Tree Pattern Matching and Subset Matching in Randomized $O(n \log^3 m)$ Time

Richard Cole* Ramesh Hariharan†

Abstract

The main goal of this paper is to give an efficient algorithm for the Tree Pattern Matching problem. We also introduce and give an efficient algorithm for the Subset Matching problem.

The Subset Matching problem is to find all occurrences of a 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, $1 \leq j \leq m$. We give an $O((s+n)\log^3 m)$ randomized algorithm for this problem, where s denotes the sum of the sizes of all the sets.

Then we reduce the Tree Pattern Matching problem to a number of instances of the Subset Matching problem. This reduction takes linear time and the sum of the sizes of the Subset Matching problems obtained is also linear. Coupled with our first result, this implies an $O(n \log^3 m)$ 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 [4]). Actually, in these applications, the tree need not be binary and the edges may be labelled; however, as shown in [2], 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.

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 [4] 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 [5] 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 [2] 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

*Courant Institute, New York University, cole@cs.nyu.edu. This work was supported in part by NSF grants CCR9202900 and CCR9503309.

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

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.

We will 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. All matches of the pattern in the text are sought with the restriction that the spine of the pattern must match a portion of the spine of the text.

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 Spine Pattern Matching problem is readily reduced to the Subset Matching Problem in linear time.

In the Subset Matching problem, the pattern and the text are strings of sets, i.e., each location is a set of characters drawn from some alphabet set. 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).

Subset Matching is not as well studied a problem. It can easily be solved in $O((n+s)\sqrt{m \log m})$ time by reduction to the String Matching with Don't-Cares problem (this algorithm is described in Section 4). Here, n is the text length, m is the pattern length, and s is the sum of the sizes of the text and pattern sets.

We obtain the following two results.

• We show that the Subset Matching problem can be solved in $O((n+s)\log^3 m)$ by a randomized algorithm. The main idea behind this algorithm is that occurrences of the pattern in the text are preserved if all occurrences of a particular character in the text are given an arbitrary shift (see Fig.4) and all occurrences of this character in the pattern are also shifted by the same amount. Shifting the characters in the text by random amounts and shifting the characters in the pattern by corresponding amounts facilitates encoding the pattern and the text sets with short strings; the resulting coded strings form an instance of the String Matching with Don't-Cares problem. This problem can be solved in $O(n \log m)$ time [3].

• We show that the Tree Pattern Matching problem can be Turing reduced to the Spine Pattern Matching problem in linear time. This reduction may create several instances of the Spine Pattern Matching problem, but the sum of the sizes of these instances is linear. We then reduce each instance of the Spine Pattern Matching problem in linear time to an instance of the Subset Matching problem. As a result, we obtain an $O(n\sqrt{m \log m})$ deterministic algorithm and an $O(n \log^3 m)$ randomized algorithm for Tree Pattern Matching. Further, it follows that a faster algorithm for the Subset Matching problem will lead to a faster algorithm for the tree pattern matching problem. Note the contrast with the Dubiner-Galil-Magen algorithm [2], which reduces the Tree Pattern Matching problem to $O(\sqrt{m})$ instances of the String Matching with Don't-Cares problem, each problem having potentially linear size.

The reduction to the Subset Matching problem 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 text string for an instance of the subset matching problem; the subtrees hanging from this path determine the various text sets.

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

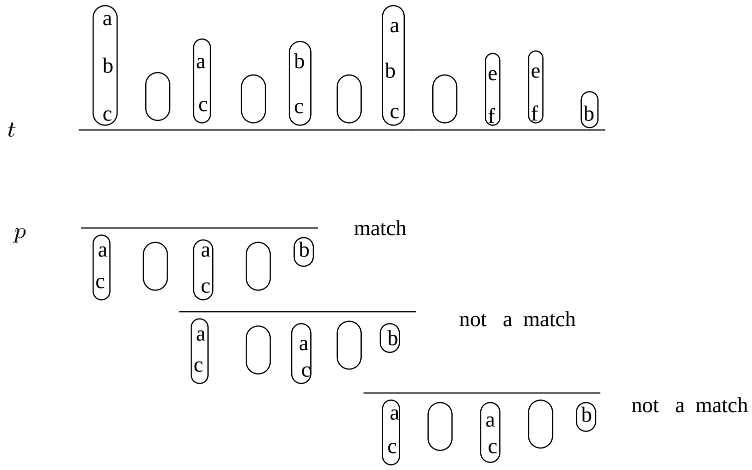

Figure 1: Example of Subset Matching.

describes the algorithms for the Subset Matching problem, and Section 5 describes the reduction from the Tree Pattern Matching problem to the Spine Pattern Matching problem.

2 Definitions

Tree Pattern Matching: We consider binary ordered 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.

Subset Matching: The text t is a string of length n and the pattern p is a string of length m. Each text location $t[i]$ and each pattern location $p[i]$ is a set of characters (and not a single character) drawn from some alphabet set (see Fig.1). Strings in which each location is a set of characters will be called *set-strings* to distinguish them from ordinary strings. Pattern p is said to match text t at location i if $p[j] \subseteq t[i+j-1]$, for all j, $1 \leq j \leq m$. Let s denote the sum of the sizes of all the sets, i.e., $\sum_{i=1}^{m} |p[i]| + \sum_{i=1}^{n} |t[i]|$. Let $\mathcal{P}$ denote the set $\cup_{i=1}^{m} p[i]$. Let $\mathcal{T}$ denote the set $\cup_{i=1}^{n} t[i]$. Let $\mathcal{A} = \mathcal{P} \cup \mathcal{T}$.

Convolution and the Don't-Cares Matching Problem. The Don't-Cares Matching problem requires finding all occurrences of a pattern string over the alphabet $\{1, ?\}$ in a text string over the alphabet $\{0, 1\}$. Here ? is a don't-care character which matches both a 0 or a 1 in the text; however, it can match only a single character and not a string of characters. Fisher and Paterson [3] showed how to solve this problem in $O(n \log m)$ time, where n is the text size and m is the pattern size. This algorithm is based upon the idea of *convolving* the two strings.

The *convolution* of a string $t[1 \ldots n]$ and $p[1 \ldots m]$ is defined as the string $t * p$ such that $(t * p)[i] = \sum_{j=1}^{m} (t[i-j+1]p[j])$, $1 \leq i \leq n-m+1$. Out of range values are assumed to be 0 in this definition. $t * p$ can be computed in $O(n \log m)$ time [1]. This operation is useful for various kinds of string matching.

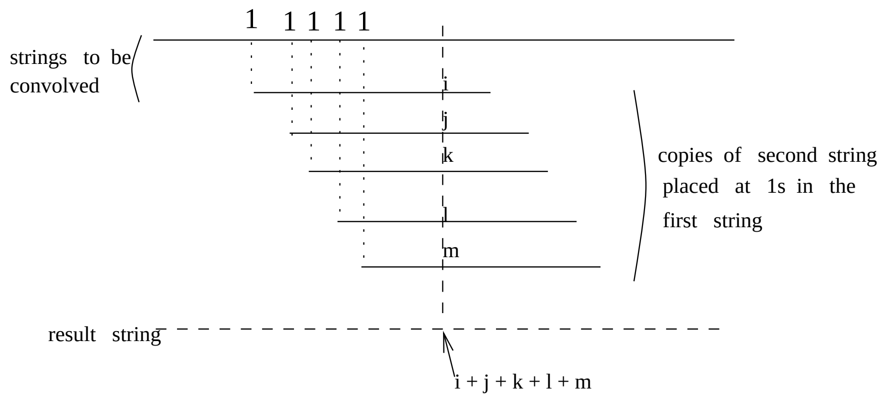

Figure 2: Convolution.

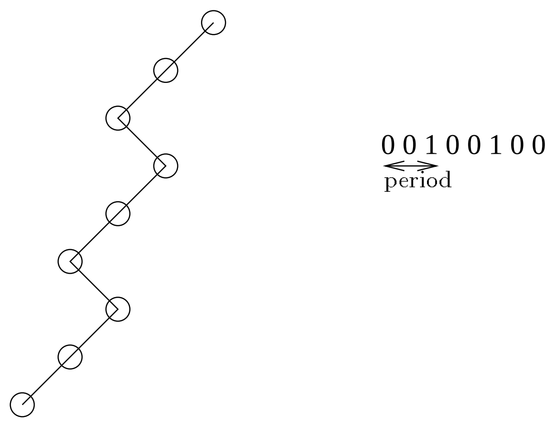

Figure 3: A Path and its Associated String

For our purpose, it will useful to look upon convolving two binary strings in the following way (see Fig.2). A copy of p is placed starting at each location in t containing a 1. The value of $(t * p)[i]$ will be just the sum of the values at locations aligned with $t[i]$ in the various copies of p.

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.3: 1 represents a left edge and 0 a right edge). The *period* of a string $s[1 \ldots |s|]$ is the smallest number j 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 [6].

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 $s[1 \ldots j]$.*

3 Reducing Spine Pattern Matching to Subset Matching

The spines of the pattern p and the text t will define the set-strings for the subset matching problem. The subsets in these set-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 whose parent is on the spine. These subsets are obtained by labelling the nodes of the off-spine subtrees as follows (see Fig.7). 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.

There is one more detail: it is necessary to record whether each non-root node on the spines is a left or right child of its parent. But this is easily done by adding the element L or R to each subset according as the associated spine node is a left or right child (the spine roots are special cases for which nothing is added to the corresponding sets).

In fact, in our application the spines of p and t will both have period θ. We define the set A of *anchor nodes* for t to be the set of nodes on its spine at distance an integer multiple of θ from the first node of t. Each anchor node begins a new instance of the period θ. Clearly, a match of p can only occur at an anchor node. Thus, in fact we do not need to record the child information in the subsets. Rather, if S is the set of matches for the Subset Matching problem instance, without the child information being recorded, then the set of legal matches corresponding to the Spine Pattern Matching instance is $S \cap A$. In our application, as we will see, there will be additional restrictions placed on the anchor nodes.

Clearly, each match in the instance of the Subset Matching Problem has a corresponding match in the instance of the Spine Pattern Matching problem and conversely. Also, clearly, this is a linear time reduction.

4 The Subset Matching Algorithm

We make the following simplifying assumptions.

Assumption 1: $\mathcal{T} = \mathcal{P}$. Every character in $\mathcal{T}$ which is not in $\mathcal{P}$ plays no role and can be discarded. Further, if $\mathcal{P}$ has a character which is not present in $\mathcal{T}$ then the pattern cannot match the text anywhere.

Assumption 2: $n \leq 2m$. A longer text can be divided into pieces of length $2m$ with consecutive pieces overlapping in m characters. All occurrences of p in each of these pieces are found. Since each occurrence of the pattern occurs completely within some piece, this suffices to find all occurrences of p in t.

Assumption 3: $s \leq 6m$. When s is larger we split up the current problem instance into a number of instances of total size $O(m+s)$ such that $s \leq 6m$ for each instance. This is done by partitioning the characters as follows.

We partition $\mathcal{A}$ into subsets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ such that for each subset $\mathcal{A}_l$, $1 \leq l < k$, $3m < \sum_{i=1}^{m} |p[i] \cap \mathcal{A}_l| + \sum_{i=1}^{n} |t[i] \cap \mathcal{A}_l| \leq 6m$, and $\sum_{i=1}^{m} |p[i] \cap \mathcal{A}_k| + \sum_{i=1}^{n} |t[i] \cap \mathcal{A}_k| \leq 6m$; this is always possible because the sum of the pattern and text sizes is at most $m + 2m = 3m$. Next, we create

k instances of the subset matching problem by restricting the pattern and text sets to have only characters from $\mathcal{A}_l$ in the lth instance, $1 \leq l \leq k$. The sum of the pattern and text set sizes is at most $6m$ for each instance. Since $3(k-1)m \leq s$, the total size of all k instances is at most $k(m+2m)+6km = 9km \leq 9(\frac{s}{3}+m) = O(m+s)$. Solving all k above instances of the subset matching problem and taking the intersection of the matches found solves the original problem.

We will show that instances of the subset matching problem with $\mathcal{T} = \mathcal{P} = \mathcal{A}$, $n \leq 2m$ and $s \leq 6m$ can be solved in $O(m\sqrt{m \log m})$ time by a deterministic algorithm, and in $O(m \log^3 m)$ time with high probability by a randomized algorithm. It follows that the problem for general n and s can be solved in $O((n+s)\sqrt{m \log m})$ by a deterministic algorithm and $O((n+s)\log^3 m)$ time with high probability by a randomized algorithm.

4.1 The Deterministic Algorithm

We describe the classical deterministic algorithm next. This algorithm considers two cases, namely, when some character in $\mathcal{P}$ occurs in fewer than $\sqrt{m \log m}$ text sets, and when every character in $\mathcal{P}$ occurs in at least $\sqrt{m \log m}$ text sets.

In the first case, the pattern can only occur at $\sqrt{m \log m}$ text locations at most. Whether the pattern matches or not at each such location can be determined in $O(m)$ time per location by checking explicitly that each pattern set is a subset of the aligned text set.

In the second case, $|\mathcal{P}| \leq \frac{6m}{\sqrt{m \log m}} = 6\sqrt{\frac{m}{\log m}}$. We consider each character in $\mathcal{P}$ individually. For each such character a, we obtain a distinct instance of the Subset Matching problem by replacing each pattern and text set X by $X \cap \{a\}$. Clearly, solving each of these instances and taking the intersection of the matches found will solve the original instance of the Subset Matching problem. We show how to solve each such instance in $O(m \log m)$ time. The time bound of $O(m\sqrt{m \log m})$ for the original problem follows.

Note that the instances of the Subset Matching problem obtained above are quite special in that each set is either $\{a\}$ or ϕ. Each instance can be reduced to the problem of Don't-Cares Matching as follows. Construct a new text from the original text by replacing each text set $\{a\}$ by 1 and each text set ϕ by 0. Similarly, construct a new pattern from the original pattern by replacing each pattern set $\{a\}$ by 1 and each pattern set ϕ by '?', a don't-care character, which matches both 1 and 0. Then this instance of the Subset Matching problem can be solved by simply finding all occurrences of the new pattern in the new text.

4.2 The Randomized Algorithm

Our randomized algorithm consists of two procedures. At its heart is a procedure which determines all occurrences of p in t in $O(m \log^3 m)$ time. This procedure could be faulty in that it could also declare some non-occurrences of p to be occurrences. However, it never misses any occurrences and it declares false occurrences with probability $O(\frac{1}{m})$. This is coupled with a procedure that checks whether the output produced by the above procedure is correct, i.e., that each declared occurrence is actually an occurrence. The checking procedure takes $O(m \log^2 m)$ time. The checking procedure uses internal results of the matching procedure, in addition to the set of candidate matches. We do not have a good general purpose verification procedure. As an aside, we note that if a verification of the matches is not needed, there is a faster variant of the matching procedure running in time $O(m \log^2 m)$.

The Limited Subset Matching Problem. First, we consider the following special case of the Subset Matching problem, which we call the *Limited Subset Matching* problem. Suppose there

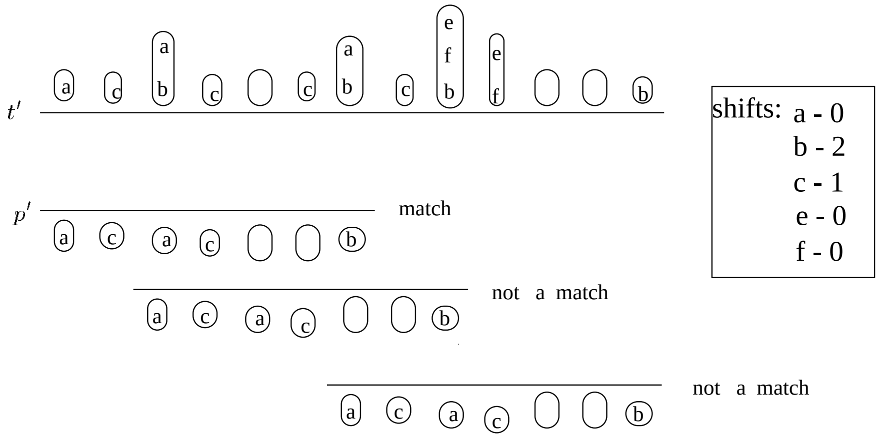

Figure 4: The set strings t', p'.

are only three possible types of sets: the empty set Φ, singleton sets, and the set of all elements Σ. We can reduce this matching problem to a Don't-Cares matching problem of size $O(m \log m)$ and thereby solve it in $O(m \log^2 m)$ time as follows. Each set is encoded by a length $2\lceil \log \mathcal{A} \rceil$ string. A new pattern p' is created; it consists of the concatenation of the encodings of the subsets in p, in the order they occur in p. A new text t' is defined analogously. The codes are chosen so that there is a match of p in t beginning at the location with index $i+1$ if and only if there is a match of p' in t' beginning at the location with index $1 + i \cdot 2\lceil \log \mathcal{A} \rceil$, where $i \geq 0$ is an integer.

This association of matches is an immediate result of the following constraints on the coding.

1. The code of an empty set in p' matches the code of any set in t'.
2. The code of a set Σ in p' matches only the code of Σ in t'.
3. The code of of a singleton set $\{a\}$ in p' matches the code of either the set Σ or $\{a\}$ in t'.

The following code satisfies the above properties. The empty sets are encoded respectively by strings of all ?'s for p and all 0's for t. Σ is encoded by strings all 1's for both p and t. Each character is given a distinct binary code of length $\lceil \log \mathcal{A} \rceil$, written as a sequence of ?'s and 1's for characters from sets in p, and with 0's replacing the ?'s for characters from sets in t. Let x denote the encoding of character a. Then the set $\{a\}$ is encoded by $x\bar{x}$, where $\bar{x}$ denotes the complement of x. Note that the encoding of a singleton set in p comprises neither all 1's nor all ?'s; an analogous statement holds for singleton sets in t. Now, it is straightforward to verify that the constraints on the coding do hold for this particular code.

The General Problem. Next, we show how to solve the general Subset Matching problem with a probability of error of at most $1/m$ in time $O(m \log^3 m)$. We follow this with a verification procedure that runs in time $O(m \log^2 m)$, which determines whether a false match was declared.

The idea of the algorithm for the general Subset Matching problem is to treat all non-empty, non-singleton sets as the set Σ. In order to avoid many non-matches being declared matches, characters have to be shifted in order to make non-empty, non-singleton sets uncommon, in a sense to be made precise later.

Shifting Characters by Random Amounts. We obtain a new text t' and a new pattern p' as follows. Consider a particular character $a \in \mathcal{A}$ and let $r(a)$ be a uniformly distributed random number in the range $1 \ldots 12m$. Let $t[i_1], t[i_2], \ldots, t[i_k]$ and $p[j_1], p[j_2], \ldots, p[j_l]$ denote the locations in t and p, respectively, which contain the character a. Then the locations in t' which contain a are the locations $t'[r(a)+i_1], t'[r(a)+i_2], \ldots, t'[r(a)+i_k]$. The locations in p' which contain a are the locations $p'[r(a)+j_1], p'[r(a)+j_2], \ldots, p'[r(a)+j_l]$. In other words, all the a's in the text and the pattern are shifted by $r(a)$. This is done for all the characters in $\mathcal{A}$. Each such character is shifted by a possibly different independent random amount (see Fig.4).

Finding p' in t' Suffices. We claim that every occurrence of p in t corresponds to a unique occurrence of p' in t' and vice versa. Since each character $a \in \mathcal{A}$ is shifted by the same amount in the pattern and in the text, there exists a match of p in t with $p[1]$ aligned with $t[j]$ if and only if there exists a match of p' in t' with $p'[1]$ aligned with $t'[j]$.

The random shifts above are performed so that p' and t' have a key property, which is stated in Lemma 4.1. This lemma shows that for any fixed text location $t'[j]$, $1 \le j \le n+12m$, $t'[j]$ is an empty set with probability at least $\frac{1}{2}$.

Lemma 4.1 *For each index j and character $b \in \mathcal{A}$, $Pr(b \in t'[j]) \le \frac{\#b}{12m}$, where $\#b$ is the number of text sets in which b occurs. $Pr(t'[j]) \ne \phi) \le \frac{s}{12m} \le \frac{1}{2}$.*

Proof. Only $\#b$ shifts of the $12m$ possible shifts can cause one of the b's in the text to appear in $t'[j]$. Therefore, $Pr(b \in t'[j]) \le \frac{\#b}{12m}$. The second part of the lemma follows from the fact that the sum of all set sizes is s which is bounded by $6m$ by Assumption 3 above. □

Lemma 4.2 *For each index j and character $a \in \mathcal{A}$, $Pr(t'[j] \ne \phi | a \notin t'[j]) \le \frac{1}{2}$ and $Pr(t'[j] - \{a\} \ne \phi | a \in t'[j]) \le \frac{1}{2}$.*

Proof. This follows by essentially the same argument as the proof of Lemma 4.2. □

Reduction to Limited Subset Matching. Each non-empty, non-singleton set in p' and t' is replaced by the set Σ giving new strings p'' and t''. The matching problem on p'' and t'' is an instance of Limited Subset Matching, for which we have already described the algorithm. The remaining issue is to bound the probability of declaring false matches, for clearly every actual match is found.

False Match Probability. It is easy to see that for each occurrence of p' in t' there is a corresponding occurrence of p'' in t''. Further, it is easy to see that a false match can occur with $p'[1]$ aligned with $t'[k]$ only if for some index i and character $a \in \mathcal{A}$:

1. $a \in p'[i], a \notin t'[k+i-1]$, and,

2. $t'[k+i-1]$ is non-empty, non-singleton.

By Lemma 4.2 applied with $j = k+i-1$, the probability of obtaining a false match with $p'[1]$ aligned with $t'[k]$ is at most $\frac{1}{2}$.

After Repeating $2\log m + 4$ Times. We repeat the above process $2\log m + 4$ times. Each repetition is an independent process consisting of the following steps:

1. Obtain p', t' from p, t using independent random shifts for each character and each repetition.

2. Obtain p'', t'' from p', t'.

3. Solve the Limited Subset Matching problem to find all occurrences of p'' in t''. From these, infer all matches of p in t.

4. For each k, $1 \leq k \leq n-m+1$, if p matches t with $p[1]$ aligned with $t[k]$ in each repetition then declare an an occurrence of p starting at $t[k]$.

Each repetition takes $O(m \log^2 m)$ time, giving $O(m \log^3 m)$ time overall.

Overall Failure Probability. Fix a value of k, $1 \leq k \leq n-m+1$. The probability of declaring an occurrence of p starting at $t[k]$ when there is no such occurrence is at most $\frac{1}{2^{2\log m+4}} = \frac{1}{16m^2}$. Over all k, this is at most $\frac{n-m+1}{16m^2} \leq \frac{1}{8m} < \frac{1}{2m}$. The probability of a false match is thus at most $\frac{1}{2m}$. Therefore, the above algorithm produces correct results with probability at least $1-\frac{1}{2m}$ and takes time $O(m \log^3 m)$.

Checking Correctness. We show how to check the correctness of the output produced by the above algorithm in $O(m \log^2 m)$ time as well. If the output is detected to be incorrect the entire procedure is repeated.

For each character a in each set $t[i]$ in t, we check whether it appears in a singleton set in some repetition. If this is not true for some character a in some set $t[i]$, we declare a failure (even though the output generated above might be correct). We can afford to do this since the probability of this event is $\frac{1}{2m}$ as is shown in Lemma 4.3, below.

Lemma 4.3 *With probability at least $1-\frac{1}{2m}$, each character a in each set in t appears in a singleton set in t' in some repetition.*

Proof. Consider a particular set $t[i]$, a particular character $a \in t[i]$ and a particular repetition. In this repetition, the probability that the set in t' containing this character a is not a singleton set is at most $\frac{1}{2}$ by Lemma 4.2. The probability that this character a appears in a non-singleton set in t' in each repetition is thus at most $\frac{1}{2^{2\log m+4}} = \frac{1}{16m^2}$. The probability that some character a in some set $t[i]$, $1 \leq i \leq n$, appears in a non-singleton set in t' in each repetition is at most $\frac{6m}{16m^2} < \frac{1}{2m}$. The lemma follows. □

For the remaining outputs, the checking procedure declares a mismatch only if one is present. Thus the probability of a failure being reported by the checking procedure is at most $\frac{1}{m}$.

The Check. For each set $t[i]$, $1 \leq i \leq n$, and each character $a \in t[i]$, we determine the number of claimed matches in which p overlaps $t[i]$ such that the set in p aligned with $t[i]$ contains a. We call this quantity the *score* of a. For each set $t[i]$, $1 \leq i \leq n$, we define its *score* as the sum of the cardinalities of the sets in p which are aligned with $t[i]$ in the various claimed matches.

We check that the score of each set in t is the sum of the individual scores of its elements. Lemma 4.4 shows that this check suffices, i.e., the output is correct if and only if the above scores are consistent.

Lemma 4.4 *If each claimed match of p in t is indeed a match then the score of each set in t is the sum of the individual scores of its elements.*

If some claimed match of p in t is a false match then the score of some set in t is not the sum of the individual scores of its elements.

Proof. Consider a text set $t[i]$. Let $\mathcal{S}_i$ denote the set of all pattern sets which are aligned with $t[i]$ in some claimed match.

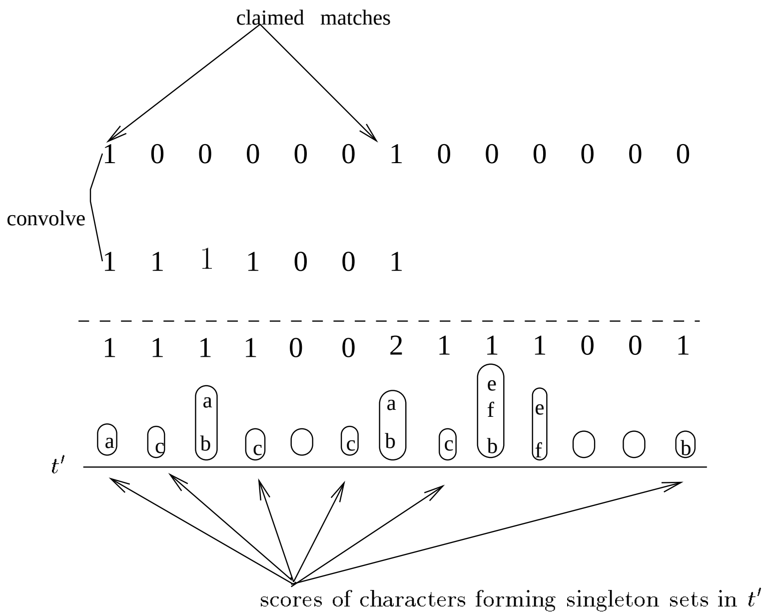

Figure 5: Individual Character Scores.

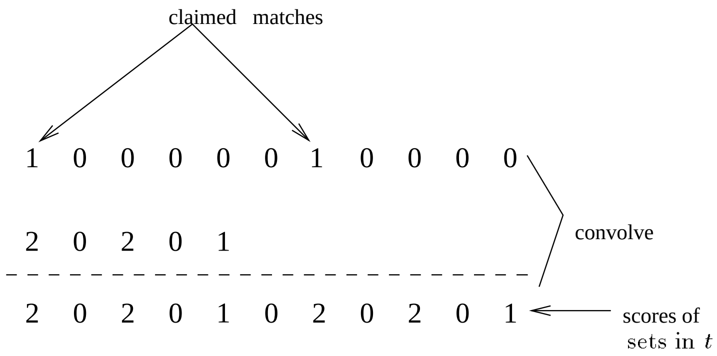

Figure 6: Set Scores.

For each character $a \in t[i]$ with score k, we charge one unit to each of the k pattern sets in $\mathcal{S}_i$ containing a. Clearly, every pattern set $p[j] \in \mathcal{S}_i$ is charged an amount equal to $|t[i] \cap p[j]|$ in this process. Further, the total charge on the sets in $\mathcal{S}_i$ is exactly the sum of the scores of the characters in $t[i]$. When every claimed match is correct, $|t[i] \cap p[j]| = |p[j]|$ and the first part of the lemma follows. When some claimed match is a false match, then for some text set $t[i]$, there exists $p[j] \in \mathcal{S}_i$ such that $|t[i] \cap p[j]| < |p[j]|$; then the sum of the scores of the characters in $t[i]$ is less than the sum of cardinalities of the pattern sets in $\mathcal{S}_i$. The second part of the lemma follows. □

Computing Character Scores. This is done in $O(m \log^2 m)$ time as follows. Consider p', t' in a particular repetition. Obtain a new string t'' from t' by putting a 1 at the start of each claimed occurrence of p' in t', and 0's elsewhere (see Fig.5). Obtain a new string p'' from p' by putting a 1 for each singleton set and a 0 elsewhere. Convolve p'' and t'' to obtain to obtain a string c in $O(m \log m)$ time (see Fig.2). For each singleton set $t'[j] = \{a\}$ which resulted due to shifting the character a in set $t[i]$ in t, $c[j]$ gives the score of a. This is so because our coding ensures that in each claimed match of p' in t', the set in p' aligned with the singleton set $t'[j]$ is identical to it or empty. But it is already guaranteed that each character a in each set of t appears in a singleton set in some string t'; consequently, By Lemma 4.3, performing the above procedure for each repetition gives the scores for all characters which appear in the various sets in t. The total time taken is $O(m \log^2 m)$.

Computing Set Scores. These can be computed in $O(m \log m)$ time as follows. A new string is obtained from t by putting a 1 at the start of each claimed occurrence of p and a 0 elsewhere. A new string in obtained from the pattern by replacing each location by the cardinality of the set at that location (see Fig.6). Then the convolution of these two strings is performed is $O(m \log m)$ time to obtain a string c. $c[i]$ gives the score of $t[i]$ (see Fig.2).

5 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 $O(\frac{n}{m})$ 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.

5.1 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.

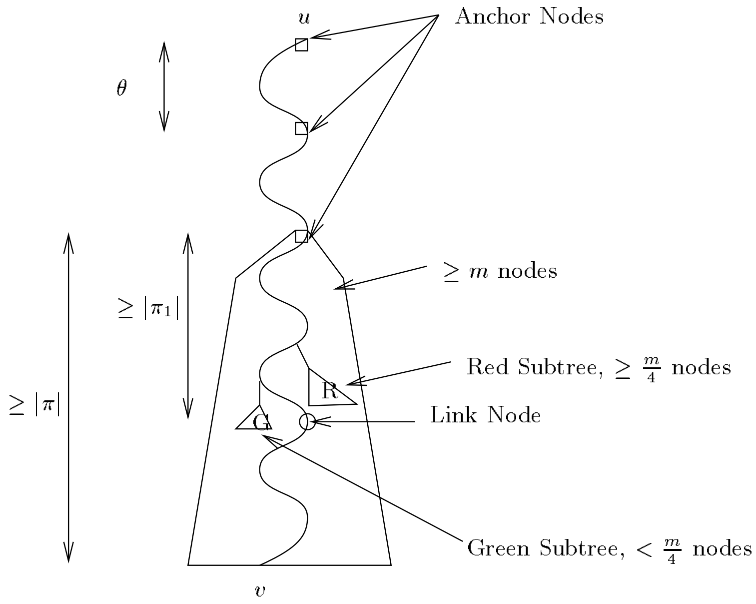

Figure 7: A θ-Path in C.

5.2 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. This path is *maximal* if extending it to the parent 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 $|t_l| \geq \frac{m}{4}$. Note that l is a node on π_2 (and π_2 includes the last node on π_1). The *anchor* nodes w on this path satisfy:

1. The distance from u to w is an integer multiple of θ.
2. The distance from w to l is at least $|\pi_1|$.
3. The distance from w to v is at least $|\pi|$.
4. t_w has at least m nodes.

We form a collection C of maximal θ-paths in t. This collection comprises all the paths which also satisfy the following properties (see Fig.7).

1. The subtree of t rooted at the start node of each path has size at least m.
2. The path from the start node to the link node in each path is at least as long as π_1, and thus has length at least θ.
3. Each path is at least as long as π.
4. Consider the subtrees of t hanging from a path in C. Classify them as *red* subtrees if they have at least $\frac{m}{4}$ nodes and as *green* subtrees otherwise. The subtrees hanging from the first

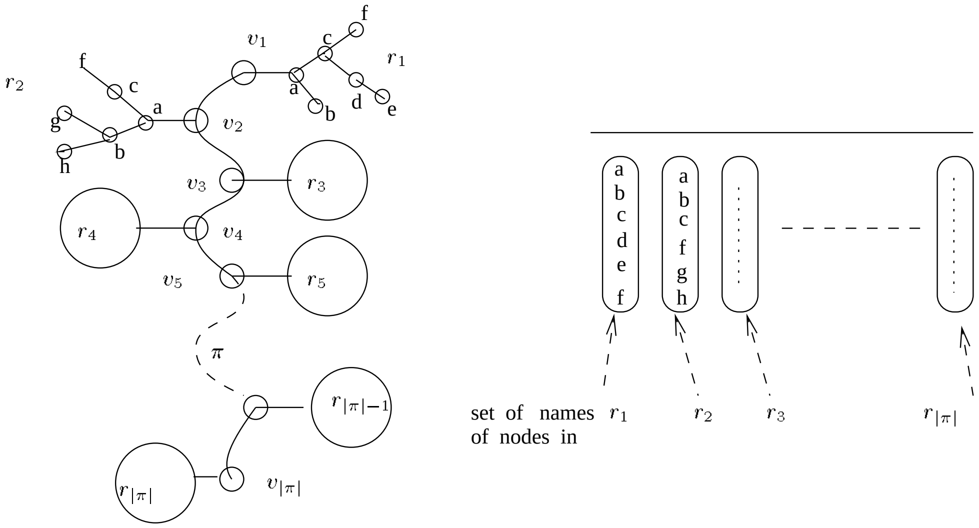

Figure 8: The Spine and its Associated Set String.

$|\pi_1|$ nodes of the path together have at least $\frac{m}{2} - |\pi_1|$ nodes. Further, if one of these subtrees is red and all other subtrees hanging from the entire path are green then the path along with all the green subtrees together have at least $\frac{m}{2}$ nodes.

Note that these paths need not be disjoint; however their combined length will still be $O(n)$ as we shall show later.

The Path Decomposition Algorithm. The decomposition is obtained using the following algorithm and takes time proportional to the sum of the lengths of the paths in C, which is $O(n)$. To achieve this time, a Knuth-Morris-Pratt type automaton is used; the details are left to the reader.

$S \leftarrow$ The set of nodes in t;
While S is not empty
  Pick the node $u \in S$ closest to the root of t;
  Traverse the maximal θ-path in t_u starting at u;
  Add the above path to the collection C if the four properties
    mentioned above are satisfied;
  Remove u and all anchor nodes on the above path from S;

Off-Spine Subtrees of p and t. Let $v_1, v_2, \ldots, v_{|\pi|}$ be the nodes on π in increasing order of distance from the root (see Fig.8). Let r_i be the off-spine subtree for v_i, $1 \leq i \leq |\pi|$. Note that some of the r_i's may be empty.

Lemma 5.1 *Consider a subtree r_i of p rooted at some node v whose parent is in π_1. Then* $m - |r_i| \geq \frac{m}{2}$.

Proof. The proof is an easy consequence of the fact that π_1 is a centroid path. $\Box$

Suppose all the r_i's are overlaid as in the reduction from Spine Pattern Matching to Subset Matching; let $\mathcal{R}$ denote the combined tree of all the overlaid r_i's. Consider a maximal θ-path π' in t. Let v be a node on π' and v' be the child of v, if any, which is not on π'. The off-spine subtree of v for this instance of the Spine Pattern Matching problem is $t_{v'} \cap \mathcal{R}$.

Fact 1 *$t_{v'} \cap \mathcal{R}$ can be found in time $O(\min\{|t_{v'}|, m\})$.*

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 one of the following: either the subtrees hanging from the first $|\pi_1|$ nodes of the path together have fewer than $\frac{m}{2} - |\pi_1|$ nodes, or one of these subtrees is red while all other subtrees hanging from the entire path are green and the path along with all the green subtrees together has fewer than $\frac{m}{2}$ nodes. That the pattern cannot match at w if the first condition holds follows from the definition of the spine of the pattern. That the pattern cannot match at w when the second condition holds follows from Lemma 5.1.

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.

5.3 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 that 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 $trunc(\rho)$. 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. $trunc(\rho)$ provides the spine of the text instance.

As stated in Fact 1, determining s_i takes $O(\min\{|s_i|, m\})$ time. Clearly, there is a match of p rooted at a node on $trunc(\rho)$, if and only if there is a match at the same location in the corresponding Spine Pattern Matching problem instance.

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. The size of the text obtained from ρ is $O(\sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$.

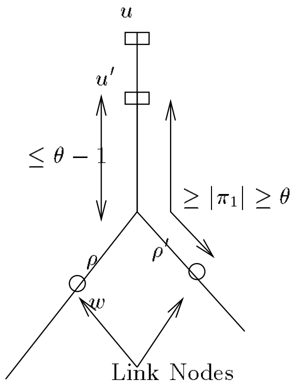

Figure 9: Overlap is at most $\theta - 1$.

5.4 Showing $O(n)$ Time

Eventually, we will seek to bound the number of red subtrees. We will do this by identifying a set of $O(n/m)$ nodes of t, called *marked* nodes. Each red tree will be assigned to either a marked node or a path, and each marked node and path will received at most a constant number of red subtrees. We will also show in Section 5.4.1 that there are $O(n/m)$ paths; 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 5.2 *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.

Lemma 5.3 *Consider two paths ρ, ρ' in C starting at nodes u and u', respectively (see Fig.9). 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 5.4 *If the first θ edges are removed from each path in C then the resulting collection of paths is node disjoint. Also, as the link node is not among the first θ nodes, the link node of ρ' cannot lie on ρ.*

Corollary 5.5 *Consider two paths $\rho, \rho' \in C$ starting at u, u', respectively, where u is closer to the root of t than u'. Let v be a node in a green subtree hanging from $trunc(\rho)$. Then v cannot be in any green subtree hanging from $trunc(\rho')$.*

Proof. By Corollary 5.4, $trunc(\rho)$ and $trunc(\rho')$ are disjoint. Let G be the green subtree hanging from $trunc(\rho)$ and containing v. Then, for v to be in a green subtree hanging from $trunc(\rho')$, $trunc(\rho')$ must lie 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.

As we will show in Section 5.4.1, the number of paths in C is $O(\frac{n}{m})$. The fact that the sum of the lengths of the paths in C is $O(n)$ follows immediately from Corollary 5.4 and the fact that $\theta \leq m$. That the algorithm then takes $O(n)$ time is shown below.

The time taken to check whether or not the pattern matches at the start node of each path in $O(n)$ over all paths. Next, we consider each path ρ in C with the first θ edges removed. The resulting collection of paths is disjoint by Corollary 5.4. It remains to show that the time taken to set up the Spine Pattern Matching matching problems over all such truncated paths in C is $O(n)$. Consider one such path and the subtrees of t hanging from this path. Each green subtree contributes an off-spine subtree no larger than itself to the Spine Pattern Matching problem, while each red tree contributes a subtree of size at most m. Note that if there are k red subtrees hanging from this path, then there must also be $k-1$ marked nodes on the path. Further, by Corollary 5.5, no node appears in a green subtree for two different paths. This suggests the following charging scheme. Each node in a green tree is charged once while each marked node on the above path is charged m. If $k > 1$, this accounts completely for the contribution of all the green and the red subtrees to the Spine Pattern Matching problem. If $k = 1$ then the path itself is charged m for the contribution of the single red subtree. Since the number of marked nodes and number of paths in C is $O(\frac{n}{m})$, the total charge over all nodes in t will be $O(n)$.

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

Lemma 5.6 *Let ρ, ρ' be as in Lemma 5.3 (see Fig.9). Then ρ' cannot overlap a node in ρ which is a 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 5.4, the link node of ρ' is not on ρ. If ρ' overlaps a node w' in ρ which is a 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 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 5.4, 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.10).

Lemma 5.7 *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.10).*

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 5.4, 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.

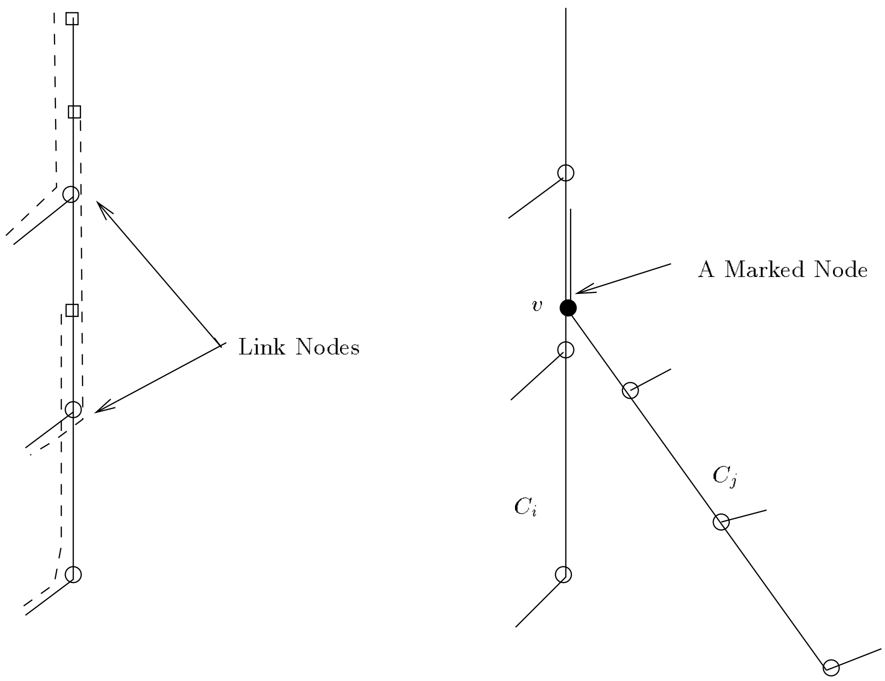

Figure 10: A Chain of Three Paths and Two Overlapping Chains.

Next, suppose ρ' indeed overlaps the link node w of ρ. By Lemma 5.6, $v = w$. 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 5.6, 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 5.8 *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 5.4, $x \leq \theta - 1$. 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, i.e., $z - y \leq x \leq \theta - 1$. Then $z - y$ must be a period of the path from the root of ρ' to its link node. 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 5.9 *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 5.10 *Consider a path on a chain; the only paths on its chain it may overlap are its immediate predecessor and successor.*

Lemma 5.11 *The number of chains is* $O(\frac{n}{m})$.

Proof. It is easy to see that each chain must branch away from every other chain. Each chain is charged to the furthest node from the root of t which it shares with some other chain. By Lemma 5.7, the charged nodes are marked nodes. Each marked node is charged at most twice, for a chain causes a charge to a node only if it is the sole chain along one of the edges descending from the charged node. The lemma then follows from Lemma 5.2. $\Box$

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

Proof. By Lemma 5.11, $|C_f| = O(\frac{n}{m})$.

Consider C_e next. The analysis for C_o is identical. By Corollaries 5.9 and 5.10, 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 5.2. Consider a path ρ in C_e which does not have a marked node.

Except possibly for one subtree, each subtree of t hanging from from ρ is green. By Property 4 of paths in C, the subtrees hanging from the first $|\pi_1|$ nodes of ρ together have at least $\frac{m}{2} - |\pi_1|$ nodes; further, if one of these trees is red then the sum of the sizes of the green subtrees plus the length of ρ is at least $\frac{m}{2}$. 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, 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 $O(\frac{n}{m})$. $\Box$

This leads to the following theorem.

Theorem 5.13 *There is a linear time reduction from the Tree Pattern Matching problem to various instances of the Subset Matching problem, but of overall linear size. There is an* $O((n+s)\sqrt{m \log m})$ *deterministic algorithm and an* $O((n+s)\log^3 m)$ *randomized algorithm for the Subset Matching Problem, where n is the text length, m is the pattern length, and s is the sum of text and pattern set sizes. Consequently, there is an* $O(n\sqrt{m \log m})$ *deterministic algorithm and an* $O(n \log^3 m)$ *randomized algorithm for Tree Pattern Matching, where n is the text size and m is the pattern size.*

References

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

[2] M. Dubiner, Z. Galil, E. Magen. Faster tree pattern matching. Proceedings of the *31st IEEE FOCS*, 1990, pp. 145–150.

[3] 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.

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

[5] S.R. Kosaraju. Efficient tree pattern matching. Proceedings of the *30th IEEE FOCS*, 1989, pp. 178–183.

[6] M. Lothaire. Combinatorics on Words.