The Problem of Context Sensitive String Matching

Venkatesan T. Chakaravarthy and Rajasekar Krishnamurthy

Computer Science Department,
University of Wisconsin, Madison, WI 53706, USA,
{venkat,sekar}@cs.wisc.edu,
WWW home page: http://www.cs.wisc.edu/~{venkat,sekar}

Abstract. In the context sensitive string matching problem, we are given a pattern and a text. The pattern is a string over variables and constants and the text is a string of constants. The goal is to find if there is a mapping from variables to strings of constants so that on applying this mapping to the pattern we get the given text. Languages like Perl and Python support such a sophisticated string matching. The problem is known to be NP-Complete. In this paper, we consider a weighted version of this problem that checks how close the pattern can be matched with the text. We show that this variation is MAXSNP-Complete and cannot be approximated within a factor of 3313/3312. We show that even the restriction, where the pattern consists of variables only, is NP-Complete and MAXSNP-Complete. When the alphabet is bounded, we give an approximation algorithm for this restriction.

1 Introduction

String matching is a well studied problem and has applications in a wide variety of fields. Recently, various modifications of the exact string matching problem have been considered. Applications in computational biology and computer vision have motivated the study of approximate string matching [5, 6], where different matching relations like swapped matching [2], "don't cares" [8] and overlap matching [3] have been proposed.

There is another interesting variation of exact string matching, which we call *context sensitive string matching*. Here, the pattern is allowed to have variables and the goal is to map variables to strings, such that, when the variables in the pattern are replaced by the corresponding strings, we get the text. This kind of string matching is supported by languages like Perl and Python. Moreover, some of the complex list processing capabilities of languages like Prolog and Lisp are captured by this problem. The problem was first considered by Angluin [4] in the context of finding patterns common to a set of strings. We next define the problem formally.

Problem Definition: The input consists of a set of *constants* Σ, a set of *variables* V, a *pattern* $P \in (V + \Sigma)^+$ and a *text* $T \in \Sigma^+$. An *assignment* σ is a mapping $\sigma : V \rightarrow \Sigma^+$. $\sigma(P)$ denotes the string obtained by replacing each

occurrence of a variable X in P, by $\sigma(X)$. If the strings $\sigma(P)$ and T are the same, σ is a *matching assignment*. The goal of the problem is to determine whether such a matching assignment exists. If so, then P matches T.

For example, let $\Sigma = \{a, b, c, d, e\}$, $V = \{X, Y\}$, $P = XaXY$ and $T =$ "*abaabcde*". Let σ be an assignment that maps X to "*ab*" and Y to "*cde*". Then $\sigma(P)$ is same as the text and σ is a matching assignment. On the other hand, if $P = XX$ and $T =$ "*aaa*", then it is easy to see that there cannot be any matching assignment. Note that each occurrence of a variable X in the pattern has to be replaced by the same string[1] and no variable can be mapped to the null string.

The context sensitive string matching problem was shown to be NP-Complete, even over a binary alphabet, by Angluin [4]. Just like how exact string matching was extended to approximate string matching, we extend context sensitive string matching to a weighted version. When there is no matching assignment between the pattern and the text, we want to find how close the pattern can be matched with the text. To explore this possibility of approximate matching, we generalize the problem to a weighted version. We show that this problem is MAXSNP-Complete and cannot be approximated within a factor of 3313/3312. We also consider various restrictions of the problem. When the pattern consists of variables only, we show that the problem remains NP-Complete and MAXSNP-Complete. We also give a simple approximation algorithm for this restriction, when the alphabet is bounded.

We refer to the set of constants Σ as the *alphabet*. We represent the constants by small letters and special symbols, and the variables by capital letters. Throughout the paper, we refer to the context sensitive string matching problem simply as CS-Matching.

Related Work

The concept of pattern, in the above sense, has been considered in many scenarios. In [4], the problem of finding a "good" pattern that matches a given set of strings was considered. In that context, the language $L(P)$ of a pattern P is defined to be the set of all strings that can be matched (under some assignment) to P. Given a text T and a pattern P, then the CS-Matching problem is membership testing of T in $L(P)$. This was shown to be NP-Complete in [4]. In this paper, we consider the weighted version of the problem, where we attempt to find an assignment that matches the pattern to the text as close as possible.

In our definitions, we required that each variable is assigned to a non-null string. Such patterns are called non-erasing in literature. If we allow null-string assignments also, the patterns are called erasing. Decidability problems about the equivalence and inclusion among patterns, in both these settings, was studied in [4, 11, 15]. The generative power of pattern languages were considered in [7, 14], from a formal language theoretic perspective. While we allow a variable to be mapped to any string (from Σ^+), they consider the power of pattern

[1] This has been referred to as *uniform substitution* in literature.

languages when this mapping is restricted to specific languages (like regular and context-free languages).

CS-Matching has also been considered in the context of unavoidability testing of a pattern. An infinite text T is said to avoid a pattern P, if P does not match any substring of T. A pattern is unavoidable if no infinite text avoids it. Prior work like [10, 13, 17] deals with identifying necessary and sufficient conditions for a pattern to be unavoidable. Our problem differs in that the goal here is to check how close the pattern matches a given text of finite length.

When each variable occurs only once in the pattern, the pattern can be treated as a regular expression. Thus, CS-Matching is a generalization of the well known regular expression string matching problem.

2 Weighted Context Sensitive String Matching

For a pattern P and a text T, there may not be any matching assignment. In this case, we want to see how close the pattern can be matched to the text. Towards that end, we first define a weighted version of the problem. We then show that this weighted version is MAXSNP-Complete.

There is a natural way to extend the CS-Matching problem to a weighted version. Here, apart from the usual Σ, V, P and T, the input also includes two positive numbers α, the *matching-cost*, and β, the *mismatch-cost*, with $\beta > \alpha$. We call an assignment σ to be a *feasible assignment* if $|\sigma(P)| = |T|$. The cost of a feasible assignment σ is the distance between $\sigma(P)$ and T. Let $T = a_1 a_2 \dots a_n$ and $\sigma(P) = b_1 b_2 \dots b_n$. The distance is given by $\Sigma d(a_i, b_i)$, where $d(a_i, b_i)$ is α, if $a_i = b_i$ and β otherwise. If σ is not a feasible assignment then its cost is ∞. We require that the input instances have at least one feasible assignment[2]. Then the weighted CS-Matching problem is to find the optimal assignment.

As CS-Matching is known to be NP-Complete [4], we consider approximation algorithms for the weighted version. If α and β are allowed to be arbitrary, we cannot have any constant factor approximation algorithm, unless $NP = P$. This is due to the fact that, one can reduce CS-Matching to the weighted version, by setting $\alpha = 0$ and $\beta = 1$. Now, any constant factor approximation algorithm can be used to solve the NP-Complete CS-Matching problem in polynomial time. Similarly, if we set $\alpha = 1$ and β to be a large value (like n^2), we would get the same result. So, it is interesting to consider the case when α and β are constants, with $\beta > \alpha > 0$. We discuss the case where $\alpha = 1$ and $\beta = 2$. Our results can be generalized for any constants α and β and the inapproximability bound will vary accordingly.

1,2-Matching is defined to be the weighted CS-Matching problem, where the matching cost is 1 and the mismatch cost is 2. We can define α, β-Matching similarly, for any constants, $\beta > \alpha > 0$. Let the length of the input text be $|T| = n$. Then any feasible assignment has a cost between n and $2n$. As we require that the input instance should have at least one feasible assignment, the

[2] In Section 3, we give a polynomial time algorithm to check whether a given instance of the problem has some feasible assignment.

optimal cost lies between n and $2n$. We define the difference between the cost of an assignment and n to be the *extra-cost* for that assignment.

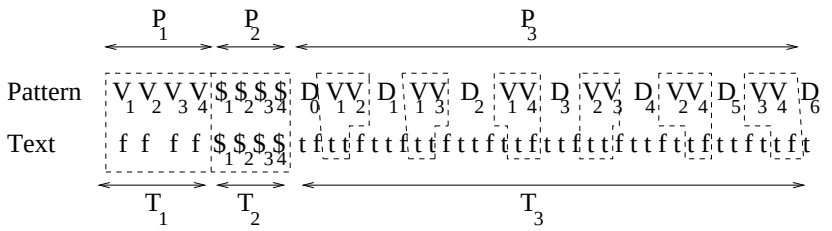
We next prove that the 1,2-Matching problem is MAXSNP-Complete. We first prove the result when the size of the alphabet is allowed to be arbitrary. Then we adapt the proof to get the same result, even over a binary alphabet. Note that over a unary alphabet, any feasible assignment is optimal. As a corollary, we get that the α, β-Matching problem is MAXSNP-Complete.

THEOREM **1** *The 1,2-Matching problem is MAXSNP-Complete.*

Proof : In Section 3, we will give a 2-approximation algorithm for the 1,2-Matching problem. It is known that a problem is in MAXSNP iff it has some constant factor approximation algorithm [16]. So the problem is in MAXSNP and we proceed to prove that it is MAXSNP-Hard.

The vertex cover problem on 3-regular graphs is known to be MAXSNP-Complete [1]. We give an L-reduction from the vertex cover problem on 3-regular graphs to our problem. As the first step, we present an algorithm that given a 3-regular graph produces an instance of our problem. Let G be the input graph with n vertices $v_1, v_2, \ldots, v_n$ and m edges $e_1, e_2, \ldots, e_m$. Note that, as the graph is 3-regular, $m = 1.5n$. We use the alphabet $\Sigma = \{\$_1, \$_2, \ldots, \$_n, t, f\}$. For each vertex v_i of the graph, we add a variable V_i to V, the variable set. We then add $m+1$ dummy variables $D_0, D_1, D_2, \ldots, D_m$ to V. A *dummy variable* is one that occurs exactly once in the pattern. The output pattern P has three parts, $P = P_1P_2P_3$. Here, $P_1 = V_1V_2 \ldots V_n$, $P_2 =$ "$\$_1\$_2 \ldots \$_n$" and $P_3 = D_0s_1D_1s_2D_2 \ldots D_{m-1}s_mD_m$ are called the vertex, dollar and edge segments respectively. In the edge-segment, s_i encodes the i^{th} edge. For example, if the i^{th} edge $e_i = (v_j, v_k)$, then $s_i = V_jV_k$. The text $T = T_1T_2T_3$ is also made of three segments. T_1 is the string "$fff...f$" of length n. T_2 is the same as $P_2 =$ "$\$_1\$_2 \ldots \$_n$". T_3 is made up of m *blocks* of the string "$tfttft$". This completes the construction. Refer to Figure 1 for an example, where the input graph is K_4, the complete graph on four nodes.

Let VC^* be an optimal vertex cover of G. We first have to show that the optimal assignment has cost, at most, $\alpha|VC^*|$, for some constant α. First note that the length of the text is $|T| = 11n$. We exhibit an assignment σ^* with cost $11n + |VC^*|$. For each vertex $v_i \in VC^*$, map the variable V_i to "t". For each vertex $v_i \notin VC^*$, map the variable V_i to "f". With this mapping the vertex segments of P and T, namely P_1 and T_1 get aligned with an extra-cost of $|VC^*|$. The dollar-segments P_2 and T_2 align with no extra-cost. We show how to map the dummy variables to strings so that there is no extra-cost in aligning the edge segments P_3 and T_3. Since the mapping is based on a vertex cover, for any edge $e_i = (v_j, v_k)$, the corresponding string V_jV_k has been mapped to one of "tt", "tf" or "ft". All these are available as substrings in "$fttf$" which occurs in each block of T_3. So we can make V_jV_k align with the text without any extra-cost, by making the dummies "eat" the "left-over" text in each block. Since the variables for each edge get mapped within "$fttf$" of a block "$tfttft$", each dummy variable will get mapped to a string of positive length. We exhibit this idea in Figure 1.



$$\sigma(V_1) = \sigma(V_2) = \sigma(V_3) = \text{“}t\text{”}; \sigma(V_4) = \text{“}f\text{”}; \sigma(D_0) = \text{“}tf\text{”}; \sigma(D_1) = \text{“}fttf\text{”};$$

$$\sigma(D_2) = \text{“}fttft\text{”}; \sigma(D_3) = \text{“}ttf\text{”}; \sigma(D_4) = \text{“}fttft\text{”}; \sigma(D_5) = \text{“}ttft\text{”}; \sigma(D_6) = \text{“}t\text{”}$$

Fig. 1. Constructing an assignment for the vertex cover $\{V_1, V_2, V_3\}$ of the graph K_4

The extra-cost for this assignment σ^* is $|VC^*|$ (the cost incurred in matching P_1 with T_1). We want to find a constant α such that, $cost(\sigma^*) \leq \alpha|VC^*|$. As G is a 3-regular graph, each vertex can cover at most 3 edges. Thus, $|VC^*| \geq n/2$. We already showed that $cost(\sigma^*) = 11n + |VC^*|$. Using these facts, with some simple arithmetic, we get that $cost(\sigma^*) \leq 23|VC^*|$.

We next give an algorithm that takes as input an assignment σ and outputs a vertex cover VC, with the property that, $|VC| - |VC^*| \leq cost(\sigma) - cost(\sigma^*)$, where VC^* is an optimal vertex cover and σ^* is an optimal assignment. To find VC, we first make certain transformations to σ to attain a feasible assignment with the following two properties:-

Property 1 : For each V_i, $\sigma(V_i) =$ “t” or $\sigma(V_i) =$ “f”.
Property 2: For each edge $e_i = (v_j, v_k)$, $\sigma(V_j)$ or $\sigma(V_k)$ is “t”.

We also ensure that we do not increase the cost of σ, in doing this transformation. Then, we include all the vertices that got mapped to “t” to VC.

If $|\sigma(V_i)| > 1$, P_2 will not align with T_2, incurring a minimum extra-cost of n. We can get an equally good assignment, by mapping each variable V_i to “t”. The mappings for the dummy variables can be changed appropriately so that P_3 exactly matches T_3. The extra cost for this assignment is n and we can take this to be σ.

We can now assume that, for all V_i, $|\sigma(V_i)| = 1$. If for some V_i, $\sigma(V_i) = \$_j$, then an extra-cost of 3 will be incurred for this variable. This is because, V_i appears three times in the edge-segment P_3 and the \$'s do not appears in T_3. So we can change $\sigma(V_i)$ to be “t” or “f”, by taking majority of the three text symbols to which V_i is aligned in the edge-segment. This change in mapping will actually decrease the cost. So, we can now assume that σ satisfies Property 1.

Next we convert σ to be block-respecting. In other words, for each edge $e_i = (v_j, v_k)$, the string $s_i = V_jV_k$ is aligned within the substring “$fttf$” of the i^{th} block of T_3. Let us consider the first edge $e_i = (v_j, v_k)$ that violates this condition. By shortening $\sigma(D_{i-1})$ and lengthening $\sigma(D_i)$ appropriately, we can make e_i to be block-respecting. This idea is expressed in Figure 2.

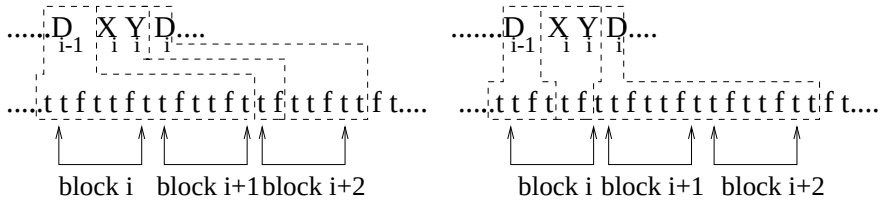


Fig. 2. Making the i^{th} edge block respecting.

By repeating this process, we can convert σ to be block-respecting without any additional cost.

Next we ensure that for each edge $e_i = (v_j, v_k)$, at least one of V_j or V_k is mapped to "t". If V_j and V_k are both mapped to "f", we pick one of them arbitrarily, say V_j, and map it to "t". This would increase the cost by 1, due to the mismatch in the vertex segment. We then change $\sigma(D_{i-1})$ appropriately, so that, V_jV_k is aligned to "tf" in the text. Previously, V_jV_k was mapped to "ff". Since T_3 does not contain the substring "ff", this would have caused a mismatch. By aligning V_jV_k to "tf", we remove this mismatch and reduce the cost by 1. This compensates the increase in cost in the vertex segment. We need to handle the other two places in the edge-segment where V_j occurs, without increasing the cost. As V_j is mapped to "t" now, the strings that we need to match can only be "tt", "tf" or "ft". All these are substrings of "$fttf$". They can be accommodated by adjusting the mapping to the appropriate dummy variable so that there is no increase in cost. So the total change in cost is 0.

We can now assume that σ satisfies Property 1 and Property 2. We construct a set VC, by adding the vertex v_i to it iff $\sigma(V_i) =$ "t". This would be a vertex cover. $cost(\sigma) = 11n + |VC|$. Cost of σ^* is, at least, $11n + VC^*$. Otherwise, we can use the above mentioned procedure to get a vertex cover smaller than VC^*. This proves that, $|VC| - |VC^*| \leq cost(\sigma) - cost(\sigma^*)$. We have proved that our reduction is an L-reduction. ♣

COROLLARY **1** *The 1,2-Matching problem cannot be approximated within a factor of 3313/3312.*

PROOF: It is known that the vertex cover problem is not approximable within a factor of 145/144, even on graphs with maximum degree at most three [12]. For ease of exposition, we gave the L-reduction from 3-regular graphs. The same proof works even for graphs with maximum degree at most three. In the above L-reduction we ensured that the cost of the optimal solution for the output instance is no more than 23 times the optimal vertex cover of the input graph. We also gave a way to translate a given solution σ of the matching problem into a vertex cover VC, such that $|VC| - |VC^*| \leq cost(\sigma) - cost(\sigma^*)$, where VC^* and σ^* are the corresponding optimal solutions. Using these, we can get the required inapproximability bound. ♣

We next adapt the above proof to show that even when the alphabet is restricted to be binary, the problem is MAXSNP-Complete.

THEOREM **2** *The 1,2-Matching problem is MAXSNP-Complete, even if the alphabet is binary.*

Proof: We adapt the proof of Theorem 1 here. We use the same L-reduction with some changes. Our alphabet is now $\{t, f\}$. The pattern is $P = P_1P_2P_3$, where P_1 and P_3 are same as in the original L-reduction. P_2 is now a string "$ff\ldots f$" of length $1.5n$. The text $T = T_1T_2\hat{T}_2T_3$ is now made of four segments. T_1 and T_3 are as in the original proof. T_2 is the string "$ff\ldots f$" of length $1.5n$. $\hat{T}_2$ = "$tt\ldots t$", a string of length n.

By adapting the original proof, we can translate an optimal vertex cover VC^* into an assignment with an extra-cost of $|VC^*|$ (note that D_0 can be mapped appropriately to match $\hat{T}_2$ also). As the length of the text is now $12.5n$, we can show that the cost of the optimal assignment is, at most, $26|VC^*|$.

We next modify the algorithm that converts a given assignment σ to a vertex cover VC. The main issue is that $|\sigma(V_i)|$ could be more than 1, for some V_i. We cannot use the argument given in the previous proof to address this issue, as we no longer have n different special symbols. We make a series of transformations to σ, so that $\forall i, |\sigma(V_i)| = 1$, without increasing the cost of σ.

We shall first transform σ so that the first "f" of P_2 is matched within T_2. If σ does not have this property, P_2 is matched completely within $\hat{T}_2$ and T_3. Notice that any substring of $\hat{T}_2T_3$ of length l has, at least, $2/3 * l$ "t"'s. So, P_2 will incur an extra-cost of at least, $|P_2| * 2/3 = n$. By mapping all V_i's to "t", we can get an equally good assignment (with extra-cost of n).

We can now assume that the first "f" of P_2 is matched within T_2. Now, if a suffix of P_2 is matched to a prefix of T_3, then the entire $\hat{T}_2$ will be matched within P_2, incurring an extra cost of n. So we can assume that P_2 is matched within T_2 and $\hat{T}_2$.

We shall now transform σ so that $|\sigma(V_i)| = 1$, for all V_i. Suppose $|\sigma(V_i)| > 1$, for some V_i. Let $\sigma(V_i) = ayb$, where $a, b \in \{t, f\}$ and $y \in (t + f)^*$. V_i occurs in three places in P_3, as the first variable or as the second (this would depend on how the edge was represented; as $(v_i, \_)$ or as $(\_, v_i)$). If the majority is the first place, we map V_i to "b", else we map it to "a". Such a change in the mapping has to be accounted for in four places, once in P_1 and thrice in P_3. First let us consider its occurrence in the vertex-segment P_1. We have shrunk $\sigma(V_i)$. The substring $V_{i+1}\ldots V_n$ of P_1 will slide to the left over T_1 and T_2 that are made of f's only. So, there is no change in cost for P_1. The sliding also causes one or more f's of P_2, which were originally matched with a prefix of $\hat{T}_2$, to now match with a suffix of T_2. As $\hat{T}_2$ is a string of t's and T_2 is a string of f's, the cost decreases by at least 1. To avoid "disturbing" the alignments in the edge-segments, we increase the length of $\sigma(D_0)$ by $|y|+1$ and the sliding has no effect in P_3. Now let us consider the three occurrences of V_i in P_3. Let us assume that V_i occurs a majority number of times as the first variable (the other case is handled similarly). Consider each of the three places where V_i occurs in the edge-segment. Let e_r be the edge in consideration. Then, we set $\sigma(D_{r-1}) \leftarrow \sigma(D_{r-1})ay$. This does not change the cost. If V_i is the second variable of e_r, we set $\sigma(D_r) \leftarrow yb\sigma(D_r)$. This could increase the cost by one, as a and b may be different. The latter case will occur at

most once, as we took majority. So we have made $|\sigma(V_i)| = 1$, without increasing the cost.

Repeating the above procedure for each variable, we have managed to change σ so that each V_i is either mapped to "t" or to "f". σ now satisfies Property 1 of the original proof. We can continue as in the original proof to satisfy Property 2 and get the required result. ♣

COROLLARY **2** *The 1,2-Matching problem cannot be approximated within a factor of 3313/3312, even when the alphabet is binary.*

PROOF: A straightforward adaptation of Corollary 1 to the above theorem would yield a bound of 3745/3744. This can be improved to 3313/3312 if we could reduce the length of the text from $12.5n$ to $11n$. We do this by using the block "$tfttf$" instead of "$tfttft$" in T_3 and adding a single constant t at the end of the text. The proof of the above theorem has to be modified slightly to handle this. ♣

COROLLARY **3** *The α, β-Matching problem is MAXSNP-Complete and cannot be approximated within a factor of* $1 + \frac{1}{144(\beta+21\alpha)(\beta-\alpha)}$.

3 Restrictions to Weighted CS-Matching

We showed that the 1,2-Matching problem is MAXSNP-Complete. In this section, we give a 2-approximation algorithm for the problem. We also consider whether restrictions of the problem have better upper bounds. The first restriction we consider is when the number of variables is bounded by a constant. We show that this problem can be solved in polynomial time. In the second restriction, we require that the pattern be a string made of variables only. We show that this problem is NP-Complete and also MAXSNP-Complete. We give an approximation algorithm for this problem, when the alphabet is bounded. The following lemma is useful in analyzing the two restrictions.

LEMMA **1** *We can verify whether a given instance of the weighted CS-Matching problem has a feasible assignment or not in polynomial time.*

PROOF: Let the input pattern contain k variables, $X_1, X_2, \ldots, X_k$, each occurring $a_1, a_2, \ldots, a_k$ times respectively. Let the length of the text be n, the total length of constants in the pattern be c and set $n' = n - c$. It can be seen that a feasible assignment exists iff the equation

$$a_1 * x_1 + a_2 * x_2 + \cdots + a_k * x_k = n'$$

has positive integer solutions in the unknowns $x_1, \ldots x_k$.

We give a simple algorithm based on dynamic programming to solve the above equation. We compute a $n' \times n'$ boolean array A, where $A_{p,q}$ is true iff there exists an assignment of positive integers to the x_i's, such that $\sum x_i = p$ and $\sum a_i x_i = q$. We set $A_{k,l}$ to true, where $l = \sum a_i$. We set all other entries of

the first k rows to false. To compute the rest of the array, we use the recurrence relation,

$$A_{p,q} = \bigvee_{i=1}^{k} A_{p-1,q-a_i}$$

Finally, there exists a positive integer solution to the equation iff for some p, $A_{p,n'}$ is true. ♣

The above algorithm can be modified to find a feasible assignment. Any feasible assignment has cost at most $2n$, where n is the length of the text. The optimal assignment has cost at least n. So we have a 2-approximation algorithm.

COROLLARY **4** *There is a 2-approximation algorithm for the 1,2-Matching problem.*

We next discuss a problem that occurs in the analysis of the restricted versions that we consider below.

Fixed Length Weighted CS-Matching Problem: As usual, the input consists of a text T of length n and a pattern P over k variables $V_1, V_2, \ldots, V_k$, where the variable V_i occurs a_i times in P. We are also given a sequence of k integers $l_1, l_2, \ldots, l_k$ satisfying the equation, $\sum a_i * l_i = n - c$, where c is the total length of the constants in P. Among all assignments σ that satisfy the condition, $\forall i \ |\sigma(V_i)| = l_i$, the goal is to find an assignment with the least cost.

The above problem can be solved in polynomial time. We can compute the required mapping for each variable V_i as follows. As the lengths of all variables are fixed, the substrings in the text that align with each occurrence of V_i are also fixed. Let these be $y_1, y_2, \ldots, y_{a_i}$, each of length l_i. We set $\sigma(V_i)$ to $a_1 a_2 \ldots a_{l_i}$, where a_j is a constant that occurs the maximum number of times in the j^{th} position of the strings $y_1, y_2, \ldots, y_{a_i}$. It can be seen that this assignment has the least cost of all assignments that satisfy the specified lengths.

3.1 Restriction 1: Bounded Number of Variables

THEOREM **3** *If the number of variables is bounded by a constant K, an optimal assignment can be found in polynomial time.*

PROOF: Finding the assignment σ can be viewed as a two stage process. We first fix the length of the mapping for each variable and then fix the actual mapping. Once we fix the lengths for $\sigma(V_i)$'s, we get an instance of the *fixed length weighted CS-Matching* problem, which was shown to be solvable in polynomial time. As each variable can only have length between 1 and n, the number of ways in which we can fix the lengths is bounded by n^K. So, when K is a constant, the problem can be solved in polynomial time. ♣

3.2 Restriction 2: Variable-Only Patterns

THEOREM **4** *The CS-Matching problem is NP-Complete even when the pattern is restricted to be a string of variables only and the alphabet is restricted to be binary.*

PROOF: We give a reduction from 1in3SAT. In the 1in3SAT problem, given a 3SAT formula, we need to check if there is a truth assignment that satisfies exactly one literal in each clause. This problem is known to be NP-Complete [9].

Let the input formula for the 1in3SAT problem be over n boolean variables $x_1, x_2, \ldots x_n$ and contain m clauses. Our reduction outputs an instance of the matching problem over the alphabet $\Sigma = \{\$, a\}$ and the variable set V. For each boolean variable x_i, we add two variables X_i and $\overline{X_i}$ to V. We also add another variable Z to V. We output the pattern

$$ZX_1\overline{X_1}ZX_2\overline{X_2}Z \ldots ZX_n\overline{X_n}Zs_1Zs_2Z \ldots ZS_mZ$$

where s_j is a string that encodes the j^{th} clause. For example, if the j^{th} clause is $(x_1 \vee \overline{x_2} \vee x_3)$ then s_j would be the string $X_1\overline{X_2}X_3$. The text is the string $\$(aaa\$)^n(aaaa\$)^m$. This defines the polynomial reduction. We next show that the formula has a 1in3 satisfying assignment iff the pattern matches the text.

First, if the input formula has a 1in3 satisfying assignment ϕ, we find a matching assignment σ as follows:

- Set $\sigma(Z) = \$$.
- If $\phi(X_i)$ is true, then set $\sigma(X_i) =$"aa" and $\sigma(\overline{X_i}) =$ "a".
- If $\phi(X_i)$ is false, then set $\sigma(X_i) =$ "a" and $\sigma(\overline{X_i}) =$ "aa".

It can be seen that σ is a matching assignment.

Next, let us assume that the pattern matches the text via a matching assignment σ. Then $\sigma(Z)$ has to start and end with \$. As the number of Z's in the pattern is same as the number of \$'s in the text, Z has to be mapped to \$. So, for any pair of variables X_i and $\overline{X_i}$ either X_i is mapped to "aa" and $\overline{X_i}$ to "a" or vice versa. We then exhibit a 1in3 satisfying assignment for the input formula. We set x_i to true, if X_i is mapped to "aa", and to false, otherwise. As the pattern and the text are matched, any string s_j is aligned with the string "$aaaa$". Thus, of the three variables in s_j, exactly one is mapped to "aa" and the other two are mapped to "a". This implies that the truth assignment we constructed is a 1in3 satisfying assignment. ♣

THEOREM **5** *The 1,2-Matching problem is MAXSNP-Complete even when the over Variable-Only patterns and binary alphabet.*

PROOF: We shall adapt the proof of Theorem 2 to achieve this result. The only changes we require in the original L-reduction are that, $P_2 = ZZ \ldots Z$ and $T_2 = ff \ldots f$, where both are of length $11n$. The same proof can be used to show that we can construct an assignment whose cost is, at most, 45 times $|VC^*|$. For the other direction of the L-reduction, we can use the same proof, if we can first ensure that Z is mapped to "f". In any feasible assignment, Z cannot be mapped to a string of length greater than 1, because Z occurs $11n$ times and the text is of length $22n$. Every substring of the text of length l has at least $l/3$ f's. So, if Z is mapped to "t", there is an extra-cost of $11n/3$. But, we can easily construct an assignment of extra-cost n. Hence we can assume that Z is mapped to "f" and proceed with the original proof. ♣

THEOREM **6** *For Variable-Only patterns, when the alphabet size is bounded by a constant D, we can approximate the 1,2-Matching problem within a factor of $1 + \frac{D-1}{D}$.*

PROOF: We first use the algorithm given in Lemma 1 to check if there is some feasible assignment. The algorithm can be modified to return the lengths of the assignment for each variable. We now have an instance of the fixed length weighted CS-Matching problem. We use the algorithm given for the latter problem to find the best assignment with these lengths. Since we use the majority rule in fixing the assignment, we are guaranteed to have at least one match for every $(D-1)$ mismatches. As the pattern has variables only, the constructed assignment has cost at most $n(1 + \frac{D-1}{D})$. Hence we have the required approximation bound. ♣

4 Conclusions and Open Problems

The problem of context sensitive string matching, which has practical applications, is known to be NP-Complete. We presented a weighted version of the problem and showed it to be MAXSNP-Complete, even over a binary alphabet. We also showed that the problem cannot be approximated within a factor of 3313/3312. We considered an interesting restriction where the pattern is made of variables only and showed that even under this restriction, the problem is NP-Complete and MAXSNP-Complete. We also gave an approximation algorithm for this case, when the alphabet size is bounded.

There are several open problems in this area. We gave an approximation algorithm in the case where the pattern is void of constants and the alphabet is bounded. Finding an approximation algorithm without either or both of these restrictions could be the next step. Designing improved approximation algorithms even with both the restrictions is another avenue for research. Obtaining better inapproximability bounds is another interesting area. Studying the problem where the pattern can be matched to a substring of the text has practical ramifications.

Acknowledgments

We thank Jin-Yi Cai and Christine Heitsch for useful discussions and comments.

References

1. P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proc. 3rd Italian Conf. on Algorithms and Complexity, Lecture Notes in Computer Science, 1203*, pages 288–298. Springer-Verlag, 1997.
2. A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. In *Proc. 38th IEEE Conf. on Foundations of Computer Science (FOCS)*, pages 144–153, 1997.

3. A. Amir, R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. Overlap matching. In *Proceeding of the Twelfth Annual Symposium on Discrete algorithms (SODA)*, pages 279–288, 2001.
4. D. Angluin. Finding patterns common to a set of strings. In *Journal of Computer and Systems Sciences*, volume 21, pages 46–62, 1980.
5. A. Apostolico and Z. Galil (eds.). *Pattern Matching Algorithms*. Oxford Univ. Press, 1997.
6. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford Univ. Press, 1994.
7. J. Dassow, Gh. Paun, and A. Salomaa. Grammars based on patterns. In *Intl. Journal on Foundations of Computer Science*, volume 4, pages 1–14, 1993.
8. M.J. Fischer and M.S. Paterson. String matching and other products. In *Complexity of Computation, SIAM-AMS Proceedings*, pages 7:113–125, 1974.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
10. C. Heitsch. *Computational Complexity of Generalized Pattern Matching*. Ph.D thesis, Dept. of Math., Univ. of California at Berkeley, 2000.
11. T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. In *Journal of Computer and Systems Sciences*, volume 50, pages 53–63, 1995.
12. M. Karpinski. Approximating bounded degree instances of NP-Hard problems. In *Electronic Colloquium on Computational Complexity, ECCC Report TR01-042*, 2001.
13. M. Lothaire. *Combinatorics on Words*. Addison-Wesley, 1983.
14. V. Mitrana, Gh. Paun, G. Rozenberg, and A. Salomaa. Patttern systems. In *Theoretical Computer Science*, volume 154, pages 183–201, 1996.
15. E. Ohlebusch and E. Ukkonen. On the equivalence problem for e-pattern languages. In *Theoretical Computer Science*, volume 186, pages 231–248, 1997.
16. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
17. A. I. Zimin. Blocking sets of terms. In *Math. Sbornik*, volume 119, pages 363–375, 1982.