APPLICATION OF SEMI-LOCAL LCS TO STRING APPROXIMATE MATCHING[*]

DIANNE DOE[†], PAUL T. FRANK[‡], AND JANE E. SMITH[‡]

Abstract. We present an application of semi-local lcs to approximate string matching by developing a new algorithm and improving the existing one. Our result is based on the utilization of the underlying algebraic structure of semi-local lcs with the usage of the novel data structure for submatrix maximum queries in Monge matrices. This gives two algorithms with the following running time and space complexity. TODO. The improvement of the existing algorithm not only preserves all properties but also outperforms in practice.

In addition, we show that the algorithm for semi-local lcs based on sticky braid multiplication is not perform well with the current complex recursive structure.

Key words. semi-local lcs, monge matrix, range queries, approximate matching, near-duplicate detection

AMS subject classifications. 68Q25, 68R10, 68U05

1. Introduction.

Approximate string matching is an important task in many fields such as computational biology, signal processing, text retrieval and etc. It also refers to a duplicate detection subtask.

In general form it formulates as follows: Given some pattern p and text t need to find all occurrences of pattern p in text t with some degree of similarity.

There are many algorithms that solve the above problem. Nonetheless, the number of algorithms sharply decreases when the algorithm needs to meet some specific requirements imposed by running time, space complexity or specific criterion for the algorithm itself. For example, recently there was developed an approach for interactive duplicate detection for software documentation [2]. The core of this approach is an algorithm that detects approximate clones of a given user pattern with a specified degree of similarity. The main advantage of the algorithm is that it meets a specific requirement of completeness. Nonetheless, it has an unpleasant time complexity.

The algorithm for approximate detection utilizes mainly algorithm for solving the longest commons subsequence (LCS) problem. The longest common subsequence is a well-known fundamental problem in computer science that also has many applications of its own. The major drawback of it that it shows only the global similarity for given input strings. For many tasks, it's simply not enough. The approximate matching is an example of it.

There exist generalazation for LCS called *semi-local LCS* [] which overcome this constraint. The effective theoretical solutions for this generalized problem found applications to various algorithmic problems such as bla bla add cited. For example, there has been developed algorithm for approximate matching in the grammar-compresed strings[].

Although the algorithms for *semi-local LCS* have good theoretical properties, there is unclear how they would behave in practice for a specific task and domain.

To show the applicability of semi-local lcs on practice we developed several algorithms based mainly on it and the underlying algebraic structure. As well as devel-

[*]Submitted to the editors DATE.

Funding: This work was funded by the Fog Research Institute under contract no. FRI-454.

[†]Imagination Corp., Chicago, IL (ddoe@imag.com, http://www.imag.com/~ddoe/).

[‡]Department of Applied Mathematics, Fictional University, Boise, ID (ptfrank@fictional.edu, jesmith@fictional.edu).

This manuscript is for review purposes only.

oping new algorithms we improve and significantly outperform the existing one for interactive duplicate detection for software documentation []. It should be noted that improvement preserves all properties of this algorithm. Do we need to state that ant algo is slow for current strucute of algorithm

The paper is organized as follows. Blablabla **??**, our new algorithm is in **??**, experimental results are in **??**, and the conclusions follow in **??**.

2. Preliminaries.

2.1. Approximate matching.

Describe approximate matching formally

2.2. Semi-local lcs.

Describe semi-local lcs (definition), algorithms that solves (steady and and braid reducing)

2.3. Monge matrix.

Describe monge property

Say about range queries (about soda12, soda14 and new result that we will be used)

2.4. Near-duplicate detection algorithm.

Describe luciv algo

3. Related work.

?????

could mention about approximation. Need discuss

4. Algorithm for near duplicate detection.

TODO: third phase also may be imporved/ removed. By filtering in place. Also prserves property.

We now describe an improved version of Luciv et.al. algorithm [2] by utilizing a *semi-local sa* solution. Then we present proof that improved version preserves completnesess property. It is achieved by imitating all phases of the algorithm.

4.1. Algorithm description.

The algorithm comprises three phases as in [2]. At phase one (Line 1) semi-local sa problem is solved for the pattern p against whole text t. This solution provides access to the string-substring matrix which allows performing fast queries of sa score for pattern p against every substring of text t.

At the second phase text t is scanning with a sliding window of length L_w with step 1. First, it checks that given substring w that of a maximum possible size of L_w have score that is higher or equal to a given threshold (Line 4). If no, then this interval will not further be proceeded (Line 5) else this interval will be processed as follows. First, for each prefix of text t it finds suffix that has the highest alignment score with the maximal length among all suffixes with that score. It corresponds to the searching row position for each column in string-substring matrix with associated alignment score. Second, among these suffixes, one is selected with the highest score. If several suffixes have the same score the one with maximal length is selected (Line 8). Then if selected suffix has score higher than the threshold, then it is added to set W_2.

The third phase is the same as in [2]. More precisely, on the third phrase, set W_2 is filtered out in a such way that only non-intersected intervals are left. It is simply the sorting of set W_3 by starts of intervals with following one way passage with filtering.

Algorithm 4.1 PATTERN BASED NEAR DUPLICATE SEARCH ALGORITHM VIA SEMI-LOCAL SA

Input: pattern p, text t, similiarity measure $k \in [\frac{1}{\sqrt{3}}, 1]$
Output: Set of non-intersected clones of pattern p in text t

$$k_{di} = |p| * (\frac{1}{k} + 1)(1 - k^2) \tag{4.1}$$

$$L_w = \frac{|p|}{k} \tag{4.2}$$

Comment: w_i, w_j — start and end positions of w in text t
Pseudocode:

```
1: $W = semilocalsa(p, t)$
2: $W_2 = \emptyset$
3: for $w \in t, |w| = L_w$ do
4:     if $W.stringSubstring(w_i, w_j) < -k_{di}$ then
5:         continue
6:     end if
7:     $maximums = FindMaxForColumnsBySmawk(w)$
8:     $max = FindMaxWithLenghtConstraint(maximums)$
9:     if $max \geq -k_{di}$ then
10:        add substring associated with max to $W_2$
11:    end if
12: end for
13: $W_3 = UNIQUE(W_2)$ {3rd phase unchanged}
14: for $w \in W_3$ do
15:     if $\exists w^{'} \in W_3 : w \subset w^{'}$ then
16:         $remove\ w\ from\ W_3$
17:     end if
18: end for
19: return $W_3$
```

Describe our algorithm and shows our optimization.
Proof all properties satisified.
Make note that not contraint on other metric???

5. CutMax a new approximate mathing algorithm. We now descibe several algorithms that heavily based on semi-local lcs and its underlying algebraic structure.

The first algorithm 5.1 reffers to following constraint. There should be found all non-intersected clones τ_i of pattern p from text t that has the highest similiarity score on the uncovered part of the text t i.e algorithm should performs greedy choice at each step.

First, semi-local sa problem is solved (Line 1). Then upon string-substring submatrix of semi-local sa solution is builded data structure for perforiming range queries in square martix (Lines 2-3). $IntervalsToSearch$ that holds text intervals where should be perfomed search. Lines 7-21 corresponds to seaching interval τ_i with maximal alignment within the current uncovered part of the text t. More preciesely, it refers to searching maximum value with corresponding position (row and column) in submatrix of string-substring *matrix* within bounds for text starting at ith position and ending at jth position. It is solved via range queries (Line 9). After that if foun When detected *interval* has alignment score less then therhsold it means that no clones of pattern p is presented in this part of text t and further processing should be skipped (Line 19). Otherwise, founded clone is added to final result and current part of text is splitted on two smaller part and procces in same way (Lines 13, 15). Finally, algorithm outputs a set of non-intersected interval of clones of pattern p in text t.

Theorem Algorithm 5.1 could be solved with total running time $O(|p| * |t| * |v|)$, and space $O()$. The time complexity of solving semi-local sa is $O(|p| * |t| * |v|)$. Due to fact that semi-local matrix is Monge matrix. It allows using recent achievements in range maximum queries []. More preciesely, data structure for range queries could be computed in O() time with O() space. Further, this data strucure allows to perform range query in o() time.

Lines 7-21 basically iterative version of recursion where at each level of recursion amount of quesries is doubles at worst case. The total amount of nodes at each level is following. At the bottom level of recursion is t nodes, next level is $t/2$, ith level is $t/2_i$. Thus, the overall amoount of nodes is $t + \frac{t}{2} + \frac{t}{4} + ...1 = t * (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{2^i}) =$ sum. As mentioned above complexity of query is O(). This time complexity of Lines 7-21 is O()....

THEOREM WHen v=O(1) its

Algorithm 5.1 GREEDY-PATTERN BASED NEAR DUPLICATE SEARCH ALGORITHM VIA SEMI-LOCAL SA

Input: pattern p, text t, theshold value h
Output: Set of non-intersected clones of pattern p in text t
Pseudocode:

```
1: $sa = semilocalsa(p, t)$
2: $matrix = sa.getStringSubstringMatrix()$
3: $rmQ2D = buildRMQStructure(matrix)$
4: $intervalsToSearch = \emptyset$
5: $intervalsToSearch.add((0, |t|))$
6: $result = \emptyset$
7: while $intervalsToSearch.isNotEmpty()$ do
8:     $i, j = intervalsToSearch.pop()$
9:     $interval = rmQ2D.query(i, j, i, j)$
10:    if $interval.score \geq threshold$ then
11:        $result.add(interval)$
12:        if $interval.startInclusive - i \geq 1$ then
13:            $intervalsToSearch.add((i, interval.startInclusive))$
14:        end if
15:        if $j - interval.endExclusive \geq 1$ then
16:            $intervalsToSearch.add((interval.endExclusive, j))$
17:        end if
18:    else
19:        $continue$
20:    end if
21: end while
22: return $result$
```

The second algorithm uses simple approach. It TODOOOOOOOOOOOOOOOOOOOOOOO. description

Algorithm 5.2 DFFF SEMI-LOCAL SA

Input: pattern p, text t, theshold value h
Output: Set of non-intersected clones of pattern p in text t
Pseudocode:

```
1: $sa = semilocalsa(p, t)$
2: $matrix = sa.getStringSubstringMatrix()$
3: $colmax = smawk(matrix)$
4: $colmax.filterit.score >= realThreshold.sortedByDescendingit.score$
5: $tree = buildIntervalTree()$
6: for $candidateincolmax$ do
7:    if $candidateisnotintersectedwithtree$ then
8:       $tree.add(candidate)$
9:    end if
10: end for
11: $result = tree.toList()$
12: return $result$
```

THEOREM runniing time and s oon

THEOREM WHen v=O(1) its

Lest's look at the taks of approximate mathing from the different perspective and more intuitive way.

Descirbe algroithm.

Present algo implementation with sparse table. Say that bad.

Describe optimization via monge property.

Describe complexity

6. Evaluation.

Semi-local algorithms. Show perfomance between lcs and semi-local lcs??? and poor perfomance of recursive algorithm based on steady ant?

Approximate matching algorithms. Show outperforming for different cases between luciv and our algorithm.

Show quality betwee our new algo and luciv algo (our should be better)

Show that sparse table bad when large?

7. Conclusion. Say may be succesfully be applied on practice (showed by algorithm luciv updated)

Open problem.$->$

Say that need to implement with monge2020 (what we not finished)

Improve algo based on recursive steady ant. Because it's critical for algos based on it.

df[1]

Acknowledgments. We would like to acknowledge the assistance of volunteers in putting together this example manuscript and supplement.

REFERENCES

[1] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 4th ed., 2013.

[2] D. V. Luciv, D. V. Koznov, A. Shelikhovskii, K. Y. Romanovsky, G. A. Chernishev, A. N. Terekhov, D. A. Grigoriev, A. N. Smirnova, D. Borovkov, and A. Vasenina, *Interactive near duplicate search in software documentation*, Programming and Computer Software, 45 (2019), pp. 346–355.

This manuscript is for review purposes only.