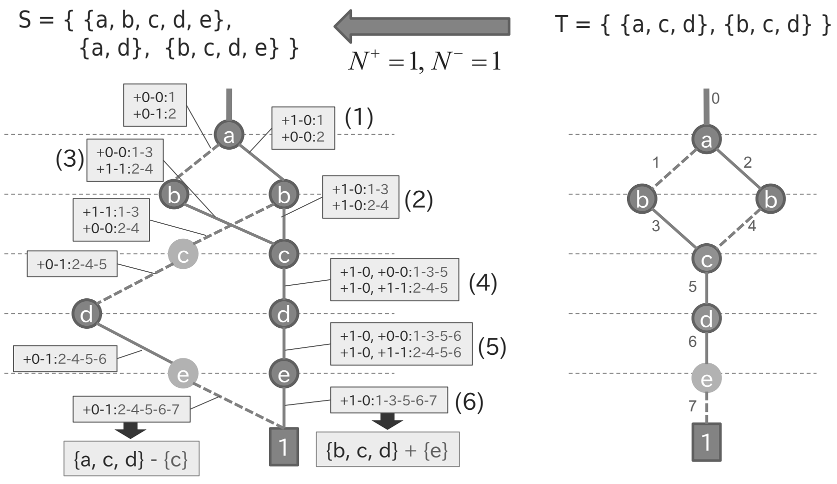


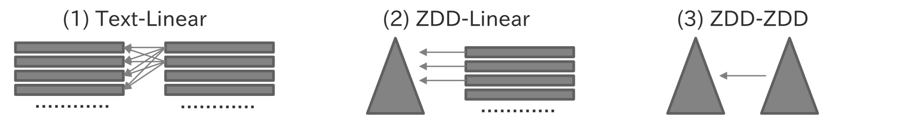
Fig. 3. Example of set-similarity joins using ZDD (2)

Table 1. Sample Data Used in Trie-Join and ZDD-based Method

Trie-Join input	ZDD-based method input
aegklorstw	x01 x05 x07 x11 x12 x15 x18 x19 x20 x23
bcegjmtvxy	x02 x03 x05 x07 x10 x13 x20 x22 x24 x25
filmrsux	x06 x09 x12 x13 x18 x19 x21 x24
dijkmqrt	x04 x09 x10 x11 x13 x17 x18 x20
aeinprst	x01 x05 x09 x14 x16 x18 x19 x20
kqrvwy	x11 x17 x18 x22 x23 x25
aefghlqvx	x01 x05 x06 x07 x08 x12 x17 x22 x24
acgknoruxy	x01 x03 x07 x11 x14 x15 x18 x21 x24 x25

our ZDD-based method are summarized in Table 2. Fig. 5 also compares performance of two systems for some selected results in Table 2 where the horizontal axis plots the number of results and the vertical axis plots the execution times (sec).

As can be seen in the table, we concluded that our system could achieve the same or better performance than Trie-Join for large scale problems. For example, in Table 2, the execution time with Trie-Join is 436.6 (sec) to generate approximately 149 million results for 1000000 records and edit distance = 2. On the other hand, the execution time with the ZDD-based method for 1,000,000 records and $N^+ = N^- = 2$ is 1039.0 (58.5 + 980.5) (sec) to generate approximately 809 million results which is over 5 times as much as the case of edit distance = 2 on Trie-Join (the set of 149 million results by Trie-Join is a proper subset of 809 million results by our ZDD-based method).



Data Set Size	Search Data Set Size	Condition		Execution Time (sec)			Num. of Visit Nodes	
				(1) Text-Linear	(2) ZDD-Linear	(3) ZDD-ZDD	ZDD-Linear	ZDD-ZDD
1,000,000	1000	add=1	delete=1	37.4	3.2	2.3	10,657,783	781,112
1,000,000	10,000	add=1	delete=1	406.6	30.8	6.5	106,268,058	1,363,943
1,000,000	100,000	add=1	delete=1	4055.9	310.2	20.1	1,055,439,664	1,987,588
1,000,000	1,000,000	add=1	delete=1	-	3124.0	98.9	10,537,412,252	2,616,617
10,000,000	1,000	add=1	delete=1	406.7	7.7	7.7	23,353,846	3,175,805
10,000,000	10,000	add=1	delete=1	4061.5	79.6	24.9	236,444,648	7,425,564
10,000,000	100,000	add=1	delete=1	39989.3	793.8	81.6	2,369,599,985	13,450,436
10,000,000	1,000,000	add=1	delete=1	-	7953.7	331.4	23,732,345,100	19,627,528

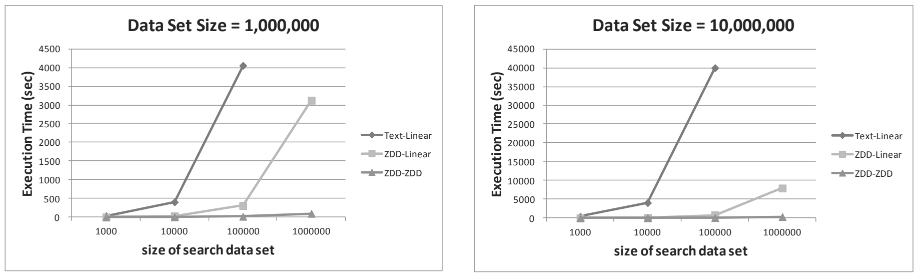


Fig. 4. Performance Evaluation

Since these two systems assumed different constraints, we could not conclude quantitative discussions preciously. Our system, however, could achieve at least potential ability comparable with Trie-Join for the problems of similarity joining with ordered sets.

5 Applications

This section presents some application results with real data sets that concern research topics. We used the data sets from DBLP and NSF data collections.

5.1 DBLP Research Titles

This experiment took into account the paper titles in a DBLP xml data set[5] whose tags include "article" and "inproceedings". The total size of records in this experiment is 863580.

We extracted the "publish year" and the "title" from the data set, and classified them into three collections according to the publish year, i.e., data set 1 (–1997 : 158706 records), data set 2 (1998 – 2007 : 348882 records), and data set 3 (2008– : 355992 records). In this experiment, we excluded words, each of

[5] http://dblp.uni-trier.de/xml

Table 2. Comparison of ZDD-based method and Trie-Join

ZDD-based Method

Size of DB1	Size of DB2	Search Condition	Num. of Results	Exec.Time(sec)		
				ZDD Setting	Search	
		add $\leq$ 1,delete $\leq$ 0	16,624		1.8	(Z1-1)
		add $\leq$ 0,delete $\leq$ 1	16,664		1.6	(Z1-2)
		add $\leq$ 1,delete $\leq$ 1	244,675		4.8	(Z1-3)
		add $\leq$ 2,delete $\leq$ 0	68,530		3.1	(Z1-4)
100,000	100,000	add $\leq$ 0,delete $\leq$ 2	68,775	5.4	2.5	(Z1-5)
		add $\leq$ 2,delete $\leq$ 1	1,293,882		11.1	(Z1-6)
		add $\leq$ 1,delete $\leq$ 2	1,295,137		10.7	(Z1-7)
		add $\leq$ 2,delete $\leq$ 2	8,713,274		37.2	(Z1-8)
		add $\leq$ 1,delete $\leq$ 0	401,823		12.6	(Z2-1)
		add $\leq$ 0,delete $\leq$ 1	405,813		10.2	(Z2-2)
		add $\leq$ 1,delete $\leq$ 1	5,922,847		31.5	(Z2-3)
		add $\leq$ 2,delete $\leq$ 0	1,649,671		19.2	(Z2-4)
500,000	500,000	add $\leq$ 0,delete $\leq$ 2	1,681,117	27.9	16.4	(Z2-5)
		add $\leq$ 2,delete $\leq$ 1	31,163,852		84.4	(Z2-6)
		add $\leq$ 1,delete $\leq$ 2	31,554,584		86.1	(Z2-7)
		add $\leq$ 2,delete $\leq$ 2	210,967,890		368.7	(Z2-8)
		add $\leq$ 1,delete $\leq$ 0	1,532,292		27.6	(Z3-1)
		add $\leq$ 0,delete $\leq$ 1	1,563,436		23.1	(Z3-2)
		add $\leq$ 1,delete $\leq$ 1	22,695,485		67.7	(Z3-3)
		add $\leq$ 2,delete $\leq$ 0	6,312,234		40.0	(Z3-4)
1,000,000	1,000,000	add $\leq$ 0,delete $\leq$ 2	6,524,292	58.5	36.2	(Z3-5)
		add $\leq$ 2,delete $\leq$ 1	119,123,804		210.6	(Z3-6)
		add $\leq$ 1,delete $\leq$ 2	121,636,513		208.0	(Z3-7)
		add $\leq$ 2,delete $\leq$ 2	809,214,292		980.5	(Z3-8)

Trie-Join

Size of DB1	Size of DB2	Search Condition	Num. of Results	Exec.Time(sec)	
100,000	100,000	edit distance=1	78,315	1.1	(T1-1)
		edit distance=2	1,746,849	19.8	(T1-2)
500,000	500,000	edit distance=1	1,827,303	8.2	(T2-1)
		edit distance=2	40,789,678	159.1	(T2-2)
1,000,000	1,000,000	edit distance=1	6,701,562	23.1	(T3-1)
		edit distance=2	149,904,112	436.6	(T3-2)

whose frequency in all databases was less than 10. The size of Σ is 23224 (i.e., 23224 words in total).

The experimental results are summarized in Table 3, where we can see the results for similarity joins between data sets 1 and 2, and between data sets 2 and 3. For example, 6-a (published in 1981) and 6-b (published in 2004) are extracted as similar research papers by different authors. Since both 6-a and 6-b involve the recognition of maximal planar graphs, these two research are intimately related with each other. On the other hand, the main theme in the paper of 1-a is the computational complexity of real sequences, while the theme in 1-b is the descriptive complexity for binary sequences. In this case, there seems to be no deep relationship among these two researches.

In fact, although we can recognize that other characteristics (e.g., research abstract) for research papers should be included if the results are practically in use, similar research activities between decades can be detected by using set similarity joins for item collections.

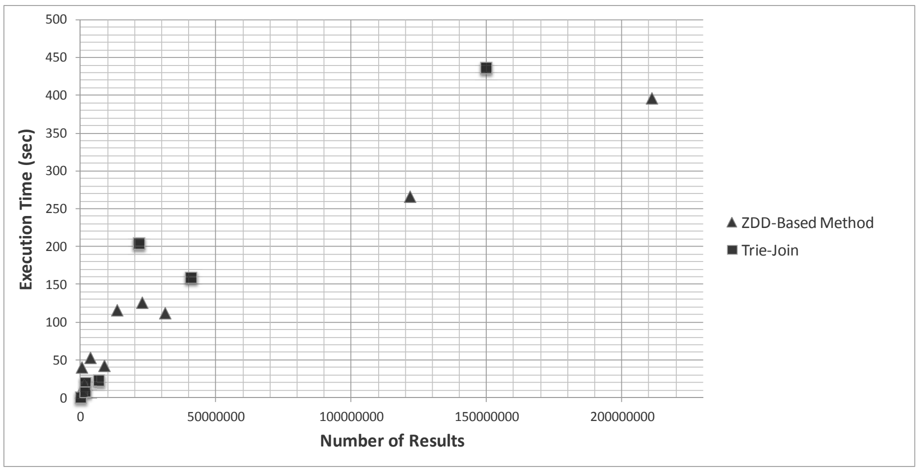


Fig. 5. Performance Comparison with ZDD-based Method and Trie-Join

5.2 NSF Research Abstracts

We applied our system to the NSF data collection as another experiment, which consists of NSF Research Award Abstracts from 1990 to 2003. There are 129000 entries (title, period, budget, abstract, area key, ...) in the database, and the bag-of-word data for the total records. The data set is disclosed in the UCI Machine Learning Repository[6].

We divided the data into "before 1996" and "after 1996" in this experiment, and we focused on entries whose abstracts include the word "computer". We created data sets, each of which consists of words in the title and the abstract. In this experiment, the size of Σ is 3444 (i.e., 3444 words in total).

Part of the experimental results are listed in Table 4, where $N^+ = 3$ and $N^- = 3$, which shows that relevant research activities are detected as well as those from the DBLP experiments.

For example, 1-a (1992) is a project for studying basic techniques for parallel algorithms, while 1-b (1999) is a workshop where more practical applications of parallel computing such as fluid dynamics are discussed. In this experiment, we can conclude that similar research projects can be detected as well as the case of DBLP application.

6 Conclusion

In this paper, we described a new approach to similarity joins for general item set collections under the constraints of the number of added and deleted items.

[6] NSF Research Award Abstracts in UCI Machine Learning Repository http://archive.ics.uci.edu/ml/datasets/NSF+Research+Award+Abstracts+1990-2003

Table 3. Experimental Results for DBLP database

Experiments for articles on DBLP (1) (– 1997 : 1998 – 2007)

Num.	Year	Title
1-a	1987	On The Complexity of Computable Real Sequences.
1-b	2001	Descriptive complexity of computable sequences
2-a	1982	Approximations for the waiting time distribution of the M/G/c queue.
2-b	2004	Mean Waiting Time Approximations in the G/G/1 Queue.
3-a	1979	On the connectivity of cayley graphs.
3-b	2005	Parameters of connectivity in (a, b)-linear graphs.
4-a	1993	An Affinity-Based Dynamic Load Balancing Protocol for Distributed Transaction Processing Systems.
4-b	2006	Dynamic Load Balancing Protocol for Locally Distributed Systems.
5-a	1989	Efficient monotone circuits for threshold functions.
5-b	2006	Monotone circuits for monotone weighted threshold functions.
6-a	1981	The reconstruction of maximal planar graphs. I. Recognition.
6-b	2004	A simple recognition of maximal planar graphs.

Experiments for articles on DBLP (2) (1998 – 2007 : 2008 –)

Num.	Year	Title
1-a	2005	k-Center problems with minimum coverage.
1-b	2008	Asymmetric k-center with minimum coverage.
2-a	2006	On complexity of multistage stochastic programs.
2-b	2008	On Stability of Multistage Stochastic Programs.
3-a	2006	A remote laboratory for electrical engineering education.
3-b	2011	Developing a remote laboratory for engineering education.
4-a	2006	Arboricity and tree-packing in locally finite graphs.
4-b	2008	Locally finite graphs and embeddings.
5-a	2005	Transforming semantics by abstract interpretation.
5-b	2009	Abstract interpretation of resolution-based semantics.
6-a	2006	PolicyUpdater: a system for dynamic access control.
6-b	2008	A privacy-aware access control system.

We introduced a matching algorithm based on Zero-suppressed Binary Decision Diagrams (ZDDs), which are special types of Binary Decision Diagrams (BDDs). ZDDs can represent huge databases efficiently, especially for sparse data collections. We developed efficient algorithms and pruning techniques for two ZDD structures.

We presented some experimental results from evaluating performance with other methods including Trie-Join, which is well known as an efficient implementation for similarity joins, although it is based on slightly different problem setting. As a result, our approach could achieve comparable results with Trie-Join for the problems we presented in this paper.

We also showed experimental results with actual huge data collections such as those from DBLP research titles and NSF research abstracts to demonstrate the availability in real applications.

Future works include extending our results in several directions such as :

– We intend to investigate various pruning techniques or filtering techniques exploited in other systems [1, 2, 6, 17] to adopt them in our system. Some

Table 4. Experimental Results for NSF database

Num.	Year	Title and Abstract
1-a	1992	Design of Parallel Algorithms – support for postdoctoral associate, – experimental computer science – developed for idealized parallel computers on real parallel computers, – load balancing techniques
1-b	1999	WORKSHOP: Parallel CFD'99 International Conference – computational fluid dynamics research on parallel computers, – parallel software system, – case studies from fluid dynamics, – early experiences on teraflops-class computers
2-a	1993	Constructing, Maintaining, and Searching Geometric Structures – construction and maintenance of geometric structures, – efficiently searching in such structures, – computational geometry unsolved problems, – dynamic maintenance of geometric structures, – computer graphics and computer vision, – sequential and parallel computation models
2-b	1999	Towards Simpler Algorithms in Computational Geometry – design and analysis of algorithms for large amounts of geometric data, – efficient algorithms for fundamental problems in computational geometry, – computer graphics and computer vision, – geometric optimization, – construction of basic geometric structures, – randomization, approximation, and techniques for correcting pessimistic worst-case analyses
3-a	1991	Undergraduate Computer Integrated Design Laboratory – establishment of an undergraduate computer integrated design laboratory, – development of a unified education program, – computer graphic simulation that gives insight into complex phenomena
3-b	1998	The Development of a Communication Networks Laboratory at Queens College – computer communication networks laboratory, – design and implementation of the Token Ring and Ethernet Local Area Networks, – undergraduate laboratory and an associated laboratory manual

pruning techniques [6, 17], such as length pruning and single branch pruning would also be especially helpful in our system.

- We need to classify items to various classes such as editable, essential, and requisite items, and we need to define transformation costs to replace these items. These extensions would make the results quite effective for real use.
- We intend to investigate more enhanced algorithms using ZDDs such as valuable ordering in ZDD construction or pruning techniques. In fact, it is well known that the valuable ordering in ZDDs is sensitive to performance in some cases.
- Sequence BDD [5, 8] offers considerable promise as a basic computation framework instead of ZDD in dealing with string similarity problems. Sequence BDD shares the same common sub-sequence as DAG structures, and can provide compact representations for manipulating sets of strings. We

are now investigating the algorithms for sequential similarity joins using the sequence BDD framework.

References

1. A. Arasu, V. Ganti, R. Kaushik : Efficient Exact Set-Similarity Joins, In Proc. of 32nd International Conference on Very Large Data Bases (VLDB 2006), 2006
2. R. J. Bayardo, Y. Ma, R. Srikant : Scaling Up All Pairs Similarity Search, In Proc. of 16th international conference on World Wide Web, 2007
3. R. E. Bryant : Graph-based algorithms for Boolean function manipulation, IEEE Transactions on Computers, Vol. 35 Issue 8, 1986
4. S. Chaudhuri, V. Ganti, R. Kaushik : A Primitive Operator for Similarity Joins in Data Cleaning, In Proc. of 22nd International Conference on Data Engineering (ICDE '06), 2006
5. S. Denzumi, R. Yoshinaka, S. Minato, H. Arimura : Efficient Algorithms on Sequence Binary Decision Diagrams for Manipulating Sets of Strings, Hokkaido University, TCS Technical Reports, TCS-TR-A-11-53, 2011
6. J. Feng, J. Wang, G. Li : Trie-join: a trie-based method for efficient string similarity joins, The VLDB Journal 21:437.461, 2012
7. D. E. Knuth : The Art of Computer Programming, Vol. 4, No.1, Bitwise Tricks & Techniques, pp.117-126, Addison-Wesley, 2009
8. E. Loekito, J. Bailey, J. Pei : A Binary decision diagram based approach for mining frequent subsequences, Knowledge and Information Systems, 24(2), 2010
9. S. Minato : Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, In Proc. of 30th ACM/IEEE Design Automation Conference (DAC'93), 1993.
10. S. Minato : VSOP (Valued-Sum-of-Products) Calculator for Knowledge Processing Based on Zero-Suppressed BDDs, Federation over the Web, LNAI 3847, 2006.
11. S. Minato : Implicit Manipulation of Polynomials Using Zero-Suppressed BDDs, In Proc. of IEEE The European Design and Test Conference, 1995.
12. M. Neuhaus, H. Bunke : An Error-tolerant Approximate Matching Algorithm for Attributed Planar Graphs and its Application to Fingerprint Classification, In Proc. of Joint IAPR International Workshops, SSPR 2004 and SPR 2004, 2004
13. K. Oflazer : Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction, Computational Linguistics, 22(1), 1996
14. K. Oflazer : Error-tolerant Tree Matching, In Proc. of 16th conference on Computational Linguistics (COLING '96), 1996
15. K. Shimizu, K. Tsuda : SlideSort: All Pairs Similarity Search for Short Reads, Bioinformatics, 27(4), 2011.
16. Y. Shirai, K. Tsuruma, Y. Sakurai, S. Oyama, S. Minato : Incremental Set Recommendation Based on Class Differences, In Proc. of 16th Pacific-Asia Conference on Knowledge Discovery and Data Mining, LNAI 7301, 2012
17. J. Wang, J. Feng, G. Li : Trie-Join: Efficient Trie-based String Similarity Joins with EditDistance Constraints, In Proc. of the VLDB Endowment, 3(1-2), 2010
18. C. Xiao, W. Wang, X. Lin : Ed-join: an efficient algorithm for similarity joins with edit distance constraints, In Proc. of VLDB Endowment, 1(1), 2008
19. C. Xiao, W. Wang, X. Lin, J. X. Yu : Efficient Similarity Joins for Near Duplicate Detection, In Proc. of 17th International Conference on World Wide Web, 2008