Function Matching: Algorithms, Applications and a Lower Bound

Amihood Amir, Bar-Ilan University
Yonatan Aumann, Bar-Ilan University
Richard Cole, New York University
Moshe Lewenstein, IBM Research
Ely Porat, Bar-Ilan University

Abstract

We introduce a new matching criterion – *function matching* – that captures several different applications. The *function matching problem* has as its input a text T of length n over alphabet Σ_T and a pattern $P = P[1]P[2]\cdots P[m]$ of length m over alphabet Σ_P. We seek all text locations i for which, for some function $f : \Sigma_P \rightarrow \Sigma_T$ (f may also depend on i), the m-length substring that starts at i is equal to $f(P[1])f(P[2])\cdots f(P[m])$.

We give a randomized algorithm which, for any given constant k, solves the function matching problem in time $O(n \log n)$ with probability $\frac{1}{n^k}$ of declaring a false positive. We give a deterministic algorithm whose time is $O(n|\Sigma_P| \log m)$ and show that it is almost optimal in the newly formalized convolutions model. Finally, a variant of the third problem is solved by means of two-dimensional parameterized matching, for which we also give an efficient algorithm.

1 Introduction

In the traditional pattern matching model, one seeks exact occurrences of a given pattern in a text, i.e. text locations where every text symbol is *equal* to its corresponding pattern symbol. In the *parameterized matching* problem, introduced by Baker [7], one seeks text locations where there exists a bijection f on the alphabet for which every text symbol is equal to *the image under* f of the corresponding pattern symbol. In the applications we will describe below, f cannot be a bijection. Rather, it should be simply a function. More precisely, P matches T at location i if for every element $a \in \Sigma$, *all* occurrences of a have the same corresponding symbol in T. In other words, unlike in parameterized matching, there may be a *several* different symbols in the pattern which are mapped to the same text symbol.

Consider the following problems where parameterized matching is insufficient and function matching is required.

<u>Programming Languages:</u> There is a growing class of real-time systems applications where software codes are embedded on small chips with limited memory, e.g. chips in appliances. In these applications it is important to have as small a number of memory variables as possible. A similar problem exists in compiler design, where it is desirable to minimize the register-memory traffic, and re-use global registers as much as possible. This need to compact code by global register allocation and spill code minimization is an active research topic in the programming languages community (see e.g. [14, 13]).

Automatically identifying functionally equivalent pieces of such compact code would make it easier to reuse these pieces (and, for example, to replace multiple such pieces by one piece in embedded code). Baker's parameterized matching was a first step in this direction. It identified codes that are identical, up to a one-to-one mapping of the variable names. This paper considers a generalization that identifies codes in which the mapping of the variable names (or registers) is possibly a many-to-one mapping. This identifies a larger set of candidate code portions which might be functionally equivalent (the equivalence would depend on the interleaving of and updates to the variables and so would require further postprocessing for confirmation).

Computational Biology: The Grand Challenge *protein folding* problem is one of the most important problems in computational biology (see e.g. [12]). The goal is to determine a protein's tertiary structure (how it folds) from the linear arrangement of its peptide sequence.

This is an area of extremely active research and a myriad of methods have and are being considered in attempts to solve this problem. One possible technique that is being investigated is *threading* (e.g. [8, 19]). The idea is to try to "thread" a given protein sequence into a known structure. A starting point is to consider peptide subsequences that are known to fold in a particular way. These subsequences can be used as patterns. Given a new sequence, with unknown tertiary structure, one can seek known patterns in its peptide sequence, and use the folding of the known subsequences as a starting point in determining the full structure. However, a subsequence of different peptides that bond in the same way as the pattern peptides may still fold in a similar way. Such functionally equivalent subsequences will not be detected by exact matching. Function matching can serve as a filter, that identifies a superset of possible choices whose bondings can then be more carefully examined.

Image Processing: One of the interesting problems in web searching is searching for color images (e.g. [16, 6, 3]). The simplest possible cases is searching for an icon in a screen, a task that the Human-Computer Interaction Lab at the University of Maryland was confronted with. If the colors are fixed, this is exact two-dimensional pattern matching [2]. However, if the color maps in pattern and text differ, the exact matching algorithm would not find the pattern. Parameterized two dimensional search is precisely what is needed. If, in addition, we are faced with a loss of resolution in the text, e.g. due to truncation, then we would need to use a two dimensional function matching search.

The above examples are a sample of diverse application areas encountering search problems that are not solved by state of the art methods in pattern matching. This need led us to introduce, in this paper, the *function matching* criterion, and to explore the *two dimensional parameterized matching* problem.

Function matching is a natural generalization of parameterized matching. However, relaxing the bijection restriction introduces non-trivial technical difficulties. Many powerful pattern matching techniques such as automata methods, subword trees, dueling and deterministic sampling assume transitivity of the matching relation (see [10] for techniques). For any pattern matching criteria where transitivity does not exist, the above methods do not help.

Examples of pattern matching with non-transitive matching relation are string matching with "don't cares", less-than matching, pattern matching with mismatches and swapped matching. It is interesting to note that the efficient algorithms for solving the above problems all used convolutions as their main tool. Convolutions were introduced by Fischer and Paterson [11] as a technique for solving pattern matching problems with wildcards, where indeed the match relation is not transitive.

It turns out that many such problems can be solved by a "standard" application of convolutions (e.g. matching with "don't cares", matching with mismatches in bounded finite alphabets, and swapped matching). Muthukrishnan and Palem were the first to explicitly identify this application method and introduced a *boolean convolutions model* [15] with locality restrictions and obtained several lower bounds in this model. Since the introduction of the boolean convolutions model, several papers appeared using general, rather than boolean convolutions. In this paper we provide a formal definition for a more general convolutions model that broadens the class of problems being considered.

The new convolutions model encapsulates the solution to many non-standard matching problems. Even more importantly, a rigorous formal definition of such a model is useful in proving lower bounds. While such bounds do not lower bound the solution complexity in a general RAM, they do help in understanding the limits of the convolution method, hitherto the only powerful tool for non-standard pattern matching.

There are three main contributions in this paper.

1. A solution to a number of search problems in diverse fields, achieved by the introduction of a new type of generalized pattern matching, that of function matching.
2. A formalization of a new general convolutions model. This leads to a deterministic solution. We prove that this solution is almost *tight* in the convolutions model. We also present an efficient randomized solution of the function matching problem.
3. Solutions to the problem of exact search in color images with different color maps. This is done via efficient randomized and deterministic algorithms for two-dimensional parameterized and function matching.

In section 2 we give the basic definitions and present progressively more efficient deterministic solutions, culminating in a $O(n|\Sigma_P|\log m)$ algorithm, where $|\Sigma_P|$ is the pattern alphabet size. We also present a Monte Carlo algorithm that solves the function matching problem in time $O(n \log m)$ time with failure probability no larger than $\frac{1}{n^k}$, where k is a given constant. In section 3 we formalize the new convolution model. We then show a lower bound proving that our deterministic algorithm is tight in the convolutions model and discuss the limitations of that model. Finally, in section 4 we present a randomized algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 \log n)$ with probability of false positives no larger than $\frac{1}{n^k}$, for given constant k. We also present a deterministic algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 \log^2 m)$.

2 Algorithms

The key notion is that of a *cover*.

Definition: Let U and V be equal length strings. Symbol τ in U is said to *cover* symbol σ in V if every occurrence of σ in V is aligned with an occurrence of τ in U (i.e. they occur in equal index locations). U is said to cover σ in V if there is some symbol τ in U covering σ. Finally, the cover is said to be an *exact* cover if every occurrence of τ in U is aligned with an occurrence of σ in V.

Definition: There is a *function match* of V with U if every symbol occurring in V is covered by U (but this relation need not be symmetric). If each of the covers is an exact cover the match is a *parameterized match* (and this relation is symmetric).

The term function match arises by considering the mapping from V's symbols to U's symbols specified by the match; it is a plain function in a function match and it is one-to-one in a parameterized match. In both cases the function is onto.

Definition: Given a text T (of length n) and a pattern P (of length m) the function matching problem is to find the alignments (positionings) of P such that P function matches the aligned portion of T. Note that every match may use a different function to associate the symbols of P with those in the aligned portion of T.

As is standard, we can limit T to have length at most $2m$, by breaking T into pieces of length $2m$, successive pieces overlapping by $m-1$ symbols. It is straightforward to give an $O(nm)$ time algorithm for function matching; it simply checks each possible alignment of the pattern in turn, each in time $O(m)$. This is left to the reader.

We start by outlining a simple $O(n|\Sigma_P||\Sigma_T|\log m)$ time algorithm, where Σ_P and Σ_T are the pattern and text alphabets, respectively. This algorithm finds, for each pair $\sigma \in \Sigma_P$ and $\tau \in \Sigma_T$, those alignments of the pattern with the text for which τ covers σ. This will take time $O(n \log m)$ for one pair. A function matching exists for an alignment exactly if every symbol occurring in P is covered.

Definition: The σ*-indicator* of string U, $\chi_\sigma(U)$ is a binary string of length U in which each occurrence of σ is replaced by a 1, and every other symbol occurrence is replaced by 0.

The procedure used the strings $\chi_\sigma(P)$ and $\chi_\sigma(T)$. For each alignment of $\chi_\sigma(P)$ with $\chi_\sigma(T)$ it computes the dot product of $\chi_\sigma(P)$ and the aligned portion of $\chi_\sigma(T)$. But the product is exactly the number of occurrences of σ in P aligned with occurrences of τ in T. This is a cover of σ by τ exactly if the dot product equals the number of occurrences of σ in P. The dot products, for each alignment of $\chi_\sigma(P)$ with $\chi_\sigma(T)$, are all readily computed in $O(n \log m)$ time by means of a convolution [11]. We have shown:

Theorem 1 *Function matching can be solved deterministically in time* $O(n|\Sigma_P||\Sigma_T| \log m)$.

We obtain a faster algorithm by determining simultaneously for all τ occurring in T and for one σ occurring in P, those alignments of P for which some τ covers σ. This is done in time $O(n \log m)$ and is repeated for each σ yielding an algorithm with running time $O(n|\Sigma_P| \log m)$. Our algorithm exploits the following observation.

Lemma 1 *Let* $a_1, ..., a_k$ *be natural numbers. Then* $k \sum_{h=1}^{k} (a_h)^2 = (\sum_{h=1}^{k} a_h)^2$ *iff* $a_i = a_j$, *for* $1 \leq i < j \leq k$.

The algorithm uses the strings T and T_2, where T_2 is defined by $T_2[i] = (T[i])^2$, $i = 0, ..., n-1$. For each σ, for each alignment of P with each of T and T_2, the dot product of $\chi_\sigma(P)$ with the aligned portion of each of T and T_2 is computed. By Lemma 1 T covers σ in a given alignment exactly if the dot product of P with the aligned portion of T_2 is k times larger than the dot product of P with T, where k is the number of occurrences of σ in P. This yields:

Theorem 2 *The function matching problem can be solved deterministically in time* $O(n|\Sigma_P| \log m)$.

We seek further speedups via randomization. We give a Monte Carlo algorithm that, given a constant k, reports all function matches and with probability at most $\frac{1}{n^k}$ reports a non-match as a match. Our first step is to reduce function matching to *paired function matching.* In paired function matching the pattern is a *paired string*, a string in which each symbol appears at most twice. We then give a randomized algorithm for paired function matching.

For the reduction we create a new text T', whose length is $2n$, and a new pattern P', whose length is $2m$. There will be a match of P with T starting at location i in T exactly if there is a match of P' starting at location $2i-1$ in T'. T' is obtained by replacing each symbol in T by two consecutive instances of the same symbol; e.g. if $T = abca$ then $T' = aabbccaa$. To define P', a little notation is helpful. Suppose symbol σ appears k times in P. Then new symbols $\sigma_1, \sigma_2, ..., \sigma_{k+1}$ are used in P'. The ith occurrence of σ is replaced by the pair of symbols σ_i, σ_{i+1}. e.g. if $P = aababca$ then $P' = a_1a_2a_2a_3b_1b_2a_3a_4b_2b_3c_1c_2a_4a_5$. It is easy to see that function matches of P' in T' and of P in T correspond as described above. Thus it remains to give the algorithm for paired function matching.

This algorithm replaces the symbols of P' and T' by integers, chosen uniformly at random from the range $[1, 2n^{k+1}]$ as follows. For the text T', for each symbol σ, a random value v_σ is chosen, and each occurrence of σ is replaced by v_σ, forming a string T''. For the pattern P', for each symbol σ, occurring twice, a random value u_σ is chosen. The first occurrence of σ is replaced by u_σ and the second occurrence by $-u_\sigma$; if a symbol occurs once it is replaced by value 0. This forms string P''. Now, for each possible alignment of P'' with T'', the dot product of P'' with the aligned portion of T'' is computed. Clearly, if there is a function match of P with T, the corresponding dot product evaluates to 0. We show that when there is a function mismatch, the corresponding dot product is non-zero with high probability.

If there is a function mismatch then there is a symbol σ in P'' aligned with distinct symbols τ and ρ in T''. Imagine that the assignment of random values assigns values v_τ, v_ρ, u_σ last. Consider the dot product expressed as a function of v_τ, v_ρ, u_σ; it has the form $A+Bv_\tau+Cv_\rho+u_\sigma(v_\tau - v_\rho)$ (assuming the τ and ρ aligned with σ appear in left to right order), where A, B, and C are the values obtained after making all the other random choices. It is easy to see that there is at most a $\frac{2}{2n^{k+1}} = \frac{1}{n^{k+1}}$ probability of this polynomial evaluating to 0. As there are $n - m + 1$ possible alignments of P with T, the overall failure probability is at most $\frac{1}{n^k}$. We have shown:

Theorem 3 *There is a randomized algorithm for function matching that, given a constant k, runs in time $O(kn \log m)$; it reports all function matches and, with probability at least $1 - \frac{1}{n^k}$ reports no mismatches as matches.*

3 Lower Bounds

The unfettered nature of the function matching problem is what makes it difficult. Traditional pattern matching methods such as automata, duels or witnesses, apparently are of no help since there is no transitivity in the function matching relation. Moreover, it is far from evident whether one can set rules during a pattern preprocessing stage that will allow text scanning, since the relationship between the text and pattern is quite loose. This is what pushed us to consider convolutions as the method for the upper bound. Unfortunately, our deterministic algorithm's complexity is no better than that of the naive for alphabets of unbounded size.

Whenever resorting to a randomized algorithm, it behooves the algorithm's developer to explain why they randomized. In this section we give evidence for the belief that an efficient deterministic solution to the problem, if such exists, may be very difficult. We do it by showing a lower bound of $\Omega(\frac{m}{b})$ convolutions with b-bit inputs and outputs for the function matching problem in the convolutions model. Convolutions, as a tool for string matching, were introduced by Fischer and Paterson [11]. Muthukrishnan and Palem [15] considered a Boolean convolutions model with locality restrictions for which they obtained a number of lower bounds. We did not find a formal definition of general convolutions as a resource in the literature. Recent uses of convolutions with non-Boolean inputs led us to broaden the class of convolutions being considered for lower bound proofs. In fact, Muthukrishnan and Palem proved a lower bound of $\Omega(\log \Sigma)$ boolean convolutions for string matching with wildcards with alphabet Σ; but their lower bound does not hold for more general convolutions as indicated by Cole and Hariharan's recent two convolution algorithm [9]. Our model does not cover all conceivable convolutions-based methods. However, it broadens the class for which lower bounds can be proven. The next subsection formally defines the general convolutions model that we propose.

3.1 The Convolutions Model

We begin by defining the class of problems that are solved by the convolutions model.

Definition: A *pattern matching problem* is defined as follows:

MATCH RELATION: A binary relation $M(a, b))$, where $a = a_0...a_k, b = b_0...b_\ell$ and $a, b \in \Sigma^*$.

INPUT: A *text array*, $T = T[0], ..., T[n-1]$, and a *pattern array* $P = P[0], ..., P[m-1]$, $P[i], T[j] \in \Sigma$, $i = 0, ..., m-1$, $j = 0, ..., n-1$.

OUTPUT: The set of indices $S \subseteq \{0, ..., n-1\}$ where the pattern P *matches*, i.e. all indices i where $M(P, T_i)$, and T_i is the suffix of T starting at location i. We also call the output set of indices the *target elements.*

Example: *String Matching with Don't Cares*

The match relation is defined as follows. Let $\Sigma = \{0, 1\}$. Let $\phi \notin \Sigma$ (ϕ is the *don't care symbol*). Let $|a| = k$ and $|b| = \ell$. If $k > \ell$ then there is no match. Otherwise, a matches b iff $a_i = b_i$ or $a_i = \phi$ or $b_i = \phi$, $i = 0, ..., k-1$. The text and pattern arrays are $T = T[0], ..., T[n-1]$ and $P[0], ..., P[m-1]$, respectively. The target elements are all locations i in the text array T where there is an exact occurrence of P (where ϕ matches both 0 and 1).

As its name suggests, the convolutions model uses convolutions as basic operations on arrays. Another basic operation it uses is *preprocessing.* There is a difference, however, between pattern and text preprocessing. We place no restriction on the pattern preprocessing. The text preprocessing, however, must be *local.* When proving lower bounds in the convolutions model, we are mainly interested in the *number* of convolutions necessary to achieve the solution, rather than the time complexity of the solution (this is akin to counting the number of comparisons in the comparison model for sorting).

Definition: Let g be a pattern preprocessing function. A *g-local text preprocessing function* $f_g : N^n \to N^n$ is a function for which there exists n functions $f_g^j : N \to N$, such that $(f_g(T))[j] = f_g^j(T[j]),\ j = 0, ..., n-1$.

In words, the "locality" of function f_g is manifested by the fact that the value in index j of $f_g(T)$ is computed based solely on the pattern preprocessing (output $g(P)$), the index j, and the value of $T[j]$.

Examples:

1. Let T be an array. Then $\chi_a(T)$ is clearly a local array function, since the only index of T that participates in computing $\chi_a(T)[j]$ is j.

2. Let T be an array of numbers. The function f such that $f(T)[j] = T[j] - (\sum_{i=0}^{n-1} T[i])/n$ is not a local array function.

We now have all the building blocks of the convolutions model.

Definition: The *convolutions computation model* is a specialized model of computation that solves a subset of the pattern matching problems. Given a pattern matching problem whose input is text T and pattern P, a solution in the convolutions model has the following form. Let $g_i,\ i = 1, ..., h(n)$ be pattern preprocessing functions, and let $f_{g_i},\ i = 1, ..., h(n)$ be the corresponding local text preprocessing functions. The model also uses a parameter b.

1. Compute $h(n)$ convolutions $C_i \leftarrow f_{g_i}(T) \otimes g_i(P),\ \ i = 1, ..., h(n)$, with b-bit inputs and outputs.
2. Compute the matches as follows. The decision of whether location j of the text is a match is decided by a computation whose inputs are a subset of $\{C_i[j] \mid i = 1, ..., h(n)\}$.

Example: (see other examples in Appendix A.)

1. *Exact String Matching with Don't Cares* This problem's solution was provided by Fischer and Paterson [11] is in the convolutions model. The two convolutions are:

   $C_1 \leftarrow \chi_0(T) \otimes \chi_1(P)$

   $C_2 \leftarrow \chi_1(T) \otimes \chi_0(P)$

   The text locations i for which $C_1[i] = C_2[i] = 0$ are precisely the match locations.

3.2 Lower Bounds

The solutions we presented in section 2 for the function matching problem were also in the convolutions model. The following theorem shows that our algorithm's complexity is almost tight in the convolutions model.

Theorem 4 *The function matching problem requires $\Omega(\frac{m}{b})$ convolutions in the convolutions model.*

Proof: We will show that the *word equality problem* can be linearly reduced to the function matching problem. The word equality problem is:

INPUT: Two m bit words, $W_1 = W_1[0], ..., W_1[m-1]$ and $W_2 = W_2[0], ..., W_2[m-1]$.

DECIDE: Whether $W_1 = W_2$ (i.e. $W_1[i] = W_2[i],\ i = 0, ..., m-1$) or not.

The following communication complexity lower bound for the word equality problem is known. Suppose processor PA starts with word W_1 and processor PB with word W_2. Then to decide word equality they need to exchange $\Omega(m)$ bits [20].

We show that any algorithm for function matching in the convolutions model using $h(m)$ b-bit convolutions can be used to solve the word equality problem with a transmission of $b \cdot h(m)$ bits, implying $h(m) = \Omega(\frac{m}{b})$.

We consider the operation of the function matching algorithm on the following pattern and text. $T = W_1 W_2$, the concatenation of W_1 and W_2, and $P = 1, 2, \cdots, m, 1, 2, \cdots, m$. Note that T function matches P if and only if $W_1 = W_2$.

Now suppose that function matching is solved by some algorithm F in the convolutions model. F computes $h(m)$ convolutions $C_1, ..., C_{h(m)}$ and then uses the results of $C_i[1],\ i = 1, ..., h(m)$ to decide whether there is a function-match.

Note that for every convolution $C = A \otimes B$, $C[1] = \sum_{h=0}^{2m-1} A[h]B[h]$. However, this is equal to $\sum_{h=0}^{m-1} A[h]B[h] + \sum_{h=0}^{m-1} A[h+m]B[h+m]$. For each convolution, PA will compute $\sum_{h=0}^{m-1} A[h]B[h]$, which is based solely on $T[1], \cdots, T[m]$ and P, and PB will compute $\sum_{h=0}^{m-1} A[h+m]B[h+m]$, which is based solely on $T[m+1], \cdots, T[2m]$ and P. PA will then transmit its b-bit result to PB for each of the $h(m)$ convolutions used by F, and PB can at this point determine the result of the word equality problem. $\square$

It is important to be careful in interpreting the results of the convolutions model in a RAM complexity model since complexity in the convolutions model is measured by number of convolutions, rather than RAM operations. When evaluating the number of operations it takes to compute a convolution one must consider the number of bits in a RAM word. The standard in the pattern matching literature is an $O(\log m)$ bit word and the currently fastest known algorithm for computing convolutions is by using the FFT. Its time complexity is $O(n \log m)$ word operations. Thus, the number of RAM operations required to compute function matching in the convolutions model would appear to be $\Omega(nm)$. Of course, conceivably, by ingenious use of special case convolutions one might be able to evaluate the convolutions more quickly, though no such approach has occurred to us.

4 Two Dimensional Algorithms

The one dimensional parameterized matching problem was efficiently solved in [5]. However, as discussed in [3], the move to two dimensions implies a possible computational difficulty if no separable attributes exist. Parameterized matching is not separable – if all columns (or rows) have parameterized matches, it does not necessarily imply that the entire matrix has a parameterized match. Thus we are forced to seek other approaches.

Our Problem:

INPUT: Two dimensional text T of size $n \times n$, and two dimensional pattern P of size $m \times m$.

OUTPUT: All locations $[i, j]$ in T where there is a parameterized (function) occurrence of the pattern.

First we show how to reduce two-dimensional function matching to the one-dimensional case, yielding an $O(n^2 \log n)$ work randomized algorithm. We then show how, with an additional $O(n^2 \log n)$ work, to solve two-dimensional parameterized matching, again with a randomized algorithm. Finally, we give a deterministic algorithm for parameterized matching, which takes $O(n^2 \log^2 m)$ time.

The two-dimensional text T is written in row major order to give a one-dimensional text T'. The pattern is padded with wildcards (or equivalently, new characters, each appearing once) to produce m rows of length n; the padded pattern is written in row major order to give a one-dimensional pattern P'. Clearly, there is a match of P at location (i, j) in T exactly if there is a match of P' at location $n(i-1)+j$ in T'. We have shown:

Corollary 1 *There is a randomized algorithm for two-dimensional function matching, which, when given a constant k, runs in time $O(kn^2 \log n)$, reports all function matches, and with probability at most $\frac{1}{n^k}$ falsely reports a mismatch as a match.*

In a parameterized match, the number of distinct symbols in the aligned portion of the text must equal the number of distinct symbols in the pattern. Amir, Church and Dar [3] gave an $O(n^2 \log n)$ time algorithm for this problem, the character count problem: determine, for each $m \times m$ subarray of an $n \times n$ array, the number of distinct characters appearing in the subarray. So we have shown:

Corollary 2 *There is a randomized algorithm for two-dimensional parameterized matching, which, when given a constant k, runs in time $O(kn^2 \log n)$, reports all parameterized matches, and with probability at most $\frac{1}{n^k}$ falsely reports a mismatch as a match.*

Next, we give an efficient deterministic algorithm for two-dimensional parameterized matching (time $O(n^2 \log^2 m)$). It uses one convolution on a size $O(n^2 \log m)$ vector, which can also be viewed as $O(\log m)$ convolutions on size n^2 vectors. As in one-dimension, this is considerably more efficient than what is known for function matching. Incidentally, we note that this convolution is outside the convolutions model.

It is helpful to recall the one-dimensional parameterized matching algorithm, due to Amir, Farach, and Muthukrishnan [7]. It is similar to the Knuth-Morris-Pratt string matching algorithm. The key idea is to recode the occurrences of each symbol in terms of their separation; namely, if symbol a occurs in the pattern (or text) at locations with indices $i_1 \leq i_2 \leq \ldots \leq i_k$, these occurrences of a are replaced by the numbers $0, i_2 - i_1, i_3 - i_2, \ldots, i_k - i_{k-1}$, respectively. For each symbol occurrence, this is simply the distance to the nearest occurrence of the same symbol to the left, if any. Except for the first occurrence of each symbol, a parameterized match in the original text and pattern corresponds to a standard match in the recoded text and pattern. One perspective on this is that all occurrences of the same symbol in the pattern (and the text) have been connected into a structure; identifying a match becomes a question of finding the alignments for which the structures in the pattern and text match. We will seek a similar construction for the two-dimensional problem. However, this will now require creating connections to $O(\log n)$ neighbors of each symbol occurrence.

Our solution has the following form. For each occurrence I of a symbol in the pattern (resp. in the text) the relative locations of some $8 \log n$ instances $J_1, J_2, \ldots$ of the same symbol (resp. in the pattern and text) are recorded. We say that J_ℓ is a neighbor of I, for $\ell = 1, 2, ...$, and also that I and J_ℓ are *linked.* If I is in position (w, y) and J_ℓ is in position (x, z), their relative position is recorded as $(x - w, z - y)$. Each potential neighbor is selected according to a specific rule, which may or may not identify a neighbor (e.g., a rule could be: the next occurrence of the symbol to the right in the same row). The rules for pattern and text will be slightly different.

The collections of selected symbols have the following properties:

(i) For each alignment of the pattern with the text, for each symbol a, if the occurrences of a in the pattern and text match, then for each occurrence I_p of a in the pattern and the aligned occurrence I_t in the text, the neighbor of I_p selected by the ith rule for the pattern is aligned with the neighbor of I_t selected by the ith rule for the text (the converse need not be true, however).

(ii) All occurrences of a in the pattern are linked, for each symbol a.

To see why the converse need not hold in Property (i) consider the rule "the next instance of this symbol to the right in the same row"; since the text may extend further to the right than the pattern, for a given alignment this rule could yield a symbol occurrence in the text and not in the pattern (the text symbol in question would be to the right of the pattern) and of course this does not preclude a match.

The text and pattern are recoded using the following encoding for each symbol occurrence. Each symbol occurrence is encoded by an equal length sequence of $O(\log n)$ relative positions, ordered as follows: the relative position of the symbol occurrence yielded by rule (1), by rule (2), by rule (3), and so on. If a rule yields no occurrence this is recoded by the "relative position" $(0,0)$. The matching problem is made one-dimensional by writing the recoded arrays in row major order using rows of length n (the missing entries in the pattern are replaced by sequences of $O(\log n)$ pairs $(0,0)$). Treating $(0,0)$ as a wildcard, it is easy to see that a parameterized match of the original pattern with the text corresponds exactly to a standard match with wildcards of the recoded pattern with the recoded text. The recoded text has length $O(n^2 \log m)$, thus this wildcard matching can be solved in time $O(n^2 \log m \log n)$ [9] (and by standard techniques this can be reduced to $O(n^2 \log^2 m)$ time).

We turn to the selection of neighbors. For the moment we suppose the pattern dimension $m = 2^i$ for some integer i. For each location (x,y) in the pattern, we divide most of the remainder of the pattern (excluding location (x,y)) into $8 \log n - 4$ disjoint rectangles. Each rectangle provides one neighbor.

The first four rectangles comprise row x and column y partitioned at location (x,y), i.e., (i) the points (x,z) with $z > y$, (ii) the points (x,w) with $w < x$, (iii) the points (z,y) with $z > x$, and (iv) the points (w,y) with $w < x$. Next, we describe how the quadrant below and to the right of (x,y) is divided into contiguous rectangles. Each rectangle comprises a distinct selection of contiguous rows, covering all columns from $y+1$ to m, starting at row $x+1$. From top to bottom, the sequence of rectangles have the following number of rows: $1, 2, 4, \ldots, m/4 = 2^{i-2}, m/4, m/8, \ldots, 2, 1, 1$, with the series stopping at the last rectangle that fits inside the pattern. This may mean that a portion of the quadrant is left uncovered.

Suppose a is the symbol in location (x,y). Each rectangle is traversed in column major order to find the first occurrence of an a, if any. These are the neighbors of the a in location (x,y). Analogous partitionings and traversals in directions away from location (x,y) are used for the other quadrants.

A very similar partitioning is used on the text, except that now each rectangle extends through $n-1$ columns or to the right boundary of the text, whichever comes sooner. (This is for the SE quadrant; other quadrants are handled similarly.)

Clearly Property (i) above holds. It remains to show Property (ii).

Lemma 2 *Let (w,y) and (x,z) be two locations in the pattern both holding symbol a. Then they are linked.*

Proof: Clearly, if $w = x$ there is a series of links along row x connecting these two locations. Similarly if $y = z$.

So WLOG suppose that $w < x$ and $y < z$. We claim that either (x,z) lies in one of the rectangles defined for location (w,y) or (w,y) lies in one of the rectangles for (x,z) (or possibly both). Suppose that $2^{k-1} < w \leq 2^k \leq m/2$. Then for (x,z) to lie outside one of (w,y)'s rectangles, $x > n - 2^{k-1}$ (for rows $w, w+1, [w+2, w+3], \ldots, [w+m/4, w+m/2-1], [w+m-m/2, w+m-m/4-1], \ldots, [w+m-2^{k+1}, w+m-2^k-1]$ are all included in (w,y)'s rectangles, and $w \geq 2^{k-1}+1$). The symmetric argument for location (x,z) shows that (w,y) lies in one of z's rectangles if $x > n - 2^{k-1}$. This argument does not cover the case $w = 1$, but then (w,y)'s rectangles cover every row, nor the case $w > m/2$, but then (x,z)'s rectangles cover row w.

WLOG suppose that (x,z) lies in one of (w,y)'s rectangles. It need not be that (x,z) is a neighbor of (w,y), however. Nonetheless, by induction on $z-y$, we show they are linked. The base case, $y = z$, has already been demonstrated. Let (u,v) denote the neighbor of (w,y) in the rectangle containing (x,z). Then $y < v \leq z$. By induction, (u,v) and (x,z) are linked and the inductive claim follows. $\square$

It remains to show how to identify the neighbors. This is readily done in $O(m^2 \log^2 m)$ time in the pattern and $O(n^2 \log n \log m)$ time in the text (and using standard additional techniques, in $O(n^2 \log^2 m)$) time). We describe the approach for the pattern.

The idea is to maintain, for each symbol a, a window of 2^i rows, for $i = 1, 2, \ldots, \log m - 2$,

and in turn to slide each window down the pattern. In the window the occurrences of a are kept in a balanced tree in column major order. For each occurrence of a, its neighbors in the relevant window are found by means of $O(\log m)$ time searches. Thus, over all symbols and neighbors the searches take time $O(m^2 \log^2 m)$. To slide a window one row down requires deleting some symbol instances and adding others. This takes time $O(\log m)$ per change and as each symbol instance is added once and deleted once from a window of each size this takes time $O(m^2 \log^2 m)$ over all symbols and windows. (It is helpful to have a list of each character in row major order so as to be able to quickly decide which characters to add and to delete from the sliding window, but these lists take only $O(m^2)$ time to prepare for all the symbols.)
The preprocessing of the text is similar.
To extend this algorithm to arbitrary n, we simply expand the pattern to size $2^i \times 2^i$ by padding it with wildcards. We have shown:

Theorem 5 *There is an $O(n^2 \log^2 m)$ time algorithm for two-dimensional parameterized matching.*

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 16(6):1039–1051, 1987.

[2] A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two dimensional pattern matching. *SIAM J. Comp.*, 23(2):313–323, 1994.

[3] A. Amir, K. W. Church, and E. Dar. Separable attributes: a technique for solving the submatrices character count problem. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 400–401, 2002.

[4] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.

[5] A. Amir, M. Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49:111–115, 1994.

[6] G.P. Babu, B.M. Mehtre, and M.S. Kankanhalli. Color indexing for efficient image retrieval. *Multimedia Tools and Applications*, 1(4):327–348, Nov. 1995.

[7] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th Annual ACM Symposium on the Theory of Computing*, pages 71–80, 1993.

[8] J. H. Bowie, R. Luthy, and D. Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253:164–176, 1991.

[9] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. *Proc. of 34th ACM Symposium on the Theory of Computing (STOC)*, pages 592-601, 2002.

[10] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[11] M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation, R.M. Karp (editor), SIAM-AMS Proceedings*, 7:113–125, 1974.

[12] Jr. K. Merz and S. M. La Grand. *The Protein Folding Problem and Tertiary Structure Prediction.* Birkhauser, Boston, 1994.

[13] W. C. Kreahling and C. Norris. Profile assisted register allocation. In *Proc. ACM Symposium on Applied Computing (SAC)*, volume 2, pages 774–781, Como, Italy, March 2000.

[14] G-Y. Lueh, T. Gross, and A-R. Adl-Tabatabai. Fusion-based register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(3):431–470, May 2000.

[15] S. Muthukrishnan and K. Palem. Non-standard stringology: Algorithms and complexity. In *Proc. 26th Annual Symposium on the Theory of Computing*, pages 770–779, 1994.

[16] M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[17] U. Vishkin. Optimal parallel pattern matching in strings. *Information and Control*, 67(1-3):91–113, Oct/Nov/Dec 1985.

[18] U. Vishkin. Deterministic sampling - a new technique for fast pattern matching. *SIAM J. Comp.*, 20:303–314, 1991.

[19] J. Yadgari, A. Amir, and R. Unger. Genetic algorithms for protein threading. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *Proc. 6th Int'l Conference on Intellingent Systems for Molecular Biology (ISMB 98)*, pages 193–202. AAAI, AAAI Press, 1998.

[20] A. C. C. Yao. Some complexity questions related to distributed computing. In *Proc. 11th ACM Symposium on the Theory of Computing (STOC)*, pages 209–213, 1979.

A Additional Examples of the Convolutions Model

1. *Exact String Matching with Don't Cares*
   Details within paper.

2. *Approximate Hamming Distance over a fixed bounded Alphabet*
   This problem was considered for unbounded alphabets in [1]. For bounded alphabets, the problem is defined in the convolutions model as follows. The matching relation $M_e(a,b)$ is all pairs of substrings over alphabet $\Sigma = \{1,...,k\}$ for which $|a| \leq |b|$ and the number of *mismatches* between a and b (i.e. the indices j for which $a_j \neq b_j$) is no greater than e.

   the solution in the convolutions model is as follows. Compute the convolutions:

   $C_i \leftarrow \chi_i(T) \otimes \chi_{\bar{i}}(P), \ i = 1, \cdots, k,$ where

   $$\chi_{\overline{a}}(x) = \begin{cases} 1 & \text{if } x \neq a \\ 0 & \text{if } x = a \end{cases}$$

   The match locations are all indices j where $\sum_{i=1}^{k} C_i[j] \leq e$.

3. *Less-than Matching over a fixed bounded Alphabet*
   This problem was considered for unbounded alphabets in [4]. For bounded alphabets, the problem is defined in the convolutions model as follows. The matching relation $M(a,b)$ is all pairs of substrings over alphabet $\Sigma = \{1,...,k\}$ for which $|a| \leq |b|$ and $a_j \leq b_j \ \forall j = 0,...,|a|-1$.

   the solution in the convolutions model is as follows. Compute the convolutions:

   $C_i \leftarrow \chi_i(T) \otimes \chi_{<i}(P)$, i=1,..., k, where

   $$\chi_{<i}(x) = \begin{cases} 1 & \text{if } x < i \\ 0 & \text{if } x \geq i \end{cases}$$

   The match locations are all indices j where $\sum_{i=1}^{k} C_i[j] \neq 0$.

4. Our solutions in Section 2 are also algorithms in the convolutions model.