(In)approximability Results for Pattern Matching Problems

Raphaël Clifford and Alexandru Popa

Department of Computer Science
University of Bristol
Merchant Venturer's Building
Woodland Road, Bristol, BS8 1UB
United Kingdom
{clifford,popa}@cs.bris.ac.uk

Abstract. We consider the approximability of three recently introduced pattern matching problems which have been shown to be **NP**-hard. Given two strings as input, the first problem is to find the longest common parameterised subsequence between two strings. The second is a maximisation variant of generalised function matching and the third is a a maximisation variant of generalised parameterised matching. We show that in all three cases there exists an $\epsilon > 0$ such that there is no polynomial time $(1-\epsilon)$-approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$. We then present a polynomial time $\sqrt{1/OPT}$-approximation algorithm for a variant of generalised parameterised matching for which no previous approximation results are known.

1 Introduction

We investigate the complexity and approximability of three recently introduced classes of pattern matching problems. Given two strings, typically termed the pattern and text, the traditional pattern matching question is to determine the minimum number of operations required to transform the pattern to either the whole or some portion of the text. The challenge arises in the specific detail of the definition of an operation or more generally in how the distance between two strings is measured. Popular examples have included pattern matching under the Hamming norm [1,9] and the edit distance [10,11] where efficient polynomial time algorithms are known. The algorithms are highly dependent on the distance measure being considered. Recent results have shown that when two or more different measures of distance are combined, the resulting problem may be **NP**-hard despite the individual measures sometimes permitting efficient linear time solutions [3,6].

The first problem we consider is known as longest common parameterised subsequence (LCPS). This combined problem is introduced by Keller et. al. in [8] and introduces the property of parameterisation into the classic and extensively studied problem of determining the longest common subsequence (LCS) between two strings. The term parameterisation, as introduced by Baker [5] in the pattern matching context, refers to the relabelling of the characters of the input so as to transform the pattern into a match for the text. A particular example given for LCPS is the practical question of comparing two code fragments, an original, and a suspected copy, after the alleged copy has been edited both by inserting new code or comments and also by possibly renaming variables. LCPS was previously shown to be **NP**-hard to solve exactly. We prove a stronger bound. We show that it is also hard to approximate within a $(1-\epsilon)$ factor, for some $\epsilon > 0$ unless $\mathbf{P} = \mathbf{NP}$. One consequence is that it is unlikely that any PTAS can be found.

Raphaël Clifford, Alexandru Popa: (In)approximability Results for Pattern Matching Problems, pp. 52–62 .
Proceedings of PSC 2010, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04597-8

We then consider a class of problems introduced in 2004 by Amir and Nor where individual characters in the pattern are permitted to match entire substrings of the text [3]. These are called generalised matching problems and for optimisation variants of both generalised function matching (GFM) and generalised parameterised matching (GPM) **NP**-hardness results are also known [6]. In generalised matching a character of the pattern can be mapped to a substring of the text. The main difference between GFM and GPM is that in the latter case, no two characters can map to the same substring. These problems arise from a natural extension of parameterised matching and function matching, which was first considered by Amir et al. [2] and can be applied to problems where one wants to determine the structure of a text in terms of repeated patterns. For example, if $t = annabobanna$ and $p = ABA$, then the mapping $A \rightarrow anna$ and $B \rightarrow bob$ is valid both in the GFM and GPM models. However, if $t = bobbobbob$ and $p = ABA$, then $A \rightarrow bob$ and $B \rightarrow bob$ is permitted in the GFM model, but not for GPM [6].

We show that GFM and GPM are also hard to approximate within a $(1 - \epsilon)$ factor unless $\mathbf{NP} = \mathbf{P}$. Finally, we present the first polynomial time $\sqrt{1/OPT}$-approximation algorithm for GPM, in the case when the length of the text is at least twice the length of the pattern.

2 Hardness of the longest common parameterised subsequence problem

The *longest common parameterised subsequence (LCPS)* problem attempts to combine the LCS measure with parameterised matching. The definition of the LCPS problem is the following:

Problem 1. [8] The input consists of two strings of the same length $t = t_1 t_2 \dots t_n$ and $p = p_1 p_2 \dots p_n$ over an alphabet Σ. The goal is to find a permutation $\pi : \Sigma \rightarrow \Sigma$ such that the LCS between $\pi(t_1)\pi(t_2)\dots\pi(t_n)$ and $p_1 p_2 \dots p_n$ is maximized.

The problem is introduced by Keller et. al. in [8] where they show it is **NP**-hard. In this paper we prove that the problem is hard to approximate within $(1 - \epsilon)$, for some $\epsilon > 0$.

Before we present the main result, we give the definition of gap-preserving reduction between two maximisation problems. A similar definition is presented in [12] for the case when the first problem is a minimisation problem and the second is a maximisation problem.

Definition 2. *Assume Π_1 and Π_2 are maximisation problems. A gap-preserving reduction from Π_1 to Π_2 comes with four parameters (functions) f_1, α, f_2 and β. Given an instance x of Π_1 it computes in polynomial time, an instance y of Π_2 such that:*

$$OPT(x) \geq f_1(x) \Rightarrow OPT(y) \geq f_2(y)$$

$$OPT(x) < \alpha(|x|) f_1(x) \Rightarrow OPT(y) < \beta(|y|) f_2(y)$$

In our proofs we make use of the following remark which is also stated in [12].

Remark 3. [12] A gap-preserving reduction from Π_1 to Π_2 with the above parameters implies that if problem Π_1 cannot have a polynomial-time α-approximation, then problem Π_2 cannot have a polynomial-time β-approximation.

To prove the inapproximability result for the LCPS problem we use a gap-preserving reduction from MAX-3SAT(13) to LCPS. MAX-3SAT(13) is a variant of MAX-SAT problem in which each clause has exactly three literals (where a literal is either a variable or its negation) and each variable appears in at most 13 clauses. In [4] Arora proves that MAX-3SAT(13) cannot be approximated within a factor of $1 - \delta/19$, if MAX3-SAT cannot be approximated within a factor of $1 - \delta$ [4]. In [7] Håstad proves that MAX3-SAT cannot be approximated within a factor better than 7/8 and thus, MAX-3SAT(13) cannot be approximated within a factor of 151/152, unless $\mathbf{P} = \mathbf{NP}$.

Consider a MAX-3SAT(13) instance ϕ with n variables and m clauses. The alphabet Σ of the LCPS instance consists of a special character \$ which has the role of a delimiter between different blocks of text (the usefulness of this character becomes clearer when we present the reduction) and two characters corresponding to each variable x of ϕ, x_T and x_F. We denote by $\$^{15}$ the string formed by concatenating 15 \$ symbols.

The reduction is the following. For each variable x we add to the text t the gadget $x_T x_F x_T x_F \$^{15}$ and to the pattern p the gadget $x_T x_F x_F x_T \$^{15}$. Then, for each clause $x_1 \vee x_2 \vee x_3$ we add to the pattern the gadget $x_{3T} x_{2T} x_{1T} \$^{15}$ (notice that the variables are placed in reverse order), and to t, the gadget $x_{1V} x_{2V} x_{3V} \$^{15}$, where x_{iV}, for every $i = \{1, 2, 3\}$ is:

$$x_{iV} = \begin{cases} x_{iT} \text{ if } x_i \text{ is a non-negated variable} \\ x_{iF} \text{ if } x_i \text{ is a negated variable} \end{cases}$$

Example 4. Consider the formula:

$$\phi = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z)$$

The corresponding instance of the LCPS problem is:

$$t = x_T x_F x_T x_F \$^{15} y_T y_F y_T y_F \$^{15} z_T z_F z_T z_F \$^{15} x_T y_T z_T \$^{15} x_T y_F z_T \$^{15}$$
$$p = x_T x_F x_F x_T \$^{15} y_T y_F y_F y_T \$^{15} z_T z_F z_F z_T \$^{15} z_T y_T x_T \$^{15} z_T y_T x_T \$^{15}$$

Theorem 5 summarises the main inapproximability result for LCPS.

Theorem 5. *There exists an $\epsilon > 0$ such that the LCPS problem does not have a $(1 - \epsilon)$-approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$.*

Proof. We have to prove that this is a gap-preserving reduction from MAX-3SAT(13). Specifically, we prove that there exists a constant $\sigma > 0$ such that:

$$OPT(\phi) = m \Rightarrow OPT(t, p) = f(|t|, |p|) \quad (1)$$
$$OPT(\phi) < (1 - \epsilon)m \Rightarrow OPT(t, p) < (1 - \sigma) f(|t|, |p|) \quad (2)$$

where $OPT(\phi)$ is the optimal value of MAX-3SAT(13) problem on the instance ϕ, $OPT(t, p)$ is the maximum LCPS on the text t, and pattern p constructed using the above reduction, and $f(|t|, |p|) = 3n + 15(n + m) + m$. These implications, together with Remark 3 prove the theorem.

Now, we explain why the two implications are true. The reduction forces the \$ symbol to be matched with itself under the permutation of the alphabet. Suppose that the \$ symbol is matched with another character, say x. In this case the LCPS can be at most $4n + 3m$ (this is the number of the characters in the text and in the

pattern which are different from \$), assuming that we can ideally match everything else. However a longer LCPS can be achieved by simply matching the \$ symbols alone.

The mapping \$ to \$ imposes a lot of structure on the other possible mappings: each pair (x_T, x_F) is either matched to (x_T, x_F), meaning that the variable x is assigned to *True* in the formula ϕ, or is matched to (x_F, x_T), meaning that the variable x is assigned to *False*. Since each variable appears in at most 13 clauses it is not optimal to match x_T or x_F with a character corresponding to any other variable y. This matching stops a block of 15 \$ symbols from the text to match a block of 15 \$ symbols from the pattern, but it can add no more than 13 to the LCPS length.

Thus, each variable gadget adds 3 to the final LCPS. Then, each block of dollars adds 15 to the final LCPS, independent of the other matchings. Finally, a gadget corresponding to a satisfied clause contributes exactly one to the LCPS. It cannot contribute more than one, if more than one literal is true in that clause, since we have placed the literals in reverse order in the pattern.

Therefore, if all the m clauses of ϕ can be satisfied, then the LCPS length has to be equal to $3n + 15(m + n) + m$. On the other hand, if at most $1 - \epsilon$ clauses of ϕ can be satisfied, then LCPS is at most $3n + 15(m + n) + (1 - \epsilon)m$, since the best strategy is to match a character according to the assignment of the corresponding variable: if a variable x is set to true in the assignment that maximises the number of satisfied clauses, then $\pi(x_T) = x_T$; otherwise $\pi(x_T) = x_F$.

To complete the proof of the second implication, we now explicitly calculate the value of σ. Since the optimal value of the LCPS is less than $3n+15(m+n)+(1-\epsilon)m$, we want the latter to be equal to $(1-\sigma)(3n+15(m+n)+m)$. By solving this equation we find the value of σ. Formally:

$$3n + 15(m + n) + (1 - \epsilon)m = (1 - \sigma)(3n + 15(m + n) + m)$$

Therefore,

$$\sigma = 1 - \frac{3n + 15(m + n) + (1 - \epsilon)m}{3n + 15(m + n) + m} = \frac{\epsilon m}{3n + 15(m + n) + m}$$

Since each variable appears in at most 13 clauses we can set $m = 13n$ and we get $\sigma = 13\epsilon/76$.

Therefore, the LCPS problem cannot be approximated within a factor of 0.99887, unless $\mathbf{P} = \mathbf{NP}$. □

3 Hardness of Generalised Function Matching

The two problems we consider in this section are termed generalised function matching (GFM) and generalised parameterised matching (GPM). Their formal definitions follow:

Problem 6. [6](GFM) Given a pattern p over an alphabet Σ_p and a text t over an alphabet Σ_t, determine if there exists a mapping f from Σ_p to Σ_t^+ such that $t = f(p_1)f(p_2)\ldots f(p_m)$.

Problem 7. [6](GPM) The problem of generalised parameterised matching (GPM) is defined in an analogous way to GFM except that f is now required to be an injection.

Recently, Clifford et. al. [6] prove that the two problems are **NP**-Complete under a wide range of conditions. They also define an optimisation version of the GFM problem, which is connected to the classical Hamming distance.

The Hamming similarity between two strings of the same length is the number of positions in which the two strings are equal. For input text t and pattern p, we are interested in the maximum Hamming similarity between p and any string p' of the same length which has a GFM match with t. As the original GFM problem is **NP**-Complete, this optimisation problem is **NP**-Hard. In the rest of the paper we refer to the "Hamming similarity", simply as "similarity".

To simplify the description of our approximation algorithms, we introduce the idea of a wildcard symbol which we define here.

Definition 8. *A* wildcard *is a special character which can be mapped to any substring of the text.*

We can replace the wildcard characters with regular characters as follows. In the GFM, just replace the wildcards with distinct characters. In the GPM, we partition the pattern alphabet into groups by which substring of the text they are mapped to. Each partition has only one non-wildcard character in it. Replace all the wildcard characters in a partition by the single non-wildcard character that are in the same partition.

We now show that there exists an $\epsilon > 0$ such that the problem of generalised function matching does not have a $(1-\epsilon)$-approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$, via a gap preserving reduction from MAX3-SAT(13).

Consider a MAX-3SAT(13) instance ϕ with n variables and m clauses. The construction starts by placing $m+n$ \$ symbols to the beginning of both the pattern and the text. The text alphabet Σ_t has only two symbols, \$ and 0 with \$ serving as a delimiter in both the pattern and text. The pattern alphabet Σ_p includes the delimiter, a pair a_i and A_i for each variable and a distinct symbol c_i for each clause. The A_i's represent the negation of the variables a_i. The constructed pattern and text contain an equal number of \$ characters which forces \$ to map to \$ under any valid function. Also, as we show later, replacing some of the \$ symbols with wildcards cannot yield to an optimal strategy. We prove that the minimal Hamming distance between the pattern and the text is achieved when exactly one variable from each unsatisfied clause is replaced by a wildcard, where a wildcard, as we mention before, is a special character which is allowed to match any non-empty substring of the text.

For each variable a_i, we add to the text the string $\$^{13}000\$^{13}$ 13 times and to the pattern $\$^{13}a_iA_i\$^{13}$ 13 times. In this way a variable can be mapped to 00 or 0. Replacing all the 13 variables with wildcards is not optimal since a variable appears in at most 13 clauses and, therefore, at most 13 clauses can be satisfied. To fix notation we say that 0 represents True and 00 represents False. For each clause, we add to the text the string $\$^{13}000000\$^{13}$ (6 zeros) and to the pattern the string $\$^{13}xyzc_i\$^{13}$ where x, y, z are the variables from the clause (or their negations, as appropriate) and c_i is a different symbol for each clause.

Example 9. Consider the following instance of MAX3-SAT(13),

$$(x \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \ .$$

The GFM input is the following:

$$t = \$\$\$\$\$ \overbrace{\$^{13}000\$^{13} \ldots \$^{13}000\$^{13}}^{13} \overbrace{\$^{13}000\$^{13} \ldots \$^{13}000\$^{13}}^{13}$$

$$\overbrace{\$^{13}000\$^{13}\ldots\$^{13}000\$^{13}}^{13}\$^{13}000000\$^{13}\$^{13}000000\$^{13}$$

$$p = \$\$\$\$\$\overbrace{\$^{13}xX\$^{13}\ldots\$^{13}xX\$^{13}}^{13}\overbrace{\$^{13}yY\$^{13}\ldots\$^{13}yY\$^{13}}^{13}$$
$$\overbrace{\$^{13}zZ\$^{13}\ldots\$^{13}zZ\$^{13}}^{13}\$^{13}xyzc_1\$^{13}\$^{13}XyZc_2\$^{13}$$

The inapproximability result for the GFM problem is stated in Theorem 10.

Theorem 10. *There exists an $\epsilon > 0$ such that the GFM problem does not have a $(1-\epsilon)$-approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$.*

Proof. We have to prove that this is a gap-preserving reduction from MAX-3SAT(13). Specifically, we prove that there exists a constant $\sigma > 0$ such that conditions (1) and (2) from the proof of Theorem 5 are satisfied, where $OPT(\phi)$ is the optimal value of the MAX-3SAT(13) problem on the instance ϕ, $OPT(t,p)$ is the maximum similarity on the text t, and the pattern p constructed using the above reduction, and $f(|t|,|p|) = (n+m) + 26 \cdot 13n + 26m + 2 \cdot 13n + 4m$. These implications, together with Remark 3 prove the theorem.

Now, we explain why the two implications are true. If ϕ is satisfiable, then there must be a GFM solution for p and t. This follows as we are guaranteed that not all the symbols from a clause can be mapped to 00 and therefore c_i can be matched to a nonempty substring in each clause gadget.

Now, suppose that at most $(1-\epsilon)m$ clauses of ϕ can be satisfied. Then, we must have at least ϵm wildcards in order to have a valid matching, one wildcard for each unsatisfied clause. This wildcard replaces one of the variables in that clause and matches only one 0 (notice that since this is an unsatisfied clause all the variables are matched to 00 and the dummy variable c_i does not have any 0's to be matched to) and gives the possibility of the last variable to be matched to a 0.

If one replaces all the dollar signs from the beginning with a wildcard the number of wildcards used is $n+m$, which is not optimal. If you replace with wildcards a block of 13 \$'s which separates the variable gadgets, then you can satisfy at most 13 new clauses, at a price of 13 wildcard symbols, which is not an improvement to the strategy presented in the previous paragraph. If you replace with wildcards a block of 13 \$'s which separates the clause gadgets, then you can satisfy at most 2 new clauses. The last option is to try to allow a variable to have an inconsistent assignment in order to satisfy more clauses, but to do this you need to place at least 13 wildcards and since a variable appears in at most 13 clauses, then this is not optimal.

Therefore, if at most $(1-\epsilon)m$ clauses of ϕ can be satisfied, then the maximum similarity has to be less or equal to $(n+m) + 26 \cdot 13n + 26m + 2 \cdot 13n + 4(1-\epsilon)m$. We want:

$$(n+m) + 26 \cdot 13n + 26m + 2 \cdot 13n + 4(1-\epsilon)m =$$
$$(1-\sigma)((n+m) + 26 \cdot 13n + 26m + 2 \cdot 13n + 4m)$$

As before, we can set $m = 13n$ and, therefore:

$$(1-\sigma) = \frac{716 + 52(1-\epsilon)}{768}$$

Therefore, the GFM problem cannot be approximated within a factor of 0.99955, unless $\mathbf{P} = \mathbf{NP}$. □

4 Hardness of Generalised Parameterised Matching

In this section we present an inapproximability result for a maximum similarity variant of GPM. In this variant, for input text t and pattern p, we are interested in the maximum similarity between p and any string p' of the same length which has a GPM match with t.

We now show that there exists an $\epsilon > 0$ such that the problem of generalised parameterised matching does not have a $(1-\epsilon)$-approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$, via another gap preserving reduction from MAX3-SAT(13).

Consider a MAX-3SAT(13) instance ϕ with n variables and m clauses. Fix an ordering of the variables and for a variable x define $L(x)$ to be the position of this variable according to this ordering. Fix also an ordering of the clauses and for a clause c we define $L(c)$ to be the position of the clause according to this ordering (we overload the notation from the variables and it should be clear from the context to which one we are referring to).

The text alphabet Σ_t has $n + m + 1$ symbols, \$ and 1, 2, ..., $n + m$, with \$ serving as a delimiter in both the pattern and text. The pattern alphabet Σ_p includes the delimiter \$, two characters x, X for each variable x and two characters for each clause c, $w_{L(c)}$ and $w'_{L(c)}$. X represents the negation of the variable x. The constructed pattern and text contain an equal number of \$ characters which forces \$ to map to \$ under any valid injective function. Also, as we show later, replacing some of the \$ symbols with wildcards cannot yield to an optimal strategy.

The construction starts by placing $m + n$ \$ symbols to the beginning of both the pattern and the text. For a variable x, we add to the text the string $\$^{13}L(x)L(x)L(x)\$^{13}$ 13 times and to the pattern $\$^{13}xX\$^{13}$ 13 times. In this way the variable x can be mapped either to $L(x)$ or $L(x)L(x)$. Replacing all the 13 variables with wildcards is not optimal since a variable appears in at most 13 clauses and, therefore, at most 13 clauses can be satisfied. To fix notation we say that for a variable x, $L(x)$ represents True and $L(x)L(x)$ represents False.

For the clauses we have the following construction. Consider that a clause c has literals x, y, z. Then we add to the text the string $\$^{13}L(x)L(x)n + L(c)L(y)L(y)n + L(c)L(z)L(z)n + L(c)\$^{13}$ and to the pattern the string $\$^{13}xw_{L(c)}yw_{L(c)}zw_{L(c)}\$^{13}$ where x, y, z are the variables from the clause, or their negations, as appropriate.

In the end we add for each clause the string $\$^{13}n + L(c)\$^{13}$ 13 times to the text and the string $\$^{13}w'_{L(c)}\$^{13}$ 13 times to the pattern. In this way the character $w'_{L(c)}$ is forced to match with $n + L(c)$ and no other characters can match $L(c)$. Also, if we want to make other characters to match $L(c)$ by replacing $w'_{L(c)}$ with wildcards, then the cost is too high (at least 13).

Example 11. Consider the following instance of MAX3-SAT(13),

$$(x \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \ .$$

The GPM input is the following:

$$t = \$\$\$\$\overbrace{\$^{13}111\$^{13}\ldots\$^{13}111\$^{13}}^{13}\overbrace{\$^{13}222\$^{13}\ldots\$^{13}222\$^{13}}^{13}$$

$$\overbrace{\$^{13}333\$^{13}\ldots\$^{13}333\$^{13}}^{13}\$^{13}114224334\ \$^{13}115225335\$^{13}\overbrace{\$^{13}4\$^{13}}^{13}\overbrace{\$^{13}5\$^{13}}^{13}$$

$$p = \$\$\$\$\$\overbrace{\$^{13}xX\$^{13}\ldots\$^{13}xX\$^{13}}^{13}\overbrace{\$^{13}yY\$^{13}\ldots\$^{13}yY\$^{13}}^{13}$$

$$\overbrace{\$^{13}zZ\$^{13}\ldots\$^{13}zZ\$^{13}}^{13}\$^{13}xw_1yw_1zw_1\$^{13}\$^{13}Xw_2yw_2Zw_2\$^{13}$$

$$\overbrace{\$^{13}w'_1\$^{13}}^{13}\overbrace{\$^{13}w'_2\$^{13}}^{13}$$

The inapproximability result for the GPM problem is stated in Theorem 12.

Theorem 12. *There exists an $\epsilon > 0$ such that the GPM problem does not have a $(1-\epsilon)$-approximation algorithm, unless* $\mathbf{P} = \mathbf{NP}$.

Proof. We have to prove that this is a gap-preserving reduction from MAX-3SAT(13). Specifically, we prove that there exists a constant $\sigma > 0$ such that conditions (1) and (2) are satisfied, where $OPT(\phi)$ is the optimal value of the MAX-3SAT(13) problem on the instance ϕ, $OPT(t,p)$ is the maximum similarity on the text t, and the pattern p constructed using the above reduction, and $f(|t|,|p|) = (n+m) + 26 \cdot 13n + 26m + 26 \cdot 13m + 2 \cdot 13n + 13m + 4m$. These implications, together with Remark 3 prove the theorem.

Now, we explain why the two implications are true. We first prove that if ϕ is satisfiable, then there must be a GPM solution for p and t where exactly $f(|t|,|p|)$ characters are not replaced with wildcards. We describe such a matching. The \$ symbol from the pattern matches the \$ symbol from the text and $w'_{L(c)}$ matches $L(c)$ for any clause c. If a variable x is assigned to True, then the character x matches $L(x)$ and X matches $L(x)L(x)$. Otherwise, if a variable x is assigned to False, then the character x matches $L(x)L(x)$ and X matches $L(x)$. Then, in every satisfied clause c, the character $w_{L(c)}$ after the True literal (we choose one literal arbitrarily if there are more than one True) is left unchanged an the other two are replaced with an wildcard. Since all the clauses are satisfied, in a clause gadget four characters are not replaced with a wildcard.

Now, suppose that at most $(1-\epsilon)m$ clauses of ϕ can be satisfied. Then, we must have at least $3\epsilon m$ wildcards in order to have a valid matching, three wildcards for each unsatisfied clause. These wildcards replace all three $w_L(c)$ characters in an unsatisfied clause c.

If we replace all the dollar signs from the beginning with a wildcard the number of wildcards used is $n+m$, which is not optimal. If we replace with wildcards a block of 13 \$'s which separates the variable gadgets, then you can satisfy at most 13 new clauses, at a price of 13 wildcard symbols, which is not an improvement to the strategy presented in the previous paragraph. If we replace with wildcards a block of 13 \$'s which separates the clause gadgets, then you can satisfy at most 2 new clauses. If we replace the characters w' with wildcards, then we have to use 13 wildcards to satisfy one clause and again the cost is too high. The last option is to try to allow a variable to have an inconsistent assignment in order to satisfy more clauses, but to do this we need to place at least 13 wildcards. Since a variable appears in at most 13 clauses, this, again, is not optimal.

Therefore, if at most $(1-\epsilon)m$ clauses of ϕ can be satisfied, then the maximum similarity has to be less or equal to $(n+m)+26\cdot 13n+26m+26\cdot 13m+2\cdot 13n+13m+(1-\epsilon)4m+3\epsilon m$. We want:

$$(n+m)+26\cdot 13n+26m+26\cdot 13m+2\cdot 13n+13m+(1-\epsilon)4m+3\epsilon m = (1-\sigma)((n+m)+26\cdot 13n+26m+26\cdot 13m+2\cdot 13n+13m+4m)$$

As before, we can set $m=13n$ and, therefore:

$$(1-\sigma)=\frac{5331-13\epsilon}{5331}$$

Therefore, the GPM problem cannot be approximated within a factor of 0.999983, unless $\mathbf{P}=\mathbf{NP}$. □

5 A $\sqrt{1/OPT}$-approximation algorithm for Generalised Parameterised Matching

In this section we present a $\sqrt{1/OPT}$-approximation algorithm for the maximum similarity for GPM in a special case where the text is at least twice as long as the pattern, where OPT is the maximum similarity between the pattern and any string p' of the same length which has a GPM match with the text.

In [6] Clifford et. al. present a $\sqrt{k/OPT}$-approximation algorithm for the maximum similarity for GFM, for any fixed k. Unlike GFM, for the maximum similarity for GPM, no approximation algorithms are known.

Informally, the $\sqrt{1/OPT}$-approximation algorithm works as follows. We define the length of the text t to be n and the length of the pattern to be m. We divide the input instances in two cases, which are treated separately: if the pattern alphabet $|\Sigma_p| \geq \sqrt{m}$ or if it is less than $\sqrt{m}$. The entire procedure is described by Algorithm 1.

Algorithm 1 A $\sqrt{1/OPT}$ approximation for maximum GPM similarity

Input: A pattern $p=p_1p_2\ldots p_m$ over the alphabet Σ_p and a text $t=t_1t_2\ldots t_n$ over the alphabet Σ_t.

1. If $|\Sigma_p| \geq \sqrt{m}$, then choose a set S of $\sqrt{m}$ distinct characters from the pattern p and find a generalised matching using Algorithm 2 and *output* it.
2. If $|\Sigma_p| < \sqrt{m}$, then:
   (a) iterate over each character $c \in \Sigma_p$ and each substring s of t;
      i. create a new pattern $p^{c,s}$ from p by replacing every character different from c with a wildcard;
      ii. compute the maximum similarity between $p^{c,s}$ and t using Algorithm 3.
   (b) *output* the pattern $p^{c,s}$ which has the highest maximum GPM similarity with t.

In the first case, select a set S of size $\sqrt{m}$ distinct characters from p, which also contains p_m, the last character of the pattern, and process p from left to right. We construct a new pattern that has similarity at least $\sqrt{m}$ to p. If p_i is not in S, then just change it to the character in t to which it is currently aligned. If it is in S, then leave it unchanged but skip $2,3,4,\ldots$ places in the text. The mapping is from characters to themselves for those positions that are not in S and from characters to substrings of length $2,3,4,\ldots$. for those in S. Algorithm 2 presents this process.

Algorithm 2 Computes a generalised parameterised matching of a pattern p, which has at least $\sqrt{m}$ distinct characters, with the text t

Input: A pattern $p = p_1 p_2 \dots p_m$ over the alphabet Σ_p, a set S of exactly $\sqrt{m}$ distinct characters from p and a text $t = t_1 t_2 \dots t_n$ over the alphabet Σ_t.

1. $j := 1$; $k := 2$
2. for $i = 1$ to m do:
   (a) If $i = m$ match p_m with $t_j t_{j+1} \dots t_n$. *return*;
   (b) If $p_i \notin S$, then change it to character t_j and match it with t_j. Set $j := j + 1$;
   (c) If $p_i \in S$, then match it with the string $t_j t_{j+1} \dots t_{j+k-1}$. Set $j := j + k$; $k := k + 1$

Output: The matching between p and t.

In the latter case we fix one character in turn and we replace everything else by a wildcard. Then, we solve the problem independently for each substring s of the text t and then we choose the substring for which the similarity is the highest. The similarity is computed using dynamic programming (Algorithm 3). We define the function $f(i, j)$ to be the best solution to the problem for $t_1 t_2 \dots t_j$ and $p_1 p_2 \dots p_i$. When the algorithm finishes, the maximum similarity is stored in $f(m, n)$.

We finally output the pattern $p_{c,s}$ with the maximum similarity over all characters and all substrings.

Algorithm 3 Computes maximum GPM similarity of a pattern $p_{c,s}$ with the text t

Input: A pattern $p^{c,s} = p_1^{c,s} p_2^{c,s} \dots p_m^{c,s}$ over the alphabet Σ_p with exactly one non-wildcard character and a text $t = t_1 t_2 \dots t_n$ over the alphabet Σ_t.

$$f(i,j) = \begin{cases} 0, & \text{if } i = 0 \text{ or } i > j \\ \max\{f(i-1,j-1), f(i-1,j-2), \dots, f(i-1,i-1)\}, & \text{if } p_i^{c,s} \neq c \\ \max_k\{f(i-1,j-k) + I(t_{j-k+1} \dots t_j = s)\}, & \text{if } p_i^{c,s} = c \end{cases}$$

Output: $f(m, n)$ - the maximum GPM similarity between $p_{c,s}$ and t.

In order to prove the correctness of Algorithm 1 we need the following lemmas.

Lemma 13. *If $|t| \geq 2|p|$ Algorithm 2 computes a GPM match between p and t.*

Proof. The number of characters in S is $\sqrt{m}$. Each character which is not in S matches only one character of the text and therefore the number of characters from the text used is $m - \sqrt{m}$. Characters from S use $2 + 3 + \dots + \sqrt{m} + 1$ characters of the text. Therefore, the total number of characters of the text used in the matching is $(\sqrt{m}+1)(\sqrt{m}+2)/2-1+m-\sqrt{m}$. Since $n \geq 2m$, $(\sqrt{m}+1)(\sqrt{m}+2)/2-1+m-\sqrt{m} \leq n$. The matching is valid (i.e. all the characters are mapped injectively) since the strings that are mapped to characters from S have different length. □

Lemma 14. *Algorithm 3 computes the GPM similarity between $p_c^{c,s}$ and t.*

Proof. We must first show that the dynamic programming procedure computes the right function and then that it runs in polynomial time. We can see immediately that $f(0, i) = 0\ \forall i$ because in this case the pattern is empty. Also, $f(i, j) = 0\ \forall i > j$ because every character of the pattern must map at least one character from the text, even if it is replaced by a wildcard. The computation of $f(i, j)$ has two cases.

- $p_i^{c,s} \neq c$. In this case we cannot increase the number of characters in our set that can be mapped. However we know that $p_i^{c,s}$ is set to a wildcard and therefore we find the maximum of the previous results for different length substrings that the wildcard maps to.

– $p_i^{c,s} = c$. We can either map $p_i^{c,s}$ to s and increase the number of mapped characters by one, which can only happen if $t_{j-|s_z|+1} \dots t_j = s$ or we do the same as in the previous case. □

We are now prepared to prove the main result of this section.

Theorem 15. *Algorithm 1 is a $\sqrt{1/OPT}$-approximation algorithm if the length of t is at least twice the length of p.*

Proof. Let M be the maximum GPM-similarity over all $p^{c,s}$. We know that $OPT \leq M \cdot |\Sigma_p|$ since M is the maximum over all characters of p. Therefore, either M or $|\Sigma_p|$ have to be greater than or equal to $\sqrt{OPT}$. The total running time of the algorithm is polynomial in n and m. □

Acknowledgments. The second author is funded by an EPSRC PhD studentship.

References

1. K. R. Abrahamson: *Generalized string matching.* SIAM journal on Computing, 16(6) 1987, pp. 1039–1051.
2. A. Amir, A. Aumann, R. Cole, M. Lewenstein, and E. Porat: *Function matching: Algorithms, applications, and a lower bound*, in Proceedings of the 30*th* International Colloquium on Automata, Languages and Programming (ICALP), 2003, pp. 929–942.
3. A. Amir and I. Nor: *Generalized function matching*, in Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC), 2004, pp. 41–52.
4. S. Arora: *Probabilistic checking of proofs and the hardness of approximation problems*, PhD thesis, UC Berkeley, 1994.
5. B. S. Baker: *A theory of parameterized pattern matching: algorithms and applications*, in Proceedings of the 25*th* Annual ACM Symposium on the Theory of Computing (STOC), 1993, pp. 71–80.
6. R. Clifford, A. W. Harrow, A. Popa, and B. Sach: *Generalised matching*, in Proceedings of the 16th International Symposium on String Processing and Information Retrieval (SPIRE), 2009, pp. 295–301.
7. J. Håstad: *Some optimal inapproximability results.* J. ACM, 48(4) 2001, pp. 798–859.
8. O. Keller, T. Kopelowitz, and M. Lewenstein: *On the longest common parameterized subsequence.* Theoretical Computer Science, 410(51) 2009, pp. 5347–5353.
9. S. R. Kosaraju: *Efficient string matching.* Manuscript, 1987.
10. V. Levenshtein: *Binary codes capable of correcting spurious insertions and deletions of ones.* Problems of Information Transmission, 1 1965, pp. 8–17.
11. V. Levenshtein: *Binary codes capable of correcting insertions and reversals.* Soviet Physics Doklady, 10(8) 1966, pp. 707–710.
12. V. V. Vazirani: *Approximation Algorithms*, Springer, 2004.