Pattern Matching in Hypertext*

Amihood Amir† Moshe Lewenstein‡ Noa Lewenstein§
Georgia Tech and Bar-Ilan University | Bar-Ilan University | Bar-Ilan University

Abstract

The importance of hypertext has been steadily growing over the last decade. Internet and other information systems use hypertext format, with data organized associatively rather than sequentially or relationally.

A myriad of textual problems have been considered in the pattern matching field with many non-trivial results. Nevertheless, surprisingly little work has been done on the natural combination of pattern matching and 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 [14] pioneered the study of pattern matching in hypertext and defined a hypertext model for pattern matching. Akutsu [2] developed an algorithm that can be used for exact pattern matching in a tree-structured hypertext. Park and Kim [16] considered regular pattern matching in hypertext. They developed a complex algorithm that works for hypertext with an underlying structure of a DAG.

In this paper we present a much simpler algorithm achieving the same complexity which runs on *any* hypertext graph. We then extend the problem to *approximate* pattern matching in hypertext, first considering hamming distance and then edit distance. We show 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.

Key Words: Design and analysis of algorithms, combinatorial algorithms on words, pattern matching, hypertext, pattern matching on hypertext.

---

*A preliminary version of this paper appeared in WADS 97.

†Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8770; amir@cs.biu.ac.il; Partially supported by NSF grant CCR-96-101709, BSF grant 96-00509, and a Bar-Ilan University Internal Research Grant.

‡Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; moshe@cs.biu.ac.il.

§Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; noa@cs.biu.ac.il; Partially supported by the Israel Ministry of Science and the Arts grant 8560. This work is part of Noa Lewenstein's Ph.D. dissertation.

1 Introduction

Most digital libraries today appear in *hypertext* form [15], with links between text and annotations, and, in multimedia libraries, between the text and pictures, video and voice. The World Wide Web is, perhaps, the most widely known hypertext system. In contrast, classical pattern matching (e.g. [6, 11]) has dealt primarily with unlinked textual files. Recently, there has been interest in nonstandard matching, such as dictionary matching ([4, 3]) or dynamic indexing ([9, 17]) but, surprisingly, there has been no concerted effort in analyzing string matching in hypertext.

In a pioneering paper by Manber and Wu [14] a first attempt is made to define pattern matching in hypertext. They suggest the concept of viewing a hypertext library as a general graph of unlinked files. For a formal definition, see section 2.

Akutsu [2], presented an algorithm that would be capable of pattern matching in a hypertext with an underlying tree structure. Park and Kim [16] also use the Manber and Wu model and present a general pattern matching algorithm. Their algorithm assumes that the hypertext links form an acyclic digraph.

We were motivated by a natural example of hypertext to seek a pattern matching algorithm in more general graphs. The application that motivated our research is the *Responsa Judaica library.* Hebrew Law has evolved over two thousand years and is recorded in the Bible, Mishna, Tosephta, Babylonian and Jerusalem Talmud, Maimonides, Shulchan Aruch, Tur, and thousands of published responsa. The links between the various works create a general hypertext digraph. In this context, one may want to search only a main text, but it is possible that a search is desired through the hyperlinks.

Some of the issues involved are discussed in a recent manuscript of Fraenkel and Klein [10]. An M.Sc. thesis in the Math and CS department at Bar-Ilan [5] deals with a hypertext system for the Babylonian Talmud. All existing pattern matching algorithms in the hypertext model ([14, 16] assume that the hypertext links form an acyclic digraph. In the Responsa environment, the text files present a general digraph.

In this paper we present the *first* algorithm for pattern matching in hypertext where the hypertext links form a general graph. The complexity of our algorithm is the same as that of the Park and Kim algorithm, $O(N + |E|m)$, where N is the overall size of the text, m is the pattern length and E is the edge set of the hypertext.

We extend this result to *approximate* matching in hypertext. We begin with the hamming distance as our metric. Some very surprising insights are achieved. In classical approximate pattern matching the error locations are *symmetric.* It does not matter if the errors occur in the text or in the pattern. In approximate matching in hypertext there are distinctly different cases.

This paper is the *first in the literature* to identify cases where the error location is not symmetric. We show that it is important to understand whether the mismatches occur in pattern or in the text.

We consider three flavors of the problem: with mismatches in the hypertext only, mismatches in the pattern only and mismatches in both. The first and third turn out to be $\mathcal{NP}$-Complete. For the second type, we present an algorithm that runs in time $O((N\sqrt{m} + m\sqrt{N})\sqrt{\log m} + |E|m)$, where N is the overall size of the text, m the pattern length and E the edge set of the hypertext, over unbounded alphabets. For fixed bounded alphabets the running time is $O(N \log m + |E|m)$.

Finally, we consider edit distance. Once again, this comes in three variations and the variations allowing errors in the hypertext are $\mathcal{NP}$-Complete. If we allow errors in the pattern only, then it is polynomial. We present an algorithm of time complexity $O(Nm \log N + |E|m)$. This problem was already considered by Manber and Wu [14]. As previously mentioned, they assume that the underlying digraph is acyclic. Our algorithm works for an arbitrary graph.

There are some other differences between the Manber-Wu algorithm and ours. Their algorithm also does not handle mismatches while ours does. Their algorithm works for non-uniform error cost. Although we considered uniform error cost, our algorithm is constructed in a manner that can be adapted to non-uniform error cost. The time complexity of the Manber-Wu algorithm uses different measures but worst case analysis gives the same results as our algorithm. On the other hand, their algorithm is more space efficient.

2 Pattern Matching in Hypertext

Formally a *hypertext* above 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*.

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 texts upon this path, where T''_{v_k} is a prefix of T_{v_k} and T'_{v_1} is a suffix of T_{v_1} beginning at location l of T_{v_1}. We say that W *matches* on the path $v_1, ..., v_k$ beginning at location l of v_1 or that $v_1 : l$ is a *W-match location*. In general we say that W *matches* in H. Note that for a one-character hypertext we may disregard the location since there is only one location per vertex.

The *Pattern Matching in Hypertext* problem is defined as follows:
INPUT: A pattern P and a hypertext H.
OUTPUT: All P-match locations.

Initially we solve the problem by transforming a given hypertext into a one-character hypertext. The transformation is done by taking every text T_v and splitting v into $|T_v|$ vertices (saving for each new vertex its origin). For the remainder of this section, we consider the hypertext $H = (V, E, T)$ to be a one-character hypertext. In the next section we discuss general hypertext graphs.

We would like to find all the P-match locations within the hypertext. Since our hypertext is a one-character hypertext, every path that $P = p_1 p_2 ... p_m$ matches upon is exactly of length m, the length of pattern P. To be precise, the path is of the form $v_1, v_2, v_3, ..., v_m$ such that $T_{v_1} = p_1$, $T_{v_2} = p_2$, ..., $T_{v_m} = p_m$. The idea is to create a digraph $G^{(P,H)} = (V^{(P,H)}, E^{(P,H)})$ that depends on the hypertext and the pattern, such that every path of length m in the digraph will represent a match in the hypertext. Corresponding to each vertex in the hypertext there will be m vertices in the digraph that will represent the m pattern locations. Similar to an edge in the hypertext, an edge in the digraph will represent two consecutive characters, but the edge will also represent their location in the pattern, i and $i+1$. In addition, the edge represents a match between the hypertext character in the source of the edge and the ith pattern character.

Formally, we define the digraph in the following way:

$$V^{(P,H)} = \{v^i \mid v \in V, \;\; 1 \leq i \leq m\} \cup \{s, f\}$$

$$E^{(P,H)} = \{(s, v^1) \mid v \in V\} \cup \{(v^i, u^{i+1}) \mid (v,u) \in E, \;\; T_v = p_i\} \cup \{(v^m, f) \mid T_v = p_m\}.$$

It is easy to see from the definition that $G^{(P,H)}$ is a DAG and the longest path in $G^{(P,H)}$ is of length $m+1$ (the additional 1 is for the initial vertex s). For our algorithm we need the following lemma.

Lemma 1 *P matches on path $v_1, v_2, ..., v_m$ in H iff $s, v_1^1, v_2^2, ..., v_m^m, f$ is a path in $G^{(P,H)}$.*

Proof: Follows from the definition of $G^{(P,H)}$. □

Algorithm for Pattern Matching in Hypertext

1. Construct a one-character hypertext $H' = (V', E', T')$ from the original hypertext $H = (V, E, T)$.
2. Construct $G^{(P,H')}$ from the pattern P and the hypertext H'.
3. Do a depth first search in $G^{(P,H')}$ starting from s. Denote by *true* every vertex v^i where there is a path from v^i to f.
4. For every vertex v in the hypertext, check v^1 in the digraph and if marked true announce the vertex and location in H corresponding to vertex v in H' as a P-match location.

Time: Building H' and $G^{(P,H')}$ takes time linear to the size of H', P and $G^{(P,H')}$. The size of $G^{(P,H')}$, where $H' = (V', E', T')$ is $O(m|E'|)$ and this is clearly the largest of the three. Step 3, can likewise be implemented in linear time. Step 4 takes $O(|V'|)$ time. The size of V' is at most $|V| + N$ and the size of E' is at most $|E| + N$ since for every new vertex in the one-character hypertext there is exactly one edge introduced. Therefore the overall time for the algorithm takes $O(m|E'|)$=$O(mN + m|E|)$. □

Correctness: Follows from the construction of $G^{(P,H)}$ and Lemma 1. □

3 Improved Algorithm for Pattern Matching in Hypertext

We take an approach similar to the previous algorithm without transforming the hypertext into a one-character hypertext. Hypertext vertices may now contain text longer than one-character and possibly even longer than the pattern length. We use conventional pattern matching techniques to find instances of the pattern within vertices that have text longer than the pattern length . To find the other instances of the pattern, i.e. those that cross at least one hyperlink, we will extend the idea from the previous section.

As in the previous section, we create a digraph to model the hypertext and the pattern, but in this case there will be two sets of m vertices for every vertex in the hypertext instead of one set. These two sets will model comparisons of subpatterns rather than comparisons of characters only. In the case of the one-character hypertext, there is only one possible entrance and one possible exit from every node and only the relative location in the pattern varies. That forced making m copies of every node. Now, however, there may be more than one entrance and more than one exit from each node, *in addition* to the location. For example, a node may end in a number of pattern prefixes, start with a number of pattern suffixes, and be a number of internal subpatterns. To allow for all these possibilities, we make $2m$ copies of every node, encompassing all "entrance" locations and all "exit" locations.

For a match of T_v from location i to location j in the pattern we set an edge from the ith vertex in the first set to jth vertex in the second set. (These edges are described in the first part of $E^{(P,H)}$ defined below.) We also need to consider a match beginning in the middle of the vertex's text or ending in the middle of the vertex's text. (These edges are described in the second and third part of $E^{(P,H)}$ defined below.)

We will use the following notation. Let x be a string and k an integer. Denote the k length suffix of x by *Suf(x,k)* and the k length prefix of x by *Pref(x,k)*.

We now give a formal definition of the digraph. Let $H=(V,E,T)$ be a hypertext and $P=p_1p_2...p_m$ be a pattern then $G^{(P,H)}=(V^{(P,H)},E^{(P,H)})$ is defined in the following way.

$$
\begin{array}{lll}
V^{(P,H)} = & \{\overline{v}^i \mid v\in V,\ \ 1\le i\le m\}\ \cup\ \{\underline{v}^i \mid v\in V,\ \ 1\le i\le m\}\cup\{s\} & \\[1ex]
E^{(P,H)} = & \{(\overline{v}^i,\underline{v}^{i+|T_v|-1}) \mid & |T_v|<m,\ \ 1<i<m-|T_v|+1,\ \ T_v=p_i...p_{i+|T_v|-1}\}\ \cup \\
& \{(\overline{v}^1,\underline{v}^k) \mid & Suf(T_v,k)=p_1...p_k,\ \ 1\le k<\min\{m,|T_v|\}\}\ \cup \\
& \{(\overline{v}^k,\underline{v}^m) \mid & Pref(T_v,m-k+1)=p_k...p_m,\ \max\{1,m-|T_v|+1\}<k\le m\}\ \cup \\
& \{(\underline{v}^i,\overline{u}^{i+1}) \mid & (v,u)\in E,\ \ 1\le i<m\}\cup \\
& \{(s,\overline{v}^1) \mid & v\in V\}.
\end{array}
$$

Lemma 2 *Let $v\in V$ and l be a location in T_v. If $|T_v|-l+1<m$ then there is a P-match location at $v:l$ in H iff there exists a path in $G^{(P,H)}$ beginning with $s,\overline{v}^1,\underline{v}^{|T_v|-l+1}$ and ending with $\underline{u}^m$ for some $u\in V$.*

Proof: Assume that there is a P-match location at $v:l$ in H with $|T_v|-l+1<m$. Let $v,v_2,...,v_k$ be a path such that P matches upon it beginning at location l of v, i.e. $P=T'_vT_{v_2}T_{v_3}...T_{v_{k-2}}T_{v_{k-1}}T''_{v_k}$. Since $|T'_v|=|T_v|-l+1$, $(\overline{v}^1,\underline{v}^{|T_v|-l+1})\in E^{(P,H)}$. It is also immediate that $(s,\overline{v}^1)\in E^{(P,H)}$. Moreover, since the prefix of length $|T''_{v_k}|$ of T_{v_k} matches at pattern location $m-|T''_{v_k}|+1$, $(\overline{v}_k^{m-|T''_{v_k}|+1},\underline{v}_k^m)\in E^{(P,H)}$. Also, for every $1<i<k$, $(\overline{v}_i^{\Sigma_{j=1}^{i-1}|T_{v_j}|+1},\underline{v}_i^{\Sigma_{j=1}^{i}|T_{v_j}|})\in E^{(P,H)}$ because $T_{v_i}<m$ and T_{v_i} appears at pattern location $\Sigma_{j=1}^{i}|T_{v_j}|+1$. Since $(v_i,v_{i+1})\in E$, for $1\le i<k$, it follows that $(\underline{v}_i^{|T_{v_i}|},\overline{v}_{i+1}^{|T_{v_i}|+1})\in E$. Therefore, we have a path that ends with $\underline{v}_k^m$ as required.

Conversely, a careful analysis of the construction of $G^{(P,H)}$ shows that any path from s to $\underline{u}^m$ for some $u\in V$ must be of the form $s,\overline{v}_1^1,\underline{v}_1^{i_1},\overline{v}_2^{i_1+1},\underline{v}_2^{i_2},...,\underline{v}_{k-1}^{i_{k-1}},\overline{u}^{i_{k-1}+1},\underline{u}^m$. Similar to the other direction if $i_1=|T_v|-l+1$ it is relatively straightforward to see that this represents a path beginning in v_1 upon which P matches at location l of v_1. $\square$

Algorithm for Pattern Matching in Hypertext

1. For every vertex v in H do standard pattern matching with pattern P and text T_v.

2. Based on the results of step 1, announce the internal matches as P-match locations.
3. Build $G^{(P,H)}$ from the pattern P and the hypertext H, using step 1 for the first three types of edges.
4. Do a depth first search in $G^{(P,H)}$ starting from vertex s saving at every vertex $w \in V^{(P,H)}$ true, if there is a path from w to $\underline{u}^m$ for some $u \in V$.
5. For every vertex v in the hypertext, check $\overline{v}^1$ in the digraph $G^{(P,H)}$ and if marked true do the following: For each $\underline{v}_j$ s.t. $(\overline{v}^1, \underline{v}^j) \in E^{(P,H)}$, if $\underline{v}^j$ is marked true, announce $v : |T_v| - j + 1$ a P-match location.

Correctness: Matches internal to a vertex will be detected in step 1 and announced in step 2. Now, let $v : l$ be a P-match location that crosses at least one hyperlink. Since it crosses a hyperlink it must be the case that $|T_v| - l + 1 < m$. The conditions of step 4 and step 5, where we set $j = |T_v| - l + 1$, together with Lemma 2 shows that we indeed find this P-match location and announce it in step 5.

Time: Step 1 takes $O(N)$ time since pattern matching is linear for text and N is the overall text size. Step 2 is included in the complexity of the previous step. For step 3, note the size of $G^{(P,H)}$. There are $O(|V|m)$ vertices and there are $O(|E|m)$ edges. Constructing the edges is immediate for those with source s and those of the form $(\underline{v}^i, \overline{u}^{i+1})$. For the others we use the results of the pattern matching from step 1. A possible implementation can be done by slightly modifying the KMP algorithm. Therefore, the construction of $G^{(P,H)}$ is linear in its size. So, step 3 is $O(|E|m)$. Step 4 is, once again, linear in the size of $G^{(P,H)}$ and the time for step 5 is bounded by the size of $G^{(P,H)}$. Therefore, the algorithm runs in $O(N + |E|m)$ time.

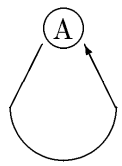
4 Approximate Pattern Matching in Hypertext - Hamming Distance

Approximate pattern matching is one of the well-researched problems in pattern matching. Often the text contains errors and searching for an exact match is not sufficient. We present algorithms that search for three types of approximations: a “closest” solution, all “closest” solutions, and for solutions not exceeding k errors. The “closest” solution depends on what type of errors are considered. A mismatch is one of the most common errors. The number of mismatches between two equal length strings is called the *hamming distance*. *Approximate pattern matching with hamming distance* refers to the problem of finding the substring with minimum hamming distance from the pattern or finding all substrings with hamming distance less than a specified distance from the pattern.

Naturally, it would be interesting to investigate approximate pattern matching in hypertext using hamming distance as our metric. This needs some clarification. Historically, whenever hamming distance between two strings is used, it is not specified in which string the error occurred. In our context, the error may occur in the text or in the pattern. This detail is never discussed because, in strings, it really does not make a difference where the error occurred.

In hypertext this is no longer true. Consider a certain path in the hypertext passing through v, k times, where $k > 1$. If we change a character in that hypertext vertex we are changing all k

instances on that path. On the other hand, if we are changing the characters in the pattern we can change each instance to a different character. From this reasoning it follows that the fewest number of changes in the pattern required so that the pattern matches may be different than the fewest number of changes required in the hypertext. Moreover, we can always change the pattern so that the pattern will match in the hypertext but it may be the case that we cannot change the hypertext so that the pattern will match. In the example below, assuming that there are two errors in the pattern, we have a match in the hypertext. Assuming a mistake in the text can not give us a match at all!

hypertext: 

pattern: ABC

In real life, applications vary. In some applications it is reasonable to expect the mismatches to occur in the text while in others the mismatches occur in the pattern. Sometimes we may even expect errors to appear in both. Consider a hypertext similar to the internet in which information is often inserted quickly and erroneously. Here we would expect the mismatches to be in the text. On the other hand, a relatively constant and well checked hypertext, such as the Responsa project which serves as a query system, is less prone to errors. Nevertheless, for large queries we may expect mismatches in the pattern. The hypertext application actually allows for three versions of approximate pattern matching with hamming distance. In each of these versions the input is a pattern P, a hypertext $H = (V, E, T)$, and an error bound d.

Version 1: Can we change at most d characters in the hypertext H so that P will match in H?

Version 2: Can we change d characters in P so that P will match in H?

Version 3: Can we change d characters in P and H so that P will match in H?

These questions are stated as decision questions but we may also ask, what is the fewest number of changes, min_d, necessary for P to match in H? Or, find all locations where P matches in the hypertext with min_d changes. Note that when the hypertext is a DAG all three versions boil down to the same version. In general for an arbitrary graph we shall see that versions 1 and 3 are $\mathcal{NP}$-Complete and version 2 is polynomial.

Proposition 1 *The Approximate hypertext-hamming distance problem - version 1 is $\mathcal{NP}$-Complete.*

Proof: Clearly the problem is in $\mathcal{NP}$. We will reduce from directed hamiltonian path.

Let $G = (V, E)$ be a directed graph in which we seek a hamiltonian path and let $n = |V|$. Construct a hypertext H over $\Sigma = \{\sigma_1, ..., \sigma_n$ setting $H = (V, E, T)$ where $T_v = \sigma_1$ for every $v \in V$. Take pattern $\sigma_1 ... \sigma_n$, (note that $\forall i \neq j,\ \sigma_i \neq \sigma_j$) and distance $d = n - 1$. If we have a hamiltonian path then replacing the i-th vertex's text with σ_i will give us a match of P in the hypertext.

Conversely, if we can make text-changes in the hypertext (which, in our case, clearly, does not exceed n) so that P matches on some path then since $|T| = |V| = n$ and $\forall i \neq j,\ \sigma_i \neq \sigma_j$ it must be the case that the pattern matches on a hamiltonian path. $\square$

We can not use the same reduction for version 3, since any graph containing a cycle would have returned a match simply by changing all pattern characters to σ_1. However, we use a similar reduction.

Proposition 2 *The Approximate hypertext-hamming distance problem - version 3 is $\mathcal{NP}$-Complete.*

Proof: Clearly the problem is in $\mathcal{NP}$. We will reduce from directed hamiltonian cycle.

Let $G = (V, E)$ be a directed graph in which we seek a hamiltonian cycle and let $n = |V|$. Construct a hypertext H setting $H = (V, E, T)$ where $T_v = \sigma_1$ for every $v \in V$. Take pattern $\sigma_1 ... \sigma_n \sigma_1 ... \sigma_n$, where $\forall i \neq j,\ \sigma_i \neq \sigma_j$ and distance $d = n - 1$. If we have a hamiltonian cycle then replacing the i-th vertex's text with σ_i, for $i \geq 2$, making $n - 1$ changes altogether, will give us a match of P in the hypertext.

Conversely, assume we have a match of P in H with at most $n - 1$ changes on P and H together. Consider the path on which P matches in H after the changes were made, call it C, and let k be the number of different characters on C. Note, that it is always true that $k \leq n$ since there are only n characters in the whole hypertext. There are n different characters in the pattern. But since there are only k of these on C, at least $n - k$ of the pattern symbols must be changed. Since each symbol in the pattern appears twice we must make at least $2(n - k)$ changes in the pattern. Now, since we started with the same character in all vertices in the hypertext we must have made at least $k - 1$ changes in the text. So overall there are at least $2(n - k) + k - 1$ changes. Our distance is $d = n - 1$ and therefore it must be that $2(n - k) + k - 1 \leq n - 1$. This is equivalent to $n \leq k$ and since $n \geq k$ we have $k = n$. This means that after changes every vertex in the hypertext must contain a different character. This already accounts for the $n - 1$ allowed changes. So, the pattern must be in its original form.

C now serves for a hamiltonian cycle since we first visit the vertex labeled σ_1 and then the vertex labeled σ_2 and so on till we come back to the vertex labeled σ_1 (and then go for another round). $\square$

We now show a polynomial algorithm for version 2. The idea is similar to the algorithm of the previous section.

We build a digraph in a similar fashion to the digraph of section 3 but this time the digraph will be weighted. As in the previous construction, for each vertex in the hypertext we construct two sets of m vertices. All edges that were in the previous construction will also be in this digraph, all having weight zero. The weight zero expresses that there are no mismatches. We add edges that will capture the matches containing mismatches. The weight we assign to such an edge is the number of mismatches occuring between the corresponding subpattern and the vertex's text.

We denote the number of mismatches between two equal length strings x and y as *Ham(x,y)*.

Formally we define the directed weighted graph $G^{(P,H)} = (V^{(P,H)}, E^{(P,H)})$, where H is a hypertext (V, E, T) and P is a pattern $p_1 p_2 ... p_m$, as follows:

$$V^{(P,H)} = \quad \{\overline{v}^i \mid v \in V, 1 \leq i \leq m\} \ \cup \ \{\underline{v}^i \mid v \in V, 1 \leq i \leq m\} \cup \{s\}$$

$$
\begin{aligned}
E^{(P,H)} = \quad & \{(\overline{v}^i, \underline{v}^{i+|T_v|-1}, w) \mid |T_v| < m, 1 < i \leq m - |T_v| + 1, w = Ham(T_v, p_i ... p_{i+|T_v|-1})\} \cup \\
& \{(\overline{v}^1, \underline{v}^k, w) \mid 1 \leq k < m, w = Ham(Suf(T_v, k), p_1 ... p_k)\} \cup \\
& \{(\overline{v}^k, \underline{v}^m, w) \mid 1 < k \leq m, w = Ham(Pref(T_v, m-k+1), p_k ... p_m)\} \cup \\
& \{(\underline{v}^i, \overline{u}^{i+1}, 0) \mid (v,u) \in E, 1 \leq i \leq m\} \cup \\
& \{s, \overline{v}^1, 0) \mid v \in V\}.
\end{aligned}
$$

Here we denoted edges as a triplet (u, v, w) where w is the weight of the edge from u to v. We will alternate between this notation and notation of an edge as (u, v) with a weight function over all edges $w(u, v)$.

Lemma 3 *Every path in $G^{(P,H)}$ from s to $\underline{v}^m$ for some $v \in V$ is of the form $s, \overline{v}_1^1, \underline{v}_1^{i_1}, \overline{v}_2^{i_1+1}, \underline{v}_2^{i_2}, ..., \underline{v}_{k-1}^{i_{k-1}}$, $\overline{v}^{i_{k-1}+1}, \underline{v}^m$, with $1 \leq i_1 < i_2 < ... < i_{k-1} < m$.*

Proof: Let C be a path from s to some $\underline{v}^m$. The only edges leaving s enter vertices of the form $\overline{u}^1$ and all edges leaving vertices of the form $\underline{u}^i$ go to vertices of the form $\overline{w}^{i+1}$. All edges leaving vertices of the form $\overline{u}^i$ go to vertices of the form $\underline{u}^j$ for $j > i$. So the lemma follows. □

Lemma 4 *Let P be a pattern of length m and $1 \leq d \leq m$. There exists a pattern P' of length m with $Ham(P, P') = d$ such that P' matches in H iff there exists a path in $G^{(P,H)}$ from s to v^m, for some vertex $v \in V$, with path weight d.*

Proof: Assume that there exists a P' for which the lemma's conditions hold. Let $v_1, ..., v_k$ be the path in H on which P' matches beginning from location l, i.e. $P' = T'_{v_1} T_{v_2} T_{v_3} ... T_{v_{k-2}} T_{v_{k-1}} T''_{v_k}$, where $|T'_{v_1}| = |T_{v_1}| - l + 1$. It is straightforward to check that the following path exists and has weight d: $s, \overline{v}_1^1, \underline{v}_1^{|T'_{v_1}|}, \overline{v}_2^{|T'_{v_1}|+1}, \underline{v}_2^{|T'_{v_1}|+|T_{v_2}|}, ..., \overline{v}_i^{|T'_{v_1}|+\sum_{j=1}^{i-1}|T_{v_j}|+1}, \underline{v}_i^{|T'_{v_1}|+\sum_{j=1}^{i}|T_{v_j}|}, ..., \overline{v}_k^{m-|T''_{v_k}|+1}, \underline{v}_k^m$.

Conversely, according to Lemma 3 every path from s to $\underline{v}^m$ is of the form $s, \overline{v}_1^1, \underline{v}_1^{i_1}, \overline{v}_2^{i_1+1}, \underline{v}_2^{i_2}, ..., \underline{v}_{k-1}^{i_{k-1}}$, $\overline{v}_k^{i_{k-1}+1}, \underline{v}_k^m$ such that $1 \leq i_1 < i_2 < ... < i_{k-1} < m$. An analysis of the construction of $G^{(P,H)}$ shows that $(s, \overline{v}_1^1)$ and $(\underline{v}_j^{i_j}, \overline{v}_{j+1}^{i_j+1})$ are edges with weight 0 and that $(\overline{v}_j^{i_{j-1}+1}, \underline{v}_j^{i_j})$ are edges with weight $Ham(T_{v_j}, p_{i_{j-1}+1} ... p_{i_j})$ except for $(\overline{v}_1^1, \underline{v}_1^{i_1})$ and $(\overline{v}_k^{i_{k-1}+1}, \underline{v}_k^m)$ which have weight $Ham(Suf(T_{v_1}, i_1)$, $p_1 ... p_{i_1})$ and $Ham(Pref(T_{v_k}, m - i_{k-1}), p_{i_{k-1}+1} ... p_m)$, respectively.
Therefore, the overall weight $d = w(s, \overline{v}_1^1) + w(\overline{v}_1^1, \underline{v}_1^{i_1}) + w(\underline{v}_1^{i_1}, \overline{v}_2^{i_1+1}) + ... + w(\overline{v}_k^{i_{k-1}+1}, \underline{v}_k^m) = Ham(P$, $T'_{v_1} T_{v_2} T_{v_3} ... T_{v_{k-2}} T_{v_{k-1}} T''_{v_k})$, where $T'_{v_1} = Suf(T_{v_1}, i_1)$ and $T''_{v_k} = Pref(T_{v_k}, m - i_{k-1})$. So, choosing $P' = T'_{v_1} T_{v_2} T_{v_3} ... T_{v_{k-2}} T_{v_{k-1}} T''_{v_k}$ gives a pattern of length m that matches in H with $Ham(P, P') = d$.
□

The following algorithm is designed to output the minimum number of changes necessary to assure a P match. If desired, this can be adapted in a simple way to return P' that does match and is obtained by a minimum number of changes from P. Another modification of our algorithm can allow outputing, for every hypertext location, the number of changes to the pattern that are necessary in order for the pattern to match at that hypertext location.

Algorithm for Approximate Pattern Matching with Hamming Distance in Hypertext

1. For each vertex v in H for which $|T_v| \geq m$, build an array of the hamming distance between pattern P and each substring of the text $\$^{m-1} T_v \$^{m-1}$, where $\$ \notin \Sigma$.

For each vertex v in H for which $|T_v| \leq m$, build an array of the hamming distance between T_v, viewed as the pattern, and each substring of $\$^{|T_v|-1}P\$^{|T_v|-1}$, viewed as text, where $\$ \notin \Sigma$.

2. For each vertex with $|T_v| \geq m$ use the results of step 1 to find a substring of T_v of length m with least hamming distance from pattern P. Denote the distance $w(v)$.

3. Build $G^{(P,H)}$ from the pattern P and the hypertext H, using the results of step 1.

4. Do a depth first search in $G^{(P,H)}$ starting from vertex s finding the shortest path from s to $\underline{u}^m$ for some $u \in V$ and denote the length of the shortest (weighted) path w.

5. **Output:** $\min(\{w\} \cup \{w(v) \mid v \in V,\ |T_v| \geq m\})$.

Correctness: If the substring with shortest hamming distance from the pattern appears within a vertex then we will find it in step 2 and announce it in step 5. If the pattern crosses at least one hyperlink then by Lemma 4 there is a path of the form described in Lemma 4 with this weight. Since by Lemma 3 all paths from s to $\underline{v}^m$ for some $v \in V$ have this form which by Lemma 4 correspond to paths which model comparisons of P in the hypertext, it is sufficient to find the shortest path from s to some $\underline{v}^m$. This is exactly what we do in step 4. We implement it with a depth first search since $G^{(P,H)}$ is a DAG. In step 5 we announce the shortest.

Time: Assume the alphabet is unbounded. For each vertex, if $|T_v| > m$ then step 1 takes time $O(|T_v|\sqrt{m \log m})$ [1, 12] and if $m > |T_v|$ then step 1 takes time $O(m\sqrt{|T_v| \log m})$, for an overall time of $O((N\sqrt{m} + m\sqrt{N})\sqrt{\log m})$. If the alphabet is fixed, then by the Fischer-Patterson method [8] step 1 can be solved in time $O(N \log m)$. Step 2 is linear in the size of the text within the vertices, i.e. $O(N)$. Step 3 is linear in the size of $G^{(P,H)}$, which has a vertex set of size $O(|V|m)$ and edge set of size $O(|E|m)$. Step 4 is linear in the size of $G^{(P,H)}$ and step 5 costs $O(|V|)$. So, we have an overall complexity of $O((N\sqrt{m} + m\sqrt{N})\sqrt{\log m} + |E|m)$. ($O(N \log m + |E|m)$ over fixed bounded alphabets.)

5 Approximate Pattern Matching in Hypertext - Edit Distance

While it is true that mismatches are a common error in texts, often other errors occur such as accidently deleting a character or inserting a superfluous character. The minimal number of insertions, deletions and changes necessary to transfer one string into another is called the *edit distance* of these two strings. In this section we consider approximate pattern matching in hypertext with edit distance as our metric.

Similar to the previous section we have three versions to the problem and, not surprisingly similar results. We will first show that if the insertions, deletions and changes can be done only in the hypertext, then the problem is $\mathcal{NP}$-Complete. We then give an outline how to extend this result to the case when the errors occur both in the pattern and in the hypertext. Afterwards we present a polynomial algorithm for the case when errors occur in the pattern only. Note that the insertions or deletions are always on the characters whether in the pattern or in the hypertext, never on the structure of the hypertext. The underlying digraph always remains in its original form. Different from what we have seen up till now, it may be that after a deletion of a character of text in a vertex, the text is the empty word ϵ.

The *Approximate Hypertext Matching-Edit Distance in Hypertext* problem is defined as follows:
Input: A pattern P, a hypertext $H = (V, E, T)$ and a distance d.
Output: *True*, if with d error-corrections (insertions, deletions and changes) we can change the hypertext so that P matches in H. *False*, otherwise.

Proposition 3 *The Approximate Matching-Edit Distance in Hypertext problem is $\mathcal{NP}$-Complete.*

Proof: Clearly the problem is in $\mathcal{NP}$. We will reduce from directed hamiltonian path.

Let $G = (V, E)$ be a directed graph in which we seek a hamiltonian path and let $n = |V|$. Set $P = \$\sigma_1\#...\$\sigma_n\#$, where $\forall i \neq j,\ \sigma_i \neq \sigma_j$ and $\$, \# \notin \{\sigma_1, ..., \sigma_n\}$ and set the hypertext $H = (V, E, T)$ where $T_v = \$\#$ for every $v \in V$. Set the distance $d = n$. If we have a hamiltonian path then by inserting σ_i in between the \$ and the # of the i-th vertex will give us a match of P in the hypertext with n error-corrections.

Conversely, assume we have a match of P in H with at most n error-corrections on H. Since $\sigma_1, ..., \sigma_n$ do not appear in the hypertext before the error-corrections and must appear after the error-corrections and since $\forall i \neq j,\ \sigma_i \neq \sigma_j$ it must be the case that the n error-corrections are either changes or insertions intended for inserting $\sigma_1, ..., \sigma_n$ into the hypertext.

Consider the path C in the hypertext that P matches upon after the error-corrections. If this path is not hamiltonian then one of the following must be true: (a) there is a vertex that does not appear on C, (b) there is a vertex that appears twice, not including the first or last vertex on the path or (c) the first or last vertex appear a second time along the path.

If (case (a)) there is a vertex that does not appear on C then there must be a different vertex in which both σ_i and σ_{i+1} appear. But this other vertex must contain text $\sigma_i\#\$\sigma_{i+1}$. This cannot be, because at least $n-2$ error corrections are necessary for the other vertices and four corrections are necessary for the vertex containing $\sigma_i\#\$\sigma_{i+1}$.

If (case (b)) there is a vertex in C that appears twice, this vertex may not contain σ_j for any j. But then there must be a vertex with σ_i and σ_{i+1} which, as in case (a), is impossible. Case (c) is also impossible following similar reasoning. Therefore, C is a hamiltonian path. □

If we modify the definition of the problem to allow errors in the hypertext and the pattern then it is once again $\mathcal{NP}$-Complete. This can be proved reducing from the directed hamiltonian cycle problem with pattern $P = \$\sigma_1\#...\$\sigma_n\#\$\sigma_1\#...\$\sigma_n\#$, using a similar claim to Proposition 2 to show that there cannot be changes in the pattern and then the rest is similar to the proof of Proposition 3.

We now consider the version with errors in the pattern only. For this problem we present a polynomial algorithm that, once again, extends the previous ideas. We start with a formal definition.

The *Approximate Hypertext Matching-Edit Distance in Pattern* problem is defined as follows:
Input: A pattern P and a hypertext $H = (V, E, T)$.
Output: The minimum d such that with d error-corrections (insertions, deletions and changes) in the pattern, P, it will match in H.

The best known algorithms [13] for approximate pattern matching with edit distance in regular text have complexity of $O(nm)$, where n is the text length and m is the pattern length. In a hypertext a vertex may contain $O(N)$, where N is, as before, the overall size of the text in the vertices. So applying regular approximate pattern matching techniques would cost $O(Nm)$ for this vertex alone. Therefore, for simplicity, without sacrificing efficiency, we will turn the hypertext into a one-character hypertext.

As in section 2, for each vertex v in the hypertext the digraph contains the vertex set $\{v^i \mid 1 \leq i \leq m\}$, where v^i represents the comparison of T_v and p_i. The edges will be weighted in a fashion similar to the previous section. There will be edges with weight 0 to account for an exact match and edges with weight 1 to account for mismatch. We also add edges with weight 1 to account for insertion and deletion of a character.

We now describe the edges in the digraph. We have (a) edges describing an exact match of ith pattern character, (b) edges describing a mismatch at location i, (c) edges describing insertion of a character to the pattern before pattern location i, (d) edges describing deletion of the ith character of the pattern and (e) edges from the start vertex s to all the first location vertices.

Formally,

$$
\begin{array}{rl}
V^{(P,H)} = & \{v^i \mid v \in V, 1 \leq i \leq m\} \cup \{s\} \cup \{f\} \\[1ex]
E^{(P,H)} = & (a)\ \{(u^i, v^{i+1}, 0) \mid (u,v) \in E, T_u = p_i\} \cup \\
 & \quad \{(u^m, f, 0) \quad \mid T_u = p_m\} \cup \\
 & (b)\ \{(u^i, v^{i+1}, 1) \mid (u,v) \in E, T_u \neq p_i\} \cup \\
 & (c)\ \{(u^i, v^i, 1) \quad \mid (u,v) \in E,\ 2 \leq i \leq m\} \cup \\
 & (d)\ \{(v^i, v^{i+1}, 1) \mid v \in V,\ 1 \leq i \leq m-1,\ T_v \neq p_i or (v,v) \notin E\} \cup \\
 & \quad \{(v^m, f, 1) \quad \mid T_v \neq p_m\} \cup \\
 & (e)\ \{(s, v^1, 0) \qquad \mid v \in V\}.
\end{array}
$$

Lemma 5 *Let P be a pattern of length m and $1 \leq d \leq m$. There exists a pattern P' formed by d error-corrections of P such that P' matches in H iff there exists a path in $G^{(P,H)}$ from s to f with path weight d.*

Proof: Let $P = p_1 \cdots p_m,\ P' = p'_1 \cdots p'_k$. The proof of the lemma follows immediately from the following observation.

Let $e_1, ..., e_d$ be the edit operations performed on P to achieve P'. Denote each e_i by a triple $\langle j_i, E_i, \sigma_i \rangle$, where $j_i \in \{1, ..., m\}$, $\sigma_i \in \Sigma$, and $E_i \in \{M, I, D\}$. The meaning of E_i is:

$$
E_i = \begin{cases} M, & \text{means "change } p_{j_i} \text{ to } \sigma_i\text{" (mismatch);} \\ I, & \text{means "insert } \sigma_i \text{ before } p_{j_i}\text{";} \\ D, & \text{means "delete } p_{j_i}\text{".} \end{cases}
$$

Let $\#_i^I$ be the number of E_ℓ's that are I, for $\ell = 1, ..., i$. Let $\#_i^D$ be the number of E_ℓ's that are D, for $\ell = 1, ..., i$, and let $\#_i = \#_i^I - \#_i^D$.

Without loss of generality, assume that $e_1, ..., e_d$ are sorted by increasing values of j.

By definition of edit distance, the following is true:

$p_1 \cdots p_{j_1 - 1} = p'_1 \cdots p'_{j_1 - 1}$
If $E_1 = I$ then $p'_{j_1} = \sigma_1$ and $p'_{j_1+1} = p_{j_1}$.
If $E_1 = M$ then $p'_{j_1} = \sigma_1$.

$p_{j_1+1} \cdots p_{j_2-1} = p'_{j_1+\#_1+1} \cdots p'_{j_2+\#_1-1}$

.

.

.

If $E_i = I$ then $p'_{j_i+\#_{i-1}} = \sigma_i$ and $p'_{j_i+\#_{i-1}+1} = p_{j_i}$.
If $E_i = M$ then $p'_{j_i+\#_{i-1}} = \sigma_i$.
$p_{j_i+1} \cdots p_{j_{i+1}-1} = p'_{j_i+\#_i+1} \cdots p'_{j_{i+1}+\#_i-1}$

.

.

.

If $E_d = I$ then $p'_{j_d+\#_{d-1}} = \sigma_d$ and $p'_{j_d+\#_{d-1}+1} = p_{j_d}$.
If $E_d = M$ then $p'_{j_d+\#_{d-1}} = \sigma_d$.
$p_{j_d+1} \cdots p_m = p'_{k+j_d+1-m} \cdots p'_k$.

Assume now that P' matches in H. Then there exists a path $v_1, ..., v_k$ in H where P' matches, i.e. $P' = T_{v_1} T_{v_2} \cdots T_{v_k}$.

The following path in $G^{(P,H)}$ starts at s, ends at f and has weight d. Start with type (e) edge $(s, v_1^1, 0)$ followed by j_1 type (a) edges $(v_i^i, v_{i+1}^{i+1}, 0)$, for $i = 1, ..., j_1 - 1$.

Inductively assume that we have constructed a path of weight $i-1$ until $p'_{j_i+\#_{i-1}-1}$. The following three cases are possible:

1. $E_i = M$. Follow one type (b) edge $(v^{j_i}_{j_i+\#_{i-1}}, v^{j_i+1}_{j_i+\#_i+1}, 1)$.

2. $E_i = I$. Follow one type (c) edge $(v^{j_i}_{j_i+\#_{i-1}}, v^{j_i}_{j_i+\#_i}, 1)$, and one type (a) edge $(v^{j_i}_{j_i+\#_i}, v^{j_i+1}_{j_i+\#_i+1}, 0)$.

3. $E_i = D$. Follow one type (d) edge $(v^{j_i}_{j_i+\#_{i-1}}, v^{j_i+1}_{j_i+\#_i}, 1)$.

Continue with $j_{i+1} - j_i - 1$ type (a) edges, $(v^{\ell}_{\ell+\#_\ell}, v^{\ell+1}_{\ell+\#_\ell+1}, 0)$, $\ell = j_i + 1, ..., j_{i+1} - 1$ until the next edit operation.

For the final operation, end it with a type (a) edge to f if $j_d \neq m$, and with a single type (d) edge to f if $j_d = m$.

The converse is similar. Simply construct the edit operations from the path in $G^{(P,H)}$. □

Algorithm for Approximate Hypertext Matching - Edit Distance in Pattern

1. Create a one-character hypertext $H' = (V', E', T')$ from H.

2. Build $G^{(P,H')}$ from the pattern P and the one-character hypertext H'.

3. Find the shortest path from s to f and announce its weighted length.

Correctness: Follows directly from Lemma 5.

Time: Step 1 takes time linear to H' and step 2 time linear to $G^{(P,H')}$. $|V'| = O(|V| + N) = O(N)$ and $|E'| = O(|E| + N)$. $G^{(P,H')}$ has vertex size $O(|V'|m) = O(mN)$ and edge size $O(m|E'|) =$

$O(mN + m|E|)$. Using Dijkstra's algorithm implemented with Fibonacci heaps the third step can be done in $O(Nm \log N + |E|m)$ time [7].

References

[1] K. Abrahamson. Generalized String Matching. *SIAM J. Computing* **16 (6)**, 1039-1051, 1987.

[2] T. Akutsu. A Linear Time Pattern Matching Algorithm Between a String and a Tree. *Proceedings of the 4th Symposium on Combinatorial Pattern Matching*, 1-10, Padova, Italy, 1993.

[3] A. Amir, M. Farach, R. Giancarlo, Z. Galil, and K. Park. Dynamic dictionary matching. *Journal of Computer and System Sciences*, 49(2):208–222, 1994.

[4] A. Amir, M. Farach, R.M. Idury, J.A. La Poutré, and A.A Schäffer. Improved dynamic dictionary matching. *Information and Computation*, 119(2):258–282, 1995.

[5] A. Aviad. HyperTalmud: a hypertext system for the Babylonian Talmud and its commentaries. Dept. of Math and Computer Science, Bar-Ilan University, 1993.

[6] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.

[7] T.H. Cormen, C.E. Leiserson and R.L. Rivest. *Introduction to Algorithms* The MIT Press, Cambridge, Massachusetts, 1990.

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

[9] P. Ferragina and R. Grossi. Optimal on-line search and sublinear time update in string matching. *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 531–540, 1995.

[10] A. S. Fraenkel and S. T. Klein. Information Retrieval from Annotated Texts. TR-95-25, Dept. of Applied Math and Computer Science, The Weizmann Institute of Science, 1995.

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

[12] S. Rao Kosaraju. Efficient String Matching. Manuscript, 1987.

[13] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms* **10 (2)**, 157-169, 1989.

[14] U. Manber and S. Wu. Approximate string matching with arbitrary costs for text and hypertext. *IAPR Workshop on structural and syntactic pattern recognition*, Bern, Switzerland, 1992.

[15] J. Nielsen. *Hypertext and Hypermedia.* Academic Press Professional, Boston, 1993.

[16] K. Park and D.K. Kim. String matching in hypertext. *6th Symposium on Combinatorial Pattern Matching*, Helsinki, Finland, 1995.

[17] S. C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. *Proc. 36th FOCS*, 320-328, 1996.