Permuted Function Matching

Raphaël Clifford[a], Benjamin Sach[*,a]

[a]*Department of Computer Science, University of Bristol, UK*

Abstract

We consider the combination of function and permuted matching, each of which has fast solutions in their own right. Given a pattern p of length m and a text t of length n, a function match at position i of the text is a mapping f from Σ_p to Σ_t with the property that $f(p_j) = t_{i+j-1}$ for all j. We show that the problem of determining for each substring of the text, if any permutation of the pattern has a function match is in general **NP**-Complete. However where the mapping is also injective, so called parameterised matching, the problem can be solved efficiently in $O(n \log |\Sigma_p|)$ time. We then give a 1/2-approximation for a Hamming distance based optimisation variant by reduction to multiple knapsack with colour constraints.

1. Introduction

A great number of efficient algorithms for approximate pattern matching have been developed in recent years under a number of measures of similarity between strings. Despite these successes, one of the outstanding problems that remains is how to combine more than one measure of approximation simultaneously. For example, relatively simple algorithms are known for exact matching with single character wildcards [7, 6] and also separately for counting the number of mismatches at each alignment [11, 3]. However, approximate matching when both wildcards and mismatches are permitted has proved considerably more challenging. Our work is motivated not only by the theoretical interest of exploring the complexity of such combined problems but also the practical observation that in modern applications, it is increasingly common that patterns are to be found using more than one criterion simultaneously and that this requires a new toolset of algorithmic techniques.

We consider here the combination of the simple problem of permuted matching with that of function matching (defined below). We also consider combining permuted matching with parameterised matching, a restriction of function matching. We first give fast solutions for this latter problem and then show that the combined problem of permuted and function matching is **NP**-Complete. We then introduce a Hamming distance based optimisation variant of permuted

*Corresponding author

function matching and show how to derive a 1/2-approximation by reduction to a problem known as multiple knapsack with colour constraints.

Notation. *We assume throughout that the pattern p and text t have lengths m and n respectively, with $n \geq m$. The i-th pattern (text) character is denoted p_i (t_i). We also write Σ_p and Σ_t for the pattern and text alphabets, respectively.*

Definition 1 (Permuted Matching)**.** *Given a pattern p and text t, for each alignment i, report whether there exists a permutation, π, such that*

$$t_{i+j-1} = p_{\pi(j)} \text{ for all } 1 \leq j \leq m\,.$$

Permuted matching has an elementary $O(n \log |\Sigma_p|)$ time deterministic solution by maintaining a histogram of character frequencies in a sliding window of the text. This can be improved to $O(n)$ expected time using perfect hashing [8].

Definition 2 (Function/Parameterised Matching [1, 5])**.** *Given a pattern p and text t, for each alignment i, report if there is a function, $f : \Sigma_p \to \Sigma_t$, such that*

$$t_{i+j-1} = f(p_j) \text{ for all } 1 \leq j \leq m\,.$$

The problem is termed parameterised matching if the function f is restricted to be an injection (one-to-one).

The function matching problem has randomised $O(n \log m)$ and deterministic $O(nm)$ time solutions [1]. The restriction to parameterised matching is solvable in $O(n \log |\Sigma_p|)$ time [2] by a modification of the well known Knuth-Morris-Pratt algorithm. The original motivation for this restricted problem formulation was in software maintenance [5], where programs are to be considered identical even if their variable names are different.

The new problem that we consider combines these problems and determines at each alignment whether there is any permutation of the pattern for which a function or alternatively parameterised match exists.

Definition 3 (Permuted Function/Parameterised Matching)**.** *Given a pattern p and text t, for each alignment i, report if there exists a pair, (f, π) where f is a function, $f : \Sigma_p \to \Sigma_t$ and π is a permutation, such that*

$$t_{i+j-1} = f(p_{\pi(j)}) \text{ for all } 1 \leq j \leq m\,.$$

The parameterised variant restricts the function f to be injective.

Example 1. *If $p = aba$ and $t = xyyy$, then there is permuted function match at both alignments by considering the permutation baa at the first alignment and any permutation at the second. However, there is no permuted parameterised match at the second alignment as a and b cannot both be mapped to the same character y.*

In contrast to the situation for permuted function matching which we will see later on, permuted parameterised matching can be solved efficiently in $O(n \log |\Sigma_p|)$ time. We give a brief overview of the techniques required by way of introduction to the main results. The overall approach is to consider a sliding window of the same length as the pattern and update any required data structures as it is moved one position at a time to the right in the text. We maintain a balanced binary search tree (BST) of pairs (a, occ), where $occ > 0$ is the number of occurrences of symbol a in the current sliding window of the text being considered. We also maintain an array Y of length m which contains at $Y[occ]$, the number of symbols which occur exactly occ times in the current window of the text. We keep an equivalent array Y_p for p. By the problem definition, we have that there is a permuted parameterised match between p and $t[i..i+m-1]$ iff $Y_p = Y$ (at window alignment i). The array Y can be updated in $O(\log m)$ time per position in the text using the binary search tree. To compare Y_p to Y we keep a counter of the number of positions at which they differ which we can update in constant time. This gives a solution with $O(n \log m)$ time complexity.

The time complexity can be improved by observing that only alignments where $t[i..i+m-1]$ contains at most $|\Sigma_p|$ unique symbols can match. It is sufficient to maintain a balanced BST of the rightmost $|\Sigma_p|+1$ distinct characters in $t[i..i+m-1]$. This reduced BST allows updates of Y in $O(\log |\Sigma_P|)$ time.

Theorem 1.1. *Given two strings of length m and n, all permuted parameterised matches can be found in $O(n \log |\Sigma_P|)$ time and $O(n)$ space.*

As a side observation, we can see that in the data streaming model [12] where the text is provided online, one character at a time from left to right, it is possible to reduce the space further. In this model we do not need to store the entire text. Observe that at matching alignments, arrays Y and Y_p are sparse, containing only $|\Sigma_p|$ entries. By using a BST implementation, the total space can be reduced from $O(m)$ to $O(|\Sigma_p|)$.

Corollary 1.2. *For an online text, permuted parameterised matches can be found in $O(\log |\Sigma_P|)$ (unamortised) time per character using $O(|\Sigma_p|)$ space.*

2. Permuted function matching is NP-Complete

We show that permuted function matching (PFM) is **NP**-complete by reduction from the strongly **NP**-Complete problem 3-partition. In this section we will consider the case where $n = m$.

Definition 4 (3-partition [9])**.** *Assume we are given positive integers x, B, and a (multi)set of integers $A = \{a_1, a_2, ..., a_{3x}\}$ with $B/4 < a < B/2$ for all $a \in A$ and $\sum_{a \in A} a = xB$. Can A be partitioned into x (multi)sets, $S_1, S_2, \ldots, S_x$ such that $|S_i| = \sum_{a \in S_i} a = B$ for all i?*

Example 2. *Given, $A = \{1, 1, 1, 2, 2, 2, 2, 3, 4\}$, $x = 3$ and $B = 6$, can A be partitioned into subsets, S_1, S_2, S_3 s.t. $|S_1|=|S_2|=|S_3|=6$? The following*

subsets give a witness,

$$S_1 = \{1, 2, 3\} \ , \ S_2 = \{2, 2, 2\} \ , \ S_3 = \{1, 1, 4\}.$$

Permuted function matching is clearly in **NP** as any (f, π) pair can be checked in polynomial time. It only remains to show that it is also **NP**-hard.

Theorem 2.1. *The permuted function matching problem is* **NP***-hard.*

PROOF. Consider an instance of the 3-partition problem, (A, x, B) as defined above. As 3-partition is strongly **NP**-Complete we are free to assume that all numbers are encoded in unary. The importance of this will become apparent below. We wish to construct an instance of PFM such that a match exists iff there is a valid partitioning of the 3-partition instance.

The construction of the PFM instance is as follows: Let a_i be the ith integer in A. We use w^k to denote symbol w repeated k times. Let $y = 3x$ and construct the pattern p and text t as follows:

$$p = 1^{a_1} 2^{a_2} 3^{a_3} \dots y^{a_y} \quad \text{and} \quad t = 1^B 2^B 3^B \dots x^B.$$

For example, using the instance given in Example 2, we obtain:

$$p = 1^1 2^1 3^1 4^2 5^2 6^2 7^2 8^3 9^4 = 123445566778889999,$$
$$t = 1^6 2^6 3^6 = 111111222222333333.$$

For our reduction to be valid we need the length of the PFM instance constructed to be polynomial in the length of the original 3-partition instance. We denote the total length of the input, (A, x, B) by ℓ. However as the instance is encoded in unary we have that x and B are both no larger than ℓ. By observing that $m = n = xB$, we have $m + n \in O(\ell^2)$. This highlights the importance to our reduction that 3-partition is strongly **NP**-complete.

Case 1: $(A, x, B) \in$ 3-partition. Let $S_1, S_2, \dots, S_x$ be a correct partitioning of this instance. For each integer $a_i \in A$, let $f(i)$ equal the index of the (unique) subset S_j to which it is assigned. Continuing our example:

$$f(1) = 1, f(2) = 3, f(3) = 3,$$
$$f(4) = 1, f(5) = 2, f(6) = 2,$$
$$f(7) = 2, f(8) = 1, f(9) = 3.$$

Now let π be a permutation under which $f(p_{\pi(j)}) \leq f(p_{\pi(j+1)})$ for all j. We define $f_\pi(p)$ to be the string which results from applying (f, π) to p. It remains to prove that $t = f_\pi(p)$. This is clear from inspection in our example as:

$$f(p) = 133112222221113333 \text{ and } f_\pi(p) = 111111222222333333 = t\,.$$

As $f_\pi(p)$ and t are both in non-decreasing order, they are equal iff they contain the same number of occurrences of each symbol. Consider some symbol $j \in \{1, 2, \dots, x\}$. Observe that the number of occurrences of j in $f_\pi(p)$ is exactly

$\sum_{a \in S_j} a$ by the construction of f. However as the sets are satisfying, $\sum_{a \in S_j} a = B$ which is the number of occurrences of any symbol in t. Therefore $p = t$ and there is a PF-match for p and t.

Case 2: $(p, t) \in$ PFM. There exists a satisfying pair, (f, π) with $f_\pi(p) = t$. For each $a_i \in A$ we assign a_i to the subset S_j where $j = f(i)$. Recall that each integer a_i corresponds to $|a_i|$ occurrences of the i symbol in p. Therefore the size of S_j equals the number of occurrences of the symbol j in $f_\pi(p) = t$. As each symbol $j \in \{1, 2, \ldots, x\}$ occurs B times in t we have that $|S_j| = B$ as required.

3. Approximating permuted function matching

Given a pattern p and a text t (both of length n), we define an optimisation variant of the **NP**-Complete permuted function matching problem. The *permuted similarity* problem is defined to be the maximum (Hamming) similarity between p and any pattern p' which PF-matches t. We define the Hamming similarity between two equal length strings as the number of positions at which they agree which we denote by $sim(p', p)$. This is exactly the string length minus the well known Hamming distance. The related problem of approximate parameterised matching (without permutation) has also been considered under the same similarity measure [4, 10].

Definition 5 (Multiple knapsack with colour constraints [13])**.** *We are given M items to pack into N knapsacks. Each item has a weight, a value and a colour. Each knapsack has a weight limit and a number of compartments. We wish to pack items into knapsacks to maximise the packed value. However, we cannot put into a knapsack more weight than its limit or more distinct colours than the number of compartments.*

We now give a reduction which as a corollary implies the existence of a 1/2-approximation for the *permuted similarity* problem.

Theorem 3.1. *The permuted similarity problem has an approximation preserving reduction to multiple knapsack with colour constraints (MKCC) which can be computed in $O(n \log n)$ time.*

Proof. Given (p, t) we construct an instance of MKCC such that there exists a packing with value v iff there exists a string p' which PF-matches t and $sim(p', p) = v$. For each location in the text, i, we have an item, I_i, of weight 1, value 1 and colour t_i. For each symbol, $a \in \Sigma_p$ we have a knapsack, K_a with weight limit $|K_a| = |\{j | p_j = a\}|$ and 1 compartment. We consider K_a as having $|K_a|$ unit spaces, each corresponding to a location containing an occurrence of symbol a. After sorting p lexicographically in $O(n \log n)$ time both the knapsacks and the items can be easily constructed in linear time.

Assume there exists a pattern p' which PF-matches t under (f, π) such that $sim(p', p)$ is v. Let $C = \{i | p_i = p'_i\}$. For all $\pi(i) \in C$, we pack I_i into $K_{p_{\pi(i)}}$. If $\pi(i), \pi(j) \in C$ such that $p_{\pi(i)} = p_{\pi(j)}$ then $t_i = f(p'_{\pi(i)}) = f(p'_{\pi(j)}) = t_j$.

Therefore, each knapsack contains items of only one colour as required. As we enumerate a subset of pattern locations, filling their allocated spaces, no weight limit can be exceeded. Finally, observe that we pack exactly $|C| = v$ items as required.

Assume there exists a packing with value v. For all $a \in \Sigma_p$, if K_a is non-empty, it contains items of a single colour, corresponding to a text symbol b, let $f(a) = b$. If the space for pattern position j is filled with item i then let $\pi(i) = j$. Observe that $f(p_{\pi(i)}) = f(a) = b = t_i$. Let C be a list of all filled spaces and hence $|C| = v$. Alternatively, C is the list of locations for which (f, π) is a valid mapping from p to t. For all $i \notin C$, complete the permutation arbitrarily. We now give a string p' which matches t under (f, π) and has similarity v to p. If $i \in C$ then let $p'_i = p_i$. Otherwise let p'_i equal a new symbol not appearing in p (or $p'_1 p'_2 \dots p'_{i-1}$). Observe that $sim(p', p) = |C| = v$ as required. Finally, as p and p' agree on all indices in C we need only to consider $i \in \{1, 2, \dots n\}/C$ but p'_i is unique so f can be completed to make these locations match.

Corollary 1. *There is a polynomial time* 1/2*-approximation ratio algorithm for the* permuted similarity *problem which runs in time* $O(n^4)$.

Proof. Shachnai and Tamir [13] give an algorithm which approximates MKCC to within a factor $b/(b+1)$. They restrict items to have unit weight and value and every knapsack to have at least $b \geq 1$ compartments. A 1/2-approximation for PFM follows immediately from Theorem 3.1. The algorithm runs in $O(N^2M^2)$ $\in O(n^4)$ time.

4. Open Problems

An interesting direction of further research would be to find a PTAS for the similarity problem. There is a PTAS for standard multiple knapsack as well as for several interesting variants of multiple knapsack including multiple knapsack with colour constraints given a constant number of colours. However, in our case each item may have a different colour and so these results do not directly apply.

References

[1] A. Amir, A. Aumann, R. Cole, M. Lewenstein, and E. Porat. Function matching: Algorithms, applications, and a lower bound. In *Proc.* 30*th ICALP*, pages 929–942, 2003.

[2] A. Amir, M. Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Inf. Process. Lett.*, 49(3):111–115, 1994.

[3] A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

[4] Alberto Apostolico, Péter L. Erds, and Moshe Lewenstein. Parameterized matching with mismatches. *J. Discrete Algorithms*, 5(1):135–140, 2007.

[5] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the Annual ACM Symposium on the Theory of Computing*, pages 71–80, 1993.

[6] P. Clifford and R. Clifford. Self-normalised distance with don't cares. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, pages 63–70, 2007.

[7] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC '02: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 592–601, 2002.

[8] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Co., 1979.

[10] Carmit Hazay, Moshe Lewenstein, and Dina Sokol. Approximate parameterized matching. *ACM Trans. Algorithms*, 3(3):29, 2007.

[11] G. M. Landau and U. Vishkin. Efficient string matching in the presence of errors. In *FOCS '85: Proceedings of the 26th Symposium on Foundations of Computer Science*, pages 126–136, 1985.

[12] S. Muthukrishnan. Data streams: algorithms and applications. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[13] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29(3):442–467, 2001.