Tree Pattern Matching to Subset Matching in Linear Time*

Richard Cole† Ramesh Hariharan‡

Abstract

This paper is the first of two papers describing an $O(n\text{polylog}(m))$ time algorithm for the *Tree Pattern Matching* problem on a pattern of size m and a text of size n. In this paper, we show an $O(n+m)$ time Turing reduction from the Tree Pattern Matching problem to another problem called the *Subset Matching* problem. The second paper will give efficient deterministic and randomized algorithms for the Subset Matching problem. Together, these two papers will imply an $O\left(n\log^3 m + m\right)$ time deterministic algorithm and an $O\left(n\frac{\log^3 m}{\log\log m} + m\right)$ time randomized algorithm for the Tree Pattern Matching problem.

1 Introduction

In the Tree Pattern Matching problem, the text and the pattern are ordered, binary trees and all occurrences of the pattern in the text are sought. Here, the pattern occurs at a particular text position if placing the pattern with root at that text position leads to a situation in which each pattern node overlaps some text node. This problem is an important problem and has many applications (see [7]). Actually, in these applications, the tree need not be binary and the edges may be labelled; however, as shown in [5], this general problem can be converted to a problem on binary trees with unlabelled edges but with a blow-up in size proportional to the logarithm of the size of the pattern. In fact, this blow-up can also be avoided in our approach, as we will indicate in our description.

The naive algorithm for Tree Pattern Matching takes time $O(nm)$, where n is the text size and m is the pattern size. Hoffman and O'Donell [7] gave another algorithm with the same worst case bound. This algorithm decomposes the pattern into strings, each string representing a root-to-leaf path. It then finds all occurrences of each of these strings in the text tree. The first $o(nm)$ algorithm was obtained by Kosaraju [9] who first noticed the connection of the Tree Pattern Matching problem to the problem of String Matching with Don't-Cares and the problem of convolving two strings. Kosaraju's algorithm takes $O(nm^{.75}\log m)$ time. Dubiner, Galil and Magen [5] improved Kosaraju's algorithm by discovering and exploiting periodicities in paths in the pattern. They obtained a bound of $O(nm^{.5}\log m)$. This was the best bound known to date. Dubiner, Galil and Magen also made the observation that the naive algorithm actually takes $O(nh)$ time, where h is the height of the pattern.

In this paper, we show how to reduce the Tree Pattern Matching problem to the *Subset Matching* problem in linear time. The Subset Matching problem is to find all occurrences of a

*This work was supported in part by NSF grants CCR9202900, CCR9503309, CCR9800085. An abstract of this work appeared in the Proceedings of the 29th ACM Symposium on Theory of Computing, 1997.

†Courant Institute, New York University, cole@cs.nyu.edu.

‡Indian Institute of Science, Bangalore, ramesh@csa.iisc.ernet.in. This work was done in part while visiting NYU.

pattern string p of length m in a text string t of length n, where each pattern and text location is a set of characters drawn from some alphabet. The pattern is said to occur at text position i if the set $p[j]$ is a subset of the set $t[i+j-1]$, for all j, $1 \leq j \leq m$. It is required to find all text locations at which the pattern matches, i.e., each pattern set is a subset of the aligned text set (see Fig.1).

The reduction from Tree Pattern Matching to Subset Matching proceeds in two steps.

• We show that the general Tree Pattern Matching problem can be reduced to the following special case, called *Spine Pattern Matching*, by a linear time Turing reduction. In Spine Pattern Matching, there is a special path in each of the pattern and text called their spines. The spine begins at the root of its tree, and in addition each node on the spine has at most one non-spine child. Spines have additional properties as well, which will be described later. All matches of the pattern in the text are sought with the additional restriction that the spine of the pattern must match a portion of the spine of the text, i.e., nodes on the pattern spine must be aligned with nodes on the text spine. For intuition, one can think of the spine as being the path of left children starting at the root (and in fact one can reduce the general problem to this case in linear time, although we will not do so).

The above reduction may create several instances of the Spine Pattern Matching problem, but the sum of the sizes of these instances will be linear. This reduction is completely deterministic. It proceeds by using the periodicity structure of paths and by decomposing the text tree into periodic paths in a non-trivial manner. Each path then gives a spine for the Spine Pattern Matching problem.

• Next, we reduce the Spine Pattern Matching problem to the Subset Matching problem in linear time. This is, in fact, readily done. The spine of the text tree gives the text string for the Subset Matching problem; the subtrees hanging from this spine determine the various text sets. Analogous facts hold for the pattern.

The two reductions above imply that that the Tree Pattern Matching problem can be reduced to several instances of the Subset Matching Problem, the sum of the sizes of these instances being linear. Therefore, an algorithm for the Subset Matching problem yields an algorithm for the Tree Pattern Matching problem with the same time complexity.

Cole and Hariharan [2] gave a randomized algorithm for the Subset Matching problem running in time $O\left((n+s)\log^3 m\right)$, where s is the sum of the sizes of all the pattern and text sets. Subsequently, Indyk [8] gave a deterministic algorithm for the Subset Matching problem running in time $O\left((n+s)m^{\sqrt{\frac{\log\log m}{\log m}}(1+o(1))}\right)$. Finally, Cole, Hariharan and Indyk [3] gave a deterministic algorithm running in time $O\left((n+s)\log^3 m\right)$ and a randomized algorithm running in time $O\left((n+s)\frac{\log^3 m}{\log\log m}\right)$. The above algorithms will be described in a companion paper [4]. It follows that there is a deterministic algorithm running in time $O(n\log^3 m)$ and a randomized algorithm running in time $O\left(n\frac{\log^3 m}{\log\log m}\right)$ for the Tree Pattern Matching problem.

This paper is organized as follows. Section 2 gives some required definitions. Section 3 describes the reduction of the Spine Pattern Matching Problem to the Subset Matching problem. Section 4 describes the reduction from the Tree Pattern Matching problem to the Spine Pattern Matching problem.

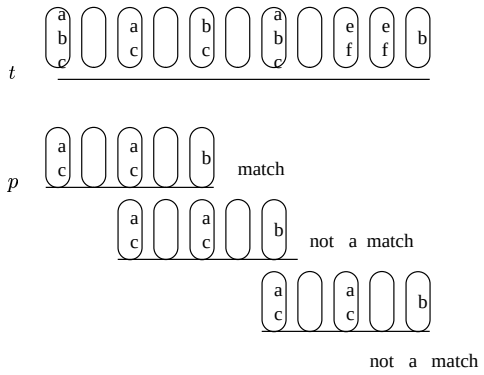

Figure 1: Example of Subset Matching.

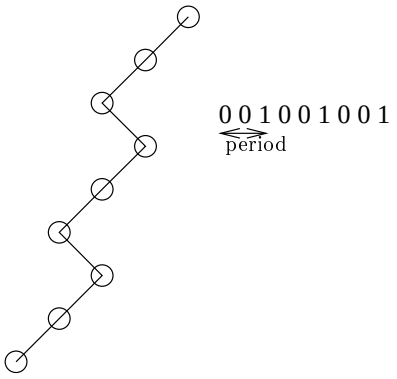

Figure 2: A Path and its Associated String

2 Definitions

Tree Pattern Matching: We consider ordered binary trees, i.e., each internal node has a left and/or a right child. The text tree t has n nodes and the pattern tree p has m nodes. The problem entails finding all nodes v in t where p matches, i.e., when the root of p is aligned with v, each node in p is aligned with a node in t.

Paths, Strings and Periods. Note that paths in trees p and t can be expressed as strings over a two character alphabet, one character signifying a left edge and the other a right edge (see Fig.2: 0 represents a left edge and 1 a right edge). The *period* of a string $s[1 \ldots |s|]$ is the smallest number $j > 0$ such that $s[i] = s[i+j]$, for all i, $1 \leq i \leq |s| - j$. If no such j exists then the period of s is defined to be $|s|$. The *period* of a path is defined to be the period of its associated string. The following lemma is classical [10].

Lemma 2.1 *If $k \leq |s| - j$ is such that the period j of s does not divide k, then the string $s[k+1 \ldots k+j]$ differs from the string $s[1 \ldots j]$.*

Spine Pattern Matching: This is a restricted version of the Tree Pattern Matching problem. In this problem, the text and the pattern each have one designated path, called their *spines*. The text and pattern spines originate at their respective roots and are maximal paths having the same period, θ say (the θ needed for Tree Pattern Matching will be determined later). In fact, both spines when represented as strings will have the form $x^k x'$, where $|x| = \theta$ and x' is a prefix of x (here, the values of k and x' could differ for the pattern spine and the text spine but x is identical for both spines). All matches of the pattern in which the pattern spine falls completely on the text spine are sought.

From maximality, it follows that both spines terminate at nodes with at most one child (a child which when added to the spine destroys its periodic structure). Since both spines have the same period θ, it follows that the pattern spine will fall completely on the text spine only if the root of the pattern is placed at certain nodes on the text spine. These nodes will occur at integer multiples of θ from the text root and will be designated "anchor nodes". Anchor nodes will have further restrictions which will be described later.

3 Reducing Spine Pattern Matching to Subset Matching

The spines of the pattern p and the text t will define the strings for the Subset Matching problem. The subsets at each location in these strings will correspond to the off-spine subtrees of the spine nodes; an *off-spine subtree* is a subtree whose root is a non-spine node but the parent of whose root is on the spine. These subsets are obtained by labelling the nodes of the off-spine subtrees as follows (see Fig.3). The key fact about this labelling is that two nodes in two distinct off-spine subtrees (both of which could be in the pattern or in the text, or alternatively, one could be in the pattern and the other in the text) get the same label if and only if the paths from these nodes to the roots of their respective off-spine subtrees represent identical strings.

The off-spine subtrees of p are labelled first. The subtrees are overlaid to form a *combined pattern subtree*; the overlaying aligns the roots of the off-spine subtrees and recursively overlays their subtrees. Then the combined pattern subtree is traversed by any convenient method, e.g. a breadth first traversal, and the nodes are labelled by the associated numbering. For each spine node, we form a subset consisting of the collection of numbers labelling the nodes of its off-spine subtree. This collection of subsets defines the pattern for the Subset Matching problem instance. The off-spine subtrees of t are labelled using the same labelling. To do this, each off-spine text subtree and the combined pattern subtree are traversed in lock-step[1]. We state the following easy fact about the time complexity of the above computation.

Fact 1 *The labels to nodes in off-spine subtrees of the pattern can be given in $O(m)$ time. The labels to any one off-spine subtree t' in the text can be given in time $O(\min\{|t'|, m\})$. The total time taken for the labelling is thus $O(n + m)$; consequently, the size of the resulting Subset Matching problem is also $O(n + m)$.*

Clearly, each match in the instance of the Subset Matching Problem beginning at a location corresponding to an anchor node has a corresponding match in the instance of the Spine Pattern Matching problem and conversely. This completes the reduction from Spine Pattern Matching to Subset Matching.

[1]Recall our remark from the introduction that the case of larger degree and labelled trees can be handled without any extra overhead. Larger degree is simply handled by the usual binarization. Labelled trees are handled by pairing the given labels with the labels obtained here.

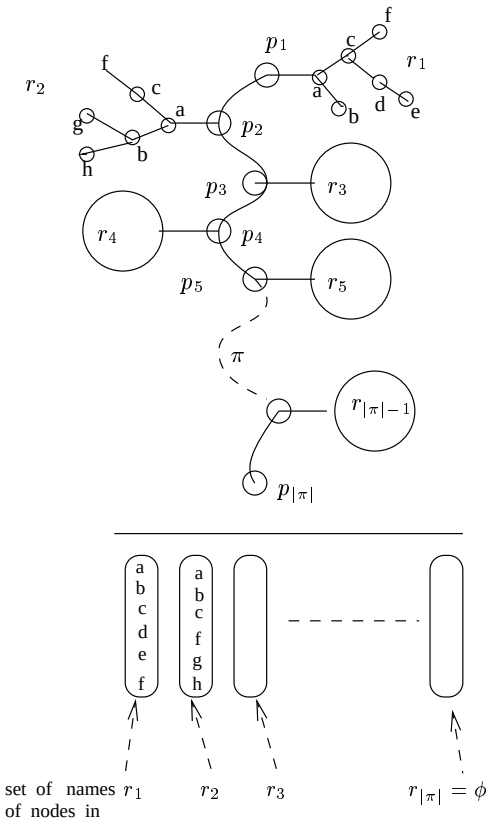

Figure 3: The Spine and its Associated Set String.

4 Reducing Tree Pattern Matching to Spine Pattern Matching

First, we identify a particular path π as the spine of p. Let π have period θ. Next, we decompose t into maximal paths with period θ; as we will show, there are $O\left(\frac{n}{m}\right)$ such paths of total length $O(n)$. We obtain a tree for each such path in a manner to be described. The sum of the sizes of these trees will also be $O(n)$. Solving the Spine Pattern Matching problem for p and each of the trees obtained from t will suffice to determine all occurrences of p in t.

Definitions. The *size* of a node v in a tree is defined to be the number of nodes in the subtree rooted at v. Let t_v denote the subtree of t rooted at a node v in t and let p_v denote the subtree of p rooted at a node v in p.

4.1 Processing the Pattern

The Spine of the Pattern. We define the spine π of the pattern p to be the following path from the root to a node with at most one child. π consists of two segments, π_1 and π_2. π_1 is a centroid path, i.e., it is obtained by moving to the child with larger size at each step, with ties broken arbitrarily. π_1 ends when a node x such that $|p_x| \leq \frac{m}{2}$ is reached. Note that $|p_x| \geq \frac{m}{4}$. Let θ be the period of π_1. π_2 is the longest path starting at x such that the path π continues to have period θ. Note that π_2 has a vertex in common with π_1.

4.2 Decomposing the Text

Definitions. A path in t from a node u to a node v in t_u is a *θ-path* if it has period θ and is identical to the spine of the pattern in the first θ locations (when both paths are viewed as strings). This path is *maximal* if extending it to the distance θ ancestor of u or either child of v results in a path which is not a θ-path (in fact, v can have only one child). The *link* node l in this path is the node closest to v such that $|t_l| \geq \frac{m}{4}$. The *anchor* nodes w on this path satisfy the following properties. These properties will be justified shortly.

1. t_w has at least m nodes.
2. The distance from w to l is at least $|\pi_1|$, and thus has length at least θ.
3. The distance from w to v is at least $|\pi|$.
4. Consider the subtrees hanging from the maximal *θ-path* starting at w. Classify them as *red* subtrees if they have at least $\frac{m}{4}$ nodes and as *green* subtrees otherwise. If all these subtrees except exactly one are green, then the green subtrees plus the path together have at least $m/2$ nodes.
5. The distance from u to w is an integer multiple of θ.

We form a collection C of maximal θ-paths in t. This collection comprises all the paths whose start nodes (which are anchor nodes) satisfy properties 1–4 above.

Note that these paths need not be disjoint; however their combined length will still be $O(n)$ as we shall show later, in Lemma 4.11 and Corollary 4.3. The algorithm for constructing these paths is given next.

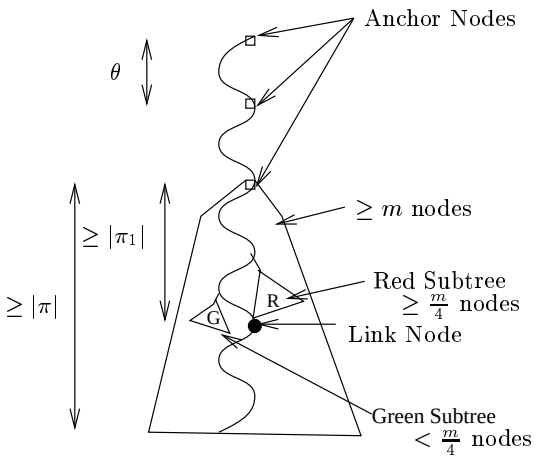

Figure 4: A θ-Path in C.

The Path Decomposition Algorithm. The decomposition is obtained using the following algorithm. For each node x in T, this algorithm first determines the longest θ-path which begins at x. This is done in $O(n)$ time using a Knuth-Morris-Pratt type automaton in conjunction with a depth-first traversal of t as in the algorithm of Hoffman and O'Donell [7]. Next, the algorithm determines those maximal θ-paths found above which satisfy Properties 1–4, discarding all other paths. In fact, this filtering can easily be done directly on the fly when the paths are being determined in $O(n)$ time. The details are left to the reader. Finally, the algorithm traverses the paths in C and determines anchor nodes satisfying the required properties. Since, as we will see, the sum of the length of paths in C is $O(n)$, the total time taken above is $O(n)$.

Significance of Anchor Nodes on θ-Paths. Note that each node w is an anchor node on some path in C except in the following four situations. In each of these situations, the pattern cannot match at w.

1. $|t_w| < m$.
2. The longest θ-path in t_w starting at w and ending at some node w' with the property that $|t_{w'}| \geq \frac{m}{4}$ is shorter than π_1.
3. The longest θ-path in t_w starting at w is shorter than π.
4. The maximal θ-path in t_w starting at w satisfies the following: the subtrees hanging from the path are all green with exactly one exception, and the sum of the sizes of these green subtrees and the length of the path itself is less than $m/2$. The pattern cannot match at w if this condition holds. This can be seen as follows. For the pattern to match at w, there must be at most one subtree hanging from π in p which can be red. By the centroid nature of π_1 and the stopping condition for π_1, this subtree can have size at most $m/2$. Therefore, π and all other subtrees hanging from it together have size at least $m/2$.

Thus determining matches of p at anchor nodes on paths in C suffices to determine all matches of p in t. Further, note that when p is placed with its root at an anchor node on some path in C, the spine of p lies completely on that path.

4.3 Processing Paths in C.

The purpose of processing a path $\rho \in C$ is to determine whether or not p matches at w, for each anchor node w on ρ. Each path ρ in C will be processed as follows.

Let u be the node at which ρ starts. u itself is an anchor node. Whether or not the pattern matches at u is determined in a brute force manner. This requires $O(m)$ time. We will show in Lemma 4.11 that there are $O(n/m)$ paths and hence the total time taken over all paths in this process is just $O(n)$.

Matches at other anchor nodes on ρ are determined differently, i.e., by reduction to an instance of the Spine Pattern Matching problem.

Consider the portion of ρ starting from the second anchor node onwards, denoted $trunc(\rho)$. $trunc(\rho)$ provides the spine of the text instance. Clearly, there is a match of p rooted at an anchor node on $trunc(\rho)$, if and only if there is a match at the same location in the corresponding Spine Pattern Matching problem instance.

Let $s_1, \ldots, s_{|\rho|-\theta}$ denote the off-spine subtrees, if any, for $trunc(\rho)$, in increasing order of distance from the start node of ρ. Some of the s_i's might not exist. By Fact 1, reducing this instance of the Spine Pattern Matching problem to the Subset Matching problem takes time $O(\sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$ (plus, of course, $O(m)$ time for processing the pattern, which is common to all the instances of the Spine Pattern Matching problem which result above).

The total time taken to process ρ is thus $O(m + \sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$. This quantity can be split into 4 parts: $O(m)$ time for checking for an occurrence of p at the first anchor node, time proportional to its size for each green subtree hanging from $trunc(\rho)$, $O(m)$ time for each red subtree hanging from $trunc(\rho)$, and $O(|\rho| - \theta)$ time for the path itself. We need to show that this sums to $O(n)$ over all paths ρ. By Lemma 4.11, there are $O(n/m)$ paths; hence the first part sums to $O(n)$ time. By Corollary 4.4, the green subtrees in the truncated paths are disjoint; hence the second part sums to $O(n)$. By Corollary 4.12, there are $O(n/m)$ red subtrees, hence the third part sums to $O(n)$. Finally, by Corollary 4.3, the truncated path lengths sum to $O(n)$, and hence the fourth part sums to $O(n)$ also. This yields $O(n)$ time overall.

4.4 Showing $O(n)$ Time

Eventually, we will seek to bound the number of red subtrees over all paths in C. We will do this by identifying a set of $O(n/m)$ nodes of t, called *marked* nodes. Each red subtree will be assigned to either a marked node or a path in C, and each marked node and each path will receive at most a constant number of red subtrees. We will also show in Section 4.4.1 that there are $O(n/m)$ paths in C; it then follows that there are $O(n/m)$ red subtrees.

Marked Text Nodes. We mark the following nodes in t: those nodes whose left and right subtrees both contain at least $\frac{m}{4}$ nodes.

Lemma 4.1 *The number of marked nodes in t is $O(\frac{n}{m})$.*

Proof. There are only $O(\frac{n}{m})$ marked nodes v with the property that all nodes in either the left subtree of v or the right subtree of v are unmarked; this is because both these subtrees have at least $\frac{m}{4}$ nodes. The number of marked nodes v such that both the left subtree of v and the right subtree of v contain marked nodes is at most 1 less than the number of marked nodes without this property. The lemma follows. □

Some Properties of Paths in C.

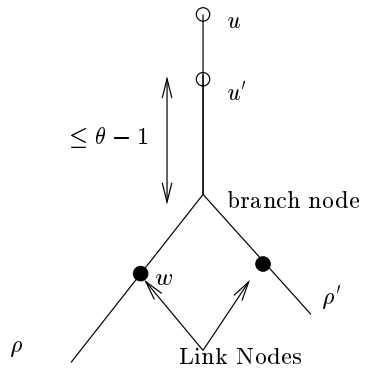

Figure 5: Overlap is at most $\theta - 1$.

Lemma 4.2 *Consider two paths ρ, ρ' in C starting at nodes u and u', respectively (see Fig.5). Suppose u' lies on ρ. Then only the first $\theta - 1$ edges of ρ' can also be present in ρ.*

Proof. From the construction of C, the length of the path between u and u' is not divisible by θ. The lemma then follows from Lemma 2.1. □

Corollary 4.3 *If the first θ edges are removed from each path in C then the resulting collection of paths is node disjoint. Hence the truncated paths have total lengths $O(n)$. Also, as the link node is not among the first θ nodes by Property 2 of paths in C, the link node of ρ' cannot lie on ρ.*

Corollary 4.4 *The green subtrees hanging from the truncated paths are all disjoint.*

Proof. It suffices to consider the green subtrees hanging from two paths ρ, $\rho' \in C$, starting at u, u', respectively, where u is a proper ancestor of u' (for if u and u' are unrelated then clearly the green trees hanging from ρ and ρ' are disjoint.) By Corollary 4.3, $trunc(\rho)$ and $trunc(\rho')$ are disjoint. For a contradiction, suppose that G is a green subtree hanging from $trunc(\rho)$ and containing v, a node in a green subtree hanging from $trunc(\rho')$. It follows from Lemma 4.2 that $trunc(\rho')$ lies within G. But $trunc(\rho')$ includes the link node l' of ρ' and the subtree of t rooted at l' contains at least $m/4$ nodes. Then G, which contains this subtree, would be red. □

4.4.1 Showing $|C| = O(\frac{n}{m})$.

Lemma 4.5 *Let ρ, ρ' be as in Lemma 4.2 (see Fig.5). Then ρ' cannot overlap a node in ρ which is a proper descendant of ρ's link node w. Therefore, if ρ' overlaps the link node w of ρ, then it branches away from ρ at w.*

Proof. By Corollary 4.3, the link node of ρ' is not on ρ. If ρ' overlaps a node w' in ρ which is a proper descendant of w, then $|t_{w'}| \geq \frac{m}{4}$, and therefore w cannot be the link node of ρ, a contradiction. □

Partitioning C into Disjoint Chains of Paths. We partition the paths in C into $O(\frac{n}{m})$ disjoint ordered groups C_i, which we call *chains*. Paths in a chain have the property that each

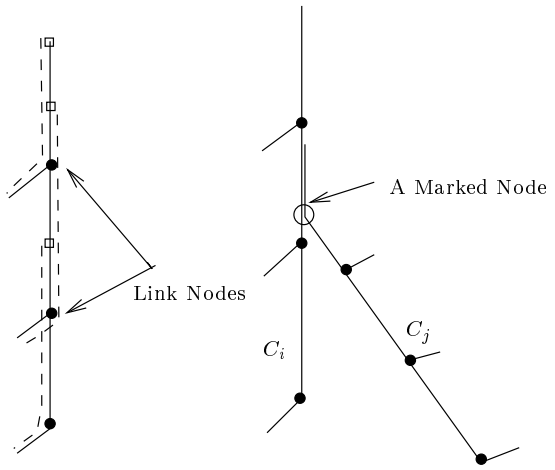

Figure 6: A Chain of Three Paths and Two Overlapping Chains.

path ρ' overlaps the link node of the previous path ρ in the chain; in addition, among all paths in C which overlap the link node of ρ, ρ' is the path whose start node is closest to the start node of ρ. Each path in C whose link node is not overlapped by any other path ends its chain. By Corollary 4.3, the portions below the link nodes in the various paths in C are all disjoint; therefore, only *cores* of various chains can overlap each other, where the core of a chain C_i is the path formed by the union of the paths in C_i, with portions below the link nodes in each path discarded (see Fig.6).

Lemma 4.6 *Consider two distinct chains C_i, C_j. Let v be the node furthest from the root of t which is common to both chains. Then v is a marked node (see Fig.6).*

Proof. Since v is in both chains, there in a path $\rho \in C_i$ and a path $\rho' \in C_j$ containing v. In addition, ρ and ρ' separate at v. Without loss of generality, assume that the start node of ρ is closer to the root than the start node of ρ'. By Corollary 4.3, the link node of ρ' does not appear on ρ. We consider two cases depending upon whether or not ρ' overlaps the link node of ρ.

First, suppose ρ' doesn't overlap the link node of ρ. Then v must be a marked node since the link nodes of both paths are descendants of v and the paths separate at v.

Next, suppose ρ' indeed overlaps the link node w of ρ. By Lemma 4.5, $v = w$. Since $\rho' \notin C_i$, there must be a path $\rho'' \in C_i$ which begins between the start nodes of ρ and ρ' and which also overlaps the link node w of ρ (ρ'' is the path next to ρ in the chain C_i). By Lemma 4.5, ρ and ρ'' must separate at $w = v$. Then ρ'' must share a child of v with ρ', which contradicts the definition of v. □

Lemma 4.7 *Consider path ρ in some chain C_i and two consecutive paths ρ', ρ'' in some chain C_j, where i may or may not equal j, with $\rho \neq \rho'$. Suppose that the start nodes of ρ, ρ', ρ'' appear in that order along the path from the root to the start node of ρ''. Then ρ'' and ρ do not have a vertex in common.*

Proof. Let x and y be respectively, the number of edges that ρ' are ρ have in common, and ρ'' and ρ' have in common. By Corollary 4.3, $x \leq \theta - 1$. Let the number of edges on the path from

the start node of ρ' to its link node be z. Recall that ρ' and ρ'' separate at the link node of ρ'. Suppose ρ'' overlaps ρ. Then, $z - y \leq x \leq \theta - 1$. $z - y$ must be a period of the path from the root of ρ' to its link node (since both ρ' and ρ'' are θ-paths). Since this path has length at least $|\pi_1|$ by Property 2 of paths in C, it has period θ. Therefore, $z - y \geq \theta$, which is a contradiction. □

Corollary 4.8 *Only the first path in chain C_j can possibly overlap some path in a chain C_i, where $i \neq j$ and the start node of C_i is an ancestor of the start node of C_j.*

Corollary 4.9 *Consider a path on a chain; the only paths on its chain it may overlap are its immediate predecessor and successor.*

Lemma 4.10 *The number of chains is $O(\frac{n}{m})$.*

Proof. We prove that the link nodes of the last paths in the various chains are incomparable, i.e., no two such nodes have an ancestor-descendant relationship. Then, since each link node v satisfies $|t_v| \geq m/4$, the lemma follows.

Suppose two paths, each last in its respective chain, have comparable link nodes. Further, suppose the link node of the second path is a descendant of the link node of the first path. Then, by Property 1 of paths in C and the definition of the link node, the start node of the second path must be an ancestor of the link node of the first path. In this case, the first path could not be the last in its chain, a contradiction. □

Partitioning C into 3 Sets. We partition C into three sets C_f, C_e, C_o. C_f comprises those paths which are the first paths in their respective chains. C_e comprises the even numbered paths in each chain and C_o comprises odd numbered paths (starting from three) in each chain.

Lemma 4.11 *The number of paths in C is $O(\frac{n}{m})$.*

Proof. By Lemma 4.10, $|C_f| = O(\frac{n}{m})$.

Consider C_e next. The analysis for C_o is identical. By Corollaries 4.8 and 4.9, the paths in C_e are non-overlapping. We show that the number of paths in C_e is $O(\frac{n}{m})$.

The number of paths in C_e which have a marked node on them is $O(\frac{n}{m})$ by Lemma 4.1. Consider a path ρ in C_e which does not have a marked node. Then, except possibly for one subtree, each subtree of t hanging from ρ is green. By Property 4 of paths in C, the sum of the sizes of the green subtrees hanging from ρ plus the length of ρ is at least $m/2$. We charge $O(\frac{1}{m})$ to each of the nodes in the green subtrees and the nodes on ρ for this path. By Property 1 of paths in C and the non-overlapping nature of paths in C_e, none of the nodes in the green subtrees can be present on any path in C_e. Further, the set of green subtrees hanging from various paths in C_e are clearly disjoint. It follows that the total charge to all nodes in t and therefore, the number of paths in C_e, is $O(\frac{n}{m})$. □

Corollary 4.12 *There are $O(n/m)$ red subtrees over all paths in C.*

Proof. If a path has k red subtrees it has $k - 1$ marked nodes (namely the parents of each red tree apart from the bottommost one on the path). As there are $O(n/m)$ marked nodes, by Lemma 4.1, and $O(n/m)$ paths, by Lemma 4.11, the result follows. □

This leads to the following theorem.

Theorem 4.13 *There is a linear time reduction from the Tree Pattern Matching problem to a collection of instances of the Subset Matching problem, of overall linear size.*

5 Further Comments

It is not completely clear this construction maps unlabelled trees to the set strings as compactly as possible, for ancestral information is lost in the reduction. Indeed, an unlabelled n-node tree can be represented using $O(n)$ bits, whereas a size n set problem in general requires $\theta(n \log n)$ bits, and will do so after our reduction. In general, n labels would require $\theta(n \log n)$ bits, so it appears the reduction is tight for labelled trees. Thus this raises the question of whether there are algorithms for unlabelled tree pattern matching that are faster by a $\theta(\log n)$ factor.

References

[1] A. Aho, J. Hopcroft, J. Ullman. Design and Analysis of Algorithms. Addison-Wesley, 1974.

[2] R. Cole, R, Hariharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. Proceedings of the *29th ACM Symposium on Theory of Computing*, 1997, pp. 66–75.

[3] R. Cole, R, Hariharan, P. Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 m)$ time. Proceedings of the *10th ACM-SIAM Symposium on Discrete Algorithms*, 1999, pp. 245–254.

[4] R. Cole, R, Hariharan, P. Indyk. Efficient deterministic and randomized algorithms for subset matching. Manuscript under preparation.

[5] M. Dubiner, Z. Galil, E. Magen. Faster tree pattern matching. Proceedings of the *31st IEEE Symposium on Foundations of Computer Science*, 1990, pp. 145–150.

[6] M.J. Fisher, M.S. Paterson. String matching and other products. *Complexity of Computation*, SIAM-AMS proceedings, ed. R.M. Karp, 1974, pp. 113–125.

[7] C.M. Hoffman, M.J. O'Donell. Pattern matching in trees. *Journal of the ACM*, 1982, pp. 68–95.

[8] P. Indyk. Deterministic superimposed coding with applications to pattern matching. Proceedings of the *38th IEEE Symposium on Foundations of Computer Science*, 1997, pp. 127–136.

[9] S.R. Kosaraju. Efficient tree pattern matching. Proceedings of the *30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 178–183.

[10] M. Crochemore, W. Rytter, *Text Algorithms*, Oxford University Press, New York, 1994, pp. 27-31.