Parameterized Matching on Non-linear Structures

Amihood Amir* Gonzalo Navarro†
Bar-Ilan University University of Chile
and
Johns Hopkins University

Abstract

The classical pattern matching paradigm is that of seeking occurrences of one string in another, where both strings are drawn from an alphabet set Σ. In the *parameterized pattern matching* model, a consistent renaming of symbols from Σ is allowed in a match. The parameterized matching paradigm has proven useful in problems in software engineering, computer vision, and other applications.

In classical pattern matching, both the text and pattern are strings. Applications such as searching in `xml` or searching in hypertext require searching strings in non-linear structures such as trees or graphs.

There has been work in the literature on exact and approximate parameterized matching, as well as work on exact and approximate string matching on non-linear structures. In this paper we explore parameterized matching in non-linear structures. We prove that exact parameterized matching on trees can be computed in linear time for alphabets in an $O(n)$-size integer range, and in time $O(n \log m)$ in general, where n is the tree size and m the pattern length. These bounds are optimal in the comparison model. We also show that exact parameterized matching on directed acyclic graphs (DAGs) is $\mathcal{NP}$-complete.

1 Introduction

The last few decades have prompted the evolution of pattern matching from a combinatorial solution of the exact string matching problem [13, 18] to an area concerned with approximate matching of various relationships motivated by computational molecular biology, computer vision, and complex searches in digitized and distributed multimedia libraries [12, 7].

The *parameterized matching* problem was introduced by Baker [10, 11]. Her main motivation lay in software maintenance, where program fragments are to be considered "identical" even if variable names are different. Therefore, strings under this model are comprised of symbols from two disjoint sets Σ and Π containing fixed symbols and parameter symbols respectively. In this paradigm, one seeks *parameterized occurrences*, i.e., occurrences up to renaming of the parameter symbols, of a string in another. This renaming is a bijection $b : \Pi \to \Pi$. An optimal algorithm for exact parameterized matching appeared in [5]. Approximate parameterized matching was investigated in [10, 15, 8]. Idury and Schäffer [16] considered multiple matching of parameterized patterns.

Parameterized matching has proven useful in other contexts as well. An interesting problems is searching for color images (e.g. [21, 9, 4]). Assume, for example, that we are seeking a given icon in any possible color map. If the colors were fixed, then this is exact two-dimensional pattern matching [3]. However, if the color map is different the exact matching algorithm would not find the pattern. Parameterized two dimensional search

*Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, +972 3 531-8770; `amir@cs.biu.ac.il`; and Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218. Partly supported by ISF grant 35/05 and an Israel-Korea bi-national research grant.

†Department of Computer Science, University of Chile, Blanco Encalada 2120, Santiago, Chile, +56 2 6892736; `gnavarro@dcc.uchile.cl`. Partially funded by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

is precisely what is needed. If, in addition, one is also willing to lose resolution, then a two dimensional function matching search should be used, where the renaming function is not necessarily a bijection [2].

Most of the above work is carried out in the traditional string matching model, where both the pattern and the text are strings (one dimensional arrays). The function matching work [2] considers both pattern and text as two dimensional arrays. However, there are applications where a string is sought in non-linear structures.

The importance of hypertext has been steadily growing over the last few decades. Internet and other information systems use hypertext format, with data organized associatively rather than sequentially or relationally. Thus, it is natural to consider pattern matching on hypertext. In contrast to regular text, hypertext has a non-linear structure and the techniques of pattern matching for text cannot be directly applied to hypertext.

Manber and Wu [19] pioneered the study of pattern matching in hypertext and defined a hypertext model for pattern matching. Akutsu [1] developed an algorithm that can be used for exact pattern matching in a tree-structured hypertext. Park and Kim [17] considered regular pattern matching in hypertext. They developed a complex algorithm that works for hypertext with an underlying structure of a DAG. Amir, Lewenstein and Lewenstein [6], developed an algorithm for pattern matching in any hypertext graph, and considered the problem of approximate pattern matching in hypertext. They showed that, in contrast to regular text, it does make a difference whether the errors occur in the hypertext or the pattern. The approximate pattern matching problem in hypertext with errors in the hypertext turns out to be $\mathcal{NP}$-Complete and the approximate pattern matching problem in hypertext with errors in the pattern has a polynomial time solution. Navarro [20] improved the polynomial complexity to $O(m(|V| + |E|))$ time and $O(|V|)$ space, for a graph $G = (V, E)$ with one letter per edge.

In this paper we investigate the natural combination of parameterized matching in hypertext. We present an optimal algorithm for pattern matching where the text is a tree, whose running time is $O(n \log \sigma)$, where n is the tree size, m is the pattern length, Σ is the alphabet of text node labels and pattern symbols, and $\sigma = \min(m, |\Sigma|)$. The time reduces to $O(n)$ if Σ is a range of integers of size $O(n)$. We then show that when the text is a directed acyclic graph (DAG), the problem is $\mathcal{NP}$-complete.

2 Problem Definition

We start by simplifying Baker's definition of parameterized pattern matching.

Definition 1 *(***Parameterized-Matching***) Let Σ be an alphabet set, $T = t[1] \cdots t[n]$ the* text *and $P = p[1] \cdots p[m]$ the* pattern*, $t[i], p[j] \in \Sigma, \quad i = 1, \ldots, n; j = 1, \ldots, m$. We say that P* parameterize-matches *or simply p-*matches *T in location j if $p[i] \cong t[j + i - 1], \quad i = 1, \ldots, m$, where $p[i] \cong t[j]$ if and only if the following condition holds:*

1. *for every $k = 1, \ldots, i - 1, \quad p[i] = p[i - k]$ if and only if $t[j] = t[j - k]$.*

*The p-*matching problem *is to determine all p-matches of P in T. Two strings S_1 and S_2 of same length are said to* parametrize-match *or simply p-*match *if $S_1[i] \cong S_2[i]$ for all i.*

Intuitively, the matching relation $\cong$ captures the notion of one-to-one mapping between the alphabet symbols. Specifically, the condition in the definition of $\cong$ ensures that there exists a bijection between the symbols from Σ in the pattern and those in the overlapping text, when they p-match. The relation $\cong$ has been defined by [5] in a manner suitable for computing the bijection.

We follow Manber and Wu [19] in defining string matching in hypertext.

Definition 2 *(***hypertext***) A* hypertext *over alphabet Σ is a triplet $H = (V, E, T)$ where (V, E) is a digraph and $T = \{T_v \in \Sigma^+ \mid v \in V\}$. If for every $v \in V$, $T_v \in \Sigma$ then we call the hypertext a* one-character hypertext*.*

It was shown [19, 5] that hypertext matching is equivalent to one-character hypertext matching, thus we will henceforth only consider one-character hypertext. For simplicity's sake we will refer to "one-character hypertext" as "hypertext".

Definition 3 *(***hypertext matching***) Let $v_1, ..., v_k$ be a path, possibly with loops, in (V, E) and let $W = T_{v_1}T_{v_2}T_{v_3}...T_{v_{k-2}}T_{v_{k-1}}T_{v_k}$ be the concatenation of the symbols upon this path. We call W the* string defined by path $v_1, ..., v_k$.

The Pattern Matching in Hypertext *problem is defined as follows:*
INPUT: *A pattern P and a hypertext H.*
OUTPUT: *All locations $v \in V$ where $P = W$ and W is a string defined by a path starting at v.*

The Parameterized Matching in Hypertext *problem is defined as follows:*
INPUT: *A pattern P and a hypertext H.*
OUTPUT: *All locations $v \in V$ where P p-matches W, and W is a string defined by a path starting at v.*

3 Parameterized Hypertext Matching on Trees

Throghout this section we assume that our trees are directed graphs where the parent points to the children; we consider the alternative case at the end. A crucial feature of trees is that each node has a unique ancestor path in the tree. This property leads to the following results.

Observation 1 *Given a deterministic finite automaton A and a tree of n nodes, it is possible to determine in time $O(n)$ all the tree nodes v for which the string labeling the path from the root to v is accepted by A.*

Proof: Compute, in constant time per tree node v, the state $s(v)$ of A upon reading the string $T_{root} \dots T_v$. Set A at the initial state and and carry out a depth-first traversal starting at the root. Upon arriving at each node v for the first time, with label T_v, set $s(v) = \delta_A(s(parent(v)), T_v)$. ∎

Corollary 1 *Exact string matching in hypertext can be performed in linear time when the hypertext structure is a tree.*

Proof: This is an immediate result of Observation 1, since the Knuth-Morris-Pratt (KMP) algorithm [18] is an automaton-based algorithm. ∎

Amir, Farach, and Muthukrishnan [5] achieved an optimal time algorithm for parameterized string matching by a modification of the KMP algorithm. In fact, the algorithm is exactly the KMP algorithm, however, every equality comparison "$x = y$" is replaced by "$x \cong y$" as defined below.

Implementation of "$x \cong y$"

Construct table $A[1], \dots, A[m]$ where $A[i]$ = the largest k, $\quad 1 \le k < i$, such that $p[k] = p[i]$. If no such k exists then $A[i] = i$.

The following subroutines compute "$p[i] \cong t[j]$" for $j \ge i$ (needed for the text scanning phase of KMP), and "$p[i] \cong p[j]$" for $j \le i$ (needed for the pattern preprocessing phase of KMP).

```
Compare(p[i],t[j])
    if A[i] = i and t[j] ≠ t[j − 1], . . . , t[j − i + 1] then return equal
    if A[i] ≠ i and t[j] = t[j − i + A[i]] then return equal
    return not equal
end

Compare(p[i],p[j])
    if i − A[i] > j − 1 and p[j] ≠ p[1], . . . , p[j − 1] then return equal
    if i − A[i] ≤ j − 1 and p[j] = p[j − i + A[i]] then return equal
```

return *not equal*
end

Theorem 1 *[5] The p-matching problem can be solved in $O(n \log \sigma)$ time, where $\sigma = \min(m, |\Sigma|)$.*

Proof: The table A can be constructed in $O(m \log \sigma)$ time as follows: scan the pattern left to right keeping track of the distinct symbols from Σ in the pattern in a balanced tree, along with the last occurrence of each such symbol in the portion of the pattern scanned thus far. When the symbol at location i is scanned, look up this symbol in the tree for the immediately preceding occurrence; that gives $A[i]$.

Compare can clearly be implemented in time $O(\log \sigma)$. For the case $A[i] \neq i$, the comparison can be done in time $O(1)$. When scanning the text from left to right, keep the last m symbols in a balanced tree. The check $t[j] \neq t[j-1], \ldots, t[j-i+1]$ in Compare($p[i]$,$t[j]$) can be performed in $O(\log \sigma)$ time using this information. Similarly, Compare($p[i]$,$p[j]$) can be performed using $A[i]$. Therefore, the automaton construction in KMP algorithm with every equality comparison "$x = y$" replaced by "$x \cong y$" takes time $O(m \log \sigma)$ and the text scanning takes time $O(n \log \sigma)$, giving a total of $O(n \log \sigma)$ time.

As for the algorithm's correctness, Amir, Farach and Muthukrishnan showed that the failure link in automaton node i produces the largest prefix of $p[1] \cdots p[i]$ that p-matches the suffix of $p[1] \cdots p[i]$. ∎

Theorem 2 *The p-matching problem in hypertext can be solved in $O(n \log \sigma)$ time, where $\sigma = \min(m, |\Sigma|)$, and where the hypertext is a tree.*

Proof: Corollary 1 guarantees that KMP can be performed on the tree in linear time. We need to assess the time it takes to perform the "$x \cong y$" operation on the tree.

For the "$x \cong y$" operation, we need to be able to access previous symbols $t[j-k]$, for $k = 1 \ldots m$, both for the case $A[i] \neq i$ and for maintaining the set of the last m characters seen. These symbols are recorded in a circular array of size m, and maintained in the obvious way as we move down and back in the tree. These are indeed the same symbols stored in the balanced tree, which must also be maintained up to date to handle the case $A[i] = i$. ∎

Theorem 3 *The $O(n \log \sigma)$ time algorithm for solving the p-matching problem in hypertext, where $\sigma = \min(m, |\Sigma|)$, and where the hypertext is a tree, is optimal.*

Proof: Amir, Farach, and Muthukrishnan [5] proved that the algorithm is optimal for the special case of a string – a non-branching tree. ∎

We note that the problem is solved in $O(n)$ time if the alphabet is integers smaller than cn for some fixed constant c. The algorithm can then use a table indexed by alphabet symbols, in place of the binary tree. This enables Compare to be performed in time $O(1)$ at the cost of $O(\sigma)$ additional space.

We also note that the problem statement refers to initial positions of matching, whereas we are finding final positions. Since all the matches have fixed length m, we can easily mark a match at the m-th level ancestor of the node instead of at the node. For this sake we need to maintain a circular array of pointers to the last m tree nodes, not only to their symbols.

Similarly, if the tree is a directed graph where children point to their parent (instead of the other way) we can first reverse all arrows and then run the original algorithm for the reverse pattern $P^R = p[m] \ldots p[2]p[1]$, which will correctly mark the beginning of matches. The correctness of this approach stems from the fact that P p-matches $t[i]t[i+1] \ldots t[i+m-1]$ iff P^R p-matches $t[i+m-1] \ldots t[i+1]t[i]$.

4 Parameterized Hypertext Matching on a DAG

Theorem 4 *The Parameterized Hypertext Matching on a DAG problem is $\mathcal{NP}$-hard.*

We reduce the 3-dimensional matching problem [14] to Parameterized Hypertext Matching on a DAG.

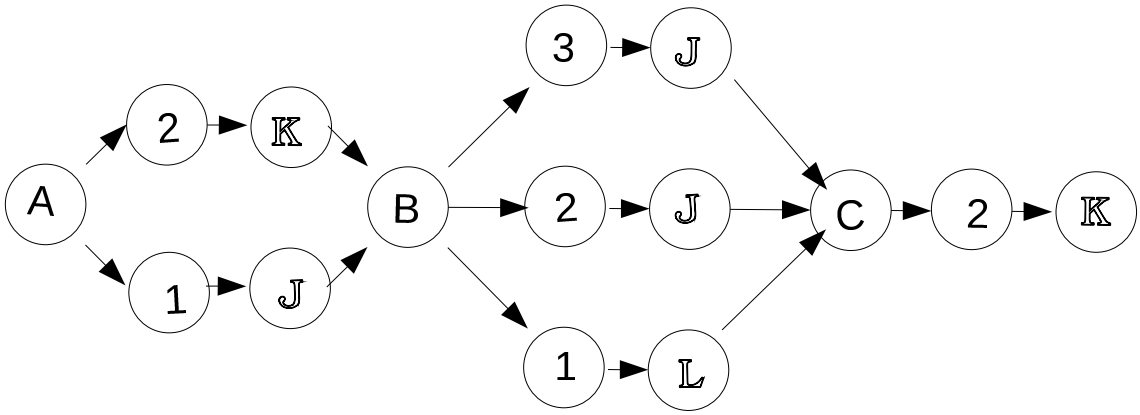

Figure 1: The constructed DAG.

Definition 4 *The* 3 Dimensional Matching problem*: (3DM)*
INPUT: *Three sets X, Y, Z of q elements each, and a set $M \subseteq X \times Y \times Z$.*
OUTPUT: *Is there a set $M' \subseteq M$ of size q where no two elements in M' agree in any coordinate. Such a set is called a* matching

Let $X = \{x_1, ..., x_q\}$; $Y = \{y_1, ..., y_q\}$; $Z = \{z_1, ...z_q\}$. Given set $M \subseteq X \times Y \times Z$, construct a labeled DAG $G = (V, E)$ as follows:

The alphabet: $\Sigma = X \cup Y \cup Z$. Assume the three sets are disjoint; otherwise they can be marked to enforce disjointness.

The vertex set V: For every triple $t_k = \langle x, y, z\rangle \in M$ define two new nodes v_k and w_k, and label them y and z respectively. In addition, define q nodes $u_1, \ldots, u_q$ labeled $x_1, \ldots, x_q$, respectively. Altogether we have $q + 2|M|$ nodes in V (of course, some different nodes v_k or w_k may have the same alphabet symbols).

The Edges: For every triple $t_k = \langle x_i, y, z\rangle \in M$, $i = 1, ..., q-1$, put the edges: $\overline{u_i v_k}$, $\overline{v_k w_k}$, and $\overline{w_k u_{i+1}}$. For the triples $t_k = \langle x_q, y, z\rangle$, put the edges $\overline{u_q v_k}$ and $\overline{v_k w_k}$.

Example: Let $X = \{A, B, C\}$, $Y = \{1, 2, 3\}$, and $W = \{\mathbb{J}, \mathbb{K}, \mathbb{L}\}$. Let $M = \{\langle A, 1, \mathbb{J}\rangle, \langle A, 2, \mathbb{K}\rangle, \langle B, 1, \mathbb{L}\rangle, \langle B, 2, \mathbb{J}\rangle, \langle B, 3, \mathbb{J}\rangle, \langle C, 2, \mathbb{K}\rangle\}$. Then the constructed DAG G appears in Figure 1.

It is clear that the construction is polynomial.

Note that any path through a node labeled with a symbol $x \in X$ traverses through at most one triple of M that begins with x. Observe also that the longest path in G is of length $3q$. It starts at the source (u_1), and its first 3 elements are one triple whose first coordinate has x_1, it is followed by a triple whose first coordinate is x_2, and so on until it ends with a triple whose first coordinate is x_q.

Now consider the string $P = x_1\ y_1\ z_1\ x_2\ y_2\ z_2 \cdots x_q\ y_q\ z_q$. Clearly, there is a parameterized match of P in G iff there is a path from the source with no repeating symbol iff there are q different triples where all first elements are different, all second elements are different and all third elements are different iff there is a 3-d matching on M. ∎

5 Conclusions

Exact string matching can be effectively done in general hypertext files. Parameterized string matching is also efficiently computable. In this paper we have shown that while parameterized pattern matching can be done in hypertext in optimal $O(n \log \sigma)$ time for tree graphs, where $\sigma = \min(m, |\Sigma|)$, as soon as the structure of the hypertext becomes more complex, i.e. a directed acyclic graph, parameterized matching is already $\mathcal{NP}$-hard.

References

[1] T. Akutsu. A linear time pattern matching algorithm between a string and a tree. In *Proc. 4th Symp. on Combinatorial Pattern Matching (CPM)*, pages 1–10, Padova, Italy, 1993.

[2] A. Amir, A. Aumann, M. Lewenstein, and E. Porat. Function matching. *SIAM Journal on Computing*, 35(5):1007–1022, 2006.

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

[4] 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.

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

[6] A. Amir, N. Lewenstein, and M. Lewenstein. Pattern matching in hypertext. *J. of Algorithms*, 35:82–99, 2000.

[7] A. Apostolico and Z. Galil (editors). *Pattern Matching Algorithms*. Oxford University Press, 1997.

[8] A. Apostolico, M. Lewenstein, and P. Erdös. Parameterized matching with mismatches. *Journal of Discrete Algorithms*, 5(1):135–140, 2007.

[9] 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.

[10] B. S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28–42, 1996.

[11] B. S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26(5):1343–1362, 1997.

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

[13] 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.

[14] 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.

[15] C. Hazay, M. Lewenstein, and D. Sokol. Approximate parameterized matching. In *Proc. 12th Annual European Symposium on Algorithms (ESA 2004)*, pages 414–425, 2004.

[16] R.M. Idury and A.A Schäffer. Multiple matching of parameterized patterns. In *Proc. 5th Combinatorial Pattern Matching (CPM)*, volume 807 of *LNCS*, pages 226–239. Springer-Verlag, 1994.

[17] D.K. Kim and K. Park. String matching in hypertext. *Proc. 6th Symposium on Combinatorial Pattern Matching (CPM 95)*, 1995.

[18] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.

[19] U. Manber and S. Wu. Approximate string matching with arbitrary cost for text and hypertext. In *Proc. Int'l Workshop on Structural and Syntactic Pattern Recognition*, pages 22–33, 1992.

[20] G. Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237:455–463, 2000.

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