Record Linkage as DNA Sequence Alignment Problem

Yoojin Hong
Penn State Univ., USA
yuh108@psu.edu

Tao Yang
Penn State Univ., USA
tuy104@psu.edu

Jaewoo Kang*
Korea Univ., Korea
kangj@korea.ac.kr

Dongwon Lee†
Penn State Univ., USA
dongwon@psu.edu

ABSTRACT

Since modern database applications increasingly need to deal with dirty data due to a variety of reasons (e.g., data entry errors, heterogeneous formats, and ambiguous terms), considerable recent efforts have focused on the **(record) linkage** problem to determine if two entities represented as relational records are approximately the same or not. In this paper, we propose a novel idea of using the popular gene sequence alignment algorithm in Biology – BLAST. Our proposal, termed as the BLASTed linkage, is based on the observations that: (1) both problems are variants of approximate pattern matching, (2) BLAST provides the statistical guarantee of search results in a scalable manner – a greatly lacking feature in many linkage solutions, and (3) by transforming the record linkage problem into the gene sequence alignment problem, one can leverage on a wealth of advanced algorithms, implementations, and tools that have been actively developed for BLAST during the last decade. In translating English alphabets to DNA sequences of `A`, `C`, `G`, and `T`, we study four variations: (1) *default* – each alphabet is mapped to nucleotides under 1, 2, and 4-bit coding schemes, (2) *weighted* – tokens are elongated or shortened proportional to their importance, making important tokens longer in the resultant DNA sequences, (3) *hybrid* – each token's lexical meaning as well as its importance are considered at the same time during translation, and (4) *multi-bit* – tokens are selected for any of 1, 2, and 4-bit coding schemes based on the cumulative distribution functions of their importance. The efficacy of our proposed BLASTed linkage schemes are experimentally validated using both real and synthetic data sets.

1. INTRODUCTION

Poor quality data is prevalent in databases due to a variety of reasons, including transcription errors, data entry mistakes, lack of standards for recording database fields, etc. To fix such errors, considerable recent work has focused on the **(record) linkage problem**[1], i.e., determine whether or not two entities represented as relational records are approximately the same. The linkage problem frequently occurs in data applications (e.g., digital libraries, search engines, customer relationship management) and gets exacerbated when data are integrated from heterogeneous sources. For instance, a customer address table in a data warehouse may contain multiple address records that are all from the same residence, and thus need to be consolidated. For another example, imagine integrating two digital libraries, say DBLP and CiteSeer. Since citations in two systems tend to have different formats, identifying all matching pairs is not always straightforward.

The linkage problem has been extensively studied in various disciplines. By and large, contemporary approaches have mostly focused on how to identify similar records "accurately" by designing new methods or models in a particular situation. However, often, newly-developed linkage solutions are hard to apply to data sets of different scenarios, let alone using additional tools to visualize or analyze the results further. One way to approach the problem is to develop a suite of linkage algorithms and tools for generic usage so that the developed linkage solutions can be used in a variety of situations. Another way is to extend an existing generic solution to solve the linkage problem so that one can leverage on the development of the generic solution and its user base. In this paper, we take the latter approach and aim at solving the linkage problem by applying one of such popular existing solutions drawn from Bioinformatics, called BLAST (*B*asic *L*ocal *A*lignment *S*earch *T*ool) [1].

The BLAST, developed in 1990, is one of the most popular (and best cited) algorithms for aligning biological sequence information such as nucleotides of DNA sequences and amino acid sequences of proteins. In its essence, given a query sequence q, BLAST finds top-k most similar sequences above a threshold from a database of sequences D using approximation[2] heuristics based on Smith-Waterman algorithm [20]. Our decision to use BLAST for the linkage problem is based on the observations that: (1) Both problems are variants of approximate pattern matching. That is, comparing two address records of "John Doe, 110 E. Foster

*Partially supported by Korea University grant.
†Partially supported by IBM and Microsoft gifts.

VLDB '08, August 24-30, 2008, Auckland, New Zealand

[1]The record linkage problem is also known as other names such as *merge-purge* [9], *de-duplication* [18], or *entity resolution* [19] problem. Throughout the paper, to be consistent, we will use the *linkage problem* to refer to all these problems.
[2]It is reported that BLAST is able to produce results that are slightly less accurate than what Smith-Waterman algorithm [20] would generate but over 50 times faster [1].

Notation	Description
q_r, q_s	a query record and sequence
d_r, d_s	a duplicate record and sequence
D_r, D_s	a database of records and sequences
t_1, t_2 ..	tokens
tf	term frequency
tf/idf	term freq./inverse document freq.

Table 1: Notations used.

Ave. State College, PA, USA" and "Dr. J. E. Doe, 110 East Foster Franklin Avenue, University Park, PA 16802" in the linkage problem is closely related to finding an alignment between the sequences `ATATCG` and `ATTAGC` using BLAST; (2) The alignment results from BLAST provide a robust similarity measure of *S-score* as well as a sound reliability measure of *E-value*[3]. Such a statistical guarantee has been greatly lacking in existing linkage solutions. Furthermore, BLAST is known for its fast running time and ability to handle a large amount of sequence data; and (3) BLAST has a wealth of advanced algorithms (e.g., nucleotide-nucleotide, protein-protein, and position-specific version), implementations (e.g., NCBI BLAST, FPGA-based BioBoost BLAST, and open source version), and tools (e.g., KoriBLAST for visualization and Parallel BLAST for parallel processing [17]) to leverage on. Therefore, if one can successfully transform the linkage problem to the gene sequence alignment problem, one can have an immediate access to a vast number of application software to use.

In the investigation of BLAST for the linkage problem, in particular, we aim to have the following *desiderata* to maximize the impact of study: (1) The kernel of BLAST algorithm and implementation should not be changed to make existing tools remained as useful. Instead, our proposal sits atop BLAST algorithm and manipulates query and database sequences; and (2) The proposed scheme should be able to identify matching records fast as well as accurately. That is, both accuracy and speed of the solution should be the evaluation metrics.

Our contributions are as follows:

- To the best of our knowledge, this is the first attempt to solve the linkage problem using the BLAST technology. We formally introduce the linkage problem and propose a framework to show how to apply BLAST algorithm.
- We propose and evaluate four different BLAST based linkage solutions to translate string data into DNA sequences while considering the relative importance of tokens in the string data.
- We compare the proposed BLAST based schemes against popular approximate pattern matching schemes such as *Jaro*, *Jaccard*, and *SoftTFIDF* and report promising results of our proposal in both speed and accuracy.

[3]E-value is the probability indicating that the S score of the current sequence could be due to by chance using the given database. For instance, an E-value less than e^{-5} of an alignment means that the alignment is very unlikely to occur by chance – thus a good match.

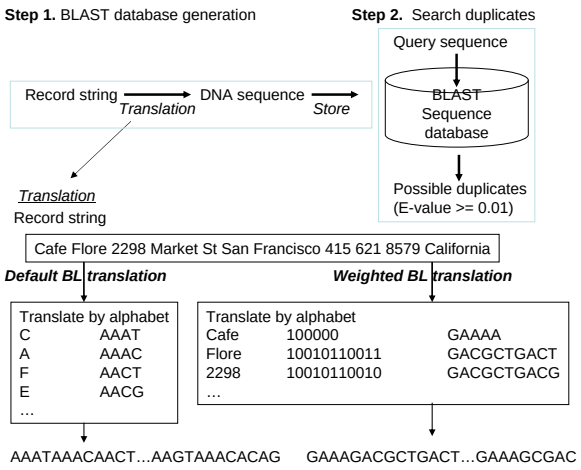


Figure 1: Overview of the BLASTed Linkage.

2. OVERVIEW OF PROBLEM AND SOLUTION

Throughout the paper, we use notations in Table 1.

Problem Overview. The linkage problem can be modeled as a *selection* problem (i.e., select top-k similar records) as well as a *threshold* problem (i.e., find all similar records above a threshold). For illustration purpose, we briefly describe the selection version of the linkage problem as follows. Imagine that given a query record q_r, there are k true duplicates, d_1, ..., d_k, in the database of records D_r. Then, the goal of record linkage is to obtain the true duplicates from the retrieved top-k records. In the best scenario, if all k true duplicates are retrieved, then we get the precision and recall of 1.0. Note that in this paper, we adopt both the selection approach and the threshold approach for further experimental analysis.

Given a set of query records Q_r and a database of records D_r, a naive nested-loop style algorithm with quadratic running time to find all duplicate records is:

for each record $r \in Q_r$
  for each record $s_i \in D_r$
    compute $dist(r, s_i)$;
  return $\{s_1, ..., s_k\}$ with the lowest $dist(r, s_i)$;

Note that what we study in this paper is *not* the algorithmic improvement of the linkage problem *nor* the study of $dist()$ function therein. Therefore, such an optimization or modeling issue is not further discussed in the paper.

Solution Overview. The basic framework of our approach, termed as the "BLASTed linkage" or "BL" in short, is illustrated in Figure 1. All strings in database D are first translated to DNA sequences by rewriting each English alphabet character to pre-assigned one, two, or four nucleotides under 1, 2, or 4-bit coding scheme, respectively. For instance, the alphabet "A" is re-written to nucleotides of `T`, `GT`, or `AAAC` under 1, 2, or 4-bit coding scheme, respectively. Then, the query sequence q is also translated in a similar manner. Finally, BLAST is invoked to find similar sequences to q from D. In addition to this default BLASTed linkage technique, we also propose three other variations depending on how we "translate" strings to DNA sequences –

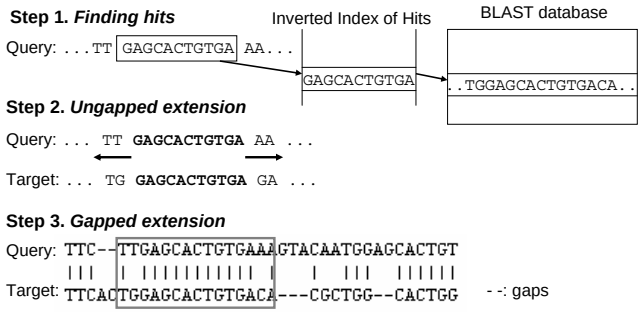


Figure 2: Three steps of BLAST.

Letter vs. Nucleotides (4-bit)							
A	AAAC	B	AAAG	C	AAAT	D	AACC
E	AACG	F	AACT	G	AAGC	H	AAGG
I	AAGT	J	AATC	K	AATG	L	AATT
M	ACAC	N	ACAG	O	ACAT	P	...

Letter vs. Nucleotides (1-bit/2-bit)							
A	T/GT	B	A/AG	C	C/CA	D	T/GC
E	A/AA	F	C/CC	G	T/GG	H	G/CT
I	G/GA	J	C/CG	K	A/AG	L	C/AT
M	T/GT	N	A/AA	O	T/GG	P	...

Table 2: Conversion tables. (Partial)

weighted, *hybrid*, and *multi-bit* BLASTed linkage.

- In the *default BLASTed linkage* (in Section 3.1), each alphabet character in record strings is translated to pre-assigned number of nucleotides.
- In the *weighted BLASTed linkage* (in Section 3.2), each token in record strings is either elongated or shortened, proportional to their associated weights, to make important tokens longer in DNA sequences.
- In the *hybrid BLASTed linkage* (in Section 3.3), each token's lexical meaning is incorporated into the translation in addition to the importance such as their weights or ranks.
- In the *multi-bit BLASTed linkage* (in Section 3.4), each token is assigned to any of 1, 2 or 4-bit coding schemes based on the cumulative distribution functions of their importance.

3. MAIN PROPOSAL: BLASTed Linkage

The original BLAST algorithm performs three basic steps to align similar sequences. These three steps are illustrated in the format of DNA sequences in Figure 2. Given a query sequence q and a database of sequences D, consider the candidate sequence c ($c \in D$). Then,

1. ***Exact Match***: BLAST first searches all sequences from D that share "hits" with q, where hits are the exactly matching continuous letters of nucleotides between c and q. By default, in BLAST, the size of a hit for DNA sequence is 11 nucleotides.

2. ***Ungapped Extension***: In the second step, if c has at least two hits in close distance, BLAST extends the two hits without gaps. During the extension, BLAST increases the score for matching pairs and decreases it for mismatching pairs. If the score drops below a certain threshold, then the process stops and the final result becomes the High-scoring Segment Pairs (HSP). The box in step 3 of Figure 2 represents HSP from the ungapped extension step.

3. ***Gapped Extension***: Last, BLAST extends the HSP (with matching score above the threshold) with gaps as long as the final score does not drop below a threshold, which is larger than the threshold of ungapped extension.

At each step, BLAST can eliminate some candidate sequences with insufficient matching scores below the thresholds to reduce the cost of expensive dynamic programming. That is, the thresholds for ungapped and gapped extension help BLAST avoid comparing the entire sequence.

The key contribution of the BLASTed linkage is to enable BLAST to do the "linkage" of query and database records. To enable this, one needs to translate query and database records into DNA sequences of `A`, `C`, `G`, and `T`. For this translation, we study four variations below.

3.1 Default BLASTed Linkage

In the default BLASTed linkage, each alphabet letter in record strings is translated to pre-assigned number of one, two, or four nucleotides depending on 1, 2, or 4-bit coding schemes, respectively. In this paper, we used three different coding schemes as follows:

4-bit scheme: An alphabet letter is translated to four nucleotides, as shown in Table 2 (top). Note that the assignment of four nucleotides to an alphabet is done such that the assigned nucleotides become "prefix-free." A prefix-free code such as Huffman code is a code where no code word is a prefix of any other codes. For instance, a code set {0, 1, 01} is not prefix-free since "0" is the prefix of "01", but a code set {000, 010, 111} is. We use 4-bit coding of Table 2 (top) borrowed from [13]. However, note that any other prefix-free assignment would work equally. Using 4-bit coding, note also that after the translation, DNA sequences are always four times longer than original English strings. In the translation, ignoring upper/lower cases, we consider 37 (= 26 + 10 + 1) cases – i.e., 26 cases for 26 English alphabets, 10 cases for 10 numbers (e.g., 0 to 9), and one case for all special characters (e.g., `@`, `#`, `$`). we used the 4-bit scheme, by default.

1-bit/2-bit schemes: Our eventual goal in this translation is *not* to have good representation of nucleotides, rather to translate two *similar* (*dissimilar*) records to two *similar* (*dissimilar*) DNA sequences so that both can be aligned correctly by BLAST. That is, whether "*F*atabase" is translated to `AACTAAACACGGAAACAAAGAAACACGCAACG` in 4-bit coding or `CTTTATGA` in 1-bit coding is not important as long as both DNA sequences can be aligned well with DNA sequences for "*D*atabase" if we use BLAST to distinguish them. Based on this observation, we also propose to use one or two nucleotide(s) to represent each alphabet letter. For example, we can group 37 cases into 4 groups, and assign one of `A`, `C`, `G`, and `T` to each group. Similarly, we can also group 37 cases into 16 groups to use two nucleotides per group. This coding scheme is shown in Table 2 (bottom).

The actual assignment of 1-bit or 2-bit nucleotides letters to each group is done randomly since errors were artificially introduced in the experimentation. However, one can exploit further heuristic features in the assignment. For in-

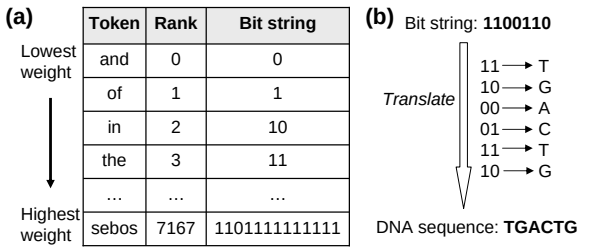


Figure 3: The sliding window coding scheme in the weighted BLASTed linkage: (a) tokens are sorted by their weights, (b) bit string is translated to DNA sequence by the sliding window of size 2.

stance, alphabet letters in the neighborhood on keyboards can be assigned to the same group. Finally, note that compared to the 4-bit coding scheme which is a lossless scheme, 1-bit and 2-bit schemes are lossy – i.e., we are unable to translate translated strings back to the original ones. Despite their lossy translation, we will show that 1-bit and 2-bit coding schemes can achieve acceptable accuracy with noticeable speed-up.

3.2 Weighted BLASTed Linkage

By the coding schemes of default BLASTed linkage, longer English letters will always have longer DNA sequences, regardless of bit schemes. For instance, when an article title "Improving RNA ..." is translated, the word "Improving" will have 36 nucleotides `AAGTACACACCCACCTACATACTCAAGTACAGAAGC` while the second word "RNA" has 12 nucleotides `ACCTACAGAAAC` under 4-bit scheme. Therefore, if there is another title such as "Improving Stem Cell ...", then BLAST is likely to find this title as a similar match since 36 nucleotides between two sequences (corresponding to "Improving") are identical. However, in both titles, the other words such as "RNA" or "Stem Cell" carry more important meaning and thus it is most likely users' intention to find articles with titles having either "RNA" or "Stem Cell", not "Improving."

To capture this intuition, in the weighted BLASTed linkage, each word token t of a record string s from a database of sequences D is associated with an importance weight w using the *tf-idf* weight (term frequency-inverse document frequency) of the traditional weighting scheme in Information Retrieval. This tf-idf weight is a statistical measure to estimate how important a word token t in the string s is within a database D such that it increases proportionally to the number of times t occurs in s but is offset by its frequency in D. For instance, for a record string "Improving RNA ...", the token "Improving" would have a lower tf-idf weight than what the token "RNA" would have, assuming that the token "Improving" occurs frequently among entire corpus of D.

Given tf-idf weight, each token can be converted to corresponding bits using the *sliding window coding*, as illustrated in Figure 3. The details are as follows:

1. Word tokens are sorted in ascending order according to their tf-idf weights – from the *least* important to the *most* important token. For a token t with the rank r (starting from 0) in this order, then, a binary string corresponding for the rank r is used as the encoded string for the token t. For instance, the third lowest rank word "in" in Figure 3(a) has the rank of #2 (since the rank starts from 0), and is thus converted to its corresponding binary representation of `11` (i.e., `11` = "third lowest"). In addition to this simple ranking-based encoding method, we also tried more sophisticated method using equi-distance histograms. For instance, using 1,000 equi-distance bins (i.e., bin #1 for weights in [0.000, 0.001), bin #2 for weights in [0.001, 0.002), etc.), all tokens are assigned to appropriate bins with similar weights (thus similar importance), and subsequently all tokens in the same bin are encoded into the same binary string (i.e., `0` for bin #0, `1` for bin #1, and `10` for bin #2, etc.)[4].

2. Once a binary bit string representation for the token t is ready, a sliding window w slides down the bit string, one bit each time, and encodes bits within the window to corresponding nucleotides. When $|w|$=2, the sliding window can have four kinds of bit strings, {`00`, `01`, `10`, `11`} (ignoring `0` and `1`), which can be encoded to `A`, `C`, `G`, and `T`, respectively. As shown in Figure 3(b), based on the binary bit string `1100110` for a particular token, six 2-bits of {`11`, `10`, `00`, `01`, `11`, `10`} are generated in the sliding window, and six letter nucleotides `TGACTG` become the final DNA sequence.

Note that since more important tokens with higher tf-idf weights are encoded to longer bits, their final DNA sequences will be much longer as well. Therefore, when BLAST aligns sequences, such elongated DNA sequences will play a more significant role than they would in the default BLASTed linkage.

One potential caveat of the rank based weighted BLASTed linkage is that each unique token in the corpus D will have unique DNA sequences assigned at the end. For instance, when there are two tokens t_1 ("Database") and t_2 ("Fatabase") in D, since they may have different tf-idf weights, their corresponding ranks may be extremely different. In turn, this will yield two completely different final DNA sequences although the two tokens appear to be similar. To mitigate this problem, we can first group tokens which must be considered as the same or similar word and assign the same DNA sequence to each token group. In the experimentation, we applied the popular Jaro-Winkler method to find all obvious typos first.

3.3 Hybrid BLASTed Linkage

Compared to those in the default BLASTed linkage, the final DNA sequences of tokens in the weighted BLASTed linkage are completely determined by their importance weights. To some extent, we become lost the lexical information of tokens. In other words, there might exist such situation that two tokens are completely different but they happen to carry equal or close importance weights in the database. For instance, let us consider a citation record such as "...proceedings of VLDB ..." and the other citation record such as "...proceedings of SIGMOD ...". The words "VLDB" and the "SIGMOD " may have close importance weights in the database. Suppose the binary bit string representations for "VLDB" and "SIGMOD" are `11000` and `11001`, respectively. When they are translated using the *sliding window coding*, the final DNA sequences for "VLDB" and "SIGMOD" become `TGAA` and `TGAC`, respectively. Obviously, this may result in a similar match between these two tokens although they are completely different.

[4]In the experimentation, the histogram based scheme did not show significant improvement over the simple rank based method, and thus is not discussed any further.

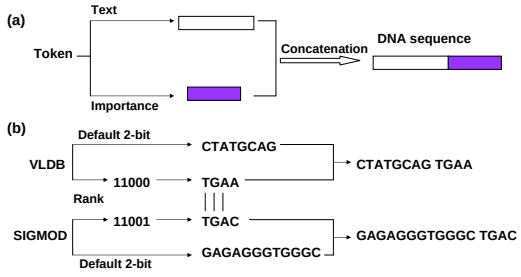


Figure 4: The coding scheme in the hybrid BLASTed linkage: (a) importance weights are appended to the lexical information of tokens, (b) comparison of token "VLDB" and "SIGMOD" using 2-bit as default coding and ranks as importance weights in the hybrid BLASTed linkage.

Motivated by this intuition, in the hybrid BLASTed linkage, we propose to incorporate the lexical information into the translation in addition to the importance weights of tokens. For each token, we append the gene sequence generated from the importance weight to the end of the gene sequence generated from the default coding scheme, as illustrated in Figure 4(a). For the above example, using this hybrid coding scheme with 2-bit as default coding and ranks as importance weights, the final DNA sequences for "VLDB" and "SIGMOD" become `CTATGCAGTGAA` and `GAGAGGGTGGGCTGAC`, respectively. (Figure 4(b)). Accordingly, the probability that BLAST finds these two tokens as a similar match is greatly reduced.

The proposed hybrid BLASTed linkage also have some advantage over the default BLASTed linkage. According to the default coding, a longer token has a longer DNA sequence based on the lexical formation. If it is more important in the corpus, then its DNA sequence generated from the importance weight is also longer. Therefore in the hybrid BLASTed linkage, by combining the two DNA subsequences, it would have even longer final DNA sequence for BLAST to distinguish. However, if it is not important in the corpus, then its DNA sequence based on the importance weight is much shorter, which in turn offsets the length of the DNA sequence generated from the default coding. Since most English words have limited length but the rank of a word in a very large database could be significantly high, the hybrid BLASTed linkage can improve the precision of pattern matching in BLAST without loss of lexical meaning of English words.

In addition, the hybrid BLASTed linkage can mitigate the synonym or typo problem which is mentioned at the end of Section 3.2. Although the two tokens "*D*atabase" and "*F*atabase" have quite different tf-idf weights and hence different gene sequences based on their ranks, their lexical formations are quite similar. Therefore, the gene sequences generated from the default coding can offset the difference between their ranks. Overall, the final DNA sequences can still be similar to reduce the chance for BLAST to find the mis-match.

3.4 Multi-bit BLASTed Linkage

In the weighted BLASTed linkage, there exists such situation that two tokens have different tf-idf weights but they have close ranks which in turn result in similar final DNA sequences after translation. The hybrid BLASTed linkage, to some extent, can mitigate this problem. However, the combination of two component gene sequences may quickly increase the length of the final DNA sequence for each token. Thus, we propose a multi-bit BLASTed linkage. The main idea is to divide all the tokens into three groups based on the actual values of tf-idf weights and then assign 1, 2 and 4-bit coding schemes to the three groups respectively to ensure that more important tokens can be translated to longer DNA sequences.

In this scheme, we regard different tokens in the corpus as possible values of a discrete random variable X, i.e., X can take on possible values $\{x_1, x_2,..., x_n\}$ where n is total # of tokens in the database. The weight of each token is regarded as the probability of X at that particular value. Note that if the weights are not normalized, i.e., the sum of the weights are not equal to 1, we need to normalize these weights first (e.g. by dividing each weight by the total sum of weights). Hence, the normalized weights construct the probability mass function(*pmf*) at x_i for the discrete random variable X. That is, the *pmf* at x_i represents $Prob(X = x_i)$. Without loss of generalization, we can re-arrange x_1 through x_n based on the ascending order of *pmf*. Then we can obtain the cumulative distribution function(*cdf*) for X. That is, the *cdf* at x_i is the sum of *pmf* at x_1 through x_i. Therefore, the *cdf* at x_i represents $Prob(X \leq x_i)$. Based on these values, we can generate the *pmf* and *cdf* curves by connecting points at each specific value of X.

The motivation of the multi-bit BLASTed linkage is that we select two cutoff values C_1 and C_2 of X so that we can assign 1-bit, 2-bit, and 4-bit coding to the tokens in these three regions respectively (suppose $C_1 < C_2$). Basically we will assign 4-bit coding to those more important tokens, therefore those tokens would fall in the range $[C_2, x_n]$ on the *cdf* curve. There are a variety of potential criteria to select the cutoff values. For instance, we can select C_1 as the 25th percentile or the first quartile, and C_2 as the 75th percentile or the third quartile. This corresponds to the following selection criterion: $Prob(X \leq C_1) = Prob(X \geq C_2) = 0.25$. More generally, given two pre-determined thresholds ϕ_1 and ϕ_2 for the lower and upper boundaries of the cumulative distribution function, we can base on the following criterion to select the cutoff values:

$$Prob(X \leq C_1) = \phi_1 \text{ and } Prob(X \geq C_2) = \phi_2$$

The detailed steps for the proposed multi-bit BLASTed linkage is shown in Algorithm 1.

4. EXPERIMENTAL VALIDATION

4.1 Set-up

Platform. All experiments presented here are done on a machine with eight Dual-core 2400MHz AMD Opteron processors and a total of 32G memory. For the evaluation of BLASTed linkage schemes, we used the `blastn` of BLAST v 2.2.13 (linux) from NCBI ftp site[5]. The `blastn` is one of BLAST implementations for searching DNA sequence database. By default, BLAST filters out regions in a query sequence that are frequent in database sequences because those regions tend not to be biologically significant. However, since those regions can be important for record

[5]ftp://ftp.ncbi/nih.gov

Name	Data	Error	Domain	# of records	Max # of duplicates	# of queries	# of targets
`cora`	real	real	citation	1,326	5	98	194
`restaurant`	real	real	restaurant info.	864	2	111	111
`celebrity`	real	real	celebrity address	2615	2	276	276
`dblp`	real	synthetic	citation	5359	5	1,369	3,991
`dbgen1`	synthetic	synthetic	mailing list	10,159	20	949	9,210
`dbgen2`	synthetic	synthetic	mailing list	9,947	19	960	8,987
`dbgen3`	synthetic	synthetic	mailing list	100,327	9	17,842	80,324

Table 3: Summary of data sets.

Algorithm 1: **Multi-bit BLASTed linkage**.

Input : A complete list of tf-idf weights for each token, ϕ_1 and ϕ_2 as two thresholds of *cdf*
Output: Two cutoff values C_1 and C_2 to separate 1, 2, and 4-bit coding schemes

normalize the weights so that $\sum weight[i] = 1$;
sort tokens by normalized weights in the ascending order;
/*initialization*/;
$n \leftarrow$ number of tokens;
$i \leftarrow 0$, $sum \leftarrow 0$;
/*find C_1*/
while $i \leq n$ *and* $sum \leq \phi_1$ **do**
  $sum \leftarrow sum + weight[i], i \leftarrow i + 1$
endwhile
$C_1 \leftarrow weight[i]$;
/*find C_2*/
$j \leftarrow n - 1$, $sum \leftarrow 0$;
while $j \geq 0$ *and* $sum \leq 1 - \phi_2$ **do**
  $sum \leftarrow sum + weight[j], j \leftarrow j - 1$
endwhile
$C_2 \leftarrow weight[j]$;
return(C_1, C_2);

matching (e.g., year and venue in citation data sets), the filter option of BLAST was set "off." Besides, the rest of the parameters were untouched.

`blastp`, BLAST implementation for protein sequences, can use various substitution matrix and each matrix assigns different score for mismatches between different pairs of amino acids. `blastn` uses simpler scoring scheme so that the same score is assigned for mismatches between any two nucleotides. Since this simple scoring scheme is fit better for the comparison of English strings, we used `blastn` in our experiments.

Even though some parameters to calculate e-value is derived from the observation of biological sequences, e-value basically indicates how significant an alignment is given length of query and database sequences. Thus, e-values of an alignments between a query and each database sequence is considered as a measure of how each database sequence is relatively similar to the query in our context.

Data Sets. Table 3 shows the summary of data sets used in experiments. All three data sets, `cora`, `restaurant`, and `celebrity`, are used as real data sets, containing *real* string data and *real* errors. Both of `cora` and `restaurant` data sets are downloaded from RIDDLE[6] data repository[7]

[6] http://www.cs.utexas.edu/users/ml/riddle/
[7] The celebrity data set was provided to us by Ned Porter at US Census Bureau.

Error Type	Probability
Typo errors exist	0.3
Single typo error	0.6
Multiple typo error	0.4
First name is dropped	0.3
First name is abbreviated	0.7
Middle name is dropped	0.2
Middle name is abbreviated	0.8
Typo errors exist	0.9
Single typo error	0.9
Multiple typo error	0.1
Venue information as a full name	0.4
Pages and month information are dropped	0.5
Numeric information has a typo	0.2

Table 4: Parameter settings used for the `dbgen1` (above) and `dblp` (bottom) data set. Note that probabilities for different error types are independent each other.

First, from the original `cora` data set with 1,296 citation records, we hand-selected 292 records of 98 citations since some of the duplicates are incorrectly labeled and it is unrealistic that duplicates of only a few entities take a large portion of the entire database records like in the original data set. Then, it is extended with 1,034 additional citation records manually collected from the Web.

Second, the `restaurant` data set contains 864 records of restaurant information, such as restaurant name, address, telephone number, and food type that they serve (e.g., Korean, Chinese, Thai), collected from the Fodor's and Zagat's restaurant guides. Each record in the data set has exactly two duplicates.

Third, the original `celebrity` data set has 2764 records of celebrities' addresses with their names. The name of a celebrity is used as a key to distinguish duplicates in the data set. In case of the records with the same address for different celebrity names, only one of them was kept. Among the remaining 2615 records, 276 records are used as query strings and each of them has an exactly one duplicate.

Next, a new citation data set `dblp` was generated using real citation records from similar venues of DBLP. To inject artificial errors, we randomly selected ten venues from similar research domains such as SIGMOD, PODS, and EDBT, and again randomly selected citations published in those venues. Given initial citations, we generated duplicates by introducing typographical errors. For instance, single typo error was generated by inserting a new character, replacing a characters by different character, deleting a character, or swapping two characters in a word. For numeric data fields (e.g, journal volumes), we only use insertion and deletion. The detailed parameter settings to generate errors are shown

in Table 4.

Finally, using the data generation tool, DBGen [9], we generated four synthetic data sets, `dbgen1` to `dbgen3`, containing mailing list information with nine attributes such as SSN, first name, middle name, last name, street number, and so on. DBGen permits us to control error rates in records in detail. Both of `dbgen1` and `dbgen2` have around 10,000 records with on average 20 duplicates per record. The third data set, `dbgen3`, have more than 100,000 records for us to study the scalability of the proposed BLASTed linkage methods. For `dbgen1`, the probability to insert an error in the SSN field is set to 0.1 and the probability to change an address completely is 0.7. In `dbgen2`, the probability for a token to have a typo (10% single vs. 90% multiple typos) in person and street name fields is substantially increased to 0.8. The `dbgen3` is similarly made, too.

Evaluation Metrics. To evaluate the efficiency and effectiveness of the proposed BLASTed linkage methods, we mainly use two evaluation metrics – speed and accuracy. In particular, to measure the speed of a method, we use the **running time** excluding any pre-processing steps of sequence translation and database preparation. To measure the accuracy, we use the average precision and recall. Suppose that T denotes a set of true matching records and R denotes a set of records retrieved by an algorithm. Then, we have: precision=$\frac{|R \cap T|}{|R|}$ and recall=$\frac{|R \cap T|}{|T|}$. Since the golden solution set and # of true matching records are available in all of the data sets, in the selection model with top-k answers, we have $R = T = k$, making both precision and recall equivalent. Therefore, hereafter, we simply refer to the accuracy as precision. In addition, we also simulate the threshold model in evaluation by changing the k in top-k. In such a case, we will present the precision-recall graph in the traditional IR sense.

To compare with the performance of BLASTed linkage proposals, we chose three string distance metrics popular in conventional record linkage methods – Jaro, Jaccard, and SoftTFIDF – from the Java-based open-source implementations of the SecondString package [5]. Recent work such as [15] reported good performance of these methods in the context of the citation matching problem. In addition, we used JaroWinkler distance to detect obvious typos. Using symbols like x, y for strings, T_x for all tokens of x, C_x for all characters of x, $CC_{x,y}$ for all characters in x common with y, and $X_{x,y}$ for # of transpositions of character in x relative to y, simple definitions of the four metrics are below: (1) $\mathbf{Jaro}(\mathbf{x},\mathbf{y}) = \frac{1}{3} \times (\frac{|CC_{x,y}|}{|C_x|} + \frac{|CC_{y,x}|}{|C_y|} + \frac{|CC_{x,y}| - X_{CC_{x,y},CC_{y,x}}}{2|CC_{x,y}|})$; (2) $\mathbf{Jaccard}(\mathbf{x},\mathbf{y}) = \frac{|T_x \bigcap T_y|}{|T_x \bigcup T_y|}$; (3) $\mathbf{TFIDF}(\mathbf{x},\mathbf{y}) = \sum_{w \in T_x \cap T_y} V(w,T_x) \times V(w,T_y)$, where $V(w,T_x) = log(TF_{w,T_y}+1) \times \frac{log(IDF_w)}{\sqrt{\sum_{w'}(log(TF_{w,T_y}+1) \times log(IDF_w))}}$ (symmetrical for $V(w,T_y)$), where TF_{w,T_x} is the frequency of w in T_x, and IDF_w is the inverse of the fraction of names in a corpus containing w; and (4) $\mathbf{JaroWinkler}(\mathbf{x},\mathbf{y}) = Jaro(x,y) + \frac{max(|L|,4)}{10} \times (1 - Jaro(x,y))$, where L is the longest common prefix of x and y. The SoftTFIDF method is a minor improvement from the TFIDF method to use "soft" token-matching idea [5].

4.2 Comparison with Baseline Approaches

Figure 5(a) shows a per-query execution time computed by dividing the total execution time per data set by the number of queries in the data set. The BLASTed linkage schemes dominated two baseline methods, Jaro and SoftTFIDF, in most of the data sets. Jaccard ran faster than BLASTed linkage methods in `cora`, `restaurant`, and `dblp` data sets.

The execution time of BLASTed linkage schemes is largely affected by the lengths of sequences to be matched due to the nonlinear nature of its matching algorithm while it is not affected so much by the number of records in the target database as it uses inverted list to prune out unrelated sequences before starting the alignment. Consequently, the difference in the performance two baseline methods, which are Jaro and SoftTFIDF, and BLASTed linkage methods is much more significant in `dblp` and `dbgen2`, due to the sizes of the data sets, than in `cora` and `restaurant` data sets.

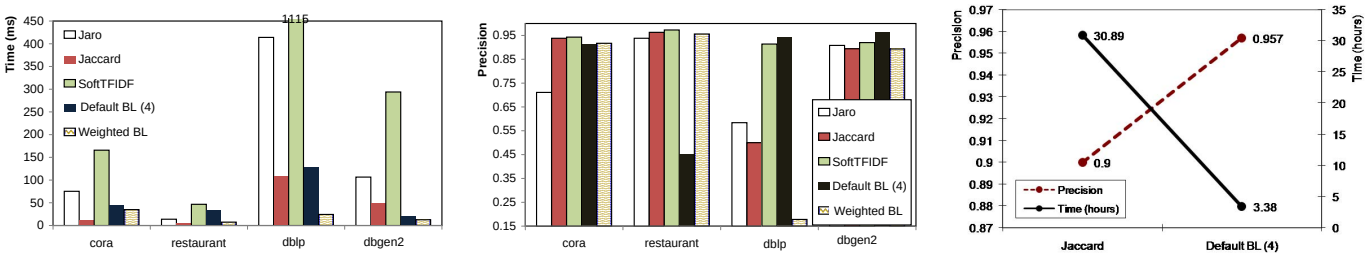
However, the cost of token-based matching methods like Jaccard increases quadratic to the number of records to match. Figure 5(c) highlights the scalability of BLASTed linkage method using the largest `dbgen3` data set, compared to Jaccard. As shown in figure 5(a), in general, Jaccard are fast for small-sized data sets. However, for a substantially large data set, thanks to the fast performance of underlying BLAST algorithm and its optimized implementation, our default BLASTed linkage with 4-bit scheme far outperforms Jaccard by an order of magnitude (3.38 vs. 30.89 hours) while maintaining higher precision (0.957 vs. 0.9).

Record linkage applications deal with large data sets and the scalability of the matching algorithm is one of their primary concerns. We can easily address the problem using BLAST and it is one of the key motivations of this work.

Figure 5(b) shows the precisions of the BLASTed linkage schemes and the baseline approaches measured on the four data sets. Graphs with other data sets show similar pattern and are omitted. As shown in the figure, Jaccard and SoftTFIDF showed slightly better precision than the BLASTed linkage schemes in `cora` and `restaurant` data sets. The `cora` data set is particularly challenging because it includes numbers of different citations sharing the same venue information. If the venue of those citations is long, the alignment algorithms can get mislead as the substantial parts of them are matched. Also, there are some numbers of entries that have the same title and authors but published in different venues. It is virtually impossible to tell the identity of the entries without looking at the actual contents of the papers.

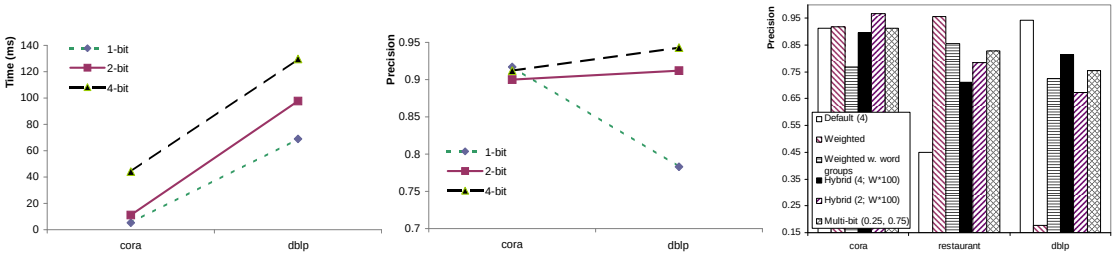
Default BLASTed linkage showed an interesting result on the `restaurant` data set. Compared to its results on the other data set, its precision was very low. This is due to the characteristics of record strings in the `restaurant` data set. The records of `restaurant` data set have restaurant name, address, telephone number, and food type. Among the record fields, the address string takes a large portion of entire record string. The `restaurant` data set include information of restaurants in a few cities in U.S., and thus many records are likely to share a portion of their addresses, such as name of street, city, and area code and so on. If two different restaurant records have large common substrings, default BLASTed linkage can easily get mislead to find them as a match.

On the other hand, weighted BLASTed linkage does not suffer from this problem as much as default BLASTed linkage does because it translates words to sequences of "different" lengths according to their weights. Common words



(a) (b) (c)

Figure 5: (a)-(b): Comparison of string distance measures vs. the default BLASTed linkage using 4-bit coding and weighted BLASTed linkage. (c)Running time (right Y-axis) and precision (left Y-axis) of the largest `dbgen3` data set of 100,327 records.



(a) Running time (ms) (b) Precision (c) Precision

Figure 6: Comparison: (a)-(b): three bit schemes in default BLASTed linkage and (c): six variations of BLASTed linkage methods using three data sets.

such as street names, cities and area codes will get low weights and translated into short sequences, reducing their impacts on alignment. As previously discussed, records of `cora` data set also share common venue names in their record strings. Therefore, precision of weighted BLASTed linkage on the `cora` data set was also better than that of default one. However, in the `cora` data set, the difference between the two was insignificant because the venue information does not take as much portion from the record as the address does in the `restaurant` data set.

Meanwhile, on both of `dblp` and `dbgen2` data sets, default BLASTed linkage showed the best performance. Especially, its performance on `dblp` is worth to note where we introduced a high error rate. The precision of Jaro and Jaccard, which showed good precisions on other data sets, decreased drastically to around 0.5. The precision of weighted BLASTed linkage on `dblp` data set is very low because the errors are introduced to tokens and weighted BLASTed linkage takes the same tokens with typographical errors as different tokens.

4.3 Comparison of 1/2/4-bit Coding Schemes

Figure 6(a)-(b) presents the execution time and precision of the default BLASTed linkage with three different bit-coding schemes. We compared the default 4-bit coding to two lossy bit-coding schemes of 1-bit and 2-bit codings. Tests performed on `cora` and `dblp` data sets are presented. As can be seen in Figure 6(a), default BLASTed linkage with 4-bit coding is the slowest taking approximately 40 and 130 milliseconds on `cora` and `dblp` data sets, respectively. On the other hand, the 1-bit coding scheme was the fastest taking only less than half of the time that 4-bit coding required.

As shown in the figure 6(b), all three coding schemes showed comparable results on `cora` data set achieving about 0.9 accuracy. On the other hand, on `dblp`, the 4-bit scheme performed slightly better than 2-bit achieving 0.94 as precision while the 1-bit scheme was the worst achieving only 0.78 as precision. The performance degradation of the lossy coding schemas are data set dependent while execution time saving is evident.

4.4 Comparison among BLASTed Linkages

In Figure 6(c), we compared six variations of BLASTed linkage method using three data sets. In the graph, default(4) represents default BLASTed linkage methods with 4-bit coding scheme. And, hybrid (4; W*100) refers hybrid BLASTed linkage method with 4-bit coding scheme and an importance weight, which is the original weight of a token multiplied by 100. Multi-bit (0.25, 0.75) represents multi-bit BLASTed linkage method with a lower cutoff value 0.25 and a higher cutoff value 0.75. The third variant "weighted BL with word group" refers to the application of weighted BLASTed linkage after we coalesce similar words (e.g., typos) into groups using JaroWinkler method.

For `cora` data set, all methods, except the "weighted BL with word group", perform well, above the precision of 0.9. The hybrid BLASTed linkage methods with 2-bit coding scheme and an importance weight of the original weight multiplied by 100 outperforms other methods. The real data/error of `restaurant` data set, weighted BLASTed linkage shows the best precision. Because the data set has high chance that a large portion of the tokens in two records are common, weighted BLASTed linkage methods improves the precision of default BLASTed linkage method as considering word weight. Finally, for the synthetic data set `dblp`, default BLASTed linkage methods outperform the rest. It is

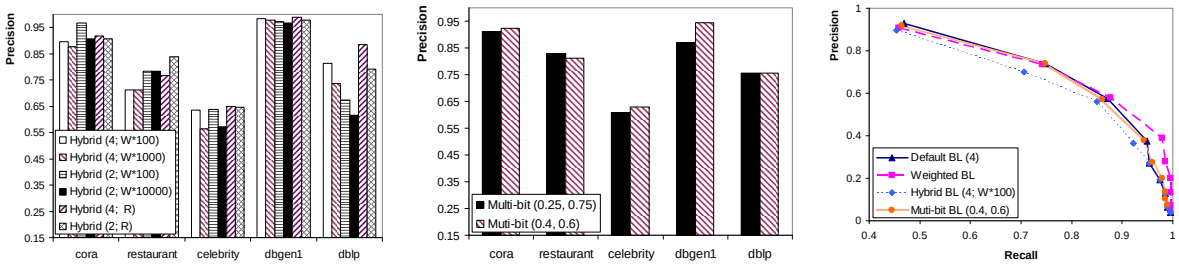


Figure 7: (a) Comparison among four variations of hybrid BLASTed linkage methods using five data sets. (b) Comparison between multi-bit BLASTed linkage methods with two cutoff parameters using five data sets. (c) Precision-recall graphs of three variations of BLASTed linkage methods using `cora` data set.

interesting that the precision of weighted BLASTed linkage method, the best method for `restaurant` data set, is very low due to typographical errors introduced in the data set.

Overall, there is no clear winner across three data sets as inferring that each BLASTed linkage method performs differently depending on the characteristics of each data set, such as the number of tokens shared in different records and types/degrees of errors in the data set. Multi-bit BLASTed linkage method does not outperform other BLASTed linkage methods in all data sets, however it seems to provide robust performance in overall. The performance of hybrid and weighted BLASTed linkage with word group is relatively consistent. Also, it seems that Default and weighted BLASTed linkage methods are affected by the characteristics of a data set the most.

4.5 Hybrid vs. Multi-bit BLASTed linkage

As shown in Figure 7, the performance of hybrid BLASTed linkage method is tested on five data sets as using six combinations of different bit coding schemes and types of importance weight. Either of a token weight or a token rank is used as importance weight for each token. The original token weight is multiplied by either of 100 or 10,000 before the weight is translated to DNA sequence. As multiplying larger value, we can increase the proportion of weight information in DNA sequence of a records. Also, as using larger number-bit scoring scheme, the proportion of lexical meaning information of each token is increased.

With the fixed importance weight, hybrid BLASTed linkage methods with 4-bit coding scheme outperforms the methods with 2-bit coding schemes in `dbgen1` and `dblp`. However, the result is the opposite in `cora` and `restaurant` data sets. In `celebrity` data set, different coding schemes almost does not affect the performance of hybrid BLASTed linkage. Next, given the same bit coding scheme, hybrid BLASTed linkage method with smaller importance weight shows better precision than that with larger importance weight in all data sets except for `restaurant` data set. Different importance weight does not change the performance of hybrid BLASTed linkage methods in `restaurant` data set.

Overall, hybrid BLASTed linkage method with larger number-bit coding scheme and smaller importance weight works better than other methods in both of `dbgen1` and `dblp`. For `celebrity` data set, hybrid BLASTed linkage with largest importance weight show better precision. Contrast to `celebrity` data set, which bit coding scheme is used is more important in `restaurant` data set and hybrid BLASTed linkage smaller value bit coding scheme outperforms that with larger value bit coding scheme. For `cora` data set, it is better to use hybrid BLASTed linkage with smaller value bit coding scheme and smaller importance weight.

To see the effect of different cutoff values for multi-bit BLASTed linkage method, two different cutoff values, such as (0.25, 0.75) and (0.4, 0.6), are used. Figure 7 shows the result of the two variations using five data sets. We cannot conclude that which variation clearly outperform the other. It is possible that optimal cutoff values for selection criterion of bit coding schemes on tokens with different weights may depend on data sets.

4.6 Discussion

In this section, we will discuss about variations of BLASTed linkage method to answer the following questions: (1) *Which variation of BLASTed linkage method would work optimal on the data set with particular characteristics?* (2) *Which variation of BLASTed linkage method performs consistently in overall?*

Default BLASTed linkage method performs consistently well on the data sets with both of low and high error rate, except for the data sets with many records sharing common tokens as much as a significant portion of the entire records. On the other hands, weighted BLASTed linkage method performs well with such data sets, as well as those with low error rate. For the data sets with relatively high error and the records sharing many common tokens, either of hybrid and multi-bit BLASTed linkage methods would work well. For the data set with high error rate, hybrid BLASTed linkage method with 4-bit coding scheme and small importance weight performs better than hybrid method with 2-bit coding scheme and large importance weight. However, for the data set with low error rate, hybrid method with 2-bit coding scheme and large importance weight outperforms. Multi-bit BLASTed linkage method seems perform consistently well for all kinds of data sets. It is true that we still need to decide the proper cutoff values for selection criterion of bit coding schemes, but the performance of multi-bit BLASTed linkage method does not change much with reasonable cutoff values as shown in figure 7(b).

The precision-recall graph in figure 7(c) is generated with precision and recall of three variations of BLASTed linkage methods as increasing k, which is the number of answers returned (e.g. top-k answers). With respect to recall, precision of default and hybrid BLASTed linkage methods is changed as following very similar curve. As recall is in-

creased, precision of multi-bit BLASTed linkage method is dropped faster than default and hybrid BLASTed linkage methods. However, the difference is not significant. Carefully selected BLASTed linkage method which is suitable for a certain data set can guarantee better performance on the data set. Either of hybrid and multi-bit BLASTed linkage methods perform robust for any kind of data sets despite.

5. RELATED WORK

The general linkage problem has been known as various names – record linkage (e.g., [7, 3]), identity uncertainty (e.g., [16]), merge-purge (e.g., [9]), citation matching (e.g., [15]), object matching (e.g., [4]), entity resolution (e.g., [18]), authority control (e.g., [21]), and approximate string join (e.g., [8, 4]) etc. Readers are referred to excellent survey papers (e.g., [22, 6]) for the latest development of the linkage problem. To our best knowledge, none of these existing works attempted to solve the linkage problem using bioinformatics framework as we did it using BLAST. In terms of distance measures to use in comparing records, Cohen et al. [5] have compared the efficacy of various string-distance metrics including Jaro and Jaro-Winkler. [15] conducted an in-depth study on the blocking issue of the linkage problem in digital library context. On the other hand, in terms of scalability issue, people have proposed the blocking technique [11], computationally efficient distance function [14], or parallel linkage [12].

To our best knowledge, there have been very few attempts to use BLAST for applications outside of gene sequence alignment. First, [2] borrows the idea of multiple sequence alignment from BLAST in building a mapping dictionary, that is a lexicon of elementary semantic expressions and corresponding natural language realizations. Second, [13][8] uses BLAST to detect gene and protein names from journal articles by viewing an entire article as a query sequence and a set of known gene and protein terms as a database of sequences to match. Finally, [10] employs BLAST in the citation field segmentation problem – i.e., split a unsegmented citation string into known set of fields such as author, title, venue, and year. In the same sprit of these works but for the first time, we propose to use BLAST to solve the *linkage* problem. Moreover, we devise/evaluate four effective variations to transform regular string data to DNA sequences.

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented the novel idea of applying BLAST, one of the most popular sequence alignment algorithms in Bioinformatics, to the (record) linkage problem. To handle various cases of the linkage problem, we proposed four variations of the BLAST based linkage schemes, and show their efficacy using both real and synthetic data sets. Overall, our proposed BLAST based schemes show promising results with good combination of accuracy and speed, compared to conventional string matching methods such as Jaro, Jaccard, and SoftTFIDF. Since BLAST has a wealth of supporting algorithms, implementations, and tools in its user community, our proposal enables users of the linkage problem to have an immediate access to rich resources.

Many directions are ahead. First, in this paper, we only focused on exploiting the basic version of BLAST for the linkage problem. Applying the very same idea using more recent and advanced BLAST versions (e.g., PSI-BLAST, MegaBLAST) is an interesting extension to try. Second, many domain-specific linkage solutions have been proposed. Comparing the performance of our BLASTed linkage schemes against other state-of-the-art (but less generic) solutions would be able to help improve the weakness of our ideas further.

[8]To our best knowledge, [13] is the first article, mentioning the idea of transforming an English alphabet to DNA sequences of `A`, `C`, `G`, and `T`.

7. REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. "Basic Local Alignment Search Tool". *J. Mol. Bio.*, 1990.

[2] R. Barzilay and L. Lee. "Bootstrapping Lexical Choice via Multiple-Sequence Alignment". In *Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[3] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. "Adaptive Name-Matching in Information Integration". *IEEE Intelligent System*, 18(5):16–23, 2003.

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. "Robust and Efficient Fuzzy Match for Online Data Cleaning". In *ACM SIGMOD*, 2003.

[5] W. Cohen, P. Ravikumar, and S. Fienberg. "A Comparison of String Distance Metrics for Name-matching tasks". In *IIWeb Workshop held in conjunction with IJCAI*, 2003.

[6] A. Elmagarmid, P. Ipeirotis, and V. Verykios. "Duplicate Record Detection: A Survey". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[7] I. P. Fellegi and A. B. Sunter. "A Theory for Record Linkage". *J. of the American Statistical Society*, 1969.

[8] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. "Text Joins in an RDBMS for Web Data Integration". In *Int'l World Wide Web Conf. (WWW)*, 2003.

[9] M. A. Hernandez and S. J. Stolfo. "The Merge/Purge Problem for Large Databases". In *ACM SIGMOD*, 1995.

[10] I.-A. Huang, J.-M. Ho, H.-Y. Kao, and S.-H. Lin. "Extracting Citation Metadata from Online Publication Lists Using BLAST". In *Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD)*, 2004.

[11] R. P. Kelley. "Blocking Considerations for Record Linkage Under Conditions of Uncertainty". In *Proc. of Social Statistics Section*, pages 602–605, 1984.

[12] H. Kim and D. Lee. "Parallel Linkage". In *ACM CIKM*, Lisbon, Portugal, Nov. 2007.

[13] M. Krauthammer, A. Rzhetsky, P. Morozov, and C. Friedman. "Using BLAST for Identifying Gene and Protein names in Journal Articles". *Gene*, 259, 2000.

[14] B.-W. On, N. Koudas, D. Lee, and D. Srivastava. "Group Linkage". In *IEEE ICDE*, Istanbul, Turkey, Apr. 2007.

[15] B.-W. On, D. Lee, J. Kang, and P. Mitra. "Comparative Study of Name Disambiguation Problem using a Scalable Blocking-based Framework". In *ACM/IEEE Joint Conf. on Digital Libraries (JCDL)*, Jun. 2005.

[16] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. "Identity Uncertainty and Citation Matching". In *Advances in Neural Information Processing Systems*. 2003.

[17] H. Rangwala, E. Lantz, R. Musselman, K. Pinnow, B. Smith, and B. Wallenfelt. "Massively Parallel BLAST for the Blue Gene/L". In *High Availability and Performance Computing Workshop (HAPCW)*, Santa Fe, NM, 2005.

[18] S. Sarawagi and A. Bhamidipaty. "Interactive Deduplication using Active Learning". In *ACM KDD*, 2002.

[19] W. Shen, X. Li, and A. Doan. "Constraint-Based Entity Matching". In *AAAI*, 2005.

[20] T. Smith and M. Waterman. "Identification of Common Molecular Subsequences". *J. Mol. Biology*, 147, 1981.

[21] J. W. Warnner and E. W. Brown. "Automated Name Authority Control". In *ACM/IEEE Joint Conf. on Digital Libraries (JCDL)*, 2001.

[22] W. E. Winkler. "The State of Record Linkage and Current Research Problems". Technical report, US Bureau of the Census, Apr. 1999.