

Generalised matching

Raphael Clifford¹, Aram W. Harrow²,
Alexandru Popa¹, and Benjamin Sach¹

¹ Department of Computer Science, University of Bristol, UK
{clifford, popa, sach}@cs.bris.ac.uk

² Department of Mathematics, University of Bristol, UK
a.harrow@bris.ac.uk

Abstract. Given a pattern p over an alphabet Σ_p and a text t over an alphabet Σ_t , we consider the problem of determining a mapping f from Σ_p to Σ_t^+ such that $t = f(p_1)f(p_2)\dots f(p_m)$. This class of problems, which was first introduced by Amir and Nor in 2004, is defined by different constraints on the mapping f . We give NP-Completeness results for a wide range of conditions. These include when f is either many-to-one or one-to-one, when Σ_t is binary and when the range of f is limited to strings of constant length. We then introduce a related problem we term *pattern matching with string classes* which we show to be solvable efficiently. Finally, we discuss an optimisation variant of generalised matching and give a polynomial-time $\min(1, \sqrt{k/\text{OPT}})$ -approximation algorithm for fixed k .

1 Introduction

We consider a class of pattern matching problems where individual characters in the pattern are permitted to match entire substrings of the text. When the substrings are restricted to have length exactly one, then the problems of finding efficient search algorithms are exactly those typically found in the rich and successful literature of combinatorial pattern matching. Our interest here lies when the substrings may have length greater than one. In many cases, it has not been known up to this point even whether polynomial-time pattern matching algorithms exist.

The problems we analyse take the following general form. The input is a pattern $p = p_1p_2\dots p_m$ and a text $t = t_1t_2\dots t_n$. We wish to find a mapping f from Σ_p to Σ_t^+ (substrings of t of length at least one) so that $t = f(p_1)f(p_2)\dots f(p_m) = f(p)$. For example, if $p = aba$ and $t = xyxx$ then if f maps $a \rightarrow x$, $b \rightarrow yy$, we say that there is a match. However if $t = xyxz$, then no such mapping can be found. The first results in this model were given by Amir and Nor [2, 3] who considered a problem called generalised function matching (GFM) with wildcards. Here single

character wildcards may occur in the input and f is allowed to be any function. They show that if the pattern alphabet has constant size, then a polynomial algorithm can be found but that the problem is NP-Complete otherwise. The problem of generalised parameterised matching (GPM) is also defined in an analogous way to GFM except that f is now required to be an injection. That is if $p = aba$ and $t = xxx$ then there is a GFM match but not a GPM match as a and b cannot both map to x in a generalised parameterised match. As the names suggest, these problems arise from a natural extension of parameterised matching, introduced by Baker [4] in 1993, and function matching, which was considered by Amir et al. [1].

Our contributions are twofold. Primarily, we answer a number of open problems posed by Amir and Nor in their initial work. Namely, we prove that both GFM and GPM are NP-Complete with or without wildcards (Sections 2 and 3) and give a $\min(1, \sqrt{k/\text{OPT}})$ -approximation for a Hamming distance based variant of GFM for fixed k (Section 5). We also extend the work to include a new variant of generalised matching based on pattern matching with string classes (Section 4). Due to space constraints, some of the proofs are omitted.

2 GFM is NP-Complete

In this Section we show that the problem of generalised function matching is NP-Complete (with or without wildcards) via a reduction from 3-SAT. Specifically, given an instance of 3-SAT with N variables and M clauses we show how to construct an instance of GFM (with length polynomial in M) which has a solution if and only if the 3-SAT instance is satisfiable. In the process we show that the reduction is valid even for quite severely restricted versions of the problem.

The construction starts by prepending two \$ 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 will include 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 will contain an equal number of \$ characters which forces \$ to map to \$ under any valid function, since any other mapping of \$ must contain \$\$ as a prefix, breaking the equality assumption.

For each variable a_i , we add to the text the string \$000\$ and to the pattern $\$a_iA_i\$$. In this way a variable can be mapped to 00 or 0. To fix

notation we say that 0 represents True and 00 represents False. For each clause, we add to the text the string \$000000\$ (6 zeros) and to the pattern the string \$xyzc_i\$ 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.

Theorem 1. *The generalised function matching problem is NP-Complete.*

Proof. GFM is in NP as any candidate mapping function f can be checked in polynomial time. To show NP-Hardness, take a 3-SAT instance ϕ and use the above construction to produce a pattern and text, called p and t respectively. We prove that there is a GFM solution for p and t if and only if ϕ is satisfiable.

If there is a GFM solution then we know that the \$ symbol in the pattern is matched to the same symbol in the text. The variable gadgets also ensure that variables have consistent assignments. The clause gadgets ensure that at least one of the three literals in each clause is assigned to the value 0, representing True. Therefore, it suffices to read the mapping found to give an assignment of truth values to variables which satisfies ϕ .

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 will be able to be matched to a nonempty substring in each clause gadget.

Finally, we observe that, with a small modification, the proof still holds under two severe restrictions:

Corollary 1 *Generalised function matching remains NP-Complete when Σ_t contains only two distinct symbols and we restrict f so that $|f(x)| \leq 2$ for all $x \in \Sigma_p$.*

3 GPM is NP-Complete

Generalised parameterised matching adds an extra constraint to the conditions set by GFM. The mapping f must now be injective.

It is instructive first to see why we cannot simply translate the reduction for GFM. The problem is that all “variable characters” are mapped to 0 or 00 in the GFM reduction so a GPM solution could never occur with more than two variables. Therefore we need to design new gadgets to overcome this difficulty.

We present a reduction from 1-in-3 SAT, an NP-Complete variant of 3-SAT [6]. 1-in-3 SAT is defined as 3-SAT but with the additional constraint that in a satisfying assignment, each clause must contain exactly one True

literal, instead of at least one for 3-SAT. Given an instance of 1-in-3 SAT, ϕ , with N variables named $a_1, a_2 \dots a_N$ and M clauses, we construct an instance of GPM which matches if and only if ϕ is satisfiable. The instance has the form, $t = \$\$V_t C_t$ and $p = \$\$V_p C_p$. Here V_t and V_p encode the variables in ϕ , and C_t and C_p encode the clauses. As in Section 2, the $\$$ symbols ensure that $\$$ maps to $\$$.

We will often say that $f(\{a, b\}) = \{x, y\}$, by which we mean that $(f(a) = x \text{ and } f(b) = y)$ or $(f(a) = y \text{ and } f(b) = x)$ and also define 0^k to be a string of 0s of length k . For example, $0^3 = 000$.

Each variable results in a pair of strings P_i, T_i . We then concatenate these to form variable gadgets $V_p = P_1 P_2 \dots P_N$ and $V_t = T_1 T_2 \dots T_N$. The components P_i, T_i are defined as $P_i = \$a_i A_i \$$ and $T_i = \$0^{4i-1} \$$.

As with the variables, each clause results in a pair of strings, P'_i and T'_i . These are concatenated as gadgets $C_p = P'_1 P'_2 \dots P'_M$ and $C_t = T'_1 T'_2 \dots T'_M$ respectively. Suppose the i^{th} clause is $(v_j \vee v_k \vee v_l)$, with $v_x \in \{a_x, A_x\}$ for $x \in \{j, k, l\}$. We then define $P'_i = \$v_j v_k v_l \$$ and $T'_i = \$0^{2(j+k+l)-1} \$$.

Lemma 1 *The mapping f gives a GPM match for pattern $\$\$V_p \$C_p$ and text $\$\$V_t \$C_t$ iff for all i , $f(\{a_i, A_i\}) = \{0^{2i-1}, 0^{2i}\}$ and $f(C_p) = C_t$.*

Lemma 1 allows us to consider only a restricted set of functions in the proof of our reduction below. The proof of the Lemma is by induction on the set of mapped strings in a matching function. We will let 0^x represent True iff x is odd. This can be seen as a generalisation of the GFM reduction where 0 represented True and 00 represented False.

Theorem 2. *Generalised parameterised matching is NP-Complete.*

Proof. Given an instance of 1-in-3 SAT ϕ we construct pattern $p = \$\$V_p \$C_p$ and text $t = \$\$V_t \$C_t$ as described above. We show that p GPM matches t iff there exists a 1-in-3 satisfying assignment for ϕ .

We define a function, f_σ for each assignment of truth values, σ . First set $f_\sigma(\$) = \$$. For each i , if variable a_i is True then let $f_\sigma(a_i) = 0^{2i-1}$ and $f_\sigma(A_i) = 0^{2i}$. Otherwise let $f_\sigma(a_i) = 0^{2i}$ and $f_\sigma(A_i) = 0^{2i-1}$. It follows from Lemma 1 that any f which creates a GPM match is equal to f_σ for some σ . Thus we do not need to consider other functions.

Assume that σ satisfies ϕ . Consider the i -th clause, $(v_j \vee v_k \vee v_l)$ and, wlog, assume v_j is True and v_k, v_l are False. Therefore, $f(P'_i)$ equals,

$$f(\$)f_\sigma(v_j)f_\sigma(v_k)f_\sigma(v_l)f(\$) = \$0^{2i-1}0^{2j}0^{2l}\$ = \$0^{2(i+j+l)-1}\$ = T'_i$$

as required. As i was arbitrary, this holds for all clauses so C_P GPM matches C_T under f_σ . Thus, by Lemma 1, p GPM matches t under f_σ .

Conversely, assume that p GPM matches t under some function f_σ . Again, consider the i -th clause, $(v_j \vee v_k \vee v_l)$. As $f_\sigma(\$) = \$$, P'_i GPM matches T'_i . Assume for a contradiction that all three literals are False,

$$f_\sigma(v_j) = 0^{2^j}, f_\sigma(v_k) = 0^{2^k} \text{ and } f_\sigma(v_l) = 0^{2^l}$$

$$f_\sigma(P'_i) = \$f_\sigma(v_j)f_\sigma(v_k)f_\sigma(v_l)\$ = \$0^{2^{j+k+l}}\$ \neq T'_i .$$

Similarly, if one or zero literals are False then $f_\sigma(T_i) = 0^{2^{j+k+l}-2}$ or $f_\sigma(T_i) = 0^{2^{j+k+l}-3}$ respectively. Therefore, exactly one literal is True and the clause is 1-in-3 satisfied. Again, as i was arbitrary, σ satisfies ϕ .

We might now ask if the GPM problem can be restricted in the same way as GFM in Corollary 1:

Corollary 2 *GPM remains NP-Complete when Σ_t contains only two distinct symbols or when we restrict f so that $|f(x)| \leq 2$ for all $x \in \Sigma_p$ (but not both).*

4 Generalised pattern matching with string classes

We now discuss a variant of generalised function matching which does permit a polynomial-time solution. The input is specified by a pattern, p , a text, t , and a set of string classes defined by a function C from characters in the pattern to sets of strings over the text alphabet. Pattern p matches text t if and only if t can be written as $s_1s_2 \dots s_m$ where $s_i \in C(p_i)$. As we will see, an important feature of the problem definition is that different occurrences of the same character in the pattern can match to different substrings if the substrings are in the same class. The problem can be seen as a generalisation of the problem known as *pattern matching with character classes* [5].

Example 1. If the text is $t = banana$, the pattern is $p = ABA$ and the classes are $C(A) = \{ban, ba, n, an, na\}$ and $C(B) = \{anan, na, b, a\}$, then we say that p matches t . A possible matching is a mapping $A_1 \rightarrow ba$, $B \rightarrow na$ and $A_2 \rightarrow na$ where A_i denotes the i th A in the pattern.

We present an $O(nmk \log n)$ solution to this problem, where k is the length of the longest string from any class. The solution is based on dynamic programming. Let $d(i, j)$ be 1 if the first j characters of p match

the first i characters of t and 0 otherwise. If $d(n, m) = 1$ then p matches t . The recurrence formula is:

$$d(i, j) = 1 \text{ iff } \exists 0 \leq \ell < i \text{ s.t. } d(\ell, j - 1) = 1 \text{ and } t[\ell + 1..i] \in C(p_j),$$

where $t[\ell + 1..i] = t_{\ell+1}t_{\ell+2} \dots t_i$ and $C(p_j)$ is the character class of p_j . We define $d(0, 0)$ to be 1. We then calculate all $d(i, j)$ values using generalised suffix trees for the classes which have been precomputed in a preprocessing stage.

Theorem 3. *Assume k is the length of the longest substring in any of the classes and $|C|$ is the total length of all the substrings classes. The pattern matching problem with string classes can be solved in $O(nmk \log n)$ time with $O(|C| \log n)$ preprocessing of the pattern.*

5 An approximation algorithm for GFM

In this Section we will introduce an optimisation version of GFM for which we are able to provide a polynomial-time approximation algorithm. Our motivation is to give a measure of how close two strings are to having a GFM match.

Hamming similarity We define the Hamming similarity between two strings of the same length to be 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.

When changing a symbol at a position in the pattern, we can choose a character that does not occur elsewhere. The new unique character will therefore be permitted to match any non-empty substring of the text. We write such unique characters using the wildcard symbol ‘*’ in order to emphasise their role in any matching.

Example 2. If $p = aba$ and $t = xyz$, then we can keep at most two positions in the pattern unchanged in order to have a GFM match. Hence the Hamming similarity is 2. One option is modify the pattern so that $p = ab*$.

We present a polynomial-time approximation algorithm that achieves a $\min(1, \sqrt{k/\text{OPT}})$ approximation ratio for the Hamming similarity problem, for fixed k . The algorithm is as follows.

1. Select all $S \subset \Sigma_p$ with $|S| = k$. There are $\binom{|\Sigma_p|}{k} \leq m^k$ such S .
 - (a) For each $i = 1, \dots, m$, set p_i^S to p_i if $p_i \in S$ or to the wildcard symbol, otherwise.
 - (b) Let H_S denote the Hamming similarity between p^S and t .
2. Let $M = \max_S H_S$.
3. Output $\max(M, |\Sigma_p|)$.

We claim that that $\text{OPT} \geq \max(M, |\Sigma_p|) \geq \min(\text{OPT}, \sqrt{k \cdot \text{OPT}}) = \text{OPT} \cdot \min(1, \sqrt{k/\text{OPT}})$. In other words,

Lemma 1. *The algorithm given above achieves an approximation ratio of $\min(1, \sqrt{k/\text{OPT}})$ for the Hamming GFM similarity problem.*

Proof. We know that $\text{OPT} \leq M|\Sigma_p|/k$ as M is the maximum for any set of k characters. Therefore $k\text{OPT} \leq M|\Sigma_p|$. It follows that either M or $|\Sigma_p|$ is $\geq \sqrt{k\text{OPT}}$. Therefore, the approximation ratio follows immediately as $\sqrt{k\text{OPT}}/\text{OPT} = \sqrt{k/\text{OPT}}$.

We now describe how to perform the first step of the algorithm in polynomial time. We choose all subsets of k characters from Σ_p and all subsets of k substrings of t and solve the problem for each independently. We fix for the moment the set of characters to be a_1, \dots, a_k and the set of substrings that these are mapped to be s_1, \dots, s_k .

The solution is based on dynamic programming. We define the function $f(i, j)$ to be the best solution to the problem for $t_1 t_2 \dots t_j$ and $p_1 p_2 \dots p_i$. We define $f(0, i) = 0 \forall i$ and $f(i, j) = 0 \forall i > j$. The Hamming GFM similarity of the entire string is $f(m, n)$. We now show how we compute $f(i, j)$ when $i \leq j$.

1. If $p_i \notin \{a_1, \dots, a_k\}$, then
$$f(i, j) = \max\{f(i-1, j-1), f(i-1, j-2), \dots, f(i-1, i-1)\}.$$
2. If $\exists z$ s.t. $p_i = a_z$, then
$$f(i, j) = \max_k \{f(i-1, j-k) + I(t_{j-k+1} \dots t_j = s_z)\},$$
where I is the indicator function.

Theorem 4. *For any k there is an algorithm that runs in time $n^{O(k)}$ and achieves an approximation ratio of $\min(1, \sqrt{k/\text{OPT}})$ for the Hamming GFM similarity problem.*

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.

- $p_i \notin \{a_1, \dots, a_k\}$. In this case we cannot increase the number of characters in our set that can be mapped. However we know that p_i will be set to a wildcard and therefore we find the maximum of the previous results for different length substrings that the wildcard maps to.
- $\exists z$ s.t $p_i = a_z$. We can either map p_i to s_z and increase the number of mapped characters by one, which can only happen if $t_{j-|s_z|+1} \dots t_j = s_z$ or we do the same as in the previous case.

The running time of the approximation algorithm is polynomial in n and m for constant k , since there are $\binom{|\Sigma_p|}{k} \leq \binom{m}{k}$ ways of selecting k characters of Σ_p and at most $\binom{n^2}{k} k!$ ways of choosing k substrings of t .

6 Acknowledgements

We thank Igor Nor and Amihoud Amir for their very helpful contributions to our understanding of the general problem area and Tom Hinton for advice regarding approximation algorithms. The first and third authors also thank the EPSRC for their support.

References

1. A. Amir, A. Aumann, R. Cole, M. Lewenstein, and E. Porat. Function matching: Algorithms, applications, and a lower bound. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 929–942, 2003.
2. A. Amir and I. Nor. Generalized function matching. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, pages 41–52, 2004.
3. A. Amir and I. Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5(3):514–523, 2007.
4. B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 71–80, 1993.
5. C. Linhart and R. Shamir. Faster pattern matching with character classes using prime number encoding. *Journal of Computer and System Sciences*, 75(3):155–162, 2009.
6. T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–226, New York, NY, USA, 1978. ACM.